УЗБЕКСКОЕ АГЕНТСТВО СВЯЗИ И ИНФОРМАТИЗАЦИИ

ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра информационных технологий

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ ПО КУРСУ «БАНКИ И БАЗЫ ДАННЫХ»

Для студентов бакалавриата направления

5521900 «Информатика и информационные технологии»

Фергана 2011

УЗБЕКСКОЕ АГЕНТСТВО СВЯЗИ И ИНФОРМАТИЗАЦИИ

ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Расулов А.М., Лазарева М.В.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ ПО КУРСУ «БАНКИ И БАЗЫ ДАННЫХ»

Фергана 2011

Расулов А.М., Лазарева М.В.
Методические указания к лабораторным работам по курсу «Информационные системы». Ташкентский университет информационных технологий. Ф: 2010, 112 с.

Под редакцией ***к.ф-.м.н., доцент Абдукадыров А.А.***

Данная работа предназначена для изучения основ построения информационных систем в различных средах, основы баз данных, проектирование баз данных, технология создания информационных систем и работа с удаленными базами данных. Пособие может быть полезным для студентов бакалавриата направления 5521900 «Информатика и информационные технологии», а также для преподавателей, аспирантов и магистров соответствующего направления.

Составители:

Расулов Акбарали Махаматович	*-кандидат физико-математических наук, доц. кафедры Информационные технологий Ферганского филиала ТУИТ*
Лазарева Марина Викторовна	*ассистент кафедры Прикладная информатика Ферганского филиала ТУИТ*

Рецензенты:

к.т.н. Джалилов М.Л.	*- зав.кафедры Прикладная информатика Ферганского филиала ТУИТ*
к.т.н., доц. Полвонов Ф.Ю.	*-зам. директор по УР и НИР Ферганского филиала ТУИТ*

Методические указания к лабораторным работам по курсу «Информационные системы» утвержден учебно-методическим советом факультета Информационных технологий Ферганский филиал Ташкентского университета информационных технологий (протокол №1 от 27 августа 2010г.)

ОГЛАВЛЕНИЕ

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ НЕОБХОДИМЫЕ ДЛЯ ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

ВВЕДЕНИЕ

Программное обеспечение за полвека своего существования претерпело огромные изменения: от программ, способных выполнять только простейшие логические и арифметические операции до сложных систем управления предприятиями. В развитии программного обеспечения всегда можно было выделить два основных направления:

- выполнение вычислений;
- накопление и обработка информации.

Хотя первоначально компьютеры предназначались главным образом для выполнения сложных математических расчетов, в настоящее время доминирующим является второе направление. Такое перераспределение основных функции, выполняемых вычислительной техникой, вполне понятно - гражданский бизнес гораздо более распространен, чем военные и научные вычисления, а снижение стоимости компьютеров сделало их доступными для совсем небольших предприятий и даже частных лиц.

Сегодня управление предприятием без компьютера просто немыслимо. Компьютеры давно и прочно вошли в такие области управления, как бухгалтерский учёт, управление складом, ассортиментом и закупками. Однако современный бизнес требует гораздо более широкого применения информационных технологий в управлении предприятием. Жизнеспособность и развитие информационных технологий объясняется тем, что современным бизнес крайне чувствителен к ошибкам в управлении. Интуиции, личного опыта руководителя и размеров капитала уже мало для того, чтобы быть первым. Для принятия любого грамотного управленческого решения в условиях неопределенности и риска необходимо постоянно держать под контролем различные аспекты финансово - хозяйственной деятельности, будь то: торговля, производство или предоставление каких-либо услуг. Поэтому современный подход к управлению предполагает вложение средств в информационные технологии. И чем крупнее предприятие, тем серьезнее должны быть подобные вложения. Они являются жизненной необходимостью в жесткой конкурентной борьбе, одержать победу сможет лишь тот, кто лучше оснащен и наиболее эффективно организован.

Поэтому студенты направления «Информатика и информационные технологии» должны хорошо знать методы проектирования баз данных, обладать знаниями создания баз данных и интерфейса пользователя. Они должны уметь пользоваться этими знаниями в своей практической деятельности.

Хотя информационные системы являются обычным программным продуктом, они имеют ряд существенных отличий от стандартных прикладных программ и систем.

В зависимости от предметной области информационные системы могут очень сильно различаться по своим функциям, архитектуре, реализации. Однако можно выделить ряд свойств, которые являются общими:

- информационные системы предназначены для сбора, хранения и обработки информации. Поэтому в основе любой из них лежит среда хранения и доступа к данным;

- информационные системы ориентируются на конечного пользователя, не обладающего высокой квалификацией в области применения вычислительной техники. Поэтому клиентские приложения информационной системы должны обладать простым, удобным, легко осваиваемым интерфейсом, который предоставляет конечному пользователю все необходимые для работы функции, но в то же время не даёт ему возможность выполнять какие-либо лишние действия.

Таким образом, при разработке информационной системы приходится решать две основные задачи:

- задачу разработки БД, предназначенной для хранения информации;

- задачу разработки графического интерфейса пользователя клиентских приложений.

В данной методической пособие рассматриваются оба аспекта разработка информационных систем.

1. ВВЕДЕНИЕ В БАЗЫ ДАННЫХ

1.1. Основные понятия

Для успешного функционирования различных организаций требуется наличие развитой информационной системы, которая реализует автоматизированный сбор, обработку и манипулирование данными.

1.1.1. Банки данных

Современной формой информационных систем являются банки данных, включающие в свой состав следующие составляющие:

- вычислительную систему;
- систему управления базами данных (СУБД);
- одну или несколько баз данных (БД);
- набор прикладных программ (приложений БД).

База данных обеспечивает хранение информации, а также удобный и быстрый доступ к данным. Она представляет собой совокупность данных различного характера, организованных по определенным правилам. Информация в БД должна быть:

- непротиворечивой;
- неизбыточной;
- целостной.

Система управления базой данных (СУБД) – это совокупность языковых и программных средств, предназначенных для создания, ведения и использования БД. По характеру применения СУБД разделяют на персональные и многопользовательские.

Персональная СУБД обеспечивает возможность создания локальных БД, работающих на одном компьютере. К персональным СУБД относятся *Paradox, DBase, FoxPro, Access* и др.

Многопользовательские СУБД позволяют создавать информационные системы, функционирующие в архитектуре "клиент-сервер". Наиболее известными многопользовательскими СУБД являются *Oracle, Informix, Sybase, Microsoft SQL Server, InterBase.*

В состав языковых средств современных СУБД входят:

- язык описания данных, предназначенный для описания логической структуры данных;
- язык манипулирования данными, обеспечивающий выполнение основных операций над данными - ввод, модификацию и выборку;
- язык структурированных запросов (SQL – Structured Query Language), обеспечивающий управление структурой БД и манипулирование данными, а так же являющийся стандартным средством доступа к удаленным БД;
- язык запросов по образцу (QBE – Query By Example), обеспечивающий визуальное конструирование запросов к БД.

Прикладные программы, или приложения, служат для обработки данных,

содержащихся в БД. Пользователь осуществляет управление БД и работу с ее данными именно с помощью приложений, которые также называют *приложениями БД.*

Иногда термин "база данных" трактуют в более широком смысле и обозначают им не только саму БД, но и приложения, обрабатывающие ее данные.

Хотя система Delphi и не является СУБД в буквальном смысле этого слова, она, тем не менее, обладает вполне развитыми возможностями СУБД. Предоставляемые Delphi средства обеспечивают создание и ведение локальных и клиент - серверных БД, а также разработку приложений для работы практически с любыми БД. Назвать Delphi обычной СУБД мешает, наверное, только то, что, с одной стороны, она не имеет своего формата таблиц (языка описания данных) и использует форматы таблиц других СУБД, например, dBase, Paradox или InterBase. Это вряд ли является недостатком, поскольку названные форматы хорошо себя зарекомендовали. С другой стороны, в плане создания приложений различного назначения, в том числе приложений БД, возможности Delphi не уступают возможностям специализированных СУБД, а зачастую и превосходят их.

1.1.2. Организация баз данных

База данных содержит данные, используемые некоторой прикладной информационной системой (например, системами "Сирена" или "Экспресс" продажи авиа и железнодорожных билетов). В зависимости от вида организации данных различают следующие основные модели представления данных в базе:

- иерархическую;
- сетевую;
- реляционную;
- объектно-ориентированную.

В иерархической модели данные представляются в виде древовидной (иерархической) структуры. Подобная организация данных удобна для работы с иерархически упорядоченной информацией, однако при оперировании данными со сложными логическими связями иерархическая модель оказывается слишком громоздкой.

В сетевой модели данные организуются в виде произвольного графа. Недостатком сетевой модели является жесткость структуры и высокая сложность ее реализации.

Кроме того, значительным недостатком иерархической и сетевой моделей является также то, что структура данных задается на этапе проектирования БД и не может быть изменена при организации доступа к данным.

В объектно-ориентированной модели отдельные записи базы данных представляются в виде объектов. Между записями базы данных и функциями их обработки устанавливаются взаимосвязи с помощью механизмов, подобных соответствующим средствам в объектно-ориентированных языках

программирования. Объектно-ориентированные модели сочетают особенности сетевой и реляционной моделей и используются для создания крупных БД со сложными структурами данных.

Реляционная модель получила свое название от английского термина relation (отношение) и была предложена в 70-х годах сотрудником фирмы IBM Эдгаром Коддом. Реляционная БД представляет собой совокупность таблиц, *связанных отношениями*. Достоинствами реляционной модели данных являются простота, гибкость структуры, удобство реализации на компьютере, наличие теоретического описания. Большинство современных БД для персональных компьютеров являются реляционными. При последующем изложении материала речь пойдет именно о реляционных БД.

1.1.3. Архитектуры информационных систем

В зависимости от взаимного расположения приложения и БД можно выделить:

- локальные БД;
- удаленные БД.

Для выполнения операций с локальными БД разрабатываются и используются так называемые локальные приложения, а для операций с удаленными БД – клиент - серверные приложения.

Расположение БД в значительной степени влияет на разработку приложения, обрабатывающего содержащиеся в этой базе данные. Delphi - приложение осуществляет доступ к БД через BDE (Borland Database Engine - процессор баз данных фирмы Borland). BDE представляет собой совокупность динамических библиотек и драйверов, обеспечивающих доступ к данным. BDE должен устанавливаться на всех компьютерах, на которых выполняются Delphi-приложения, осуществляющие работу с БД. Приложение через BDE передает запрос к базе данных, а обратно получает требуемые данные.

Локальные БД располагаются на том же компьютере, что и работающие с ними приложения. В этом случае говорят, что информационная система имеет локальную архитектуру (рис. 1.1.1). Работа с БД происходит, как правило в *однопользовательском* режиме. При необходимости можно запустить на компьютере другое приложение, одновременно осуществляющее доступ к этим же данным. Для управления совместным доступом к БД необходимы специальные средства контроля и защиты. Эти средства могут понадобиться, например, в случае, когда приложение пытается изменить запись, которую редактирует другое приложение. Каждая разновидность БД осуществляет подобный контроль своими способами и обычно имеет встроенные средства разграничения доступа.

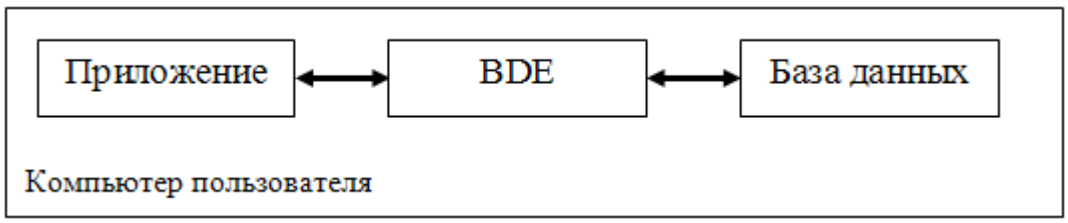


Рис. 1.1.1. Локальная архитектура.

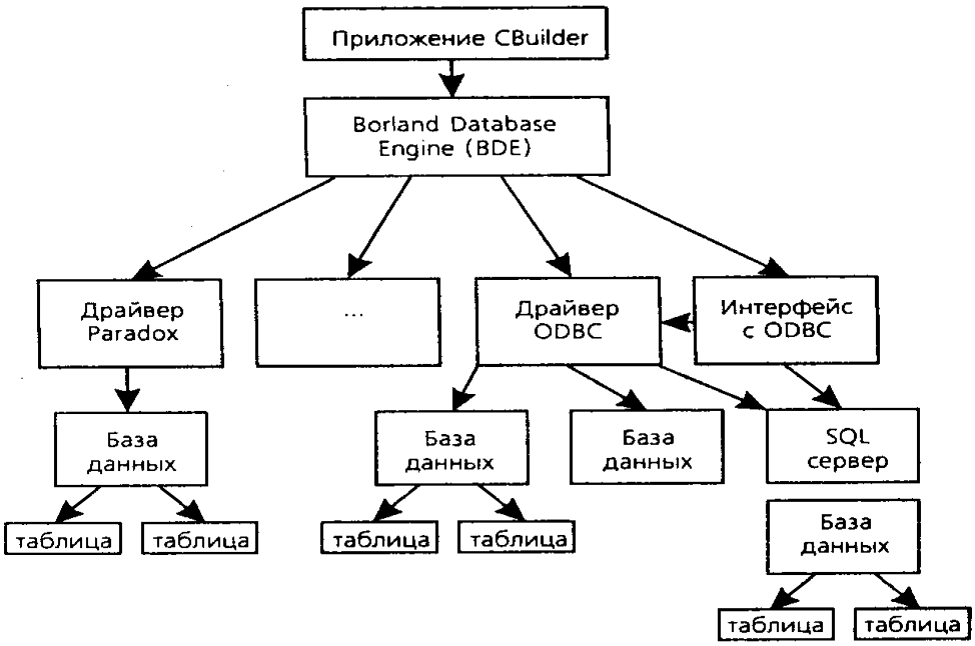


Рис. 1.1.2. Локальная и глобальная архитектура.

Для доступа к локальной БД процессор баз данных BDE использует стандартные драйверы, которые позволяют работать с форматами БД dBase, Paradox, FoxPro, а также с текстовыми файлами.

При использовании локальной БД в сети возможна организация многопользовательского доступа. В этом случае файлы БД и предназначенное для работы с ней приложение располагаются на сервере сети. Каждый пользователь запускает со своего компьютера это расположенное на сервере приложение, при этом у него запускается копия приложения. Такой сетевой вариант использования локальной БД соответствует архитектуре "файл-сервер". Приложение при архитектуре "файл-сервер" также может быть записано и на каждый компьютер сети, в этом случае приложению отдельного компьютера должно быть известно местонахождение общей БД (рис. 1.1.2, 1.1.3).

При работе с данными на каждом пользовательском компьютере сети используется локальная копия БД. Эта копия периодически обновляется данными, содержащимися в БД на сервере.

Архитектура "файл-сервер" обычно применяется в сетях с небольшим количеством пользователей, для ее реализации подходят персональные СУБД, например, Paradox или dBase. Достоинствами этой архитектуры являются

простота реализации, а также то, что приложение фактически разрабатывается в расчете на одного пользователя и не зависит от компьютера сети, на который оно устанавливается.

Однако архитектура "файл-сервер" имеет и существенные недостатки.

- Пользователь работает со своей локальной копией БД, данные в которой обновляются при каждом запросе к какой-либо из таблиц. При этом с сервера пересылается новая копия всей таблицы, данные, которой затребованы. Таким образом, если пользователю необходимо несколько записей таблицы, с сервера по сети пересылается вся таблица. В результате циркуляции в сети больших объемов избыточной информации *резко возрастает нагрузка на сеть*, что приводит к соответствующему снижению ее быстродействия и производительности информационной системы в целом.

- В связи с тем, что на каждом компьютере имеется своя копия БД, изменения, сделанные в ней одним пользователем, в течение некоторого времени являются неизвестными другим пользователям. Поэтому требуется постоянное обновление БД. Кроме того, возникает *необходимость синхронизации* работы отдельных пользователей, связанная с блокировкой в таблицах записей, которые в данный момент редактирует другой пользователь.

- Управление БД осуществляется с разных компьютеров, поэтому в значительной степени затруднена *организация контроля доступа*, соблюдения конфиденциальности и поддержания целостности БД.

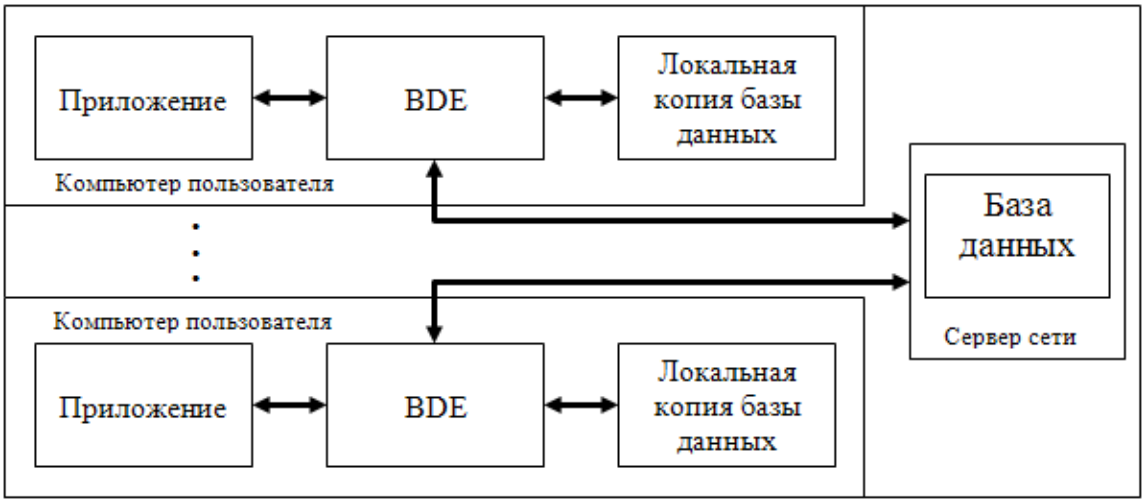


Рис. 1.1.3. Архитектура «Файл сервер».

Удаленная БД размещается на компьютере-сервере сети, а приложение, осуществляющее работу с этой БД, находится на компьютере пользователя. В этом случае мы имеем дело с архитектурой "клиент-сервер" (рис. 1.1.4), когда информационная система делится на неоднородные части - сервер и клиент БД. В связи с тем, что компьютер-сервер находится отдельно от клиента, его также называют *удаленным сервером*.

Клиент - это приложение пользователя. Для получения данных клиент формирует и отсылает запрос удаленному серверу, на котором размещена БД.

Запрос формулируется на языке SQL, который является стандартным средством доступа к серверу при использовании реляционных моделей данных. После получения запроса удаленный сервер направляет его SQL-серверу (серверу баз данных) – специальной программе, управляющей удаленной БД и обеспечивающей выполнение запроса и выдачу его результатов клиенту.

Таким образом, в архитектуре "клиент-сервер" клиент посылает запрос на предоставление данных и получает только те данные, которые действительно были затребованы. Вся обработка запроса выполняется на удаленном сервере. Такая архитектура обладает следующими достоинствами:

- Снижение нагрузки на сеть, поскольку теперь в ней циркулирует только нужная информация.
- Повышение безопасности информации, связанное с тем, что обработка запросов всех клиентов выполняется единой программой, расположенной на сервере. Сервер устанавливает общие для всех пользователей правила использования БД, управляет режимами доступа клиентов к данным, запрещая, в частности, одновременное изменение одной записи различными пользователями.
- Уменьшение сложности клиентских приложений за счет отсутствия в них кода, связанного с контролем БД и разграничением доступа к ней.

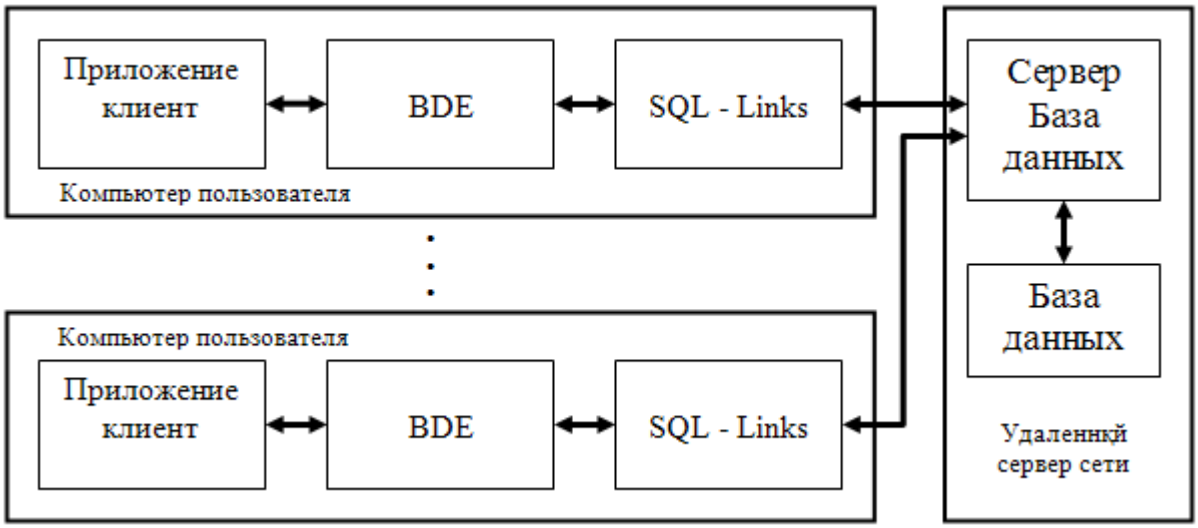


Рис. 1.1.4. Архитектура «клиент-сервер» («толстый» клиент).

Для реализации архитектуры "клиент-сервер" обычно используются многопользовательские СУБД, например, Oracle или Microsoft SQL Server. Подобные СУБД также называют промышленными, т. к. они позволяют создать информационную систему организации или предприятия с большим числом пользователей. Промышленные СУБД являются сложными системами и требуют мощной вычислительной техники и соответствующего обслуживания. Обслуживание выполняет специалист (или группа специалистов), называемый системным администратором БД (администратором). Основными задачами системного администратора являются:

– защита БД;
– поддержание целостности БД;
– обучение и подготовка пользователей;
– загрузка данных из других БД;
– тестирование данных;
– резервное копирование и восстановление;
– внесение изменений в информационную систему.

Доступ Delphi-приложения к промышленным СУБД осуществляется через драйверы SQL-Links. Отметим, что при работе с "родной" для Delphi СУБД InterBase можно обойтись без драйверов SQL-Links.

Описанная архитектура является двухуровневой – приложение клиент и сервер БД. Клиентское приложение также называют *сильным*, или "толстым", клиентом. Дальнейшее развитие данной архитектуры привело к появления трехуровневого варианта "клиент-сервер" – приложение-клиент, сервер приложений и сервер БД (рис. 1.1.5).

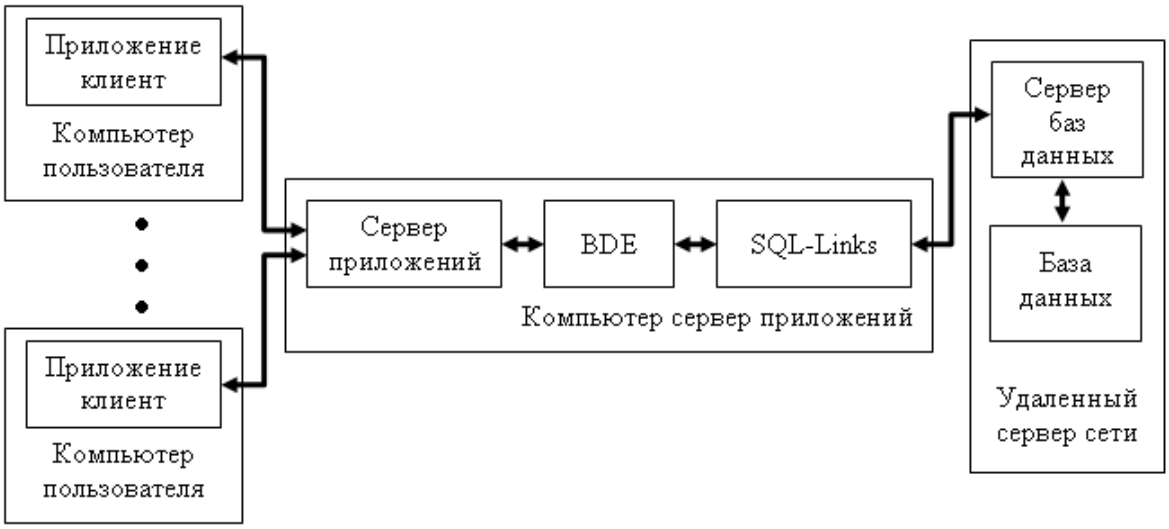


Рис. 1.1.5. Трехуровневая архитектура «клиент-сервер» («тонкий» клиент).

В трехуровневой архитектуре часть средств и кода, предназначенных для организации доступа к данным и их обработке, из приложения-клиента выделяется в сервер приложений. Само клиентское приложение при этом называют *слабым*, или "тонким", клиентом. В сервере приложений удобно располагать средства и код, общие для всех клиентских приложений, например, средства доступа к БД.

Основные достоинства трехуровневой архитектуры "клиент-сервер" состоят в следующем:

– разгрузка сервера от выполнения части операций, перенесенных на сервер приложений;
– уменьшение размера клиентских приложений за счет разгрузки их от лишнего кода;

– единое поведение всех клиентов;
– упрощение настройки клиентов – при изменении общего кода сервера приложений автоматически изменяется поведение приложений-клиентов.

Отметим, что локальные приложения БД называют *одноуровневыми*, а клиент - серверные приложения БД – *многоуровневыми.*

1.2. Реляционные базы данных

Реляционная БД состоит из взаимосвязанных таблиц. Каждая таблица содержит информацию об объектах одного типа, а совокупность всех таблиц образует единую БД.

1.2.1.Таблицы баз данных

Таблицы, образующие БД, находятся в каталоге (папке) на жестком диске. Таблицы хранятся в файлах и похожи на отдельные документы или электронные таблицы (например, табличного процессора Microsoft Excel), их можно перемещать и копировать обычным способом, скажем, с помощью Проводника Windows. Однако, в отличие от документов, таблицы БД поддерживают *многопользовательский* режим доступа, т. е. могут одновременно использоваться несколькими приложениями.

Для одной таблицы создается несколько файлов, содержащих данные, индексы, ключи и т. п. Главным из них является файл с данными, имя этого файла становится именем таблицы, которое задается при ее создании. В некотором смысле понятие таблицы и ее главного файла являются синонимами, при выборе таблицы выбирается именно ее главный файл: для таблицы dBase - это файл с расширением DBF, а для таблицы Paradox - с расширением DB. Имена остальных файлов таблицы назначаются автоматически - все файлы имеют одинаковое имя, совпадающее с именем таблицы, и разные расширения, указывающие на содержимое соответствующего файла. Расширения файлов приведены в данной главе ниже в разделе *"Таблицы формата dBase и Paradox"*. Каждая таблица БД состоит из строк и столбцов и предназначена для хранения данных об однотипных объектах информационной системы. Строка таблицы называется записью, столбец таблицы - полем (рис. 1.2.1). Каждое поле должно иметь уникальное в пределах таблицы имя.

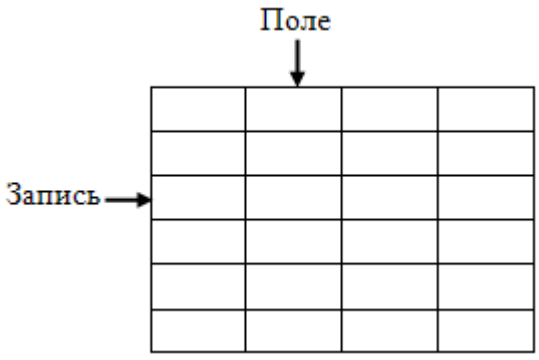


Рис. 1.2.1. Таблица базы данных.

Поле содержит данные одного из допустимых типов, например, строкового, целочисленного или даты. При вводе значения в поле таблицы автоматически производится проверка соответствия типа значения и типа поля. В случае, когда эти типы не совпадают, а преобразование типа значения невозможно, генерируется исключительная ситуация.

Особенности организации таблиц зависят от конкретной СУБД, используемой для создания и ведения БД. Например, в локальной таблице dBase и в таблице сервера InterBase нет поля автоинкрементного типа (с автоматически наращиваемым значением), а в таблице dBase нельзя определить ключ. Подобные особенности необходимо учитывать при выборе типа (формата) таблицы, т. к. они влияют не только на организацию БД, но и на построение приложения для работы с этой БД. Однако, несмотря на все различия таблиц, существуют общие правила создания и ведения БД, а также разработки приложений, которые и будут далее рассмотрены.

Основу таблицы составляет описание ее полей, каждая таблица должна иметь хотя бы одно поле. Понятие структуры таблицы является более широким и включает в себя:

- описание полей;
- ключ;
- индексы;
- ограничения на значения полей;
- ограничения ссылочной целостности между таблицами;
- пароли.

Иногда ограничения на значения полей, ограничения ссылочной целостности между таблицами, а также права доступа называют одним общим термином "ограничения".

Отметим, что отдельные элементы структуры зависят от формата таблиц, например, для таблиц dBase нельзя задать ограничения ссылочной целостности (т. к. у них нет ключей). Все элементы структуры задаются на физическом уровне (уровне таблицы) и действуют для всех программ, выполняющих операции с БД, включая средства разработки и ведения БД (например, программу Database Desktop). Многие из этих элементов (например, ограничения на значения полей или поля просмотра) можно также реализовать в приложении программно, однако в этом случае они действуют только в пределах своего приложения.

С таблицей в целом можно выполнять следующие операции:

- создание (определение структуры);
- изменение структуры (реструктуризация);
- переименование;
- удаление.

При создании таблицы задается структура и имя таблицы. При сохранении на диске создаются все необходимые файлы, относящиеся к таблице. Их имена совпадают с именем таблицы.

При изменении структуры таблицы в ней могут измениться названия и

характеристики полей, состав и наименования ключа и индексов, ограничения. Однако название таблицы и ее файлов остается прежним.

При переименовании таблица получает новое имя, в результате чего новое имя также получают все ее файлы. Для этого используются соответствующие программы (утилиты), предназначенные для работы с таблицами БД, например, Database Desktop или Data Pump.

При удалении таблицы с диска удаляются все ее файлы. В отличие от переименования, удаление таблицы можно выполнить посредством любой программы (в том числе и с помощью Проводника Windows).

1.2.2. Ключи и индексы

Ключ представляет собой комбинацию полей, данные в которых однозначно1 определяют каждую запись в таблице. Простой ключ состоит из одного поля, а составной (сложный) - из нескольких полей. Поля, по которым построен ключ, называют *ключевыми*. В таблице может быть определен только один ключ. Ключ обеспечивает:

- однозначную идентификацию записей таблицы;
- ускорение выполнения запросов к БД;
- установление связи между отдельными таблицами БД;
- использование ограничений ссылочной целостности.

Ключ также называют *первичным ключом* или *первичным (главным) индексом.*

Информация о ключе может храниться в отдельном файле или совместно с данными таблицы. Например, в БД Paradox для этой цели используется отдельный файл (ключевой файл или файл главного индекса) с расширением PX. В БД Access вся информация содержится в одном общем файле с расширением MDB. Значения ключа располагаются в определенном порядке. Для каждого значения ключа имеется уникальная ссылка, указывающая на расположение соответствующей записи в таблице (в главном ее файле). Поэтому при поиске записи выполняется не последовательный просмотр всей таблицы, а прямой доступ к записи на основании упорядоченных значений ключа.

Ценой, которую разработчик и пользователь платят за использование такой технологии, является увеличение размера БД вследствие необходимости хранения значений ключа, например, в отдельном файле. Размер этого файла зависит не только от числа записей таблицы (что достаточно очевидно), но и от полей, составляющих ключ. В ключевом файле, кроме ссылок на соответствующие записи таблицы, сохраняются, и значения самих ключевых полей. Поэтому при вхождении в состав ключа длинных строковых полей размер ключевого файла может оказаться соизмеримым с размером файла с данными таблицы. Таблицы различных форматов имеют свои особенности построения ключей. Вместе с тем существуют и общие правила, состоящие в следующем.

– Ключ должен быть уникальным. У составного ключа значения

отдельных полей (но не всех одновременно) могут повторяться.

- Ключ должен быть достаточным и не избыточным, т. е. не содержать поля, которые можно удалить без нарушения уникальности ключа.
- В состав ключа не могут входить поля некоторых типов, например, графическое поле или поле комментария.

Выбор ключевых полей не всегда является простой и очевидной задачей, особенно для таблиц с большим количеством полей. Нежелательно выбирать в качестве ключевых поля, содержащие фамилии людей в таблице сотрудников организации или названия товаров в таблице данных склада. В этом случае высока вероятность существования двух и более однофамильцев, а также товаров с одинаковыми названиями, которые различаются, к примеру, цветом (значение другого поля). Для указанных таблиц можно использовать, например, поле кода сотрудника и поле артикула товара. При этом предполагается, что указанные значения являются уникальными.

Удобным вариантом создания ключа будет использование для него поля соответствующего типа, которое автоматически обеспечивает поддержку уникальности значений. Для таблиц Paradox таким является поле автоинкрементного типа, еще одним достоинством которого является небольшой размер (4 байта). В то же время в таблицах dBase и InterBase поле подобного типа отсутствует, и программист должен обеспечивать уникальность значений ключа самостоятельно, например, используя специальные генераторы.

Отметим, что при создании и ведении БД правильным подходом считается задание в каждой таблице ключа даже в том случае, если на первый взгляд он не нужен. В примерах таблиц, которые приводятся при изложении материала, как правило, ключ создается, и для него вводится специальное автоинкрементное поле с именем Code или Number.

Индекс, как и ключ, строится по полям таблицы, однако он может допускать повторение значений составляющих его полей – в этом и состоит его основное отличие от ключа. Поля, по которым построен индекс, называют индексными. Простой индекс состоит из одного поля, а составной (сложный) – из нескольких полей.

Индексы при их создании именуются. Как и в случае с ключом, в зависимости от СУБД индексы могут храниться в отдельных файлах или совместно с данными. Создание индекса называют индексированием таблицы.

Использование индекса обеспечивает:

- увеличение скорости доступа (поиска) к данным;
- сортировку записей;
- установление связи между отдельными таблицами БД;
- использование ограничений ссылочной целостности.

В двух последних случаях индекс применяется совместно с ключом второй таблицы.

Как и ключ, индекс представляет собой своеобразное оглавление таблицы, просмотр которого выполняется перед обращением к ее записям.

Таким образом, использование индекса повышает *скорость доступа* к данным в таблице за счет того, что доступ выполняется не последовательным, а индексно-последовательным методом.

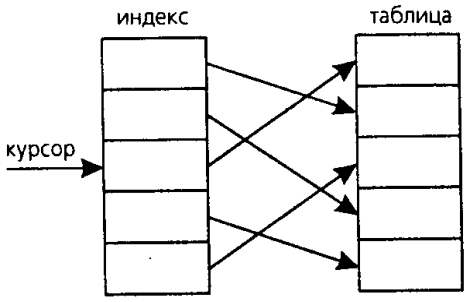


Рис. 1.2.2. Схема перемещения курсора по индексу.

Сортировка представляет собой упорядочивание записей по полю или группе полей в порядке возрастания или убывания их значений. Можно сказать, что индекс служит для сортировки таблиц по индексным полям. В частности, в Delphi записи набора Table можно сортировать только по индексным полям. Набор данных Query позволяет выполнить средствами SQL сортировку по любым полям, однако и в этом случае для индексированных полей упорядочивание записей выполняется быстрее.

Для одной таблицы можно создать несколько индексов. В каждый момент времени один из них можно сделать текущим, т. е. активным. Даже при существовании нескольких индексов, таблица может не иметь текущего индекса (текущий индекс важен, например, при выполнении поиска и сортировки записей набора данных Table).

Ключевые поля обычно автоматически индексируются. В таблицах Paradox ключ также является главным (первичным) индексом, который не именуется. Для таблиц dBase ключ не создается, и его роль выполняет один из индексов.

Таким образом, использование ключей и индексов позволяет:

- однозначно идентифицировать записи;
- избегать дублирования значений в ключевых полях;
- выполнять сортировку таблиц;
- ускорять операции поиска в таблицах;
- устанавливать связи между отдельными таблицами БД;
- использовать ограничения ссылочной целостности.

Одной из основных задач БД является обеспечение *быстрого доступа к данным* (поиска данных). Время доступа к данным в значительной степени зависит от используемых для поиска данных методов и способов.

1.2.3. Методы и способы доступа к данным

Выделяют следующие методы доступа к данным таблиц:

- последовательный;

– прямой;
– индексно-последовательный.

При последовательном методе выполняется последовательный просмотр всех записей таблицы и поиск нужных из них. Этот метод доступа является крайне неэффективным и приводит к значительным затратам времени на поиски, которые прямо пропорциональны размеру таблицы (числу ее записей). Поэтому его рекомендуется использовать только для относительно небольших таблиц.

При прямом доступе нужная запись выбирается из таблицы на основании ключа или индекса. При этом просмотр других записей не выполняется. Напомним, что значения ключей и индексов располагаются в упорядоченном виде и содержат ссылку, указывающую на расположение соответствующей записи в таблице. При поиске записи выполняется не последовательный просмотр всей таблицы, а непосредственный доступ к записи на основании ссылки.

Индексно-последовательный метод доступа включает в себя элементы последовательного и прямого методов доступа и используется при поиске группы записей. Этот метод реализуется только при наличии индекса, построенного по полям, значения которых должны быть найдены. Суть его заключается в том, что находится индекс первой записи, удовлетворяющей заданным условиям, и соответствующая запись выбирается из таблицы на основании ссылки. Это является прямым доступом к данным. После обработки первой найденной записи осуществляется переход к следующему значению индекса, и из таблицы выбирается запись, соответствующая значению этого индекса. Таким образом, последовательно перебираются индексы всех записей, удовлетворяющих заданным условиям, что является последовательным доступом.

Достоинствами прямого и индексно-последовательного методов является максимально возможная скорость доступа к данным, плата за которую – потеря памяти для хранения информации о ключах и индексах.

Указанные методы доступа реализуются СУБД и не требуют специального программирования. Задачей разработчика является определение соответствующей структуры БД, в данном случае – определение ключей и индексов. Так, если для поля создан индекс, то при поиске записей по этому полю автоматически используется индексно-последовательный метод доступа, в противном случае - последовательный метод.

При выполнении операций с таблицами используется один из следующих *способов доступа к данным:*

– навигационный;
– реляционный.

Навигационный способ доступа заключается в обработке каждой отдельной записи таблицы. Этот способ обычно используется в локальных БД или в удаленных БД небольшого размера. Если необходимо обработать несколько записей, то все они обрабатываются поочередно.

Реляционный способ доступа основан на обработке сразу *группы записей*, при этом если необходимо обработать одну запись, то обрабатывается группа, состоящая из одной записи. Так как реляционный способ доступа основывается на SQL-запросах, то его также называют SQL-ориентированным. Этот способ доступа ориентирован на выполнение операций с удаленными БД и является предпочтительным при работе с ними, хотя его можно использовать также и для локальных БД.

Способ доступа к данным выбирается программистом и зависит от средств доступа к БД, используемых при разработке приложения. Например, в приложениях, создаваемых в Delphi, реализацию навигационного способа доступа можно осуществить посредством компонентов Table или Query, а реляционного – с помощью компонента *Query*.

Таким образом, методы доступа к данным определяются структурой БД, а способы доступа – приложением.

1.2.4. Связь между таблицами

В частном случае БД может состоять из одной таблицы, например, с днями рождения сотрудников организации. Однако обычно реляционная БД состоит из набора взаимосвязанных таблиц. Организация связи (отношений) между таблицами называется *связыванием* или *соединением таблиц*.

Связи между таблицами можно устанавливать как при создании БД, так и при выполнении приложения, используя средства, предоставляемые СУБД. Связывать можно две или несколько таблиц. В реляционной БД, помимо связанных, могут быть и отдельные таблицы, и не соединенные ни с одной другой таблицей. Это не меняет сути реляционной БД, которая содержит единую информацию об информационной системе, связанную не в буквальном смысле (связь между таблицами), а в функциональном смысле – вся информация относится к одной системе.

Для связывания таблиц используются поля связи (иногда применяется термин "совпадающие поля"). Поля связи обязательно должны быть индексированными. В подчиненной таблице для связи с главной таблицей берется индекс, который также называется *внешним ключом*. Состав полей этого индекса должен полностью или частично совпадать с составом полей индекса главной таблицы. Особенности использования индексов зависят от формата связываемых таблиц. Так, для таблиц dBase индексы строятся по одному полю и нет деления на ключ (главный или первичный индекс) и индексы. Для организации связи в главной и подчиненной таблицах выбираются индексы, составленные по полям совпадающего типа, например, целочисленного.

Для таблиц Paradox в качестве полей связи главной таблицы должны использоваться поля ключа, а для подчиненной таблицы – поля индекса. Кроме того, в подчиненной таблице обязательно должен быть определен ключ. На рис. 1.2.3 и 1.2.4. показана схема связи между таблицами БД Paradox.

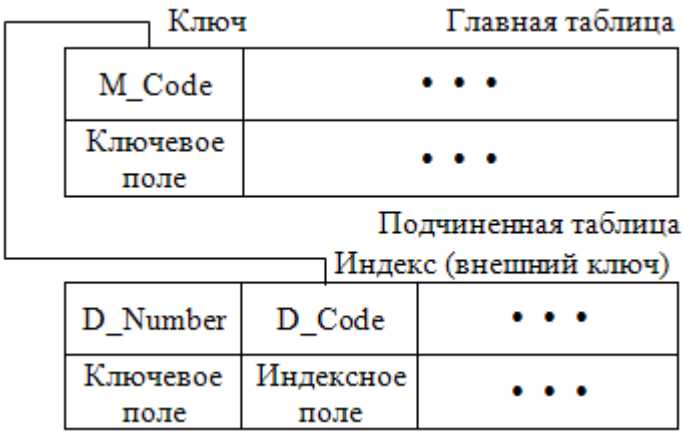


Рис. 1.2.3. Схема связи между таблицами базы данных Paradox.

В главной таблице определен ключ, построенный по полю M_code автоинкрементного типа. В подчиненной таблице определен ключ по полю D_Number.также автоинкрементного типа и индекс, построенный по полю Decode целочисленного типа. Связь между таблицами устанавливается по полям D_Code и M_code. Индекс по полю D_Code является внешним ключом. В названия полей включены префиксы, указывающие на принадлежность поля соответствующей таблице. Так, названия полей главной таблицы начинаются буквой M (Master), а названия полей подчиненной таблицы начинаются буквой D (Detail). Подобное именование полей облегчает ориентацию в их названиях, особенно при большом количестве таблиц.

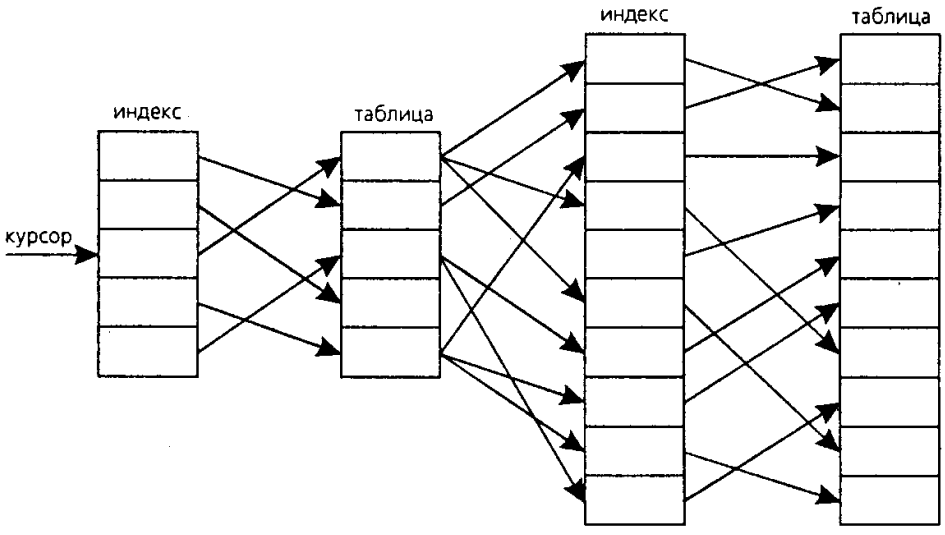


Рис. 1.2.4. Схема взаимодействия главной и вспомогательной таблиц.

Связь между таблицами определяет отношение подчиненности, при котором одна таблица является *главной* (родительской, или мастером – Master), а вторая – *подчиненной* (дочерней, или детальной – Detail). Саму связь (отношение) называют связь "главный-подчиненный", "родительский-дочерний" или "мастер-детальный". Существуют следующие виды связи:

- отношение "один-к-одному";
- отношение "один-ко-многим";
- отношение "много-к-одному";
- отношение "много-ко-многим".

Отношение "один-к-одному" означает, что одной записи в главной таблице соответствует одна запись в подчиненной таблице. При этом возможны два варианта:

- для каждой записи главной таблицы есть запись в подчиненной таблице;
- в подчиненной таблице может не быть записей, соответствующих записям главной таблицы.

Отношение "один-к-одному" обычно используется при разбиении таблицы с большим числом полей на несколько таблиц. В этом случае в первой таблице остаются поля с наиболее важной и часто используемой информацией; а остальные поля переносятся в другую (другие) таблицы.

Отношение “один-к-одному” представляет собой простейший вид связи данных, когда первичный ключ таблицы является в то же время внешним ключом, ссылающимся на первичный ключ другой таблицы. Такую связь бывает удобно устанавливать тогда, когда невыгодно держать разные по размеру данные в одной таблице.

Отношение "один-ко-многим" означает, что одной записи главной таблицы в подчиненной таблице может соответствовать несколько записей, в частном случае ни одной. Этот вид отношения встречается наиболее часто. После установления связи между таблицами, при перемещении на какую-либо запись в главной таблице в подчиненной таблице автоматически становятся доступными записи, у которых значения поля связи равно значению поля связи текущей записи главной таблицы. Такой отбор записей подчиненной таблицы является своего рода фильтрацией.

Отношение “один-ко-многим” в большинстве случаев отражает реальную взаимосвязь сущностей в предметной области. Она реализуется уже описанной парой “внешний ключ - первичный ключ”, т.е. когда определен внешний ключ, ссылающийся на первичный ключ другой таблицы. Именно эта связь описывает широко распространенный механизм классификаторов.

Типичным примером является, например, организация учета выдачи книг в библиотеке, для которой удобно создать следующие две таблицы:

- таблицу карточек читателей, содержащую такую информацию о читателе, как фамилия, имя, отчество, дата рождения и домашний адрес;
- таблицу выдачи книг, в которую заносится информация о выдаче книги читателю и о возврате книги.

В этой ситуации главной является таблица карточек читателей, а подчиненной – таблица выдачи книг. Один читатель может иметь на руках несколько книг, поэтому одной записи в главной таблице может соответствовать несколько записей в подчиненной таблице. Если читатель сдал

все книги или еще не брал ни одной книги, то для него в подчиненной таблице записей нет. После связывания обеих таблиц при выборе записи с данными читателя в таблице выдачи книг будут доступны только записи с данными о книгах, находящихся на руках этого читателя.

В приведенном примере предполагается, что после возврата книги соответствующая ей запись удаляется из таблицы выдачи книг. Вместо удаления записи можно заносить в соответствующее поле дату возврата книги.

Отношение "много-к-одному' отличается от отношения один-ко-многим только направлением. Если на отношение "один-ко-многим" посмотреть со стороны подчиненной таблицы, а не главной, то оно превращается в отношение "много-к-одному".

Отношение "много-ко-многим" имеет место, когда одной записи главной таблицы может соответствовать несколько записей подчиненной таблицы и одновременно одной записи подчиненной таблицы – несколько записей главной таблицы. Подобное отношение реализуется, например, при планировании занятий в институте и устанавливается между таблицами, в которых хранится информация о номерах аудиторий и номерах учебных групп. Так как учебная группа может заниматься в разных аудиториях, то одной записи об учебной группе (первая таблица) может соответствовать несколько записей о занимаемых этой группой аудиториях. В то же время в одной аудитории могут заниматься разные учебные группы (даже одновременно), поэтому одной записи об аудитории может соответствовать несколько записей об учебных группах (вторая таблица).

На практике отношение "много-ко-многим" используется достаточно редко. Причинами являются сложность организации связи между таблицами и взаимодействия между их записями. Кроме того, многие СУБД, в том числе Paradox, не поддерживают организацию ссылочной целостности для подобного отношения. Отметим также, что для отношения "много-ко-многим" понятия главной и подчиненной таблицы лишаются смысла.

Среди рассмотренных отношений наиболее общим является отношение "один-ко-многим". Другие виды отношений можно рассматривать как его варианты отношение "один-к-одному" представляет собой частный случай этого отношения, а отношение "много-к-одному" является его "переворотом". Отношение "много-ко-многим" можно свести к отношению "один-ко-многим", соответствующим образом преобразовав и разделив таблицы. В дальнейшем предполагается, что таблицы связаны именно отношением "один-ко-многим".

Работа со связанными таблицами имеет следующие особенности.

- При изменении (редактировании) поля связи может нарушиться связь между записями двух таблиц. Поэтому при редактировании поля связи записи главной таблицы нужно соответственно изменять и значения поля связи всех подчиненных таблиц.
- При удалении записи главной таблицы нужно удалять и соответствующие ей записи в подчиненной таблице (каскадное удаление).

– При добавлении записи в подчиненную таблицу значение ее поля связи должно быть установлено равным значению поля связи главной таблицы.

Ограничения по установке, изменению полей связи и каскадному удалению записей могут быть наложены на таблицы при их создании. Эти ограничения, наряду с другими элементами, например описаниями полей и индексов, входят в структуру таблицы и действуют для всех приложений, которые выполняют операции с БД. Указанные ограничения можно задать при создании или реструктуризации таблицы, например, в среде программы Database Desktop, которая позволяет устанавливать связи между таблицами при их создании.

Ограничения, связанные с установкой, изменением значений полей связи и каскадным удалением записей, могут и не входить в структуру таблицы (таблиц), а реализовываться программным способом. В этом случае программист должен обеспечить:

– организацию связи между таблицами;
– установку значения поля связи подчиненной таблицы ("это может также выполняться автоматически);
– контроль (запрет) редактирования полей связи;
– организацию (запрет) каскадного удаления записей.

Например, в случае удаления записи из главной таблицы программист должен проверить наличие соответствующих записей в подчиненной таблице. Если такие записи существуют, то необходимо удалить и их или, наоборот, запретить удаление записей из обеих таблиц. И в том и в другом случае пользователю должно быть выдано предупреждение.

1.2.5. Механизм транзакций

Информация БД в любой момент времени должна быть целостной и непротиворечивой. Одним из путей обеспечения этого является использование механизма транзакций.

Транзакция представляет собой выполнение последовательности операций. При этом возможны две ситуации.

– Успешно завершены все операции. В этом случае транзакция считается успешной, и все изменения в БД, которые были произведены в рамках транзакции отдельными операциями, утверждаются. В результате БД переходит из одного целостного состояния в другое.
– Неудачно завершена хотя бы одна операция. При этом вся транзакция считается неуспешной, и результаты выполнения всех операций (даже успешно выполненных) отменяются. В результате происходит возврат БД в состояние, в котором она находилась до начала транзакции.

Таким образом, успешная транзакция переводит БД из одного целостного состояния в другое. Использование механизма транзакций необходимо

– при выполнении последовательности взаимосвязанных операций с БД;

– при многопользовательском доступе к БД.

Транзакция может быть неявной или явной. *Неявная* транзакция стартует автоматически, а по завершении также автоматически подтверждается или отменяется. *Явной* транзакцией управляет программист с использованием компонента *Database* и/или средств SQL.

Часто в транзакцию объединяются операции над несколькими таблицами, когда производятся действия по внесению в разные таблицы взаимосвязанных изменений. Пусть осуществляется перенос записей из одной таблицы в другую. Если запись сначала удаляется из первой таблицы, а затем заносится во вторую таблицу, то при возникновении сбоя, например из-за перерыва, в энергопитании компьютера, возможна ситуация, когда запись уже удалена, но во вторую таблицу не попала. Если запись сначала заносится во вторую таблицу, а потом удаляется из первой таблицы, то при сбое возможна ситуация, когда запись будет находиться в двух таблицах. В обоих случаях имеет место нарушение целостности и непротиворечивости БД.

Для предотвращения подобной ситуации операции удаления записи из одной таблицы и занесения ее в другую таблицу объединяются в одну транзакцию. Выполнение этой транзакции гарантирует, что при любом ее результате целостность БД нарушена, не будет.

Еще одним примером, демонстрирующим необходимость применения механизма транзакций, является складской учет товара. При поступлении товара на склад в таблицу движения товара заносится запись с данными о названии, количестве товара и дате его поступления. Затем в таблице склада соответственно количеству поступившего товара увеличивается наличное количество этого товара. При возникновении какой-либо ошибки, связанной с записью наличного количества товара, новое значение может быть не занесено в соответствующую запись, в результате чего будет нарушена целостность БД, и она будет содержать некорректные значения. Такая ситуация возможна, например, в случае многопользовательского доступа к БД при редактировании этой записи другим приложением. Поэтому в случае невозможности внести изменения в наличное количество товара должно блокироваться и добавление новой записи в таблицу движения товара.

Для реализации механизма транзакций СУБД предоставляют соответствующие средства.

1.2.6. Бизнес-правила

Бизнес-правила представляют собой механизмы управления БД и предназначены для поддержания БД в целостном состоянии, а также для выполнения ряда других действий, например, накапливания статистики работы с БД. Отметим, что в данном контексте бизнес-правил, а являются просто правилами управления БД и не имеют отношения к бизнесу как предпринимательству.

В первую очередь бизнес-правила реализуют следующие ограничения БД:

– задание допустимого диапазона значений;
– задание значения по умолчанию;
– требование уникальности значения;
– запрет пустого значения;
– ограничения ссылочной целостности.

Бизнес-правила можно реализовывать как на физическом, так и на программном уровнях. В первом случае эти правила (например, ограничения ссылочной целостности для связанных таблиц) задаются при создании таблиц и входят в структуру БД. В дальнейшей работе нельзя нарушить или обойти ограничение, заданное на физическом уровне.

Вместо заданных на физическом уровне бизнес-правил или в дополнение к ним можно определить бизнес-правила на программном уровне. Действие этих правил распространяется только на приложение, в котором они реализованы. Для программирования в приложении бизнес-правил используются компоненты и предоставляемые ими средства. Достоинство такого подхода заключается в легкости изменения бизнес-правил и определении правил "своего" приложения. Недостатком является снижение безопасности БД, связанное с тем, что каждое приложение может устанавливать свои правила управления БД. В главе 19 приводится пример программирования бизнес-правил в приложении, связанный с каскадным удалением записей связанных таблиц.

При работе с удаленными БД в архитектуре "клиент-сервер" бизнес-правила можно реализовывать также на сервере.

1.2.7. Словарь данных

Словарь данных представляет собой совокупность определений БД и атрибутов. Словарь данных позволяет сформировать и сохранить характеристики, которые в дальнейшем можно использовать для описания БД.

Использование словаря данных позволяет:
– ускорить процесс создания БД;
– облегчить изменение характеристик БД;
– придать единообразный вид визуальным компонентам приложения.

Определение БД является специализированной БД, которая описывает структуру базы, но не содержит данных. Это описание структуры можно использовать для создания других БД.

Атрибуты представляют собой совокупности характеристик отдельных полей. Заданные через атрибуты характеристики поля при выполнении приложения устанавливаются в качестве значений соответствующих свойств визуальных компонентов, которые отображают значения поля, например, *DBGrid* или *DBText*. Приведем некоторые наиболее распространенные характеристики полей (они же свойства визуальных компонентов):

Alignment – выравнивание;

DisplayLabel – заголовок столбца (сетки DBGrid);

Readonly – недоступность поля для редактирования;

Required – требование обязательного ввода значения;
Visible – видимость;
DispiayFormat – формат отображаемого значения;
MinValue – минимальное значение;
MaxValue – максимальное значение.

Для работы со словарем данных удобно использовать программу SQL Explorer.

1.2.8. Таблицы формата dBase и Paradox

Delphi не имеет своего формата таблиц, но поддерживает, как свои собственные два типа локальных таблиц: *dBase* и *Paradox*. Каждая из этих таблиц имеет свои особенности.

Таблицы dBase являются одним из первых появившихся форматов таблиц для персональных компьютеров и поддерживаются многими системами, которые связаны с разработкой и обслуживанием приложений, работающих с БД.

Основные достоинства таблиц dBase:

- простота использования;
- совместимость с большим числом приложений.

В табл. 1.2.1 содержится список типов полей таблиц dBase IV. Для каждого типа приводится символ, используемый для его обозначения в программе Database Desktop (создания и редактирования таблиц, SQL-запросов и запросов QBE), а также описание значений, которые может содержать поле рассматриваемого типа.

Таблица 1.2.1. Форматы полей таблиц dBase IV.

Тип	Описание значения
Character (alpha)	строка символов, длиной 255 символ, содержащая любые печатаемые символы
Float (numeric)	числовое поле размером 1-20 байт в формате с плавающей точкой, значение которого может быть положительным и отрицательным. Может содержать очень большие величины, однако следует иметь в виду постоянные ошибки округления при работе с полем такого типа. Число цифр после десятичной точки (параметр Dec в DBD) должно быть по крайней мере на 2 меньше, чем размер всего поля, поскольку в общий размер включаются сама десятичная точка и знак. Диапазон -10^{308} …10^{308} точность 15 цифр.
Number (BCD)	числовое поле размером 1-20 байт, содержащее данные в формате BCD (Binary Coded Decimal). Скорость вычислений немного меньше, чем в других числовых форматах, однако точность - гораздо выше. Число цифр после десятичной точки (параметр Dec в DBD) также должно быть по крайней

	мере на 2 меньше, чем размер всего поля, поскольку в общий размер включаются сама десятичная точка и знак
Date	поле даты длиной 8 байт. По умолчанию, используется формат короткой даты (ShortDateFormat)
Logical	поле длиной 1 байт, которое может содержать только значения “истина” или “ложь” - True или False. Допускаются строчные и прописные буквы. Таким образом, в отличие от Парадокса, допускаются буквы “Y” и “N” (сокращение от Yes и No)
Memo	поле для хранения символов, суммарная длина которых более 255 байт. Может иметь любую длину. Это поле хранится в отдельном файле с расширением .DBT. Database Desktop не имеет возможности вставлять данные в поле типа Memo
OLE	поле, содержащее OLE-данные (Object Linking and Embedding) - образы, звук, видео, документы - которые для своей обработки вызывают создавшее их приложение. Может иметь любую длину. Это поле также сохраняется в отдельном файле с расширением .DBT. Database Desktop “умеет” создавать поля типа OLE, однако наполнять их можно только в приложении.
Binary	поле, содержащее любую двоичную информацию. Может иметь любую длину. Данное поле сохраняется в отдельном файле с расширением .DBT. Это полнейший аналог поля BLOb в InterBase

Таблицы dBase являются достаточно простыми и используют для своего хранения на дисках относительно мало физических файлов. По расширению файлов можно определить, какие данные они содержат.

- DBF – таблица с данными.
- DBT – данные больших двоичных объектов, или BLOB-данные (Binary Large OBject). К ним относятся двоичные, Memo и OLE-поля. Memo-поле также называют полем комментариев.
- MDX – поддерживаемые индексы.
- NDX – индексы, непосредственно не поддерживаемые форматом dBase. При использовании таких индексов программист должен обрабатывать их самостоятельно.

Имя поля в таблице dBase должно состоять из букв и цифр и начинаться с буквы. Максимальная длина имени составляет 10 символов. В имена нельзя включать специальные символы и пробел.

К недостаткам таблиц dBase относится то, что они не поддерживают автоматическое использование парольной защиты и контроль целостности связей, поэтому программист должен кодировать эти действия самостоятельно.

Таблицы *Paradox* являются достаточно развитыми и удобными для

создания БД. Можно отметить следующие их достоинства:

- большое количество типов полей для представления данных различных типов;
- поддержка целостности данных;
- организация проверки вводимых данных;
- поддержка парольной защиты таблиц.

Большой набор типов полей позволяет гибко выбирать тип для точного представления данных, хранимых в базе. Например, для представления числовой информации можно использовать один из пяти числовых типов. В табл. 1.2.2 содержится список типов полей для таблиц Paradox 7. Для каждого типа приводится символ, используемый для обозначения этого типа в программе Database Desktop, и описание значений, которые может содержать поле рассматриваемого типа.

Таблица 1.2.2. Типы полей таблиц в Paradox 7

Тип	**Обозначение**	**Описание значения**
Alpha	A	Строка символов. Длина не более 255 символов
Number	N	Число с плавающей точкой. Диапазон -10^{307}… 10^{308}. Точность 15 цифр мантиссы
Money	$	Денежная сумма. Отличается от типа Number тем, что в значении отображается денежный знак. Обозначение денежного знака зависит от установок Windows
Short	S	числовое поле длиной 2 байта, которое может содержать только целые числа в диапазоне от -32768 до 32767
Long integer		числовое поле длиной 4 байта, которое может содержать целые числа в диапазоне от -2147483648 до 2147483648
BCD	#	Число в двоично-десятичном формате, содержащее данные в формате BCD (Binary Coded Decimal). Скорость вычислений немного меньше, чем в других числовых форматах, однако точность - гораздо выше. Может иметь 0-32 цифр после десятичной точки
Date	D	Дата. Поле даты длиной 4 байта. Корректно обрабатывает високосные года и имеет встроенный механизм проверки правильности даты. Диапазон 01.01.9999 до н.э... 31.12.9999
Time	T	Время. Поле времени длиной 4 байта, содержит время в миллисекундах от полуночи и ограничено 24 часами

Timestamp	@	Дата и время. Обобщенное поле даты длиной 8 байт - содержит и дату и время
Memo	M	Строка символов. Длина не ограничена. Первые 240 символов хранятся в файле таблицы, остальные в файле с расширением MB
Formatted Memo	F	Поле, аналогичное Memo, с добавлением возможности задавать шрифт текста. Также может иметь любую длину. При этом размер, указываемый при создании таблицы, означает количество символов, сохраняемых в таблице (0-240) - остальные символы сохраняются в отдельном файле с расширением .MB.
Graphic	G	Графическое изображение. Форматы BMP, PCX, TIF, GIF и EPS. При загрузке в поле изображение преобразуется к формату BMP. Для хранения изображения используется файл с расширением MB
OLE	O	Поле, содержащее OLE-данные (Object Linking and Embedding) - образы, звук, видео, документы - которые для своей обработки вызывают создавшее их приложение. Может иметь любую длину. Смысл размера - такой же, как и в Formatted Memo. Database Desktop “умеет” создавать поля типа OLE, однако наполнять их можно только в приложении. Delphi “напрямую” не умеет работать с OLE-полями, но это легко обходится путем использования потоков
Logical	L	Поле длиной 1 байт, которое может содержать только два значения - T (true, истина) или F (false, ложь). Допускаются строчные и прописные буквы
Autoincrement	+	поле длиной 4 байта, содержащее нередактируемое (read-only) значение типа long integer. Значение этого поля автоматически увеличивается (начиная с 1) с шагом 1 - это очень удобно для создания уникального идентификатора записи (физический номер записи не может служить ее идентификатором, поскольку в Парадоксе таковой отсутствует. В InterBase также отсутствуют физические номера записей, но отсутствует и поле Autoincrement. Его с успехом заменяет встроенная функция Gen_id, которую удобней всего применять в триггерах)
Binary	B	поле, содержащее любую двоичную информацию. Может иметь любую длину. При этом размер, указываемый при создании таблицы, означает

		количество символов, сохраняемых в таблице (0-240) - остальные символы сохраняются в отдельном файле с расширением .MB. Это полнейший аналог поля BLOb в InterBase
Bytes	Y	Последовательност байтов. Длина неболее 255 байтов. Может содержат любые данные.

Имя поля в таблице Paradox должно состоять из букв (допускается кириллица) и цифр и начинаться с буквы. Максимальная длина имени составляет 25 символов. В имени можно использовать такие символы, как пробел, "#", "$" и некоторые другие. Не рекомендуется использовать символы ".", "!" и "|", т. к. они зарезервированы в Delphi для других целей.

При задании ключевых полей они должны быть первыми в структуре таблицы.

Поддержка концепции целостности данных обеспечивает правильность ссылок между таблицами. Например, если в БД имеются таблицы клиентов и заказов, то эти таблицы могут быть связаны следующим образом: каждая запись таблицы заказов ссылается через индексное поле на запись в таблице клиентов, соответствующую сделавшему заказ клиенту. Если в таблице клиентов любым способом удалить запись с информацией о каком-либо клиенте, то BDE автоматически удалит все записи, соответствующие этому клиенту, и из таблицы заказов. Подобное удаление взаимосвязанных записей называют *каскадным*.

Для полей можно определить специальный диапазон, в котором должны находиться вводимые в них значения. Кроме того, для каждого поля можно определить минимально и максимально допустимые значения. При попытке ввода в поле значения, выходящего за допустимые границы, возникает исключительная ситуация, значение не вводится и содержимое поля не изменяется. Например, для поля salary (Оклад) в качестве минимального значения логично указать ноль, тем самым в это поле запрещается ввод отрицательных значений. Максимальное значение поля Salary зависит от организации, страны и от других факторов.

Наряду с диапазоном допустимых значений, для каждого поля можно задать еще значение по умолчанию, которое автоматически заносится в поле при добавлении к таблице новой записи.

При работе с конфиденциальной информацией может потребоваться защита таблиц и их полей. Для каждой таблицы Paradox следует указать основной пароль, который используется при изменениях во всей таблице, связанных со сменой структуры или с редактированием данных в любом поле. Возможно также задание и дополнительного пароля, позволяющего ограничить доступ к конкретному полю или группе полей таблицы, а также набор операций, применимых к таблице (например, разрешение только на чтение записей таблицы без возможности их модификации). После установки паролей они будут автоматически запрашиваться, и контролироваться при любой попытке доступа к таблице.

Определенным недостатком таблиц Paradox является наличие относительного большого количества типов файлов, требуемых для хранения содержащихся в таблице данных. При копировании или перемещении какой-либо таблицы из одного каталога в другой необходимо обеспечить копирование или перемещение всех файлов, относящихся к этой таблице. Файлы таблиц Paradox имеют следующие расширения:

- DB – таблица с данными;
- MB – BLOB данные;
- PX – главный индекс (ключ);
- XG* и YG* – вторичные индексы;
- VAL – параметры для проверки данных и целостности ссылок;
- TV и FAM – форматы вывода таблицы в программе Database Desktop.

Кроме названных файлов, при работе в сети для контроля доступа к таблицам Paradox используются файлы с расширением NET.

1.3. Средства для работы с базами данных

Хотя Delphi не имеет своего формата таблиц БД, она, тем не менее, обеспечивает мощную поддержку большого количества различных СУБД – как локальных (например, dBase или Paradox), так и промышленных (например, Sybase или InterBase). Средства Delphi, предназначенные для работы с БД, можно разделить на два вида:

- инструментальные средства;
- компоненты.

К инструментальным средствам относятся специальные программы и пакеты, обеспечивающие обслуживание БД вне разрабатываемых приложений.

Компоненты предназначены для создания приложений, осуществляющих операции с БД.

Напомним, что в Delphi 6 появилось окно Обозревателя дерева объектов, которое отображает иерархическую структуру объектов текущей формы. При разработке приложений баз данных это окно удобно использовать для просмотра структуры базы данных и изменения связей между компонентами.

Кроме того, в Delphi 6 в окне Редактора кода появилась новая страница *Diagramm* (ранее она находилась в окне модуля данных), служащая для отображения и настройки взаимосвязей между элементами баз данных.

1.3.1. Инструментальные средства

Для операций с БД система Delphi предлагает следующий набор инструментальных средств.

- **Borland Database Engine** (BDE) – процессор баз данных, который представляет собой набор динамических библиотек и драйверов, предназначенных для организации доступа к БД из Delphi-приложений. BDE является центральным звеном при организации доступа к данным.
- **BDE Administrator** – утилита для настройки различных параметров

BDE.

- **Database Desktop** – программа создания и редактирования таблиц, SQL-запросов и запросов QBE.
- **SQL Explorer** – Проводник БД, позволяющий просматривать и редактировать БД и словари данных.
- **SQL Builder** – программа визуального конструирования SQL-запросов.
- **SQL Monitor** – программа отслеживания порядка выполнения SQL-запросов к удаленным БД.
- **Data Pump** – программа для переноса данных между БД.
- **IBConsole** – программа для управления удаленными БД.
- **InterBase Server Manager** – программа для запуска сервера InterBase.
- **SQL Links** – драйверы для доступа к удаленным промышленным СУБД, таким как Microsoft SQL Server или Oracle. К промышленному серверу InterBase, который поставляется совместно с Delphi и является для нее родным, доступ также можно организовать напрямую через BDE, не исцользуя драйвер SQL-Links.
- **dbExpress** – набор драйверов для доступа к базам данных SQL с помощью таких Компонентов, как SQLConnection, SQLDataSet, SQLQuery, SQLStoredProc и SQLTable. dbExpress включает в свой состав следующие драйверы:
  - InterBase - DBEXPINT.DLL;
  - DB2 - DBEXPDB2.DLL;
  - Oracle - DBEXPORA.DLL;
  - MySQL - DBEXPMYS.DLL.
- **InterBase Server** – клиентская и серверная части сервера InterBase.

Одни инструментальные средства, например, BDE Administrator и SQL Explorer, можно использовать для работы с локальными и удаленными БД, другие, например, IBConsole – для работы с удаленными БД.

1.3.2. Компоненты

Рассмотрим теперь компоненты, используемые для создания приложений БД. Кроме компонентов, Delphi также предоставляет разработчику специальные объекты, например, объекты типа Field. Как и другие управляющие элементы Delphi, связанные с БД компоненты делятся на визуальные и невизуальные.

Невизуальные компоненты предназначены для организации доступа к данным, содержащимся в таблицах. Они представляют собой промежуточное звено между данными таблиц БД и визуальными компонентами.

Визуальные компоненты используются для создания интерфейсной части приложения. С их помощью пользователь может выполнять такие операции с таблицами БД, как просмотр или редактирование данных. Визуальные компоненты также называют элементами, чувствительными к данным.

Компоненты, используемые для работы с БД, находятся на страницах

Data Access, Data Controls, dbExpress, BDE, ADO, Decision Cube, QReport и InterBase

Палитры компонентов. Некоторые компоненты предназначены - специально для работы с удаленными БД в архитектуре "клиент-сервер". Отметим, что в Палитре компонентов Delphi 6 появились новые страницы и значительно изменилось по сравнению с предыдущими версиями распределение по страницам компонентов, используемых для работы с БД. В частности, пропала страница *Midas*, а ее компоненты распределились по другим страницам.

На изрядно поредевшей странице **Data Access** (рис. 1.3.1**)** находятся невизуальные компоненты, предназначенные для организации доступа к данным:

- DataSource – источник данных;
- ClientDataSet – клиентский набор данных;
- DataSetProvider – провайдер набора данных.

Рис. 1.3.1. Страница Data Access.

Рис. 1.3.2. Страница Data Controls

На странице **Data Controls** (рис. 1.3.2) расположены визуальные компоненты, предназначенные для управления данными:

- DBGrid – сетка (таблица);
- DBNavigator – навигационный интерфейс;
- DBText – надпись;
- DBEdit – однострочный редактор (поле редактирования);
- DBMemo – многострочный редактор (панель редактирования);
- DBImage – графический образ (изображение);
- DBListBox – простой список;
- DBCornboBox – комбинированный список;
- DBCheckBox – независимый переключатель;
- DBRadioGroup – группа зависимых переключателей;
- DBLookupListBox – простой список, формируемый по полю другого набора данных;
- DBLookupcomboBox – комбинированный список, формируемый по полю другого набора данных;
- DBRichEdit – полнофункциональный тестовый редактор (поле редактирования);

– DBCtrlGrid – модифицированная сетка;
– DBChart – диаграмма.

На странице **dbExpress** (рис. 1.3.3) мы увидим компоненты, предназначенные для работы с SQL:

– SQLConnection – соединение с БД;
– SQLDataSet – набор данных;
– SQLQuery – набор данных Query;
– SQLStoredProc – вызов хранимой процедуры сервера;
– SQLTable – набор данных Table;
– SQLMonitor – монитор выполнения SQL-запросов;
– SQLClientDataSet – клиентский набор данных.

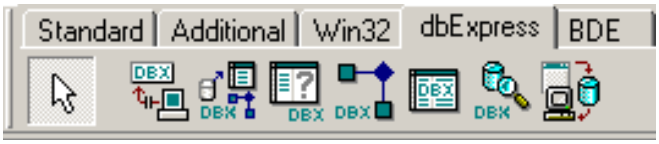


Рис. 1.3.3. Страница dbExpress

Рис. 1.3.4. Страница BDE

Страница **BDE** (рис. 1.3.4) содержит компоненты, предназначенные для управления данными с использованием BDE:

– Table – набор данных, основанный на таблице БД;
– Query – набор данных, основанный на SQL-запросе;
– StoredProc – вызов хранимой процедуры сервера;
– DataBase – соединение с БД;
– Session – текущий сеанс работы с БД;
– BatchMove – выполнение операций над группой записей;
– UpdateSQL – модификация набора данных, основанного на SQL-запросе;
– NestedTable – вложенная таблица;
– BDEClientDataSet – клиентский набор данных.

На странице **ADO** (рис. 1.3.5) расположены компоненты, предназначенные для управления данными с использованием технологии ADO (Active Data Objects):

– ADOConnection – соединение;
– ADOCommand – команда;
– ADODataSet – Набор данных;
– ADOTable – набор данных Table;
– ADOQuery – набор данных Query;

– ADOStoredProc – вызов хранимой процедуры сервера;
– RDSConnection – соединение RDS.

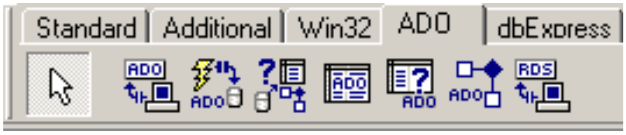


Рис. 1.3.5. Страница ADO

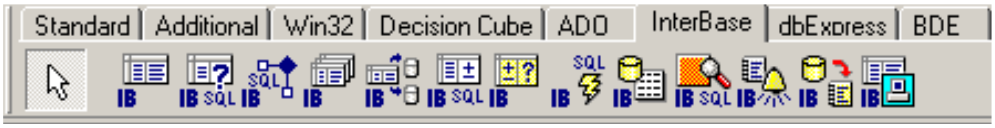


Рис. 1.3.6. Страница InterBase

Рис. 1.3.7. Страница Decision Cube

На странице **InterBase** (рис. 1.3.6) находятся компоненты, предназначенные для работы с сервером InterBase:

– IBTable – набор данных Table;
– IBQuery – набор данных Query;
– IBStoredProc – вызов хранимой процедуры;
– IBDatabase – соединение с БД;
– IBTransaction – транзакция;
– IBUpdateSQL – модификация набора данных, основанного на SQL-запросе;
– IBDataSet – источник данных;
– IBSQL – выполнение SQL-запроса;
– IBDataBaseInfo – информация о БД;
– IBSQLMonitor – монитор выполнения SQL-запросов;
– IBEvents – событие сервера;
– IBExtract – извлечение данных;
– IBClientDataSet – клиентский источник данных.

Страница **Decision Cube** (рис. 1.3.7) содержит компоненты, предназначенные для построения систем принятия решений:

– DecisionCube – куб многомерных данных;
– DecisionQuery – набор, содержащий многомерные данные;
– DecisionSource – источник многомерных данных;
– DecisionPivot – двумерная проекция многомерных данных;
– DecisionGrid – сетка для табличного представления многомерных данных;
– DecisionGraph – графическое представление многомерных данных.

И наконец, на последней странице **QReport** (рис. 1.3.8) находятся

компоненты (в основном визуальные), предназначенные для построения отчетов:

- QuickRep – отчет;
- QRSubDetail – полоса отчета для таблиц, связанных отношением "главный-подчиненный";
- QRStringsBand – строковая полоса отчета;
- QRBand – полоса отчета;
- QRChildBand – дочерняя полоса отчета;
- QRGroup – группа;
- QRLabel – надпись;
- QRDBText – текстовое поле набора данных;
- QRExpr – выражение;
- QRSysData – системная информация;
- QRMemo – многострочный текст;
- QRExprMemo – многострочное выражение;
- QRRichText – форматированный текст;
- QRDBRichText – форматированный текст поля набора данных;
- QRShape – геометрическая фигура;
- QRImage – графическое изображение;
- QRDBimage – графический образ для поля набора данных;
- QRCompositeReport – составной отчет;
- QRPreview – окно просмотра отчета;
- QRTextFilter – текстовый фильтр;
- QRCSVFilter – CSV-фильтр;
- QRHTMLFilter – HTML-фильтр;
- QRChart – диаграмма.

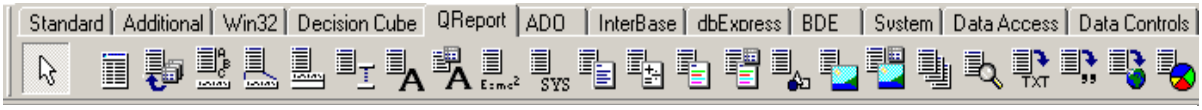


Рис. 1.3.8. Страница QReport

Названия многих компонентов, предназначенных для работы с данными, содержат префиксы, например, DB, IB или QR. Префикс DB означает, что визуальный компонент связан с данными и используется для построения интерфейсной части приложения. Такие компоненты размещаются на форме и предназначены для управления данными со стороны пользователя. Префикс QR означает, что компонент используется для построения отчетов. Эти компоненты размещаются на компоненте QuickRep отчета и его элементах, например, на полосе QRBand служат для оформления внешнего вида отчета. Префикс IB означает, что компонент предназначен для работы с сервером InterBase.

2. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

Проектирование реляционной БД заключается главным образом в разработке структур данных, т.е. в определении состава таблиц и связей между ними. При этом структура должна быть эффективной и обеспечивать:

- быстрый доступ к данным;
- отсутствие дублирования (повторения) данных;
- целостность данных.

Проектирование структуры данных (структуры БД) также называют *логическим* проектированием, или проектированием на логическом уровне.

При проектировании структур данных можно выделить три основных подхода.

- Сбор информации об объектах решаемой задачи в рамках одной таблицы (одного отношения) и последующее разбиение ее несколько взаимосвязанных таблиц на основе нормализации отношений.
- Формулирование знаний о системе (определение типов исходных данных и их взаимосвязей) и требований к обработке данных, а затем получение с помощью средств CASE схемы БД или прикладной информационной системы.
- Структурирование информации в результате системного анализа на основе совокупности правил и рекомендаций.

Проектирование может выполняться классическим способом, когда разработчик собирает и выделяет объекты системы и их характеристики, после чего вручную приводит их к требуемой структуре данных. Кроме того, для проектирования можно использовать так называемые CASE-системы, которые автоматизируют процесс разработки не только БД, но и информационной системы в целом.

2.1. Шаги проектирования базы данных

I. Первый шаг состоит в определении информационных потребностей базы данных. Он включает в себя опрос будущих пользователей для того, чтобы понять и задокументировать их требования. Следует выяснить следующие вопросы:

- сможет ли новая система объединить существующие приложения или их необходимо будет кардинально переделывать для совместной работы с новой системой;
- какие данные используются разными приложениями; смогут ли Ваши приложения совместно использовать какие-либо из этих данных;
- кто будет вводить данные в базу и в какой форме; как часто будут изменяться данные;
- достаточно ли будет для Вашей предметной области одной базы или Вам потребуется несколько баз данных с различными структурами;
- какая информация является наиболее чувствительной к скорости ее извлечения и изменения.

II. Следующий шаг включает в себя анализ объектов реального мира, которые необходимо смоделировать в базе данных.

Формирование концептуальной модели базы данных включает в себя:

- идентификацию функциональной деятельности Вашей предметной области. Например, если речь идет о деятельности предприятия, то в качестве функциональной деятельности можно идентифицировать ведение учета работающих, отгрузку продукции, оформление заказов и т.п.
- идентификацию объектов, которые осуществляют эту функциональную деятельность, и формирование из их операций последовательности событий, которые помогут Вам идентифицировать все сущности и взаимосвязи между ними. Например, процесс “ведение учета работающих” идентифицирует такие сущности как РАБОТНИК, ПРОФЕССИЯ, ОТДЕЛ.
- идентификацию характеристик этих сущностей. Например, сущность РАБОТНИК может включать такие характеристики как Идентификатор Работника, Фамилия, Имя, Отчество, Профессия, Зарплата.
- идентификацию взаимосвязей между сущностями. Например, каким образом сущности РАБОТНИК, ПРОФЕССИЯ, ОТДЕЛ взаимодействуют друг с другом? Работник имеет одну профессию (для простоты!) и значится в одном отделе, в то время как в одном отделе может находиться много работников.

III. Третий шаг заключается в установлении соответствия между сущностями и характеристиками предметной области и отношениями и атрибутами в нотации, выбранной СУБД. Поскольку каждая сущность реального мира обладает некими характеристиками, в совокупности образующими полную картину ее проявления, можно поставить им в соответствие набор отношений (таблиц) и их атрибутов (полей).

Перечислив все отношения и их атрибуты, уже на этом этапе можно начать устранять излишние позиции. Каждый атрибут должен появляться только один раз; и Вы должны решить, какое отношение будет являться владельцем какого набора атрибутов.

IV. На четвертом шаге определяются атрибуты, которые уникальным образом идентифицируют каждый объект. Это необходимо для того, чтобы система могла получить любую единичную строку таблицы. Вы должны определить первичный ключ для каждого из отношений. Если нет возможности идентифицировать кортеж с помощью одного атрибута, то первичный ключ нужно сделать составным - из нескольких атрибутов. Хорошим примером может быть первичный ключ в таблице работников, состоящий из фамилии, имени и отчества. Первичный ключ гарантирует, что в таблице не будет содержаться двух одинаковых строк. Во многих СУБД имеется возможность помимо первичного определять еще ряд уникальных ключей. Отличие уникального ключа от первичного состоит в том, что уникальный ключ не

является главным идентифицирующим фактором записи и на него не может ссылаться внешний ключ другой таблицы. Его главная задача - гарантировать уникальность значения поля.

V. Пятый шаг предполагает выработку правил, которые будут устанавливать и поддерживать целостность данных. Будучи определенными, такие правила в клиент-серверных СУБД поддерживаются автоматически - сервером баз данных; в локальных же СУБД их поддержание приходится возлагать на пользовательское приложение.

Эти правила включают:

- определение типа данных
- выбор набора символов, соответствующего данной стране
- создание полей, опирающихся на домены
- установка значений по умолчанию
- определение ограничений целостности
- определение проверочных условий.

VI. На шестом шаге устанавливаются связи между объектами (таблицами и столбцами) и производится очень важная операция для исключения избыточности данных - нормализация таблиц (см. глава 1.2.4.).

После определения таблиц, полей, индексов и связей между таблицами следует посмотреть на проектируемую базу данных в целом и проанализировать ее, используя правила нормализации, с целью устранения логических ошибок. Важность нормализации состоит в том, что она позволяет разбить большие отношения, как правило, содержащие большую избыточность информации, на более мелкие логические единицы, группирующие только данные, объединенные “по природе”. Таким образом, идея нормализации заключается в следующем. Каждая таблица в реляционной базе данных удовлетворяет условию, в соответствии с которым в позиции на пересечении каждой строки и столбца таблицы всегда находится единственное значение, и никогда не может быть множества таких значений.

После применения правил нормализации логические группы данных располагаются не более чем в одной таблице. Это дает следующие преимущества:

- данные легко обновлять или удалять
- исключается возможность рассогласования копий данных
- уменьшается возможность введения некорректных данных.

Процесс нормализации заключается в приведении таблиц в так называемые *нормальные формы*. Существует несколько видов нормальных форм: первая нормальная форма (1НФ), вторая нормальная форма (2НФ), третья нормальная форма (3НФ), нормальная форма Бойса-Кодда (НФБК), четвертая нормальная форма (4НФ), пятая нормальная форма (5НФ). С практической точки зрения, достаточно трех первых форм - следует учитывать время, необходимое системе для “соединения” таблиц при отображении их на экране. Поэтому мы ограничимся изучением процесса приведения отношений к первым трем формам.

Этот процесс включает:

- устранение повторяющихся групп (приведение к 1НФ)
- удаление частично зависимых атрибутов (приведение к 2НФ)
- удаление транзитивно зависимых атрибутов (приведение к 3НФ).

Рассмотрим каждый из этих процессов подробней.

2.2. Нормализация базы данных

Нормализация БД – это процесс уменьшения избыточности информации в БД. Метод нормализации основан на достаточно сложной теории реляционных моделей данных. Рассмотрим основные особенности и технику нормализации, не вдаваясь в теоретическое обоснование этих вопросов.

2.2.1. Избыточность данных и аномалии

При разработке структуры БД могут возникнуть проблемы, связанные:

- с избыточностью данных;
- с аномалиями.

Под избыточностью данных понимают дублирование данных, содержащихся в БД. При этом различают простое (не избыточное) дублирование и избыточное дублирование данных.

Избыточность данных при выполнении операции с ними приводит к различным аномалиям – нарушению целостности БД. Выделим аномалии:

- удаления;
- обновления;
- ввода.

Неизбыточное дублирование является естественным и допустимым, его примером является список телефонов (местных) сотрудников организации, показанный в табл. 2.2.1.

Таблица 2.2.1. Пример неизбыточного дублирования данных

Сотрудник	**Телефон**
Иванов П. Л.	*123*
Петров А. Ф.	*123*
Сидоров О. Е.	*456*
Кузнецова В. А.	*789*
Васин И. Г.	*123*

Три сотрудника имеют одинаковый номер телефона 123, что может быть, например, в случае, когда они находятся в одной комнате. Таким образом, номер телефона в таблице дублируется, однако для каждого сотрудника этот номер является уникальным. В случае удаления одного из дублированных значений номера телефона (удаления соответствующей строки таблицы) будет потеряна информация о фамилии сотрудника – Иванова П. Л., Петрова А. Ф. или Васина И. Г., что является *аномалией удаления.*

При смене номера телефона в комнате его необходимо изменить для всех сотрудников, которые в ней находятся. Если для какого-либо сотрудника этого не сделать, например, для Петрова А. Ф., то возникает несоответствие данных, связанное с *аномалией обновления.*

Аномалия ввода заключается в том, что при вводе в таблицу новой строки для ее полей могут быть введены недопустимые значения. Например, значение не входит в заданный диапазон или не задано значение поля, которое в обязательном порядке должно быть заполнено (не может быть пустым).

Теперь приведем пример *избыточного дублирования* данных. Список телефонов дополнен номерами комнат, в которых находятся сотрудники (табл. 2.2.2). При этом номер телефона указан только для одного из сотрудников – в примере для Иванова П.Л., который находится в списке первым. Вместо номеров телефонов других сотрудников этой комнаты поставлен прочерк. На практике кодировка прочерка зависит от особенностей конкретной таблицы, например, можно обозначать прочерк значением Null.

Таблица 2.2.2. Пример избыточного дублирования данных

Сотрудник	**Комната**	**Телефон**
Иванов П. Л.	17	123
Петров А. Ф.	17	-
Сидоров О. Е.	22	456
Кузнецова В. А.	8	789
Васин И. Г.	17	-

Однако при таком построении таблицы появляются проблемы.

- Номер телефона произвольного сотрудника можно получить только путем поиска в другом столбце таблицы (по номеру комнаты).
- При запоминании таблицы для каждой строки отводится одинаковая память вне зависимости от наличия или отсутствия прочерков.
- При удалении строки с данными сотрудника, для которого указан номер телефона комнаты, будет утеряна информация о номере телефона для всех сотрудников этой комнаты.

Если вместо прочерков указать номер телефона, то избыточное дублирование все равно остается. От него можно избавиться, например, с помощью разбиения таблицы на две новых (табл. 2.2.3 и 2.2.4). Разбиение – это процесс деления таблицы на несколько таблиц с целью поддержания целостности данных, т. е. устранения избыточности данных и аномалий. Таблицы связаны между собой по номеру комнаты. Для получения информации о номере телефона сотрудника из первой таблицы по фамилии сотрудника нужно считать номер его комнаты, после чего из второй таблицы по номеру комнаты считывается номер телефона.

Рассмотренное разбиение таблицы на две является примером нормализации отношений. При этом избыточность данных уменьшилась,

однако одновременно увеличилось *время доступа* к ним. Для получения номера телефона сотрудника теперь необходимо работать с двумя таблицами, а не с одной, как в предыдущем случае.

Таблица 2.2.3. Список сотрудников и номеров комнат.

Сотрудник	Комната
Иванов П. Л.	17
Петров А. Ф.	17
Сидоров О. Е.	22
Кузнецова В. А.	8
Васин И. Г.	17

Таблица 2.2.4. Список номеров комнат и телефонов.

Комната	Телефон
17	123
8	789
22	456

2.3. Приведение к нормальным формам

Процесс проектирования БД с использованием метода нормальных форм является итерационным (пошаговым) и заключается в последовательном переводе по определенным правилам отношений из первой нормальной формы в нормальные формы более высокого порядка. Каждая следующая нормальная форма ограничивает определенный тип функциональных зависимостей, устраняет соответствующие аномалии при выполнении операций над отношениями БД и сохраняет свойства предшествующих нормальных форм.

Выделяют следующую последовательность нормальных форм:

- первая нормальная форма;
- вторая нормальная форма;
- третья нормальная форма;
- усиленная третья нормальная форма, или нормальная форма Бойса-Кодда;
- четвертая нормальная форма;
- пятая нормальная форма.

Проектирование начинается с определения всех объектов, информация о которых должна содержаться в БД, и выделении атрибутов (характеристик) этих объектов. Атрибуты всех объектов сводятся в одну таблицу, которая является исходной. Затем эта таблица последовательно приводится к нормальным формам в соответствии с их требованиями. На практике обычно используются три первых нормальных формы.

Для примера спроектируем БД для хранения информации о футбольном

чемпионате страны. Будем запоминать информацию о дате матча, игравших командах и забитых голах. Сначала объединим все данные в одну исходную таблицу, имеющую следующую структуру (поля):

- дата матча;
- команда хозяев;
- команда гостей;
- игрок, забивший гол;
- признак команды;
- время.

В качестве данных о команде (хозяев и гостей) укажем ее название, город и фамилию тренера, что однозначно идентифицирует любую команду. Для игрока, забившего гол, будем указывать фамилию, а для обозначения его принадлежности к той или иной команде используем признак, например, **х** – для команды хозяев, а **г** – для команды гостей. Время гола представляет число минут, прошедших с начала матча.

Приведенная таблица имеет относительно простую структуру, на практике подобные таблицы содержат также данные о составах команд, арбитрах, времени начала игры, числе зрителей, нарушениях правил, заменах, предупреждениях и удалениях игроков и другую информацию. В структуре таблицы указаны только названия полей, поскольку тип и размерность полей на этом этапе большого значения не имеют.

Созданную таблицу можно рассматривать как однотабличную БД. Ее главным недостатком является значительная избыточность данных. Так, для каждого игрока, забившего гол, указывается дата матча и информация о команде хозяев и гостей. Дублирование данных способно привести к аномалиям, а также к существенному увеличению размера базы.

Отметим, что в таблице отсутствует информация о счете матча, т. к. ее можно получить, подсчитав голы, забитые хозяевами и гостями. Побочным явлением такого подхода является то, что если в матче не было забитых голов, то информация о соответствующем матче в таблице отсутствует. Эта ситуация исправляется автоматически при последующем разбиении исходной таблицы на таблицы матчей и голов.

2.3.1. Первая нормальная форма

Приведем исходную таблицу к первой нормальной форме, для которой должны выполняться следующие условия:

- поля содержат неделимую информацию;
- в таблице отсутствуют повторяющиеся группы полей.

Во втором и третьем полях исходной таблицы содержится информация о названии команды, городе и тренере, например, *Пахтакор Ташкент Р.Хайдаров*. Это противоречит первому требованию – информация о команде, городе и тренере является делимой. Эта информация может и должна быть разделена на три отдельных поля – название команды, город, тренер.

Согласно второму требованию, в таблице должны отсутствовать

повторяющиеся группы полей, т. е. группы, содержащие одинаковые функциональные значения. В исходной таблице таких групп нет, поэтому второму требованию она удовлетворяет. Может показаться, что повторяющимися группами полей являются поля с информацией о командах хозяев и гостей. Несмотря на то, что состав и определения названных полей полностью совпадают, эти поля имеют различное функциональное назначение и не считаются повторяющимися.

Примером таблицы, имеющей повторяющиеся группы полей, может служить табл. 2.3.1 с результатами экзаменационной сессии. В этой таблице для оценок, полученных студентами на экзаменах, необходимо создать столько полей, сколько может быть различных предметов. В связи с тем, что студент сдает не все экзамены (для них проставлены прочерки), размер записей и таблицы в целом неоправданно увеличивается. Кроме того, значительные трудности создает изменение состава предметов. Если организовать переименование предмета достаточно просто, то его удаление или добавление требует изменения структуры таблицы, что, вообще говоря, крайне нежелательно, поскольку структура таблицы обычно определяется еще на этапе проектирования БД.

Таблица 2.3.1. Результаты экзаменационной сессии

Студент	**Матема-тика**	**Информа-тика**	**Физика**	**История**	**Английский язык**
Семенов Р. О.	4	4	4	-	4
Костина В. К.	4	3	3	-	4
Папаев Д. Г.	5	5	5	-	-
Симонов П. С.	-	-	-	4	4

Вторая и третья нормальные формы касаются отношений между ключевыми и неключевыми полями.

2.3.2. Вторая нормальная форма

Ко второй нормальной форме предъявляются следующие требования:

- таблица должна удовлетворять требованиям первой нормальной формы;
- любое неключевое поле должно однозначно идентифицироваться ключевыми полями.

Записи таблицы, приведенной к первой нормальной форме, не являются уникальными и содержат дублированные данные. Так, если за одну минуту футболист забил несколько голов, то таблица будет содержать одинаковые записи. Чтобы обеспечить уникальность записей, введем в таблицу поле ключа – код матча. При этом значение ключа будет однозначно определять каждую запись таблицы.

Тогда структура таблицы примет такой вид:

– Код матча (уникальный ключ)
– Дата матча
– Название команды хозяев
– Город команды хозяев
– Фамилия тренера команды хозяев
– Название команды гостей
– Город команды гостей
– Фамилия тренера команды гостей
– Игрок
– Признак команды
– Время

Записи этой таблицы имеют значительное избыточное дублирование данных, т. к. дата матча, а также информация о командах указываются для каждого гола; Поэтому разобьем таблицу на две таблицы, одна из которых будет содержать данные о матчах, а вторая – о голах, забитых в каждом конкретном матче. Структуры этих таблиц будут следующими.

Поля таблицы матчей:
– Код матча (уникальный ключ)
– Дата матча
– Название команды хозяев
– Город команды хозяев
– Фамилия тренера команды хозяев
– Название команды гостей
– Город команды гостей
– Фамилия тренера команды гостей

Поля таблицы голов:
– Код гола (уникальный ключ)
– Код матча
– Игрок
– Признак команды
– Время

Таблицы связаны по полю Код матча, которое для таблицы матча имеет уникальное значение. Чтобы обеспечить уникальность записей таблицы голов, в нее введено ключевое поле Код гола.

2.3.3. Третья нормальная форма

Требованиями *третьей нормальной формы* являются следующие:
– таблица должна удовлетворять требованиям второй нормальной формы;
– ни одно из неключевых полей не должно однозначно идентифицироваться значением другого неключевого поля (полей).

Приведение таблицы к третьей нормальной форме предполагает выделение в отдельную таблицу (таблицы) тех полей, которые не зависят от

ключа. В таблице матчей такими являются поля с фамилиями тренеров команд, которые однозначно определяются значениями названия и города команды. (Предполагается, что в течение сезона у команды тренер не меняется, что на практике часто не выполняется, но не меняет сути рассматриваемого вопроса.) Разобьем таблицу матчей на две таблицы: одну с данными о матчах и вторую - с данными о командах. Их структура будет такой:

Таблица матчей:

- Код матча (уникальный ключ)
- Дата матча
- Код команды хозяев
- Код команды гостей

Таблица команд:

- Код команды (уникальный ключ)
- Название команды
- Город
- Фамилия тренера

Информация о команде хранится в строковых полях, которые имеют достаточно большой размер - в нашем случае приблизительно 60 символов. На практике о каждой команде необходимо запоминать больше данных, чем просто ее название, город приписки и фамилия тренера, что приводит к существенному объему информации. Поэтому для уменьшения размера записей таблицы матчей не только фамилия тренера, но и вся подробная информация о командах вынесена в таблицу команд. Вместо этого команды хозяев и гостей в таблице матчей идентифицируются кодом команды, который является уникальным ключевым значением в таблице команд.

После приведения к третьей нормальной форме база данных с информацией о футбольном чемпионате страны будет иметь структуру, показанную на рис. 2.3.1, где кроме описания таблиц обозначены также и связи между ними.

Следование требованиям нормализованных форм не всегда является обязательным. Как уже отмечалось, с ростом числа таблиц структура БД усложняется и возрастает время доступа к данным. В ряде случаев для упрощения структуры БД можно позволить частичное дублирование данных, не допуская, однако, нарушения их целостности и сохраняя их непротиворечивость.

В рассмотренной БД это относится, например, к информации о счете матча. Чтобы исключить дублирование данных, для него не отведено отдельное поле. Узнать счет матча можно с помощью двух запросов к БД, возвращающих число мячей, забитых в указанном матче хозяевами и гостями, соответственно. Запрос может иметь следующий вид:

```
SELECT COUNT(GjDwnerSign)
FROM Goal
WHERE Goal.G_Match = :MatchCode
        AND Goal.G__OwnerSign = :OwnerSign
```

Номер матча и признак команды, игрок которой забил гол, передаются в запрос через параметры MatchCode и OwnerSiqn.

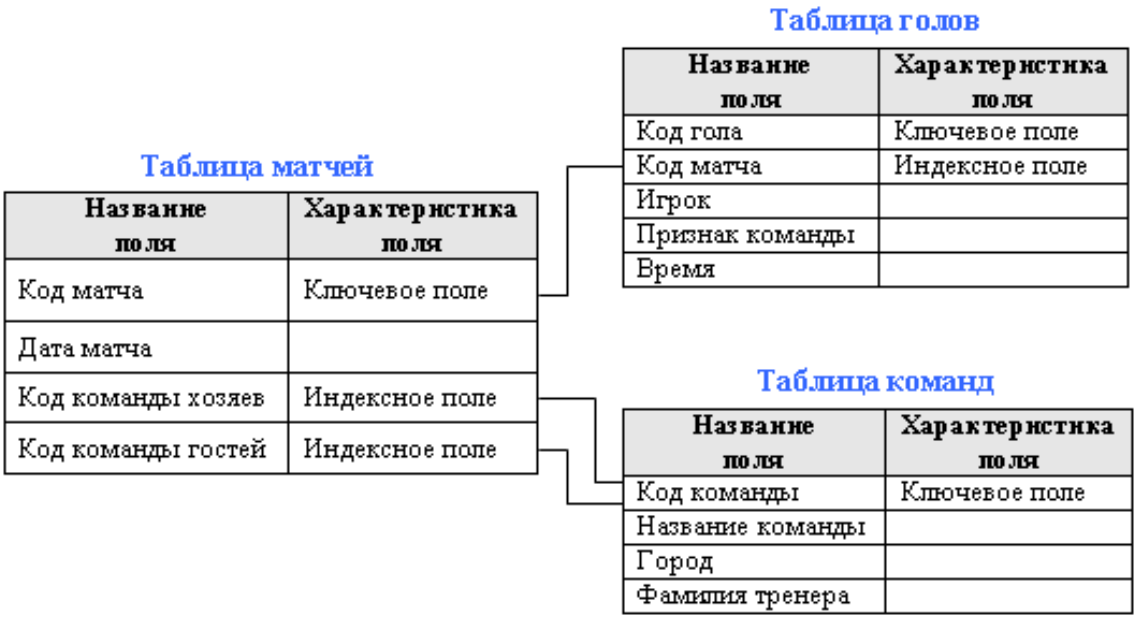


Рис. 2.3.1. Структура БД «Чемпионат по футболу».

В связи с усложнением процесса обработки данных и времени доступа к ним возможным вариантом для структуры рассмотренной БД является такой, когда счет матча запоминается в отдельном поле таблицы матчей. При этом необходимо обеспечить целостность данных при изменении БД с учетов того факта, что часть ее данных дублирована.

3. ТЕХНОЛОГИЯ СОЗДАНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Продемонстрируем возможности Delphi no работе с БД на примере создания простой информационной системы. Эту информационную систему можно разработать даже без написания кода: все необходимые операции выполняются с помощью программы Database Desktop, Конструктора формы и Инспектора объектов. Работа над информационной системой состоит из следующих основных этапов:

- создание БД;
- создание приложения.

Кроме приложения и БД, в информационную систему также входят вычислительная система и СУБД. Предположим, что компьютер или компьютерная сеть уже существуют, и их характеристики удовлетворяют потребностям будущей информационной системы. В качестве СУБД выберем Delphi.

В простейшем случае БД состоит из одной таблицы. Если таблицы уже имеются, то первый этап не выполняется. Отметим, что совместно с Delphi поставляется большое количество примеров приложений, в том числе и приложений БД. Файлы таблиц для этих приложений находятся в каталоге *c:\Program FiIes\Shared Files\Borland Shared\Data*. Готовые таблицы можно использовать также и для своих приложений.

3.1. Создание таблиц базы данных

Для работы с таблицами БД при проектировании приложения удобно использовать программу Database Desktop, которая позволяет:

- создавать таблицы;
- изменять структуры;
- редактировать записи.

Кроме того, с помощью Database Desktop можно выполнять и другие действия над БД (создание, редактирование и выполнение визуальных и SQL-запросов, операции с псевдонимами), которые будут рассматриваться позже.

Процесс создания новой таблицы начинается вызовом команды *File/New/Table* (Файл/Новая/Таблица) и происходит в интерактивном режиме. При этом разработчик должен:

- выбрать формат (тип) таблицы;
- задать структуру таблицы.

В начале создания новой таблицы в окне *Create Table* (Создание таблицы) (рис. 3.1.1) выбирается ее формат. По умолчанию предлагается формат таблицы Paradox версии 7, который мы и будем использовать. Для таблиц других форматов, например dBase IV, действия по созданию таблицы практически не отличаются.

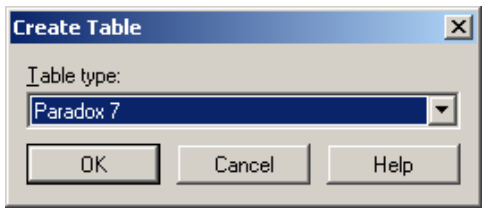


Рис. 3.1.1. Выбор формата таблицы

После выбора формата таблицы появляется окно определения структуры таблицы (рис. 3.1.2), в котором выполняются следующие действия:

– описание полей;
– задание ключа;
– задание индексов;
– определение ограничений на значения полей;
– определение условий (ограничений) ссылочной целостности;
– задание паролей;
– задание языкового драйвера;
– задание таблицы для выбора значений.

В этом списке обязательным является только первое действие, т. е. каждая таблица должна иметь хотя бы одно поле. Остальные действия выполняются при необходимости. Часть действий, такие как задание ключа и паролей, производится только для таблиц определенных форматов, например, для таблиц Paradox.

При заведении новой таблицы сразу после выбора ее формата можно не создавать структуру таблицы, а скопировать ее из другой таблицы: при нажатии на кнопку *Borrow... (Взаймы)* открывается окно *Select Borrow Table* (Выбор таблицы для заимствования) – рис. 3.1.3.

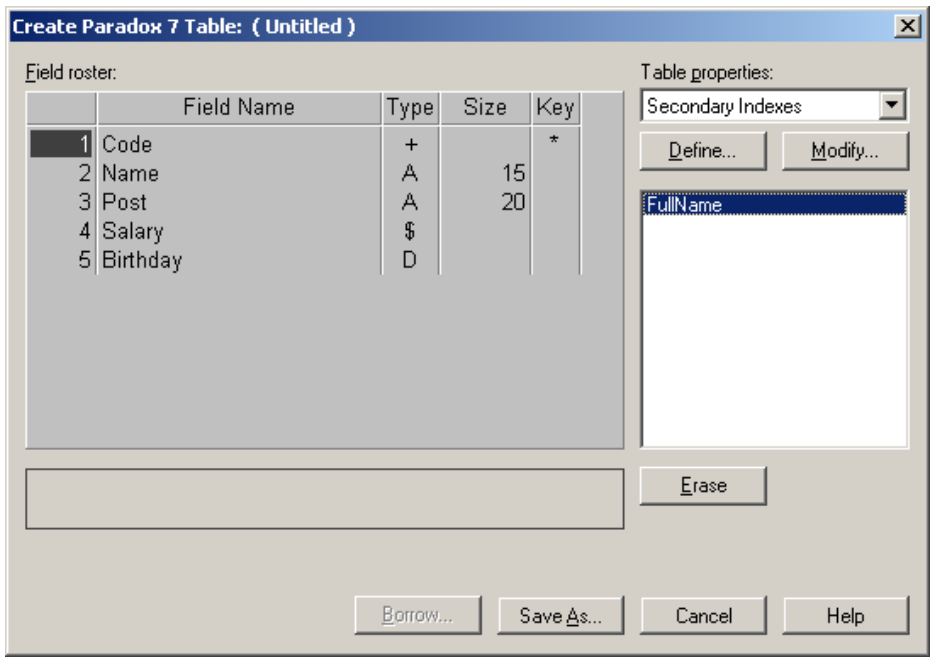


Рис. 3.1.2. Определение структуры таблицы.

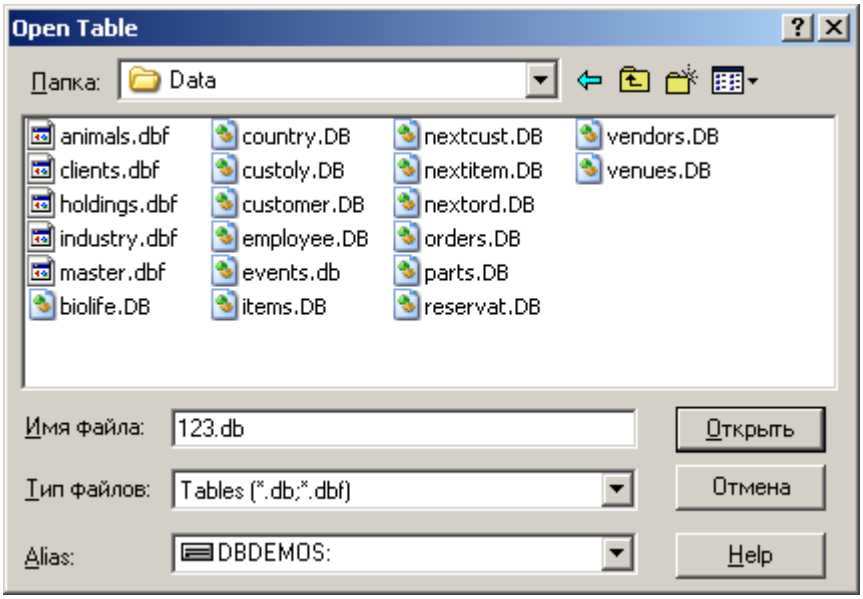


Рис. 3.1.3. Выбор таблицы для заимствования ее структуры.

В этом окне можно выбрать таблицу (главный ее файл) и указать копируемые элементы структуры, установив соответствующий флаг, например, Primary index (Первичный индекс) для ключа. После нажатия кнопки *Open* из выбранной таблицы в новую копируются описания полей, а также те элементы, для которых установлен флаг. Если какой-либо элемент в структуре копируемой таблицы отсутствует, то состояние флага не имеет значения. Например, если в выбранной таблице не определены ограничения ссылочной целостности, то в новой таблице они не появятся, даже если установлен переключатель *Referential integrity* (Ссылочная целостность).

Впоследствии скопированную структуру можно настраивать, изменяя, добавляя или удаляя ее отдельные элементы.

После определения структуры таблицы ее необходимо сохранить, нажав кнопку Save as... и указав расположение таблицы на диске и ее имя. В результате на диск записывается новая таблица, первоначально пустая, при этом все необходимые файлы создаются автоматически.

3.1.1. Описание полей

Центральной частью окна определения структуры таблицы является список *Field roster*, в котором указываются поля таблицы. Для каждого поля задаются:

- имя – в столбце *Field Name;*
- тип – в столбце *Type;*
- размер – в столбце *Size.*

Имя поля вводится по правилам, установленным для выбранного формата таблиц. Правила именования и допустимые типы полей таблиц Paradox описаны в предыдущем главе.

Тип поля можно задать, непосредственно указав соответствующий символ, например, **A** для символьного или **I** для целочисленного поля, или

выбрать его из списка (рис. 3.1.4), раскрываемого при нажатии клавиши <Пробел> или щелчком правой кнопки мыши в столбце *Type*. Список содержит все типы полей, допустимые для заданного формата таблицы. В списке подчеркнуты символы, используемые для обозначения соответствующего типа, при выборе типа эти символы автоматически заносятся в столбец *Type*.

Размер поля задается не всегда, необходимость его указания зависит от типа поля. Для полей определенного типа, например, автоинкрементного (+) или целочисленного (I), размер поля не задается. Для поля строкового типа размер определяет максимальное число символов, которые могут храниться в поле.

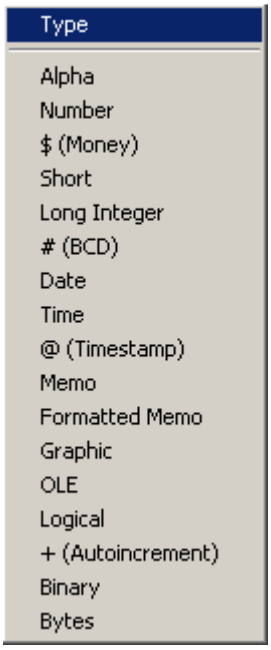


Рис.3.1.4. Типы полей Paradox 7.

Добавление к списку полей новой строки выполняется переводом курсора вниз на несуществующую строку, в результате чего эта строка появляется в конце списка. Вставка новой строки между существующими строками с описанием полей выполняется нажатием клавиши <Insert>. Новая строка вставляется перед строкой, в которой расположен курсор. Для удаления строки необходимо установить курсор на эту строку и нажать комбинацию клавиш <Ctrl>+<Delete>.

Ключ создается указанием его полей. Для указания ключевых полей в столбце ключа *(Key)* нужно установить символ *, переведя в эту позицию' курсор и нажав любую алфавитно-цифровую клавишу. При повторном нажатии клавиши отметка о принадлежности поля ключу снимается. В структуре таблицы ключевые поля должны быть первыми, т. е. верхними в списке полей. Часто для ключа используют автоинкрементное поле (см. рис. 3.1.2).

Напомним, что для таблиц Paradox ключ также называют первичным индексом (Primary Index), а для таблиц dBase ключ не создается, и его роль выполняет один из индексов.

Для выполнения остальных действий по определению структуры таблицы используется комбинированный список *Table properties* (Свойства таблицы), содержащий следующие пункты:

- **Secondary Indexes** – индексы;
- **Validity Checks** – проверка правильности ввода значений полей (выбирается по умолчанию);
- **Referential Integrity** – ссылочная целостность;
- **Password Security** – пароли;
- **Table Language** – язык таблицы (языковой драйвер);
- **Table Lookup** – таблица выбора;
- **Dependent Tables** – подчиненные таблицы.

После выбора какого-либо, пункта этого списка в правой части окна определения структуры таблицы появляются соответствующие элементы, с помощью которых выполняются дальнейшие действия.

Состав данного списка зависит от формата таблицы. Так, для таблицы dBase он содержит только пункты *Indexes* и *Table Language*.

3.1.2. Задание индексов

Задание индекса сводится к определению

- состава полей;
- параметров;
- имени.

Эти элементы устанавливаются или изменяются при выполнении операций создания, изменения и удаления индекса.

Для выполнения операций, связанных с заданием индексов, необходимо выбрать пункт *Secondary Indexes* (Вторичные индексы) комбинированного списка, при этом под списком появляются кнопки *Define* (Определить) и *Modify* (Изменить), список индексов и кнопка *Erase* (Удалить). В списке индексов выводятся имена созданных индексов, на рис. 16.2 это индекс FullName.

Напомним, что для таблиц Paradox индекс также называют вторичным индексом.

Создание нового индекса начинается с нажатия кнопки *Define*, являющейся всегда доступной. Это приводит к появлению окна *Define Secondary Index* (Задание вторичного индекса), в котором задаются, состав полей и параметры индекса (рис. 3.1.5).

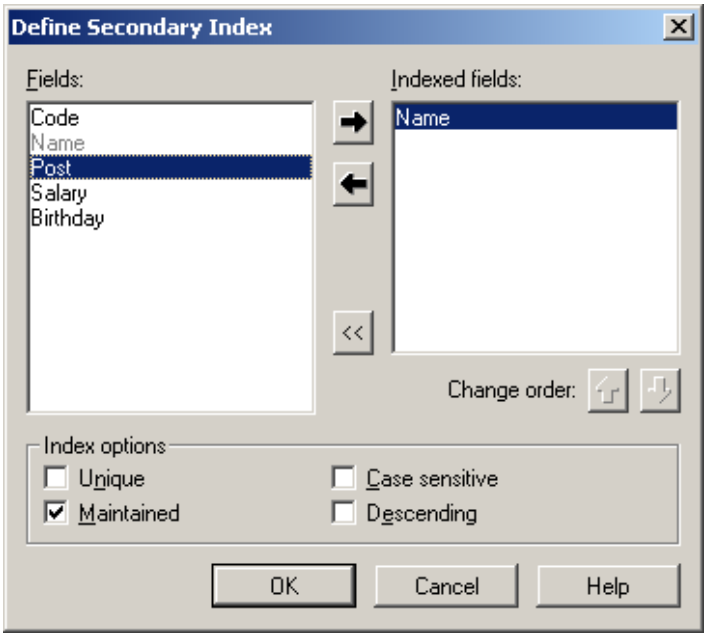


Рис. 3.1.5. Окно задания индекса.

В списке *Fields* окна выводятся имена всех полей таблицы, включая и те, которые нельзя включать в состав индекса, например, графическое поле или

поле комментария. В списке *Indexed Fields* (Индексные поля) содержатся поля, которые включаются в состав создаваемого индекса. Перемещение полей между списками выполняется выделением нужного поля (полей) и нажатием расположенных между списками кнопок с изображением горизонтальных стрелок. Имена полей, которые нельзя включать в состав индекса, выделяются в левом списке серым цветом. Поле не может быть повторно включено в состав индекса, если оно уже выбрано и находится в правом списке.

Изменить порядок следования полей в индексе можно с помощью кнопок с изображением вертикальных стрелок, имеющих общее название *Change order* (Изменение порядка). Для перемещения поля (полей) необходимо его (их) выделить и нажать нужную кнопку.

Переключатели, расположенные в нижней части окна задания индекса, позволяют указать следующие параметры индекса:

- **Unique** – индекс требует для составляющих его полей уникальных значений;
- **Maintained** – если таблица открыта, индекс автоматически не модифицируется;
- **Case sensitive** – для полей строкового типа учитывается регистр символов;
- **Descending** – сортировка выполняется в порядке убывания значений.

Так как для таблиц dBase нет ключей, то для них использование параметра *Unique* является единственной возможностью обеспечить уникальность записей на физическом уровне (уровне организации таблицы), не прибегая к программированию.

После задания состава индексных полей и нажатия кнопки ОК появляется окно *Save Index As*, в котором указывается имя индекса (рис. 3.1.6). Для удобства обращения к индексу в его имя можно включить имена полей, указав какой-нибудь префикс, например «ind». Нежелательно образовывать имя индекса только из имен полей, т. к. для таблиц Paradox подобная система именования используется при автоматическом образовании имен для обозначения ссылочной целостности между таблицами. После повторного нажатия ОК сформированный индекс добавляется к таблице, и его имя появляется в списке индексов.

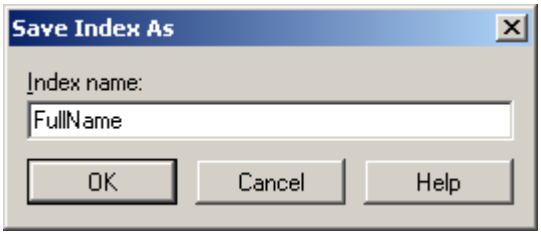


Рис. 3.1.6. Задание имени индекса.

Созданный индекс можно изменить, определив новый состав полей, параметров и имени индекса. Изменение индекса не отличается от процесса его создания. После выделения индекса в списке и нажатия кнопки **Modify** снова

открывается окно задания индекса (рис. 3.1.5). При нажатии кнопки ОК появляется окно сохранения индекса (рис. 3.1.6), содержащее имя изменяемого индекса, которое можно исправить или оставить прежним.

Для удаления индекса его нужно выделить в списке индексов и нажать кнопку *Erase.* В результате индекс удаляется без предупреждающих сообщений.

Кнопки *Modify* и *Erase* доступны, только если индекс выбран в списке.

3.1.3. Задание ограничений на значения полей

Задание ограничений на значения полей заключается в указании для полей:

- требования обязательного ввода значения;
- минимального значения;
- максимального значения;
- значения по умолчанию;
- маски ввода.

Для выполнения операций, связанных с заданием ограничений на значения полей, нужно выбрать пункт Validity Checks (Проверка значений) комбинированного списка Table Properties (рис. 3.1.2), при этом под списком появляются переключатель Required Field (Обязательное поле), поля редактирования Minimum Value, Maximum Value, Default Value, Picture (Маска ввода) и кнопка Assist (Помощь). Переключатель и поля редактирования отображают установки для поля таблицы, которое выбрано в списке (курсор находится в строке этого поля). Требование обязательного ввода значения означает, что поле не может быть пустым (иметь значение Null). Это требование действует при добавлении к таблице новой записи. До того как изменения в таблице будут подтверждены, поле должно получить какое-либо непустое значение, в противном случае генерируется ошибка. Ошибка может также возникнуть при редактировании записи, когда будет удалено старое значение поля и не присвоено новое.

Данное требование удобно использовать для так называемых *обязательных полей* таблиц, например, для поля фамилии в таблице сотрудников организации. Отметим, что обязательность ввода значения не действует на автоинкрементное поле, которое и без того является обязательным и автоматически заполняемым.

Для указания обязательности ввода значения в поле необходимо установить флаг Required Field, который по умолчанию выключен.

Для полей некоторых типов, в первую очередь числовых, денежных, строковых и даты, иногда удобно задавать диапазон возможных значений, а также значение по умолчанию. Диапазон определяется минимально и максимально возможными значениями, которые вводятся в полях редактирования Minimum Value и Maximum Value. После их задания выход значения поля за указанные границы не допускается при вводе и редактировании любым способом.

Значение поля по умолчанию указывается в редакторе Default Value. Это значение устанавливается при добавлении новой записи, если при этом для поля не указано какое-либо значение.

Отметим, что задание диапазона и значений по умолчанию возможно не для всех полей, например, они не определяются для графического поля и поля комментария. Для этих полей соответствующее поле редактирования блокируется.

В строке редактирования Picture можно задать маску (шаблон) для ввода значения поля. Ввод по маске поддерживается, например, для таких типов полей, как числовом или строковый. Его удобно использовать для ввода информации определенных форматов, например, телефонных номеров или почтовых индексов.

Для маски используются следующие символы:

- цифра;
? - любая буква (регистр не учитывается);
& - любая буква (преобразуется к верхнему регистру);
~ - любая буква (преобразуется к нижнему регистру);
@ - любой символ;
! - любой символ (преобразуется к верхнему регистру);
; - за этим символом следует буквенный символ;
* - число повторов следующего символа;

[abc] или {a,b,c} - любой из приведенных символов (a, b, или c), значения перечисляются через запятую без пробелов.

Маску можно ввести в строке Picture вручную или использовать для этого окно Picture Assistance, вызываемое нажатием кнопки Assist (рис. 3.1.7).

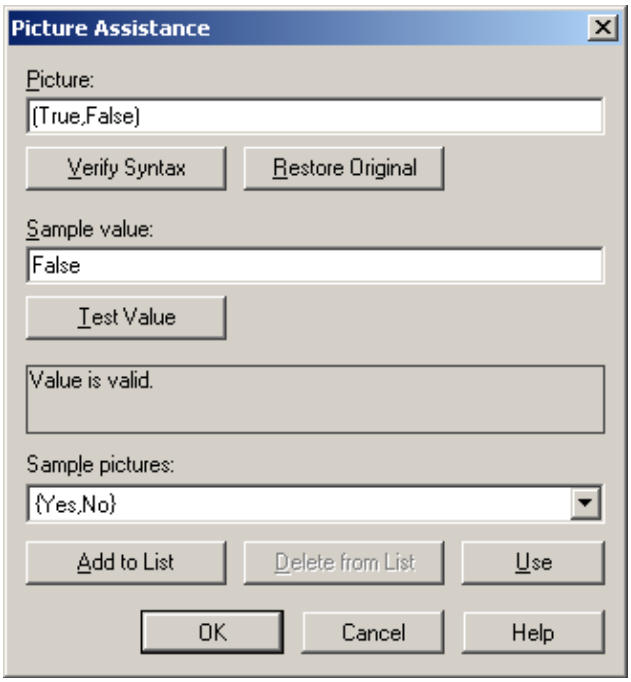


Рис. 3.1.7. Окно формирование маски.

Указанное окно помогает ввести, выбрать или откорректировать маску, а также проверить ее функционирование.

Список Sample pictures содержит образцы масок, которые выбираются нажатие кнопки Use. Выбранная маска помещается в поле редактирования Picture и доступна для изменения. Для модификации списка образцов масок служат кнопки Add to List и Delete from List: первая добавляет к списку маску, содержащуюся и поле Picture, а вторая удаляет из списка выбранную маску.

Проверка синтаксиса маски выполняется по нажатию кнопки *Verify Syntax*, результат проверки выводится в информационном поле. Кнопка *Restore Original* (Вернуть исходную) служит для восстановления начального (т. е. до начала редактирования) значения маски.

Функционирование маски можно проверить, введя в строку редактирования *Sample value* значение поля таблицы. По нажатию кнопки *Test Value* выполняется проверка введенного значения, результат проверки выводится в информационной панели.

3.1.4. Задание ссылочной целостности

Понятие ссылочной целостности относится к связанным таблицам **и** проявляется в следующих вариантах взаимодействия таблиц:

- запрещается изменение поля связи или удаление записи главной таблицы, если для нее имеются записи в подчиненной таблице;
- при удалении записи в главной таблице автоматически удаляются соответствующие ей записи в подчиненной таблице (каскадное удаление).

Для выполнения операций, связанных с заданием ссылочной целостности, необходимо выбрать пункт *Referential Integrity* комбинированного списка *Table Properties* (рис. 3.1.2). При этом, как и в случае задания индексов, появляются кнопки *Define, Modify, Erase* и список, в котором выводятся имена созданных условий ссылочной целостности.

Условие ссылочной целостности задается для подчиненной таблицы и определяется следующими элементами:

- полями связи подчиненной таблицы;
- именем главной таблицы;
- полями связи главной таблицы;
- параметрами.

Разработчик может создать, изменить или удалить условие ссылочной целостности.

Для задания условия ссылочной целостности нужно нажать кнопку *Define*, после чего появляется окно *Referential Integrity* (рис. 3.1.8).

В списке *Fields* следует выбрать поле связи и нажатием кнопки со стрелкой вправо перевести его в список *Child Fields* (Дочерние поля). Если полей связи несколько, то эти действия выполняются для каждого из них. Кнопка со стрелкой влево удаляет выбранное поле из списка полей связи.

В списке *Table* указывается главная таблица, имена таблиц выбираются

из рабочего каталога программы Database Desktop. После выбора таблицы и нажатия кнопки со стрелкой влево (рядом со списком таблиц) в список *Parent's key* автоматически заносятся ключевые поля главной таблицы.

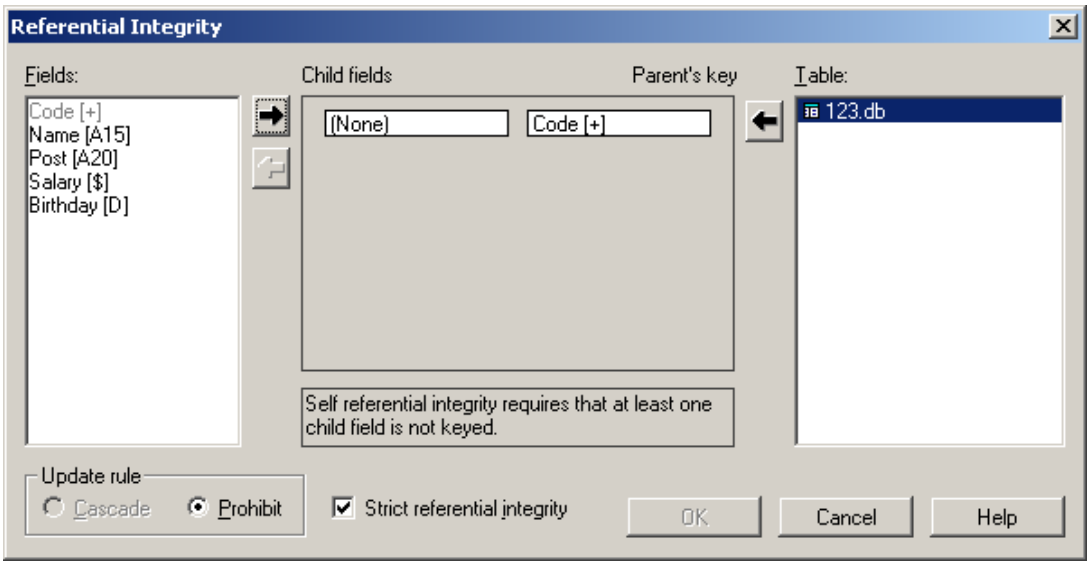


Рис. 3.1.8. Задание условия ссылочной целостности.

Параметры ссылочной целостности выбираются переключателями. Группа *Update rules* (Правила изменения) определяет вид взаимодействия таблиц при изменениях в главной таблице. Переключатель *Cascade* устанавливает режим каскадного удаления записей в подчиненной таблице при удалении соответствующей записи главной таблицы. Переключатель *Prohibit* устанавливает режим запрещения изменения поля связи или удаления записи главной таблицы, для которой имеются записи в подчиненной таблице.

Переключатель *Strict referential Integrity* (Жесткая ссылочная целостность) устанавливает защиту таблиц от модификации с использованием ранних версий программы Database Desktop (под DOS), которые не поддерживают ссылочную целостность.

После установки нужных флагов и нажатия кнопки OK появляется окно *Save Referential Integrity As*, в котором указывается имя условия (рис. 3.1.9). Напомним, что для таблиц Paradox условия ссылочной целостности именуются. Для удобства обращения к условию в его имя можно включить имена полей и таблиц, задав при этом некоторый префикс, например, *ri*. После нажатия *OK* сформированное условие ссылочной целостности добавляется к таблице, и его имя появляется в списке условий.

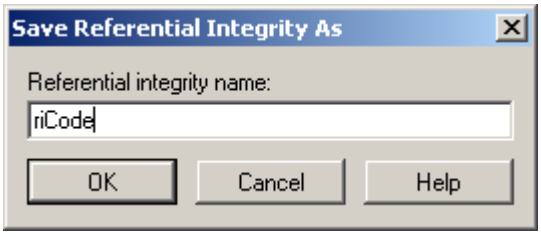


Рис. 3.1.9. Задание имени для условия ссылочной целостности.

Созданное условие ссылочной целостности можно изменить, определив новый состав полей и новые значения параметров. Изменение условия ссылочной целостности не отличается от процесса его создания: после выделения имени условия в списке и нажатия кнопки Modify открывается окно определения ссылочной целостности (рис. 3.1.8). При нажатии кнопки OK измененное условие ссылочной целостности сохраняется под тем же именем.

Для удаления условия ссылочной целостности нужно выделить его в списке и нажать кнопку Erase. Удаление производится без выдачи предупреждающих сообщений.

Кнопки Modify и Erase доступны только, если выбрано условие в списке.

3.1.5. Задание паролей

Пароль позволяет задать права доступа пользователей (приложений) к таблице. Если для таблицы установлен пароль, то он будет автоматически запрашиваться при каждой попытке открытия таблицы.

Для выполнения операций, связанных с заданием пароля, нужно выбрать строку Password Security в комбинированном списке *Table Properties* окна определения структуры таблицы (см. рис. 3.1.2). При этом под списком появляются кнопки Define и Modify. Нажатие кнопки Define вызывает окно *Password Security* (рис. 3.1.10), в котором задается пароль.

Рис. 3.1.10. Задание главного пароля.

Пароль таблицы вводится два раза – в полях Master password (Главный пароль) и Verify master password (Подтвердить главный пароль). При нажатии кнопки OK оба значения сверяются, и при их совпадении пароль принимается. Когда пароль определен, кнопка Define блокируется и становится доступной кнопка Modify изменения пароля. Ее нажатие снова вызывает окно задания пароля (рис. 3.1.10), в котором появляются кнопки Change и Delete, а поля редактирования заблокированы.

Нажатием кнопки Delete пароль удаляется, после чего его можно ввести заново. Если в качестве значения пароля указана пустая строка, то пароль для таблицы не задан. Нажатием кнопки Change поля редактирования

разблокируется и значение пароля можно изменить (в обоих полях). При этом название кнопки Change изменяется на Revert (Возврат), и ее повторное нажатие возвращает значение пароля, которое было до начала редактирования.

Рассмотренный пароль считается главным паролем, который предоставляет пользователю *полные права доступа* к таблице, включая изменение записей и структуры таблицы, в том числе смену пароля. Кроме главного пароля, можно задать для таблицы дополнительные пароли, устанавливающие пользователю *ограниченные права доступа* к таблице. Для задания дополнительных паролей нажатием кнопки Auxiliary Passwords вызывается одноименное окно (рис. 3.1.11).

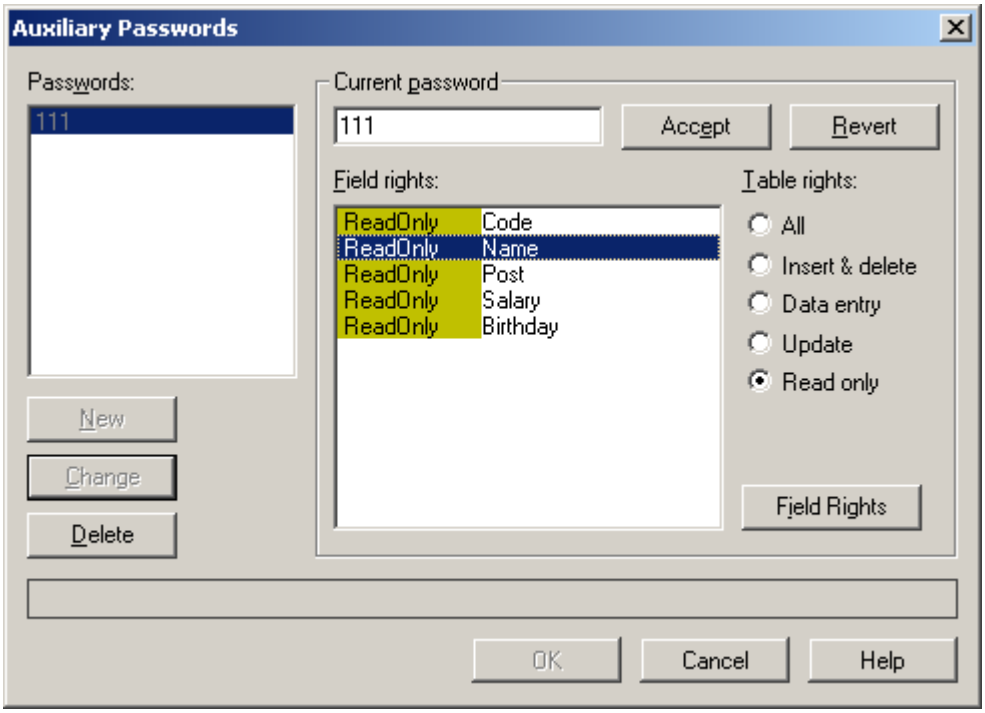


Рис. 3.1.11. Задание дополнительных паролей.

В списке *Passwords* выводятся действующие дополнительные пароли. Группа переключателей *Table rights* определяет для пароля права доступа к таблице в целом. Они могут быть следующими:

- **All** – полные права, включая изменение записей и структуры таблицы;
- **Insert & delete** – разрешены вставка и удаление, а также редактирование записей, запрещено изменение структуры таблицы;
- **Date entry** - разрешены редактирование и вставка записей, запрещены изменение структуры таблицы и удаление записей;
- **Update** – разрешены только просмотр (чтение) записей и редактирование неключевых полей;
- **Read only** – разрешен только просмотр (чтение) записей.

Права доступа к таблице действуют на все ее поля, кроме того, для каждого поля можно установить отдельные права доступа, не зависящие от прав доступа к другим полям. Права доступа к полям выводятся слева от имени

поля в списке Fields rights и содержат следующие позиции:

– **All** – чтение и изменение значения поля;
– **Read only** – только чтение значения поля;
– **None** – доступ к полю запрещен.

Смена права доступа к полю выполняется выбором поля в списке и нажатием кнопки Fields rights, при котором право циклически устанавливается очередным значением списка (All, Read only и None).

Создание пароля начинается нажатием кнопки New, после чего его имя указывается в поле редактирования Current password (Текущий пароль) и устанавливаются права доступа к таблице и ее полям. Нажатие кнопки Add заносит пароль в список дополнительных паролей.

Нажатие кнопки Change переводит выбранный и списке пароль в режим редактирования, при этом появляются кнопки Accept (Подтвердить) и Revert, а также разблокируется кнопка Delete. В процессе редактирования пароль можно изменить, удалить или оставить без изменений. После смены имени пароля и права доступа внесенные изменения утверждаются нажатием кнопки Accept. Для выхода из режима редактирования и возврата к прежним установкам пароля необходимо нажать кнопку Revert. Удаляется пароль нажатием кнопки Delete.

Отметим, что из приложения паролями можно управлять с помощью методов компонента session. Управление паролями заключается в их добавлении и удалении. Процедура AddPassword (const password: string) добавляет новый пароль, заданный параметром Password; процедура Remove Pas sword (const Password: string) удаляет указанный пароль, а процедура RemoveAllPasswords – удаляет все пароли.

3.1.6. Задание языкового драйвера

Для задания языкового драйвера нужно выбрать пункт *Table Language* (Язык таблицы) комбинированного списка Table Properties окна определения структуры таблицы (см. рис. 3.1.2). При этом под списком становится доступной кнопка Modify, открывающая окно Table Language (рис. 3.1.12). Под языком таблицы понимается языковой драйвер, используемый для этой таблицы. В поле списка Language отображается текущий драйвер.

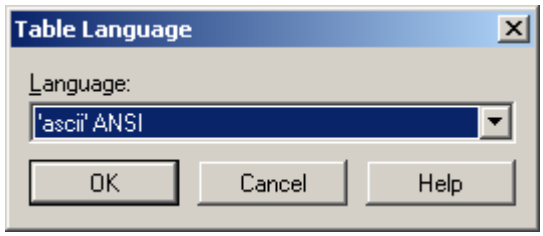


Рис. 3.1.12. Выбор языкового драйвера.

В списке можно выбрать драйвер нужного языка, для русского языка это драйвер ‘ascii’ ANSI, который корректно отображает символы русского

алфавита и выполняет с ними операции сортировки.

По умолчанию языковой драйвер определяется установками процессора баз данных BDE, поэтому целесообразно установить его (параметр *langdriver* в нужное значение, например, с помощью программы Administrator BDE. Параметры BDE и вопросы, связанные с их настройкой.

3.1.7. Задание таблицы для выбора значений

Часто возникает ситуация, когда в поле должны заноситься значения из какого-либо набора, который может формироваться различными способами. Одним и: часто используемых является вариант, когда эти значения содержатся в поле другой таблицы, а совокупность значений всех записей этого поля образует набор допустимых значений.

Для полей некоторых типов, например, строкового, числового или даты, можно определить некоторую таблицу *(таблицу выбора)*, поля которой будут использоваться для формирования набора допустимых значений. Если для поля задана таблица выбора, то в него можно ввести только значение, содержащееся в таблице выбора (в указанном поле любой записи). Задание таблицы выбора гарантирует, что в поле не будет введено недопустимое значение.

Действие набора допустимых значений распространяется также на редактирование записей таблицы программным способом: при попытке присвоить полю недопустимое значение генерируется исключительная ситуация.

Для выполнения операций, связанных с полями выбора, предназначен пункт Table Lookup (Таблица выбора) комбинированного списка Table Properties окна определения структуры таблицы (см. рис. 3.1.2). При этом под списком становится доступной кнопка Define, нажатие которой открывает окно Table Lookup (рис. 3.1.13).

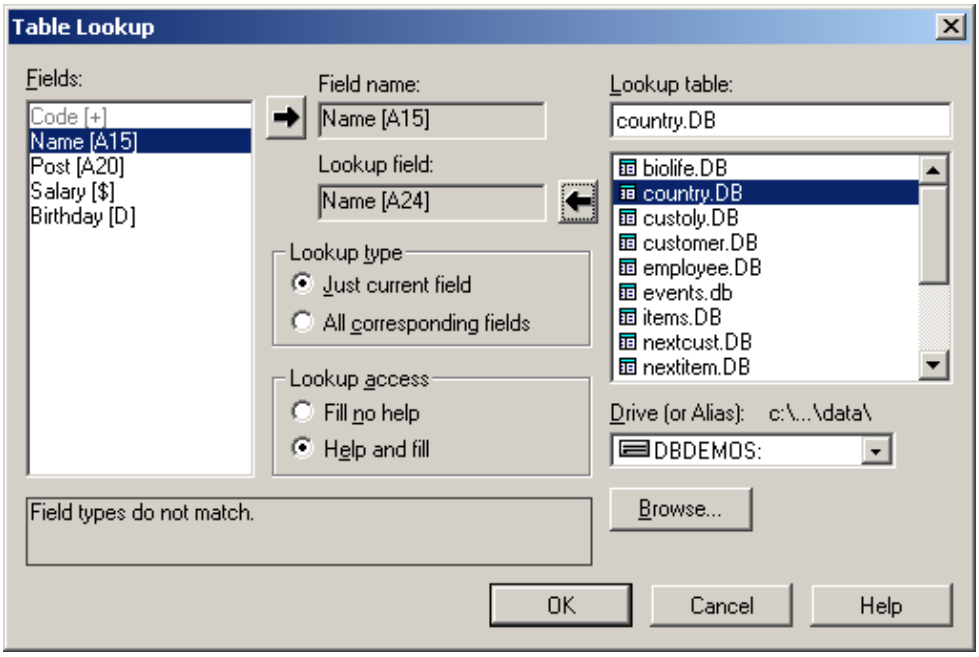


Рис. 3.1.13. Окно задания таблицы выбора.

В списке Fields выводятся имена всех полей таблицы, при этом имена полей, которые не допускают создание таблицы выбора, выделены серым цветом (например, поле автоинкрементного типа).

Имя поля, для которого задается таблица выбора, отображается в области *Field name*, для его указания следует выделить в списке *Fields* нужную строку и нажать кнопку с изображением стрелки вправо. Если указать другое поле и нажать кнопку, то имя этого поля перейдет в область *Field name* и заменит имя предыдущего поля.

В списке *Lookup table* задается таблица выбора, имя которой (имя главного файла) появляется в поле редактирования над списком. В списке *Drive* (or Alias) задается диск или псевдоним, которые определяют расположение таблица выбора. Название каталога, таблицы которого содержатся в списке Lookup table, отображается справа от названия списка *Drive* (or Alias). Таблицу также можно задать в окне Select File, вызываемом нажатием кнопки Browse (Просмотр).

После задания таблицы выбора нажатие кнопки с изображением стрелки влево переводит имя *первого ее поля*, значения которого будут использованы для формирования набора допустимых значений, в область Lookup field (Поле выбора). Типы полей обеих таблиц должны совпадать, в противном случае в информационной панели выдается сообщение об ошибке.

С помощью группы переключателей Lookup type (Тип выбора) можно задать способ взаимодействия обеих таблиц. Если включен переключатель Just current field (Только текущее поле), то таблица выбора задается только для поля, указанного в области Field name. Переключатель *All corresponding fields* назначает таблицу выбора не только указанному полю, но и всем соответствующим полям. Имена и типы этих полей должны совпадать с соответствующими полями таблицы выбора.

Группа переключателей Lookup access (Доступ к таблице выбора) определяет, каким образом пользователем используются значения из таблицы выбора.

Если включен переключатель Fill no help (Вставка без помощи), то при редактировании поля, для которого определена таблица выбора, пользователь должен знать допустимые значения этого поля. При этом термин "выбор" не совсем точно отражает взаимосвязь полей таблиц, т. к. пользователю не предлагаете! список возможных значений, из которых он может выбрать нужное ему. Пользователь сам вводит значение в поле, при этом на уровне таблицы выполняете: проверка, является ли это значение допустимым. Если значение отсутствует в указанном поле таблицы выбора, то оно не принимается.

В этом случае может помочь отображение рядом с редактируемой таблицей таблицы выбора, например, как показано на рис. 3.1.14.

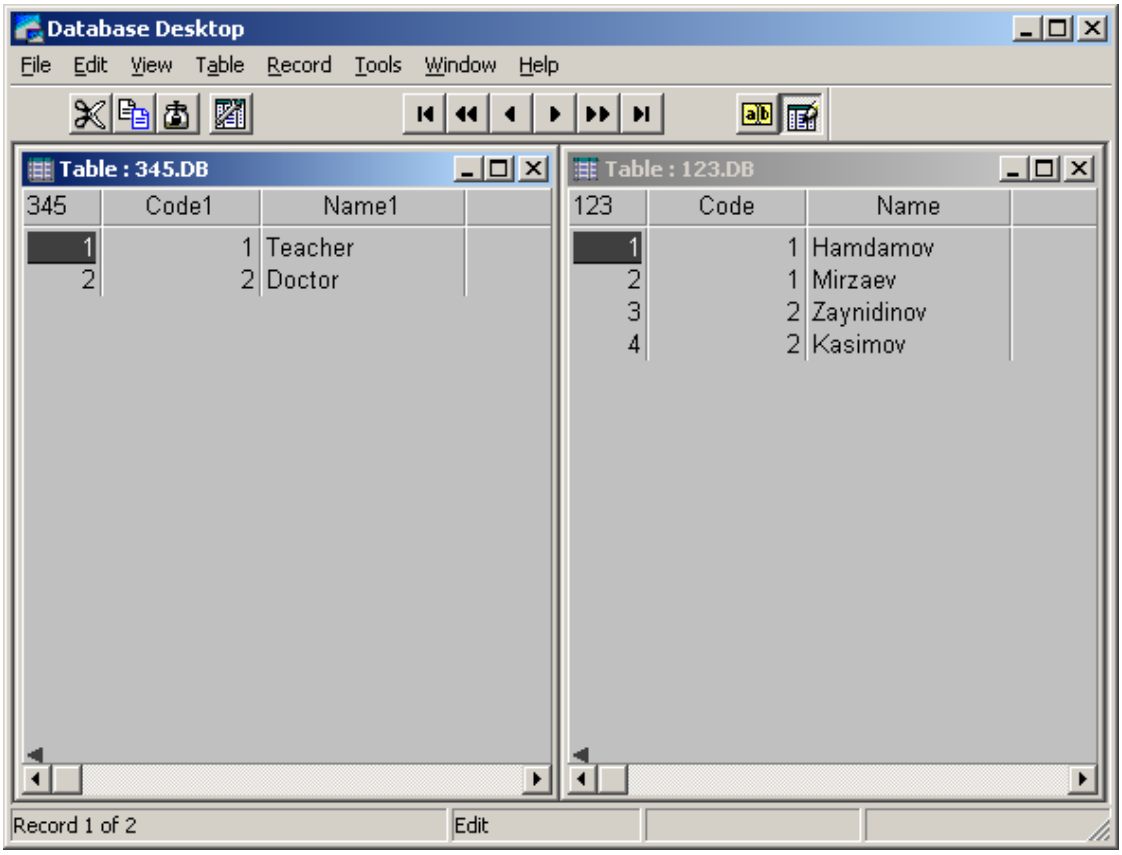


Рис. 3.1.14. Функционирования поля, для которого задана таблица выбора.

3.1.8. Просмотр списка подчиненных таблиц

Таблица, связанная с другими таблицами, является либо главной, либо подчиненной. Выше мы рассмотрели задание ссылочной целостности для подчиненной таблицы. Для главной таблицы можно просмотреть список подчиненных таблиц, отображаемый при выборе пункта *Dependent Table* (Подчиненная таблица) все того же комбинированного списка *Table Properties* (см. рис. 3.1.2). Этот список содержит имена всех таблиц, имеющих условия ссылочной целостности с участием данной таблицы.

3.1.9. Изменение структуры таблицы

Структуру существующей таблицы можно изменить, выполнив команду *Table/Restructure...* после предварительного выбора таблицы в окне программы Database Desktop. В результате открывается окно определения структуры таблицы, и дальнейшие действия не отличаются от действий, выполняемых при создании таблицы.

Если необходимо просто ознакомиться со структурой таблицы, то выполняется команда *Table/Into Structure...* В результате появляется окно определения структуры таблицы, но элементы, с помощью которых в структуру таблицы могут быть внесены изменения, заблокированы. Просмотр структуры возможен также для таблицы, с которой связаны другие приложения.

3.2. Создание приложения

Для примера рассмотрим создание приложения, позволяющего перемещаться по записям таблицы БД, просматривать и редактировать поля, удалять записи из таблицы, а также вставлять новые. Файл проекта приложения обычно не требует от разработчика выполнения каких-либо действий. Поэтому при создании приложения главной задачей является конструирование форм, в простейшем случае – одной формы.

Вид формы приложения на этапе проектирования показан на рис. 3.2.1, где на форме размещены компоненты Table1, Datasource1, DBGrid1 и DBNavigator1.

Компонент Table1 обеспечивает взаимодействие с таблицей БД. Для связи с требуемой таблицей нужно установить соответствующее значение свойствам DataBaseName, указывающему путь к БД, и TableName, указывающему имя таблицы. После задания таблицы для открытия набора данных свойству Active должно быть установлено значение True.

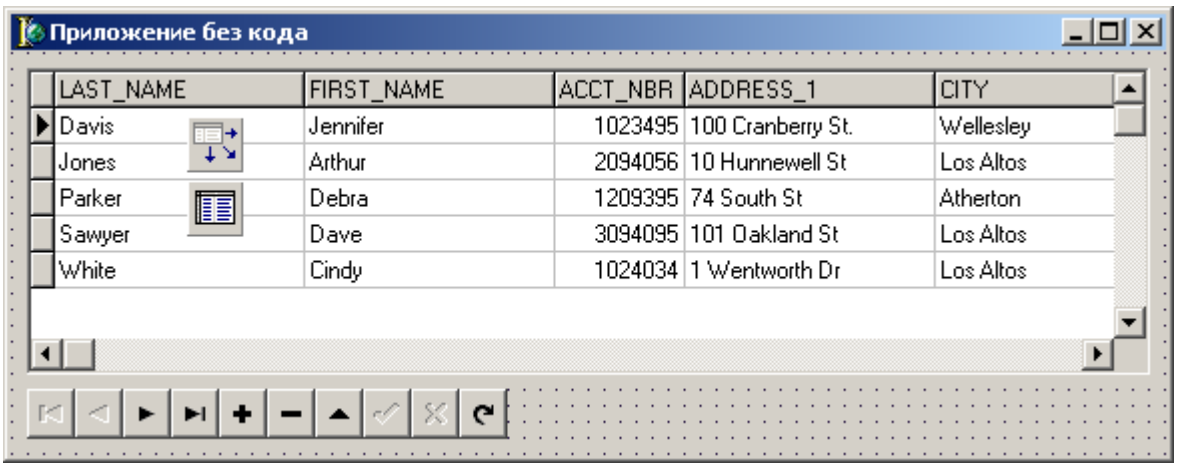


Рис. 3.2.1. Форма приложения для работы с БД.

В рассматриваемом приложении использована таблица клиентов, входящая в состав поставляемых с Delphi примеров, ее главный файл – clients.dbf. Файлы этой и других таблиц примеров находятся в каталоге, путь к которому указывает псевдоним dbdemos.

Компонент DataSource1 является промежуточным звеном между компонентом Table1, соединенным с реальной таблицей БД, и визуальными компонентами DBGrid1 и DBNavigator1, с помощью которых пользователь взаимодействует с этой таблицей. На компонент Table1, с которым связан компонент DataSource1, указывает свойство DataSet последнего.

Компонент DBGrid1 отображает содержимое таблицы БД в виде сетки, в которой столбцы соответствуют полям, а строки – записям таблицы. По умолчанию пользователь может просматривать и редактировать данные. Компонент DBNavigator1 позволяет пользователю осуществлять перемещение по таблице, редактировать, вставлять и удалять записи. Компоненты DBGrid1 и DBNavigator1 связываются со своим источником данных – компонентом

DataSource1 – через свойства DataSource.

Взаимосвязь компонентов приложения и таблицы БД и используемые при этом свойства компонентов показаны на рис. 3.2.2.

При разработке приложения значения всех свойств компонентов можно задать с помощью Инспектора объектов. При этом требуемые значения либо непосредственно вводятся в поле, либо выбираются из раскрывающихся списков. В последнем случае приложение создается с помощью мыши и не требует набора каких-либо символов с клавиатуры. В табл. 3.2.1 приведены компоненты, используемые для работы с таблицей БД, основные свойства и их значения.

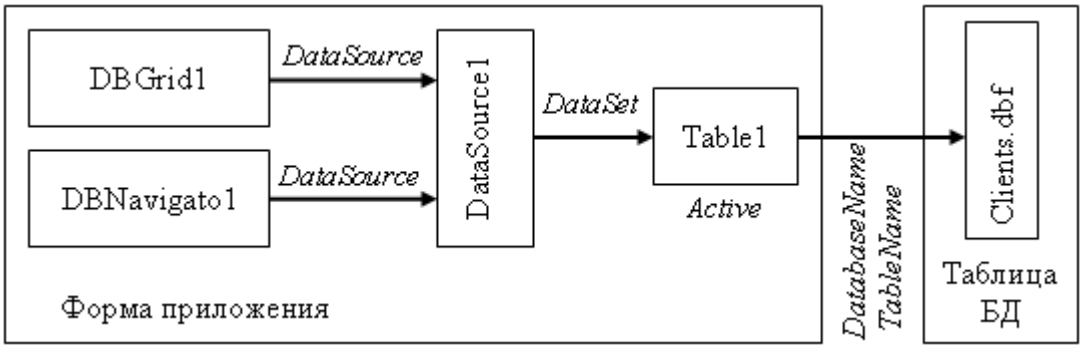


Рис. 3.2.2. Взаимосвязь компонентов приложения и таблицы БД.

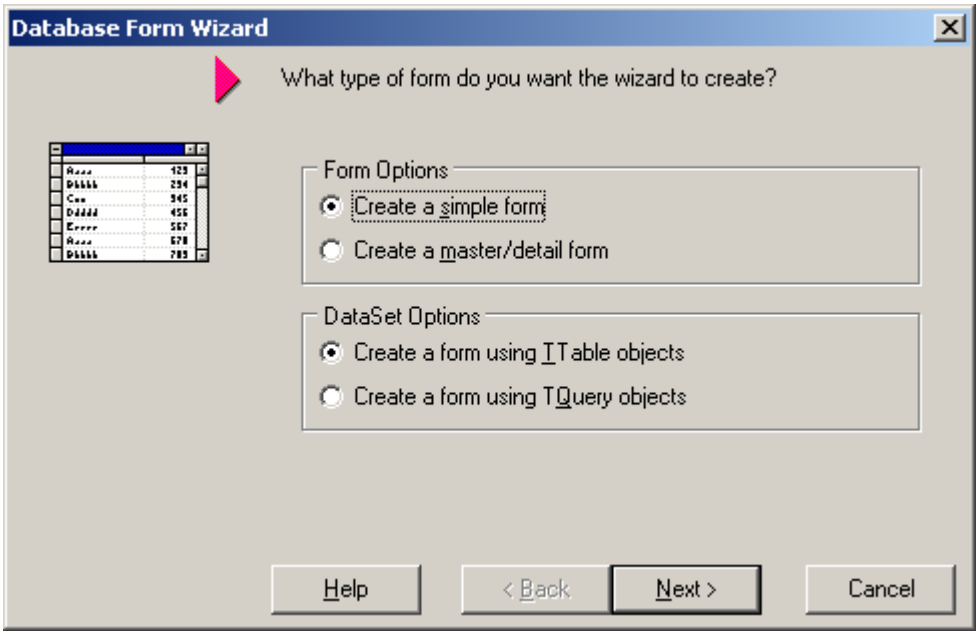


Рис. 3.2.3. Окно Database Form Wizard.

В дальнейшем при организации приложений предполагается, что названные компоненты связаны между собой именно таким образом, и свойства, с помощью которых эта связь осуществляется, не рассматриваются.

Таблица 3.2.1. Значения свойств компонентов

Компонент	Свойства	Значения
Table1	DataBaseName TableName Active	dbdemos clients.dbf true
DataSourcel	DataSet	Table1
DBGrid1	DataSource	DataSource1
DBNavigator1	DataSource	DataSource1

Для автоматизации процесса создания формы, использующей компоненты для операций с БД, можно вызвать *Database Form Wizard* (Мастер форм баз данных), показанный на рис. 3.2.3**,** Этот Мастер расположен на странице *Business* Хранилища объектов.

Мастер позволяет создавать формы для работы с отдельной таблицей и со связанными таблицами, при этом можно использовать наборы данных Table или Query.

3.3. Использование модуля данных

При конструировании формы невизуальные компоненты, используемые для доступа к данным, такие как DataSource или Table, размещаются на форме, но при выполнении приложения эти компоненты не видны. Поэтому их можно размещать в любом удобном месте формы, выступающей для них контейнером – модулем. Кроме того, для размещения невизуальных компонентов, через которые осуществляется доступ к данным, предназначен специальный объект – модуль данных.

Существует три типа модуля данных:

- простой модуль данных;
- удаленный модуль данных;
- Web-модуль.

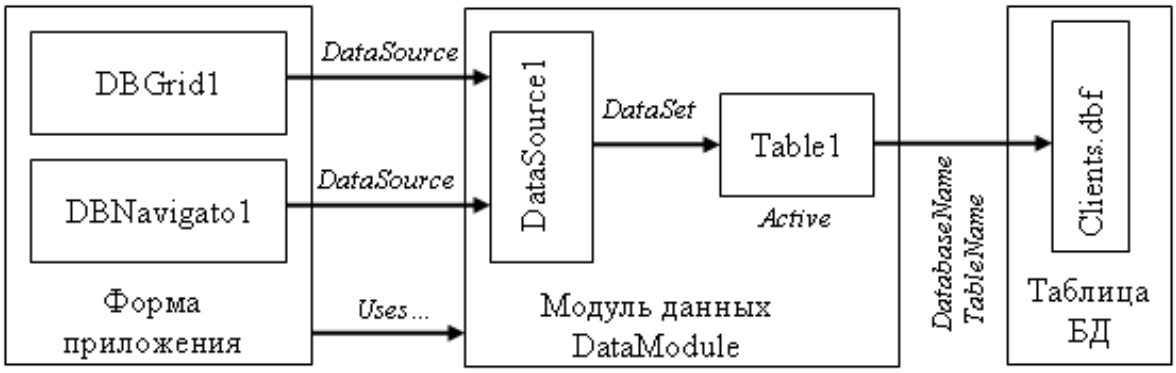


Рис. 3.3.1. Взаимосвязь компонентов приложения и таблицы БД при использовании модуля данных.

Ниже рассматривается *простой модуль* данных, который представлен объектом DataModule. Использование удаленного модуля данных и Web-модуля изучается при рассмотрении трехуровневых приложений и публикации БД в Интернете.

Если применяется простой модуль данных, то взаимосвязь компонентов приложения и таблицы БД имеет вид, показанный на рис. 3.3.1.

Модуль данных, как и форма, является контейнером для своих невизуальных компонентов, и для него создается модуль кода с расширением PAS. Добавление модуля данных к проекту выполняется командой *File/New/DataModule* главного меню Delphi. В окне модуля компоненты размещаются таким же образом, как и на форме (рис. 3.3.2). При выборе объекта в Инспекторе объектов отображаются его свойства, значения которых можно просматривать и изменять.

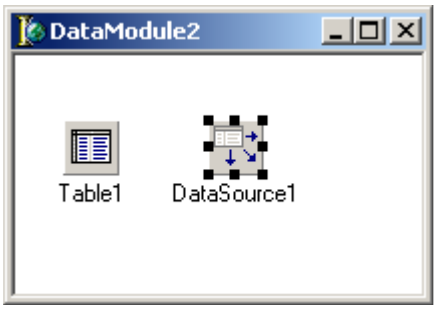


Рис. 3.3.2. Модуль данных.

При обращении к содержащимся в модуле данных компонентам для них указывается составное имя, в которое, кроме имени компонента, входит также имя модуля данных. Составное имя имеет формат

<Имя модуля данных>.<Имя компонента>

Ниже приводится пример кода, в котором осуществляется обращение к компонентам модуля данных.

```
procedure TForml.FormCreate(Sender: TObject};
begin
   DataModule2.Table1.DatabaseName:='dbdemos' ;
   DataModule2.Table1.TableName:='clients.dbf';
   DataModule2.DataSource1.DataSet:=DataModule2.Table1;
   DBGrid1.DataSource:=DataModule2.DataSource1;
   DBNavigator1.DataSource:=DataModule2.DataSource1;
   DataModule2.Table1.Active:=true;
end;
```

Для компонентов выполняется установка значений свойств, связывающих между собой эти компоненты и таблицу БД. Значения свойств устанавливаются

динамически в процессе выполнения приложения, для чего использован обработчик события создания главной формы приложения. В составных именах компонентов доступа к данным, которыми являются источник данных DataSource1 и набор данных Table1, указывается имя модуля данных DataModule2.

Чтобы обеспечить возможность доступа к компонентам модуля данных в модуле формы, в список *uses* раздела *implementation* необходимо включить ссылку на модуль данных:

```
uses  unit2;
```

Ссылку на другой модуль можно написать самостоятельно, но Delphi дает возможность вставить ее автоматически. При выборе команды *File/Use Unit...* появляется диалоговое окно *Use Unit* (рис. 3.3.3). После выбора нужного модуля и нажатия кнопки *OK* он добавляется в список.

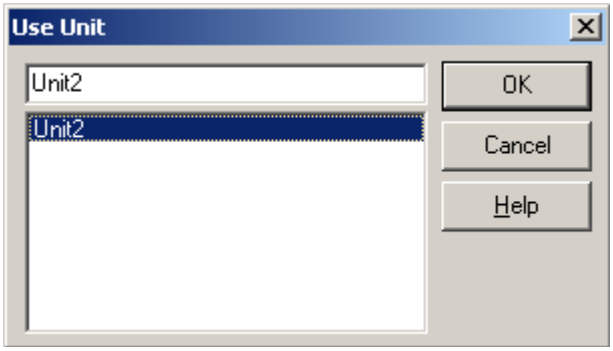


Рис. 3.3.3. Окно выбора модуля.

Если ссылка на требуемый модуль отсутствует, но в коде используется имя модуля данных, то при компиляции приложения появляется диалоговое окно *Information* (рис. 3.3.4). В нем сообщается о том, что форма ссылается на другую форму, объявленную в модуле, отсутствующем в списке *uses* формы. Для автоматического добавления модуля в список достаточно нажать кнопку *Yes*.

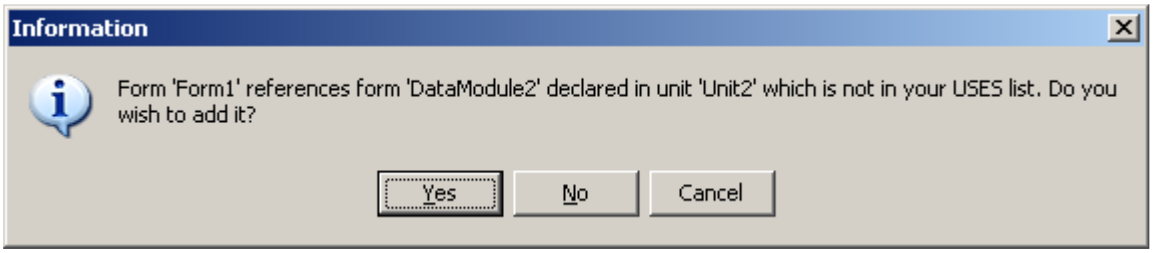


Рис. 3.3.4. Диалог добавления модуля в список *uses*.

Помимо компонентов доступа к данным, которыми являются Session, Database, Table, Query, StoredProc, BatchMove и др., в модуле данных можно

размещать также невизуальные компоненты, не имеющие прямого отношения к БД, например, ImageList, OpenDialog или Timer.

Модуль данных позволяет:

– отделить управление БД от обработки данных;
– создать модуль, совместно используемый несколькими приложениями.

Основным назначением модуля данных является *централизованное хранение* компонентов доступа к данным, а также кода для этих компонентов, в частности, обработчиков событий. В модуле данных удобно размещать код, выполняющий управление БД, например, реализацию бизнес-правил.

Использование простого модуля данных несколькими приложениями позволяет ускорить разработку приложений, т. к. готовый модуль данных впоследствии можно включать в новые приложения. Кроме того, управление БД через общий модуль дает возможность определить для всех пользователей одинаковые режимы и правила работы с базой, а также делает более простым изменение этих режимов и правил.

Однако для небольших приложений использование простого модуля данных не всегда оправдано, т. к. может затруднить, а не облегчить разработку приложения.

Удаленный модуль данных предназначен для работы с удаленными БД в трехуровневой архитектуре "клиент-сервер" и используется для создания сервера приложения – промежуточного уровня между приложением и сервером БД.

Web-модуль предназначен для работы с БД в сети Интернет и является посредником между браузером и сервером БД.

4. РАБОТА С УДАЛЕННЫМИ БАЗАМИ ДАННЫХ

Ранее были рассмотрены локальные базы данных, когда и БД, и взаимодействующее с ней приложение располагаются на одном компьютере. В данной главе мы рассмотрим некоторые особенности работы с удаленными БД, используемыми в сети, где приложение и БД располагаются на разных компьютерах.

4.1. Основные понятия

В принципе локальную БД также можно использовать для коллективного доступа, т. е. в сетевом варианте. В этом случае файлы базы данных и приложении для работы с ней располагаются на сервере сети. Пользователь запускает со своего компьютера находящееся на сервере приложение, при этом у него запускается копия приложения. Можно установить приложение и непосредственно на компьютере пользователя, в этом случае приложению должно быть извести местонахождение общей БД, заданное, например, через псевдоним. Подобный сетевой вариант использования локальной БД соответствует архитектуре "файл сервер".

Достоинствами сетевой архитектуры "файл-сервер" являются простота разработки и эксплуатации БД и приложения. Разработчик фактически создает локальную БД и приложение, которые затем просто используются в сетевом варианте. При этом не требуется дополнительное программное обеспечение, связанно с организацией работы с БД.

Однако архитектуре "файл-сервер" свойственны и существенные недостатки:

- Для работы с данными используется навигационный способ доступа, при этом по сети циркулируют большие объемы данных. В результате сеть оказывается перегруженной, что является причиной ее низкого быстродействия плохой производительности при работе с БД.
- Требуется синхронизация работы отдельных пользователей, связанная с блокировкой в таблицах тех записей, которые редактирует другой пользователь.
- Приложения не только обрабатывают данные, но и управляют самой базой данных. В связи с тем, что управление БД осуществляется с разных компьютеров, затрудняются организация контроля доступа, соблюдение конфиденциальности и поддержание целостности БД.

Из-за этих недостатков архитектура "файл-сервер", как правило, используется в небольших сетях. Для сетей с большим количеством пользователей предпочтительным вариантом (а порой и единственно возможным) является архитектура "клиент-сервер".

4.1.1. Архитектура "клиент-сервер"

В сетевой архитектуре "клиент-сервер" БД размещается на компьютере-сервере сети (сервере или удаленном сервере) и называется также удаленной

БД. Приложение, осуществляющее работу с этой БД, находится на компьютере пользователя. Приложение пользователя является *клиентом*, его также называют *приложением-клиентом*.

Клиент и сервер взаимодействуют следующим образом. Клиент формирует и отсылает запрос (SQL-запрос) серверу, на котором размещена БД. Сервер выполняет запрос и выдает клиенту в качестве результатов требуемые данные.

Таким образом, в архитектуре "клиент-сервер" клиент посылает запрос и получает только те данные, которые ему действительно нужны. Вся обработка запроса выполняется на удаленном сервере.

К достоинствам такой архитектуры относятся следующие факторы:

- Для работы с данными используется реляционный способ доступа, что снижает нагрузку на сеть.
- Приложения напрямую не управляют базой, управлением занимается только сервер. В связи с этим можно обеспечить высокую степень защиты данных.
- В приложении отсутствует код, связанный с управлением БД, поэтому приложения упрощаются.

Отметим, что сервером называют не только компьютер, но также и специальную программу, которая управляет БД. Так как в основе организации обмена данными между клиентом и сервером лежит язык SQL, то такую программу еще называют *SQL-сервером*, а БД - *базой данных SQL*. В широком смысле слова под сервером понимают компьютер, программу и саму базу данных. SQL - серверами являются промышленные СУБД, такие как: InterBase, Oracle, Informix, Sybase, DB2, Microsoft SQL Server и др. Каждый из серверов имеет свои преимущества и особенности, связанные, например, со структурой БД и реализацией языка SQL, которые необходимо учитывать при разработке приложения. Далее мы будем понимать под сервером программу (т. е. SQL-сервер), а установленную на компьютере-сервере базу данных будем называть удаленной БД.

При работе в архитектуре "клиент-сервер" приложение должно:

- выполнять соединение с сервером и отключение от него;
- формировать и отсылать запрос серверу, получая от него результаты выполнения запроса;
- выполнять обработку полученных данных.

При этом обработка данных не имеет принципиальных отличий по сравнению с обработкой данных в локальных БД.

4.1.2. Сервер и удаленная БД

Удаленная БД, как и локальная, представляет собой совокупность взаимосвязанных таблиц. Однако данные этих таблиц, как правило, содержатся в одном общем файле. Как и в случае с локальной БД, для таблиц удаленной БД могут устанавливаться связи (отношения), ограничения ссылочной целостности, ограничения на значения столбцов и т. д.

Для управления БД сервер использует:

- триггеры;
- генераторы;
- хранимые процедуры;
- функции, определяемые пользователем;
- механизм транзакций;
- механизм кэшированных изменений;
- механизм событий.

Многие из перечисленных элементов обеспечиваются возможностями языка SQL сервера, в котором, по сравнению с локальной версией, имеются существенные особенности, рассматриваемые ниже.

4.1.3. Средства работы с удаленными БД

Система Delphi обеспечивает разработку приложений для различных серверов, предоставляя для этого соответствующие средства. Отметим, что многие изложенные ранее принципы разработки приложений и средства для работы с локальными БД относятся и к работе с удаленными БД. В частности, для разработки приложений используются такие компоненты, как источник данных DataSource, наборы данных Table и Query, сетка DBGrid и др.

Для реализации реляционного способа доступа к удаленной БД необходимо использовать только средства языка SQL. Поэтому в качестве компонентов должны выбираться такие, как Query, StoredProc или UpdateSQL. Кроме того, для набора данных нельзя использовать методы, характерные для навигационного способа доступа, например, Next и Previous для перемещения текущего указателя или Edit, Insert, Append или Delete для изменения записей.

Напомним, что если при выполнении с помощью компонента Query модифицирующего БД запроса нет необходимости в получении результирующего набора данных, то этот запрос предпочтительнее выполнять с помощью метода ExecSQL.

Например:

```
Queryl,Close;
Queryl.SQL.Clear;
Queryl.SQL,Add('DELETE FROM Personnel WHERE Position='Менеджер');
Queryl.ExecSQL;
```

Для работы с таблицами и запросами по-прежнему можно использовать такие программы, как Database Desktop и SQL Explorer.

Средства Delphi, предназначенные для работы с удаленными БД, можно разделить на два вида:

- инструментальные средства;
- компоненты.

К инструментальным средствам относятся специальные программы и пакеты, обеспечивающие обслуживание БД вне разрабатываемых приложений.

В их числе могут быть выделены следующие средства:

- InterBase Server Manager – программа управления запуском сервера InterBase;
- IBConsole – консоль сервера InterBase;
- SQL Monitor – программа отслеживания порядка выполнения SQL-запросов к удаленным БД.

Отметим, что в предыдущей версии сервера вместо программы IBConsole использовалась InterBase Windows Interactive SQL (WJSQL). Кроме того, значительно упростилась программа InterBase Server Manager, а многие ее функции теперь реализованы в программе IBConsole.

Компоненты предназначены для создания приложений, выполняющих операции с удаленной БД. Перечислим наиболее важные из них:

- Database – соединение с БД;
- Session – текущий сеанс работы с БД;
- StoredProc – вызов хранимой процедуры;
- UpdateSQL – модификация набора данных, основанного на SQL-запросе;
- DCOMConnection – DCOM-соединение;
- Компоненты страниц DbExpress и InterBase Палитры компонентов.

Отметим, что многие из названных компонентов, например, Database, UpdateSQL и Session, используются также при работе с локальными БД. Так компонент Database позволяет реализовать механизм транзакций, как было показано в главе, посвященной навигационному способу доступа к данным. Однако наиболее часто эти компоненты применяются именно при работе с удаленными базами.

Часть компонентов, например, клиентский набор данных ClientDataSet и соединение с сервером DCOMConnection, предназначена для работы в трехзвенной архитектуре "клиент-сервер" ("тонкий" клиент) и используется для построения сервера приложений.

В основе операций, выполняемых с удаленными БД, как с помощью инструментальных средств, так и программно, лежит язык SQL. Например, при создании таблицы с помощью программы IBConsole (в ее окне **Interactive SQL)** необходимо набрать и выполнить SQL-запрос (оператор) Create Table. Если создание таблицы осуществляется из приложения пользователя, то для этой цели используется набор данных Query, который выполняет такой же запрос. Различие заключается в основном в том, каким образом выполняется SQL-запрос к удаленной БД.

Итак, для удаленных БД разница между средствами, используемыми в приложении, и инструментальными средствами намного меньше, чем для локальных баз данных. Поэтому далее для удаленных БД использование инструментальных средств (на примере программы IBConsole) и создание приложений рассматриваются параллельно.

4.2. Сервер InterBase

Все серверы имеют похожие принципы организации данных и управления ими. В качестве примера рассмотрим работу с сервером InterBase 6.x, который является "родным" для Delphi. Совместно с Delphi поставляются две части сервера InterBase 6.x: серверная и клиентская. Не смотря на то, что сервер InterBase поставляется совместно с Delphi, устанавливается он отдельно: после установки Delphi выдается запрос на установку сервера InterBase. Установка происходит в автоматическом режиме, основные файлы сервера копируются в подкаталог INTERBASE, находящийся в каталоге BORLAND.

Серверная часть InterBase является локальной версией сервера InterBase и используется для отладки приложений, предназначенных для работы с удаленными БД, позволяя на одном компьютере проверить их в сетевом варианте. После отладки на локальном компьютере приложение можно перенести на сетевые компьютеры без изменений, для чего нужно:

- скопировать БД на сервер;
- установить для приложения новые параметры соединения с удаленной БД.

Скопировать БД можно с помощью программ типа Проводник Windows.

Клиентская часть нужна для обеспечения доступа приложения к удаленной БД.

При разработке БД и приложений с использованием локальной версии сервера InterBase нужно иметь в виду, что она имеет ряд ограничений и может не поддерживать, например, механизм событий сервера или определяемые пользователем функции. Полноценная версия сервера InterBase приобретается и устанавливается отдельно от Delphi.

Как уже упоминалось, в основе работы с удаленной БД лежат возможности языка SQL, обеспечивающие соответствующие операции. Назначение и возможности языка SQL для удаленных БД в принципе совпадают с назначением и возможностями этого языка для локальных БД. Ниже мы рассмотрим особенности языка SQL для удаленных БД.

При рассмотрении операторов языка будем опускать несущественные операнды и элементы. При описании формата операторов языка SQL используются следующие правила:

- символы “ < ” и “ > ” обозначают отдельные элементы формата операторов, например, имена таблиц и столбцов, и при записи операторов SQL не указываются;
- в квадратные скобки заключаются необязательные элементы конструкций языка;
- элементы списка, из которого при программировании можно выбрать любой из этих элементов, разделяются знаком |, а сам список заключается в фигурные скобки.

Для наглядности зарезервированные слова языка SQL будем писать строчными буквами, а имена – прописными. Регистр букв не влияет на интерпретацию операторов языка.

4.2.1. Бизнес-правила

Как уже отмечалось, бизнес-правила представляют собой механизмы управления БД и предназначены для поддержания БД в целостном состоянии. Кроме того, они нужны для реализации ограничений БД, а также для выполнения ряда других действий, например, накапливания статистики работы с БД.

Бизнес-правила можно реализовывать на физическом и на программном уровнях. В первом случае эти правила задаются при создании таблиц и входят в структуру БД. Для этого в синтаксис оператора CREATE table включаются соответствующие операнды, например, default (значение по умолчанию). В дальнейшей работе нельзя нарушить или обойти ограничение, заданное на *физическом* уровне.

На программном уровне бизнес-правила можно реализовать в сервере и в приложении. Причем эти бизнес-правила не должны быть определены на физическом уровне. Для реализации бизнес-правил в сервере обычно используются триггеры. Достоинствами такого подхода является то, что вычислительная нагрузка по управлению БД целиком ложится на сервер, что снижает нагрузку на приложение и сеть, а также то, что действие ограничений распространяется на все приложения, осуществляющие доступ к БД. Однако одновременно снижается гибкость управления БД. Кроме того, нужно учитывать, что средства отладки триггеров и хранимых процедур сервера развиты недостаточно хорошо.

Для программирования бизнес-правил в приложении используются компоненты и их средства. Достоинство такого подхода заключается в легкости изменения бизнес-правил и возможности определить правила "своего" приложения. Недостатком является снижение безопасности БД, связанное с тем, что каждое приложение может устанавливать свои правила управления БД. В главе, посвященной навигационному способу доступа, мы рассмотрели программирование бизнес-правил в приложении на примере каскадного удаления записей в связанных локальных таблицах.

4.2.2. Организация данных

Информация всей БД сервера InterBase хранится в одном файле, который имеет расширение GDB. Размер этого файла может составлять единицы, и даже десятки гигабайт. Отметим, что аналогичный размер БД имеет СУБД Microsoft SQL Server, в то время как для более мощных СУБД Oracle и Sybase размер БД достигает десятков и сотен гигабайт.

В отличие от локальной БД, структуру которой составляли таблицы (отдельные или связанные), удаленная БД имеет более сложную структуру, которая включает в свой состав следующие элементы:

- таблицы;
- индексы;
- ограничения;
- домены;

– просмотры;
– генераторы;
– триггеры;
– функции пользователя;
– хранимые процедуры;
– исключения;
– BLOB-фильтры;
– привилегии.

Элементы структуры удаленной БД также называют *метаданными*. Слово "мета" имеет смысл "над", т. е. метаданные представляют собой данные, которые описывают структуру БД.

Для InterBase максимальное число таблиц в БД равно 65536, а максимальное число столбцов в таблице – 1000. Отметим, что таблицы InterBase имеют меньшее число допустимых типов столбцов (полей), чем таблицы локальных БД Paradox. Типы столбцов базы InterBase приведены в табл. 4.2.1.

Таблица 4.2.1. Типы полей таблиц InterBase

Тип	Описание
SMALLINT	числовое поле длиной 2 байта, которое может содержать только целые числа в диапазоне от -32768 до 32767
INTEGER	числовое поле длиной 4 байта, которое может содержать целые числа в диапазоне от -2147483648 до 2147483648
FLOAT	числовое поле длиной 4 байта, значение которого может быть положительным и отрицательным. Диапазон чисел - от 3.4*10-38 до 3.4*1038 с 7 значащими цифрами
DOUBLE PRECISION	числовое поле длиной 8 байт (длина зависит от платформы), значение которого может быть положительным и отрицательным. Диапазон чисел - от 1.7*10-308 до 1.7*10308 с 15 значащими цифрами
CHARACTER(N)	строка символов фиксированной длины (0-32767 байт), содержащая любые печатаемые символы. Число символов зависит от Character Set, установленного в InterBase для данного поля или для всей базы данных.
VARCHAR(N)	строка символов переменной длины (0-32767 байт), содержащая любые печатаемые символы. Число символов также зависит от Character Set, установленного в InterBase для данного поля или для всей базы данных

DATE	поле даты длиной 8 байт, значение которого может быть от 1 января 100 года до 11 декабря 5941 года (время также содержится)
BLOB	поле, содержащее любую двоичную информацию. Может иметь любую длину. Database Desktop не имеет возможности редактировать поля типа BLOB
ARRAY	поле, содержащее массивы данных. InterBase позволяет определять массивы, имеющие размерность 16. Поле может иметь любую длину. Однако, Database Desktop не имеет возможности не только редактировать поля типа ARRAY, но и создавать их
TEXT BLOB	подтип BLOB-поля, содержащее только текстовую информацию. Может иметь любую длину. Database Desktop не имеет возможности редактировать поля типа TEXT BLOB

В таблицах InterBase отсутствуют такие типы, как логический и автоинкрементный. Логический тип заменяется типом CHARfl), а вместо автоинкрементного типа для обеспечения уникальных значений используются генераторы и триггеры.

4.2.3. Запуск сервера

Для запуска сервера используется программа InterBase Server Manager (рис. 4.2.1), вызываемая одноименной командой главного меню Windows или через панель управления. Отметим, что в предыдущих версиях Delphi эта программа имела гораздо больше возможностей по управлению сервером InterBase и позволяла, например, управлять подключением к серверу пользователей и просматривать информацию о БД. В Delphi 6 все эти функции теперь реализует программа IBConsole.

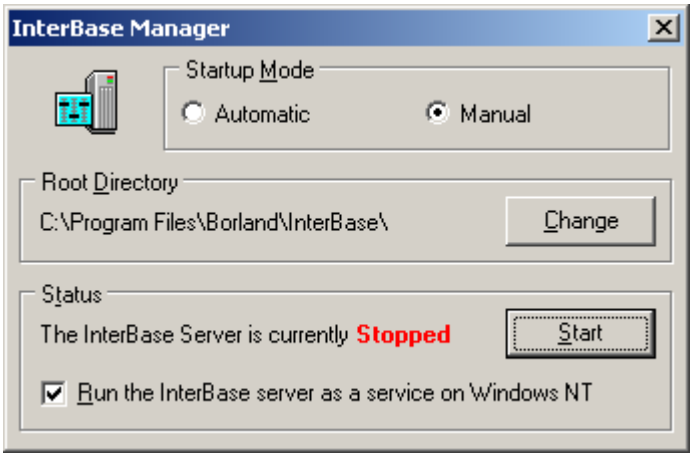


Рис. 4.2.1. Окно программы InterBase Server Manager.

Состояние сервера выводится в надписи *Status:* запущенному состоянию сервера соответствует надпись *Running,* остановленному - *Stopped.*

Сервер InterBase может запускаться автоматически или в ручном режиме, чем управляют переключатели группы *Start Mode* (Режим запуска). Если выбран переключатель *Automatic,* то сервер будет автоматически вызываться при каждом запуске (перезапуске) Windows. Если же выбран ручной запуск *(Manual),* то сервер вызывается нажатием кнопки *Start.* После запуска сервера кнопка *Start* изменяет свое название на *Stop,* и ее повторное нажатие приводит к остановке сервера.

В Windows NT сервер можно запустить как службу (service). Включенное состояние переключателя *Run the InterBase server as a service on Windows NT* указывает, что сервер InterBase запускается как служба Windows NT. Надпись *Root Directory* показывает главный каталог, в котором установлен сервер InterBase и который можно изменить, нажав кнопку *Change* и выбрав нужный каталог. Однако на практике изменять главный каталог не требуется. Если сервер InterBase запущен не как служба Windows, то в правом углу панели задач появляется значок  . Завершить работу сервера также можно, вызвав щелчком мыши на значке контекстное меню и задав команду *Shutdown.* Остальные команды контекстного меню служат для настройки сервера:

- InterBase Properties... – установка свойств InterBase;
- Properties – установка свойств.

Обычно этим параметрам уже заданы нужные значения, изменять которые нет необходимости.

При запуске сервера в качестве службы Windows NT управлять его параметрами, а также остановить сервер можно в окне *Services.*

4.2.4. Особенности приложения

Существуют основные принципы разработки и использования приложений, выполняющих операции с удаленными БД. Эти принципы являются общими для различных систем, например, таких, как InterBase, Microsoft SQL Server или Oracle. Поэтому примеры программ (приложений) для работы с БД InterBase, которые приводятся ниже, годятся и для других серверов. При разработке приложений использованы компоненты, стандартные для всех БД. Наряду с этим в Delphi имеется ряд компонентов, которые предназначены только для работы с сервером InterBase. Эти компоненты расположены на странице *InterBase* Палитры компонентов. Многие из них являются аналогами соответствующих компонентов страницы *BDE* и отличаются тем, что адаптированы специально под InterBase.

Особенность использования данных компонентов заключается в том, что для них доступ к БД осуществляется напрямую через процессор баз данных BDE. При этом нет необходимости в соответствующем драйвере SQL-Links. Компоненты страницы *InterBase* принципиально не отличаются от соответствующих компонентов других страниц, поэтому мы опускаем их подробное рассмотрение.

4.3. Соединение с базой данных

Для выполнения любых операций с БД с ней необходимо установить соединение, т. е. *открыть* БД. По завершению работы соединение нужно разорвать *(закрыть* БД). Для соединения с БД программы типа IBConsole имеют соответствующие средства, вызываемые через команды меню. При создании приложения разработчик должен организовывать соединение самостоятельно, для чего Delphi предоставляет соответствующие компоненты, в первую очередь компонент Database.

Рассмотрим, как установить соединение с удаленной БД с помощью программы IBConsole и компонентов Delphi.

4.3.1. Соединение с базой из программы IBConsole

Соединение с БД выполняется командой *Database\Connect*. После открытия БД доступна для работы, например, для изменений структуры путем добавления и удаления таблиц или для редактирования данных.

Для отключения от БД следует выполнить команду *Database\Disconnect*, при этом запрашивается подтверждение на выполнение этой операции. При утвердительном ответе соединение с БД разрывается. При необходимости также запрашивается подтверждение текущей незавершенной транзакции.

На программном уровне соединение с БД выполняет оператор *connect*, имеющий следующий формат:

```
CONNECT DATABASE "<Имя файла БД>"
USER "<Имя пользователя>" PASSWORD "<Пароль пользователя>"
```

Отключение от БД выполняет оператор *exit.*

4.3.2. Компонент Database

Компонент Database используется для соединения с БД. Если программист при разработке приложения не разместил на форме компонент Database, то для организации соединения с БД в процессе выполнения приложения будет создаваться динамический (временный) объект типа TDatabase. Динамическое создание этого объекта выгодно тем, что не требует от программиста каких-либо действий. Однако при использовании динамического объекта типа TDatabase он автоматически создается для каждого соединения с БД, что при одновременной работе с несколькими БД требует больших ресурсов Windows, чем в случае применения компонента Database.

Использование компонента Database позволяет:

- настраивать параметры соединения с БД;
- явно управлять транзакциями при работе с БД.

Компонент Database можно использовать для локальных и для удаленных БД. Например, действия, связанные с настройкой псевдонимов BDE, одинаковы для обоих типов БД.

Свойство *SessionName* типа *string* указывает компонент сеанса *session*, с которым связан компонент *Database*. Если значение этого свойства не задано (пустое значение или значение Default), то для сеанса создается динамический объект типа *TSession* (объект по умолчанию).

Свойство *AliasName* типа *string* указывает псевдоним БД. На этапе разработки приложения псевдоним выбирается из списка в Инспекторе объектов.

Расположение БД может быть определено также с помощью свойства *DriverName* типа *string*, задающего драйвер БД. Следует иметь в виду, что значения этого свойства и свойства *AliasName* являются взаимоисключающими: при установке одного из них второе автоматически сбрасывается. Если расположение БД задано значением свойства *DriverName*, то остальные параметры соединения должны быть заданы через свойство *Params*.

Свойство *DatabaseName* типа *string* задает имя БД, используемое в приложении для соединения с БД. Отметим, что это имя не совпадает ни с псевдонимом, ни с собственно именем БД (именем главного ее файла). Имя БД, заданное свойством *DatabaseName*, действует только в приложении и только для организации подключения к БД, указанной ее псевдонимом.

В последующем набор данных (обычно Query) связывается с БД через свойство *DatabaseName*, значение которого должно совпадать со значением одноименного свойства компонента *Database*, используемого для соединения. На этапе разработки приложения имя БД для набора данных выбирается в Инспекторе объектов из списка, содержащего его наряду с псевдонимами BDE.

Свойство *Params* типа *TStrings* определяет параметры соединения, для изменения которых при разработке приложения используется строковый редактор *String list editor* (рис. 4.3.1), вызываемый двойным щелчком в области значения Инспектора объектов.

В приведенном на рис. 4.3.1 примере указан имя сервера, имя пользователя и его пароль. Отметим, что имя сервера лучше задавать через псевдоним, что облегчает перенос БД на другое место.

При выполнении приложения доступ к отдельному параметру осуществляется по его индексу в массиве (списке) параметров Params. При этом необходимо учитывать, что в значение параметра также входит его название. Рассмотрим следующий пример:

```
Procedure TForm1.Button1Click(Sender: TObject);
Begin
Edit1.text:=Database1.Params [1];
End;
```

Здесь при нажатии на кнопку в поле редактирования (компонент Edit) выводится второй параметр соединения.

В частности, для параметров, приведенных на рис. 4.3.1, будет получена следующая строка:

USER NAME=SYSDBA

Кроме параметров, заданных в свойстве *Params* компонента *Database*, соединение с БД также определяется:

- установками системных параметров BDE;
- параметрами драйвера, используемого для доступа к БД;
- параметрами псевдонима БД.

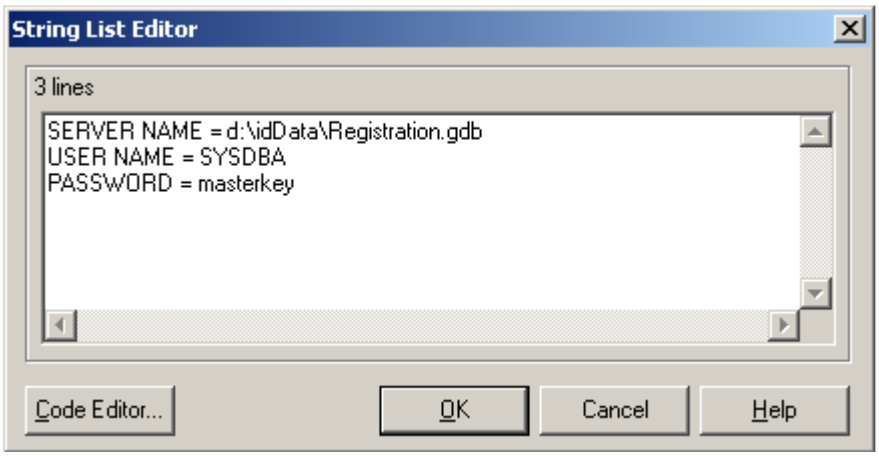


Рис. 4.3.1. Параметры соединения с БД.

Если в списке *Params* какой-либо параметр, например, *Langdriver*, не задан, он выбирается из перечисленных выше установок. Поэтому с помощью свойства *Params* нужно устанавливать (переустанавливать) только те параметры, значения которых не устраивают разработчика.

Свойство *LoginPrompt* типа *Boolean* управляет режимом отображения окна ввода имени пользователя и пароля *Database Login* (рис. 4.3.2). По умолчанию свойство имеет значение *True*, и окно появляется при первом соединении с БД. Если установить свойству LoginPrompt значение *False*, то запрос на идентификацию пользователя не выдается, и его имя и пароль должны быть указаны в параметрах соединения (свойство Params).

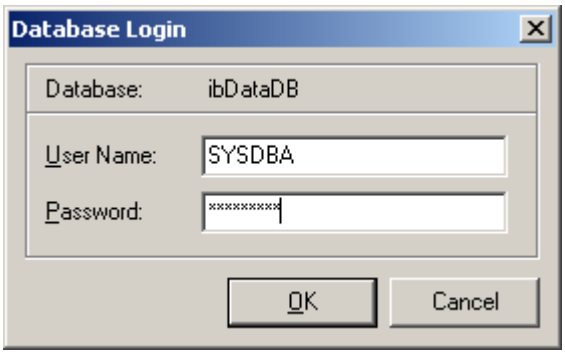


Рис. 4.3.2. Окно ввода имени пользователя и пароля.

Свойство *KeepConnection* типа *Boolean* определяет, сохранять ли соединение БД, если с ней не связан ни один набор данных. По умолчанию свойство имеет значение True, и однократно установленное соединение

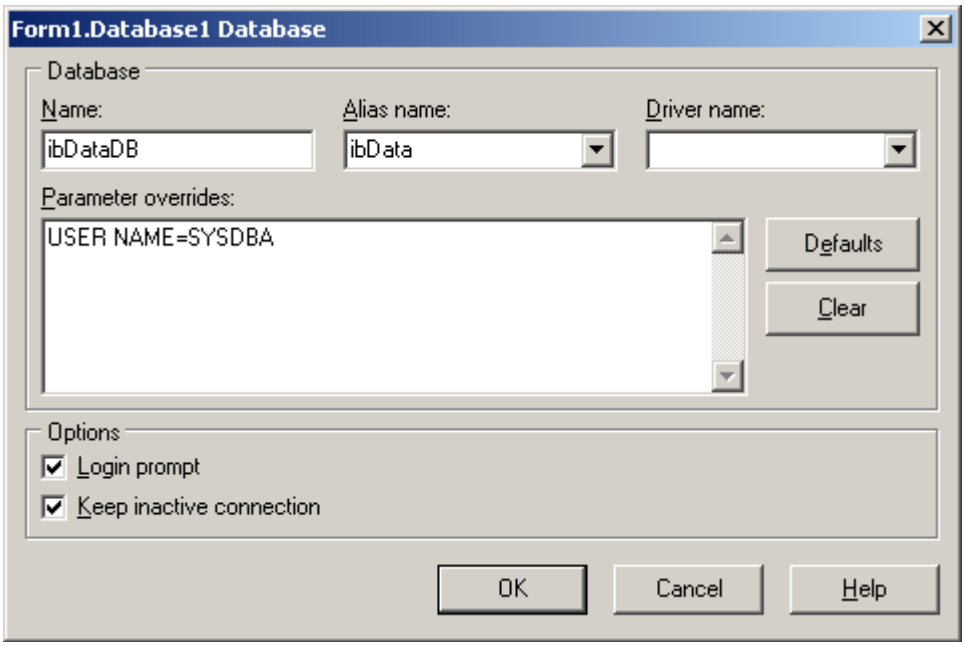
сохраняется в течение всей работы с БД. Использование постоянного соединения с БД экономит время, затрачиваемое на доступ к данным, но требует определенных системны затрат на свое поддержание. Постоянное соединение разрывается при окончании работы приложения или при программной установке значения *False* свойству *Connected*.

На этапе разработки управлять значениями названных свойств можно в окне Редактора параметров соединения, показанном на рис. 4.3.3. Оно вызываете двойным щелчком на компоненте *Database* или вызовом команды *Database editor...* контекстного меню компонента.

Управляющие элементы окна отображают и позволяют изменять значения соответствующих свойств компонента Database:

- поле редактирования **Name** – свойство DatabaseName;
- комбинированный список **Alias name** – свойство AliasName;
- комбинированный список **Driver name** – свойство DriverName;
- многострочное поле редактирования **Parameter overrides** (Новые значение параметров) – свойство Params;
- переключатель **Login prompt** (Входное сообщение) – свойство LoginPrompt;
- переключатель **Keep inactive connection** (Сохранять неактивное соединение) свойство KeepConnection.

Отметим, что на рис. 4.3.2 и 4.3.3 в качестве имени базы данных выводите именно значение свойства *DatabaseName*, а не имя (значение свойства Name компонента database).



4.3.3. Редактора параметров соединения с БД.

Свойство *Connected* типа *Boolean* определяет, установлено ли соединение с БД. Соединение с БД можно установить при разработке приложения, задав этому свойству через Инспектор объектов значение *True*.

При выполнении приложения можно также установить или разорвать соединение с БД, используя методы open и *Close*, соответственно.

Свойства *DataSetCount* и *DataSets* позволяют получить доступ к наборам данных, связанным в настоящий момент времени с БД. Свойство *DataSetCount* типа *Integer* содержит число таких наборов, а свойство *DataSets* [Index: Integer] типа *TDBDataSet* представляет собой коллекцию (массив) этих наборов данных. Доступ к отдельному набору данных можно получить по его номеру в массиве (нумерация начинается с нуля, т. е. последний набор данных имеет номер DataSetCount-l).

При выполнении операций с удаленными БД автоматически запускается механизм транзакций, в данном случае называемый *неявным*. Кроме того, программист может управлять транзакциями и явным образом, но при этом нужно учитывать, что явные транзакции несколько замедляют выполнение операций с БД. Для реализации механизма транзакций служит ряд свойств и методов компонента *Database*.

Методы *StartTransaction*, инициирующий начало транзакции, *Commit*, подтверждающий текущую транзакцию, и *Rollback*, отменяющий ее, были рассмотрены ранее, посвященной навигационному способу доступа к данным. Свойство *InTransaction* типа *Boolean* позволяет определить, существует ли активная транзакция для текущего соединения с БД. Если имеется незавершенная транзакция, то свойство имеет значение *True*, и значение *False* – в противном случае. Это свойство можно использовать, например, при подтверждении транзакции. Если активной транзакции нет, то вызов метода Commit приводит к генерации исключительной ситуации. Например:

```
if Database1.InTransaction then Database1.Commit;
```

Если имеется активная транзакция, то она подтверждается, а все сделанные в ее рамках изменения БД вступают в действие.

Свойство *TransIsolation* типа *TTransIsolation* управляет *уровнем изоляции* транзакций. Уровень изоляции определяет, каким образом транзакция взаимодействует с другими транзакциями. Ниже приводится список возможных значений свойства *TransIsolation*.

- tiDirtyRead – разрешается чтение изменений в записях БД, сделанных в рамках других транзакций (другими пользователями). В связи с тем, что на момент чтения эти изменения являются неподтвержденными и впоследствии могут быть отменены, этот уровень изоляции обеспечивает минимальнуюизоляцию (защиту) от других транзакций.
- tiReadCommitted – разрешается чтение только подтвержденных изменений в записях БД (по умолчанию). Если изменения еще не подтверждены, то читается предыдущая версия записи.
- tiRepeatableRead – считывание сразу всей БД, после чего изменения в БД, сделанные в рамках других транзакций, становятся невидимыми. Этот уровень обеспечивает максимальную изоляцию.

Для локальных БД dBase и Paradox допустимым является только значение

tiDirtyRead. Для сервера InterBase значение tiDirtyRead интерпретируется как tiReadCommitted.

4.3.3. Компонент Session

Компонент *Session* представляет собой текущий сеанс работы с БД и наряду с компонентом *Database* используется для управления соединением с БД. Однако если компонент *Database* позволяет задавать параметры одного соединения с БД или параметры соединения с одной БД, то компонент *session* управляет всеми соединениями со всеми БД. Если программист при разработке приложения не разместил на форме компонент *Session*, то будет создаваться динамический (временный) объект типа *TSession* (TSessionList).

Наряду с управлением соединениями с БД использование компонента *Session* также позволяет:

- управлять установками BDE;
- управлять паролями БД.

Свойство *SessionName* типа *string* задает имя сеанса. Это имя устанавливается программистом, и его назначение не совпадает с назначением свойства Name, которое определяет имя самого компонента *Session*. Однако в частном случае эти имена могут совпадать.

Свойства сеанса, заданного компонентом *Session*, действуют на все связанные с ним компоненты *Database*. Напомним, что компонент *Database* связывается с сеансом через свое свойство *SessionName*, значение которого при разработке приложения выбирается из списка Инспектора объектов.

С помощью свойств *DatabaseCount* и *Databases* можно получить доступ к активным БД, с которыми в настоящее время установлено соединение. Свойство *DatabaseCount* типа *Integer* содержит число таких БД, а свойство *Databases [Index: Integer]* типа *TDatabase* представляет собой коллекцию (массив) объектов *Database*, используемых для соединения с этими БД. Доступ к отдельной БД можно получить по ее номеру в массиве (нумерация начинается с нуля, поэтому последняя БД имеет номер DatabaseCount-l).

Свойство *KeepConnections* типа *Boolean* определяет, сохранять ли соединение с БД, если с ней не связан ни один набор данных. Это свойство аналогично свойству *Keepconnection* соединения с отдельной БД, заданного компонентом *Database*. Разорвать соединение с БД можно, вызвав метод *DropConnections*.

Свойства *NetFileDir* и *PrivateDir* типа string позволяют задать каталоги для хранения, управляющего сетевого файла с расширением NET (для таблиц Paradox) и рабочего каталога для хранения временных файлов сеанса, соответственно.

4.3.4. Компонент Query

Основным средством работы с серверами баз данных в Delphi является компонент Query. Основное его назначение – сформировать и выполнить SQL-запрос. Основные используемые свойства и методы:

property SQL:string – содержит текст SQL-запроса;

method Prepare – предназначен для первоначальной обработки SQL-запроса;

property Active:Boolean – показывает состояние компонента. При переводе из состояния *False* в состояние *True* выполняет те же действия, что и метод *Open*;

method Open – используется для выполнения операторов SQL (Structured Query Language) – языка структурированных запросов. Этот метод используется, когда запрос выдает результирующего набор строк. Пример – любая форма оператора SELECT;

method ExecSQL – используется для выполнения операторов DML (Data Manipulation Language) – языка манипулирования данными. Эти операторы влияют на состояние данных в базе, но не выдают никакого результирующего набора. Пример – оператор INSERT;

property Params[] – хранит параметры SQL-запроса. В тексте запроса имена параметров предваряются двоеточием;

method ParamByName – обеспечивает доступ к параметрам SQL-запроса по имени.

Компонент Query также используется при работе с хранимыми процедурами, в случае, когда последние возвращают набор, состоящий более чем из одной строки. В этом случае синтаксис оператора SELECT следующий:

SELECT ..... FROM <Имя_хранимой_процедуры>

4.3.5. Компонент StoredProc

Компонент *StoredProc* предназначен для организации работы с хранимыми процедурами. При этом хранимая процедура должна возвращать конечный набор результатов. Для работы с хранимыми процедурами, результатом работы которых является набор строк, при взаимодействии с сервером *Interbase* используется компонент *Query*.

Основные используемые свойства и методы:

property StoredProcName:string – содержит имя хранимой процедуры;

property Params[]– содержит параметры и результаты работы хранимой процедуры.

method ParamByName – обеспечивает доступ к параметрам хранимой процедуры по имени.

4.3.6. Компонент DataSource

Компонент *DataSource* используется для организации связи невизуальных компонентов доступа к базам данных, таких как *Table*, *Query*, *StoredProc* и визуальных компонентов управления базами данных, такими как *DBGrid*, *DBNavigator*, *DBText*, *DBMemo* и другими. Имя компонента DataSource используется в свойстве DataSource используемых визуальных компонентов.

Основные свойства и метода компонента:

property DataSet – имя компонента доступа к базе данных.

4.4. Средства для работы с сервером InterBase

Для работы с сервером InterBase и базами данных фирма Borland поставляет набор утилит: BDE Configuration Utility, Database Desktop, Database Explorer, Windows Interactive SQL, InterBase Communication Diagnostics Tool, InterBase Server Manager, Data Migration Expert.

4.4.1. BDE Configuration Utility

Эта утилита предназначена для управления и конфигурирования Borland Database Engine (BDE). С ее помощью можно:

- конфигурировать установленные драйверы баз данных;
- добавлять, удалять и настраивать драйверы ODBC для различных баз данных;
- создавать и удалять псевдонимы (aliases) баз данных;
- гибко настраивать свойства псевдонимом существующих баз данных;
- настраивать общесистемные параметры – языковой драйвер по умолчанию, размеры буферов и т.д.;
- настраивать представление даты в базах данных;
- настраивать представление времени;
- настраивать представление числовых величин.

4.4.2. Database Desktop

Database Desktop (DBD) – рабочий стол баз данных – средство для создания, просмотра, редактирования, удаления, реструктуризации баз данных. DBD было сконструировано для эффективной работы с различными типами баз данных и позволяет работать с таблицами, SQL-запросами и запросами «по образцу» (Query By Example – QBE).

DBD позволяет:

- создавать и удалять таблицы баз данных различного типа;
- изменять структуру таблиц;
- просматривать и редактировать таблицы;
- создавать и удалять индексы;
- управлять ссылочной целостностью таблиц;
- перемещать данные из одной таблицы в другую;
- управлять псевдонимами баз данных BDB;
- выполнять SQL-запросы;
- выполнять запросы «по образцу» – QBE.

В настоящее время вместо DBD используется Database Explorer, позволяющий выполнять почти все функции DBD.

4.4.3. Database Explorer

Database Explorer – мощное и удобное средство для управления базами данных, предназначенное для всевозможных операций над базами данных.

Database Explorer позволяет:

- управлять псевдонимами BDE;
- иерархически просматривать компоненты баз данных;
- изменять структуру баз данных и их компонентов, таких как таблицы, хранимые процедуры, представления, триггеры, генераторы, домены, исключения;
- просматривать и редактировать таблицы, хранимые процедуры, представления, триггеры, генераторы, домены, исключения;
- выполнять запросы к базам данных.
- В будущем фирма Borland предполагает перенести все функции Database Desktop на Database Explorer.

4.4.4. Windows Interactive SQL

Windows Interactive SQL (WISQL) – средство для интерактивного выполнения операций над базами данных. WISQL запускается на клиентской машине, но может подсоединяться как к локальному серверу InterBase, так и к удаленному, запущенному на любой машине в сети. WISQL позволяет:

- создавать и уничтожать базы данных;
- подсоединяться к любым базам данных в сети;
- выполнять запросы к базам данных;
- создавать модифицировать и удалять таблицы, домены, триггеры, представления, генераторы, хранимые процедуры, индексы, исключения и т.д.;
- вносить изменения в таблицы;
- получать полную информацию о базе данных, таблицах и представлениях;
- получать определение таблиц, триггеров, хранимых процедур, представлений, исключений, доменов, индексов, генераторов и т.д.;
- запускать SQL-сценарии;
- фиксировать изменения базы данных или совершать откат.

4.4.5. InterBase Communication Diagnostics Tool

InterBase Communication Diagnostics Tool (IBCD) позволяет протестировать установку связи и проанализировать существующие и потенциальные проблемы при работе с сетью. IBCD включает в себя следующие тесты:

- DB Connection — позволяет тестировать соединение с сервером Interbase, используя библиотеки клиента Interbase. Возможно проверить работу как с локальным, так и с удаленным сервером Interbase, работающим по протоколам NetBEUI, TCP/IP и IPX/SPX;
- NetBEUI — позволяет протестировать NetBEUI-соединение, не используя библиотек клиента Interbase;

– WinSock — позволяет протестировать WinSock TCP/IP соединение, не используя библиотек клиента Interbase.

4.4.6. InterBase Server Manager

InterBase Server Manager – приложение, предназначенное для администрирования локальных и удаленных баз данных и серверов InterBase. Оно запускается на машине-клиенте, но позволяет управлять базами данных на клиенте или любым сервером в сети.

Server Manager позволяет:

– управлять безопасностью баз данных (database security) – заводить новых пользователей, менять пользовательские пароли и управлять пользовательскими правами;
– выполнять резервное копирование и восстановление баз данных;
– осуществлять поддержку баз данных: проверку целостности базы данных; восстановление испорченных баз данных и очистку баз данных;
– осуществлять перезапуск баз данных.

4.4.7. Data Migration Expert

Утилита Data Migration Expert предназначена для переноса структуры и содержимого существующих локальных баз данных, реализованных с помощью СУБД Paradox или dBase, на современные серверы баз данных, типа Oracle, Informix, Interbase, Sybase или MS SQL Server.

4.5. Конфигурирование BDE для доступа к базам данных InterBase.

Для организации доступа к базе данных в BDE создается псевдоним (alias) базы данных. Тип используемого драйвера – INTERBASE. Поле SERVER NAME содержит полный адрес базы данных. Формат этого поля зависит от используемого сетевого протокола:

а) для протокола TCP/IP:
servername:/fullpath/databasename.gdb, где
servername – IP-адрес или полное имя сервера;
fullpath – полный путь к файлу базы данных;
databasename – имя файла базы данных.

б) для протокола NetBEUI:
\\servername\disk:\fullpath\databasename.gdb, где
servername – полное имя сервера в сети Microsoft;
disk – имя логического диска сервера;
fullpath – полный путь к файлу базы данных;
databasename – имя файла базы данных.

в) для протоколов IPX/SPX:
@servername/disk:fullpath\databasename.gdb, где
servername – полное имя сервера Novell;
disk – имя логического диска сервера;

fullpath – полный путь к файлу базы данных;
databasename – имя файла базы данных.

Для работы с базами данных, содержащими информацию на русском языке, необходимо указать используемый языковой драйвер в поле LANGUAGE DRIVER. При использовании альтернативной кодовой таблицы – это драйвер “Paradox Cyrr 866”, а при использовании кодировки Windows – “Pdox ANSI Cyrillic”.

4.6. Создание базы данных и ее компонентов

4.6.1. Создание базы данных

Новая база данных создается с помощью оператора DDL CREATE DATABASE, в котором в качестве параметров указываются путь к файлам базы данных, владелец базы и различные необязательные параметры. Для создания базы данных удобно использовать WISQL.

4.6.2. Создание и использование доменов

Домены – метаданные, позволяющие систематизировать и структурировать базы данных. Фактически домен – это новый тип данных, вводимый пользователем для боле гибкого управления структурой таблиц. Если в базе данных несколько таблиц используют одинаковое поле, например *Name: string(32)*, то вместо включения в определения всех таблиц этого поля описывается домен, имя которого используется в дальнейшем как имя типа столбца таблицы. Домен создается оператором CREATE DOMAIN.

4.6.3. Создание таблиц

Новая таблица создается с помощью оператора CREATE TABLE. Помимо перечисления полей и их типов, в описании таблицы могут присутствовать:

- определение первичных ключей;
- определение уникальности содержимого полей;
- определение внешних ключей;
- ограничения на значения полей;
- ограничения на таблицу.

Изменение структуры таблицы осуществляется оператором ALTER TABLE.

4.6.4. Создание триггеров

Триггеры – программы обработчики определенных событий, хранящиеся на сервере и позволяющие пользователю указать поведение сервера баз данных в определенных ситуациях. Триггеры могут быть одного из следующих типов:

- BEFORE DELETE – триггер запускается перед совершением операции удаления;

- BEFORE INSERT – триггер запускается перед совершением операции вставки;
- BEFORE UPDATE – триггер запускается перед совершением операции изменения;
- AFTER DELETE – триггер запускается после совершения операции удаления;
- AFTER INSERT – триггер запускается после совершения операции вставки;
- AFTER UPDATE – триггер запускается после совершения операции изменения.

Триггеры создаются с помощью оператора CREATE TRIGGER, в котором указывается таблица, к которой он привязан, условие запуска триггера и его порядковый номер запуска, в случае если создано несколько триггеров, запускающихся по одному условию.

4.6.5. Создание хранимых процедур

Хранимые процедуры – части программы, находящиеся и выполняющиеся на сервере. Они хранятся вместе с базой данных на сервере в откомпилированном виде и позволяют перенести часто повторяющиеся длительные операции с клиентской машины на серверную.

Хранимые процедуры создаются с помощью оператора CREATE PROCEDURE, в котором, если это необходимо, указываются параметры процедуры и возвращаемое ею значение. Тело процедуры заключено в операторные скобки BEGIN - END.

4.6.6. Создание представлений

Представления (view) являются еще одним способом перенесения определенной части работы с базами данных на сервер. С точки зрения пользователя представление - таблица, в то время как на сервере она хранится в виде процедуры (запроса), выдающей на выходе результирующий набор данных. Представления используются для форматирования содержимого таблиц, объединения данных нескольких таблиц и ограничения доступа пользователей к отдельным полям таблицы.

Представления создаются с помощью оператора CREATE VIEW, в котором указывается текст SQL-запроса (оператор SELECT), который и формирует результирующий набор.

Из программы на Delphi доступ к представлениям осуществляется как к обычным таблицам.

4.7. Уничтожение компонентов базы данных

Компоненты баз данных, такие как таблицы, представления, триггеры, хранимые процедуры, исключения, уничтожаются с помощью оператора DROP.

4.8. Организация ссылочной целостности базы данных

Для организации ссылочной целостности базы данных имеется средство организации внешних ключей (FOREIGN KEY). Предположим, имеется две таблицы, причем первая содержит поле (Field1), значение которого расшифровывается во второй таблице (вторая таблица — справочник). Ссылочная целостность организуется путем добавления в первую таблицу внешнего ключа для Field1, ссылающегося на родительский ключ из второй таблицы. После подобной процедуры таблицы становятся связанными. Добавление внешнего ключа осуществляется в операторе CREATE TABLE или ALTER TABLE предложением:

FOREIGN KEY (имя_поля) REFERENCES имя_связ_таблицы (имя_родит_ключа).

4.9. Массовое удаление и модификация данных

Массовые удаления и модификации - такие, когда изменение информации в одной таблицы влияет на содержимое других. Существует два основных способа организации сервера баз данных: каскадный (cascade) и строгий (restricted). Каскадное удаление (модификация) подразумевает удаление (модификацию) информации во всех связанных таблицах. Строгий способ предполагает запрет автоматического изменения информации в связанных таблицах. В InterBase реализован строгий способ. Для организации каскадного удаления в InterBase необходимо использовать триггеры.

5. БАЗЫ ДАННЫХ В СРЕДЕ MS ACCESS

5.1. Создание базы данных в MS Access

1. Создайте новую базу данных.
2. Создайте таблицу базы данных.
3. Определите поля таблицы как в примере (таблице 5.1.1)
4. Сохраните созданную таблицу.

Таблица 5.1.1. Примерная таблица *«Преподаватели»*

Имя поля	**Тип данных**	**Размер поля**
Код преподавателя	Счетчик	
Фамилия	Текстовый	15
Имя	Текстовый	15
Отчество	Текстовый	15
Дата рождения	Дата/время	Краткий формат поля
Должность	Текстовый	9
Дисциплина	Текстовый	11
Телефон	Текстовый	9
Зарплата	Денежный	

5.1.1. Технология создание баз данных

1. Для создания новой базы данных:

– загрузите Access, в появившемся окне выберите пункт *Новая база данных*;

– в окне "Файл новой базы данных" задайте имя вашей базы (пункт *Имя Файла***)** и выберите папку (пункт *Папка*), где ваша база данных будет находиться. По умолчанию Access предлагает вам имя базы db1, а тип файла - *Базы данные Access.* Задайте имя базу данных, а тип файла оставьте прежним, так как другие типы файлов нужны в специальных случаях;

– щелкните по кнопке «Создать».

2. Для создания таблицы базы данных:

– в окне базы данных выберите вкладку Таблицы, а затем щелкните по кнопке «Создать»;

– в окне "Новая таблица" выберите пункт Конструктор и щелкните по кнопке «ОК». В результате проделанных операций открывается окно таблицы в режим конструктора как в рисунке 5.1.1, в котором следует определить поля таблицы.

3. Для определения полей таблицы:

– введите в строку столбца Имя поля имя первого поля своей таблицы по варианту;

– в строке столбца "Тип данных" щелкните по кнопке списка и выберите соответствующий тип данных. Поля вкладки Общие установите параметры поля, как предлагает Access.

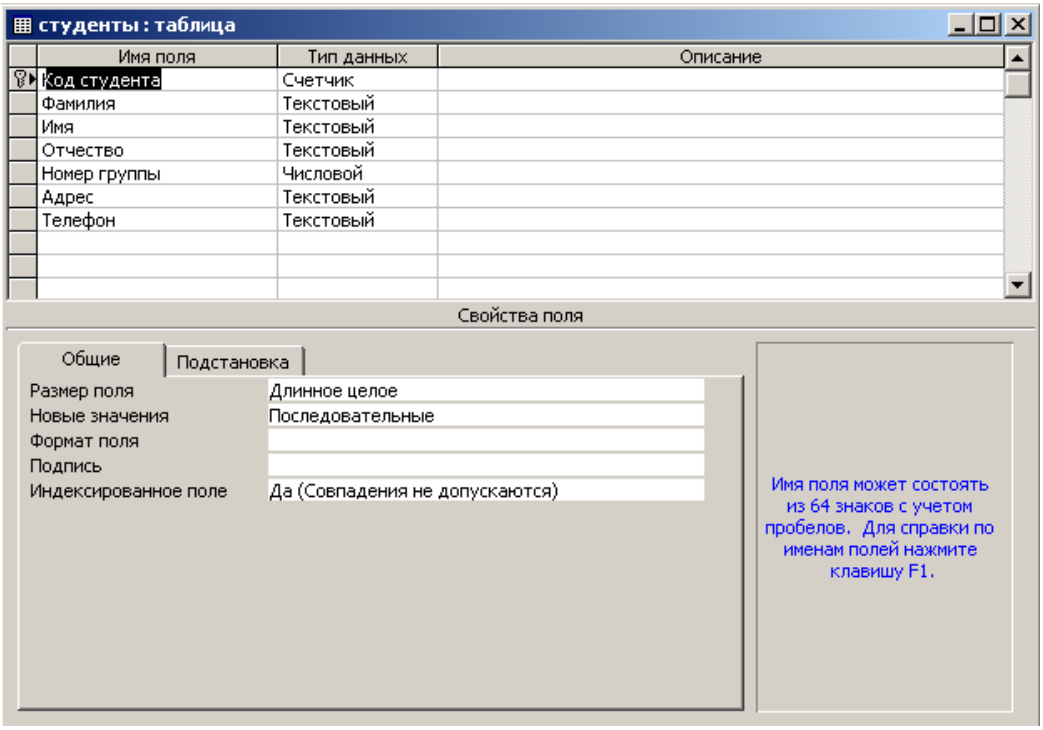


Рис. 5.1.1. Окно таблицы в режиме конструктора - в этом режиме вводятся имена и типы полей таблицы

Примечание. Заполнение строки столбца "Описание" необязательно и обычно используется для внесения дополнительных сведений о поле.

Для определения всех остальных полей таблицы базы данных выполните действия, аналогичные указанным выше.

Обратите внимание на вкладку *Общие* в нижней части экрана. Советуем изменить данные в пункте *Размер поля*, а остальные пункты оставить по умолчанию. Например, для текстового типа данных Access предлагает по умолчанию длину 50 символов. Но вряд ли поле "Фамилия" будет содержать более 15 символов, хотя лучше точно подсчитать, сколько символов в самой длинной фамилии. Не бойтесь ошибиться - в дальнейшем можно скорректировать длину поля. Для числового типа Access предлагает *Длинное целое*, но ваши данные могут быть либо небольшие целые числа (в диапазоне от -32768 до 32767) - тогда надо выбрать *Целое*, либо дробные числа - тогда надо выбрать *С плавающей точкой*.. Для выбора необходимого параметра надо щелкнуть по полю, а затем нажать появившуюся кнопку списка и выбрать необходимые данные. В результате ваша таблица будет иметь более компактный вид, а объем базы данных уменьшится.

4. Для сохранения таблицы:

– выберите пункт меню *Файл/Сохранить*;

– в диалоговом окне "Сохранение" введите имя таблицы *Преподаватели*;

– щелкните по кнопке «ОК».

Примечание. В результате щелчка по кнопке «ОК» Access предложит вам задать ключевое поле (поле первичного ключа), т.е. поле, однозначно идентифицирующее каждую запись. Для однотабличной базы данных это не столь актуально, как для многотабличной.

5.2. Заполнение базы данных

1. Введите ограничения на данные;
2. Задайте текст сообщения об ошибке, который будет появляться на экране при вводе неправильных данных;
3. Задайте значение по умолчанию;
4. Введите ограничения на данные;
5. Заполните таблицу данными согласно своему варианту и проверьте реакцию системы на ввод неправильных данных;
6. Измените ширину каждого поля таблицы в соответствии с шириной данных;
7. Произведите поиск в таблице;
8. Произведите замену данных;
9. Произведите сортировку данных по необходимым полям;
10. Просмотр созданной таблицы.

Просмотрите созданную таблицу, как она будет выглядеть на листе бумаги при печати.

Таблица 5.2.1. Примерная таблица

Код	**Фамилия**	**Имя**	**Должность**	**Дисциплина**	**Телефон**
1	Истомин	Ремир	Доцент	Физика	110-44-68
2	Миронов	Павел	Профессор	Экономика	312-21-40
3	Гришин	Евгений	Доцент	Математика	260-23-65
4	Сергеева	Ольга	Ассистент	Математика	234-85-69
5	Емец	Татьяна	Доцент	Экономика	166-75-33
6	Игнатьева	Татьяна	Доцент	Информатика	210-36-98
7	Миронов	Алексей	Доцент	Физика	166-75-33

5.2.1. Технология заполнения базы данных

1. Для задания условия на значение для вводимых данных:

– войдите в режим Конструктор для проектируемой таблицы. Если вы находитесь в окне базы данных, то выберите вкладку Таблицы и щелкните по кнопке «Конструктор»  . Если вы находитесь в режиме таблицы, то щелкните по кнопке на панели инструментов или выполните команду *Вид*, *Конструктор*;

– в верхней части окна щелкните полю, которую должны вставить условия ввода данных (например: поля "Должность");

– в нижней части окна щелкните по строке параметра *Условие на значение*;

– в появившемся окне напишите слова для ограничения (например: для поля Должность введем слово *Профессор,* затем щелкнем по кнопке 

(эта кнопка выполняет функцию ИЛИ), напишем *″Доцент″,* снова щелкнем по этой же кнопке, напишем *″Ассистент″* и щелкнем по кнопке «ОК»).

Таким образом, мы ввели условие, при котором в поле "Должность" могут вводиться только указанные значения.

2. В строке *Сообщение об ошибке* введите предложение для предупреждения при возникновении ошибок.

3. В строке *Значение по умолчанию* введите слово по умолчания.

4. Введите ограничения на данные, например в поле "Код преподавателя". Здесь ограничения надо вводить не совсем обычным способом. Дело в том, что коды не должны повторяться, а также должна быть обеспечена возможность их изменения (из-за последнего условия в этом поле нельзя использовать тип данных *Счетчик,* в котором данные не повторяются). Для выполнения второго условия пришлось задать в поле тип данных *Числовой,* а для выполнения первого условия сделайте следующее:

– щелкните по строке параметра Индексированное поле;

– выберите в списке пункт *Да* (совпадения не допускаются);

перейдите в режим *Таблица*, щелкнув по кнопке  на панели инструментов или выполнив команду *Вид, Режим таблицы*. На вопрос о сохранении таблицы щелкните по кнопке «Да».

5. Введите данные в таблицу в соответствии своему варианту как в таблице 2. Попробуйте в поле ввести любой записи, которую вставили условия. Посмотрите, что получилось. На экране должно появиться сообщение об ошибке. Введите правильное слово.

6. Для изменения ширины каждого поля таблицы в соответствии с шириной данных:

– щелкните в любой строке поля таблицы;

– выполните команду *Формат*, *Ширина столбца*;

– в появившемся окне щелкните по кнопке «По ширине данных». Ширина поля изменится;

– проделайте эту операцию с остальными полями.

7. Для поиска в таблице:

– переведите курсор в первую строку таблицы;

– выполните команду *Правка*, *Найти*;

– в появившейся строке параметра *Образец* введите искомое слово;

– в строке параметра *Просмотр* должно быть слово *ВСЕ* (имеется в виду искать по всем записям);

– в строке параметра *Совпадение* выберите из списка *С любой частью поля;*

– в строке параметра *Только в текущем поле* установите флажок (должна стоять галочка);

– щелкните по кнопке «Найти». Курсор перейдет на соответствующую запись и выделит найденное слово;

– щелкните по кнопке «Найти далее». Курсор перейдет на следующую запись и также выделит найденное слово;

– щелкните по кнопке «Закрыть» для выхода из режима поиска.

8. Для замены данных:

– переведите курсор в первую строку поля, которое производится замена данных;

– выполните команду *Правка*, *Заменить*;

– в появившемся окне в строке *Образец* введите поисковое слово;

– в строке *Заменить на* введите заменяемое слово;

– щелкните по кнопке «Найти далее». Курсор перейдет на следующую найденную запись - это то, что нам надо;

– щелкните по кнопке «Заменить». Данные будут изменены.

9. Для сортировки данных:

– щелкните по любой записи индексируемого поля;

– выполните команду *Записи*, *Сортировка*, *Сортировка по убыванию*. Все данные в таблице будут отсортированы по убыванию.

10. Для просмотра созданной таблицы:

– выполните команду *Файл*, *Предварительный просмотр*. Вы увидите таблицу как бы на листе бумаги;

– закройте окно просмотра.

5.3. Ввод и просмотр данных посредством формы

1. С помощью Мастера форм создайте форму;
2. Найдите запись, находясь в режиме формы;
3. Измените данные, находясь в режиме формы;
4. Произведите сортировку данных по убыванию;
5. Произведите фильтрацию данных;
6. Измените название поля;
7. Просмотрите форму с точки зрения того, как она будет выглядеть на листе бумаги.

5.3.1. Технология ввода и просмотра данных

1. Для создания формы:

– откройте вкладку *Формы* в окне базы данных;

– щелкните по кнопке «Создать»;

– в появившемся окне выберите (подведите курсор мыши и щелкните левой кнопкой) пункт *Мастер форм;*

– щелкните по значку списка в нижней части окна;

– выберите из появившегося списка соответствующую таблицу;

– щелкните по кнопке «ОК»;

– в появившемся окне выберите поля, которые будут присутствовать в форме. В данном примере присутствовать будут все поля, поэтому щелкните по кнопке  ;

– щелкните по кнопке «Далее»;

– в появившемся окне уже выбран вид *Форма в один столбец*, поэтому щелкните по кнопке «Далее»;

– в появившемся окне выберите стиль оформления. Для этого щелкните по словам, обозначающим стили, либо перемещайте выделение стрелками вверх или вниз на клавиатуре. После выбора стиля щелкните по кнопке «Далее»;

– в появившемся окне задайте имя формы. Остальные параметры в окне оставьте без изменений;

– щелкните по кнопке «Готово». Перед вами откроется форма в один столбец. Столбец слева - это названия полей, столбец справа - данные первой записи (в нижней части окна в строке параметра *Запись* стоит цифра "1"). Для перемещения по записям надо щелкнуть по кнопке  (в сторону записей с большими номерами) или  (в сторону записей с меньшими номерами).

2. Для поиска данных:

– переведите курсор в первую поле;

– выполните команду *Правка*, *Найти*;

– в появившемся окне в строке *Образец* введите искомое слово;

– в строке параметра *Просмотр* должно быть слово *ВСЕ* (имеется в виду искать по всем записям);

– в строке параметра *Совпадение* выберите из списка параметр *С любой частью поля;*

– в строке параметра *Только в текущем поле* установите флажок (должна стоять «галочка»);

– щелкните по кнопке «Найти». Курсор перейдет на найденную запись;

– щелкните по кнопке «Найти далее». Курсор перейдет на следующую найденную запись;

– щелкните по кнопке «Закрыть» для выхода из режима поиска;

3. Для замены данных:

– переведите курсор в первую строку поля, которое производиться замена;

– выполните команду *Правка*, *Заменить;*

– в появившемся окне в строке параметра *Образец* введите искомое слово;

– в строке параметра *Заменить на* введите заменимое слово. Обратите внимание на остальные опции - вам надо вести поиск по всем записям данного поля;

– щелкните по кнопке «Найти далее». Курсор перейдет на следующую найденную запись;

– щелкните по кнопке «Заменить». Данные будут изменены;

– щелкните по кнопке «Закрыть».

4. Сортировка данных по убыванию:

– щелкните по любой записи поля "Дата рождения";

– выполните команду *Записи, Сортировка, Сортировка по убыванию.*

Все данные в таблице будут отсортированы в соответствии с убыванием значений в выбранной поле.

5. Фильтрация данных:

– щелкните по записи для установления фильтра;

– выполните команду *Записи*, *Фильтр*, *Фильтр по выделенному*. В форме останутся только соответствующие записи;

– для отмены фильтра щелкните выполните команду *Записи*, *Удалить фильтр*. В форме появятся все данные;

6. Изменения название поля:

– перейдите в режим конструктора, выполнив команду *Вид, Конструктор;*

– щелкните правой кнопкой мыши в поле (на названии поля - оно слева, а строка справа - это ячейка для данных, свойства которых мы не будем менять). В появившемся меню выберите пункт *Свойства***.** На экране откроется окно свойств для названия поля;

– щелкните по строке с именем *Подпись*

– сотрите существующее слово и введите новое слово;

– для просмотра результата перейдите в режим формы, выполнив команду *Вид, Режим формы;*

– окончательно отредактируйте форму в режиме конструктора: увеличьте длину тех полей, где текст полностью не помещается, и уменьшите слишком большие.

7. Для просмотра созданной формы:

– щелкните по кнопке  или выполните команду *Файл, Предварительный просмотр*. Вы увидите форму как бы на листе бумаги;

– закройте окно просмотра.

5.4. Формирование запросов и отчетов для однотабличной базы данных

1. На основе таблицы созданный по варианту создайте простой запрос на выборку, в котором должны отображаться некоторые поля таблицы.

2. Отсортируйте данные запроса.

3. Сохраните запрос.

4. Создайте запрос на выборку с параметром.

5.4.1. Технология формирования запросов и отчетов

1. Для создания простого запроса:

– в окне базы данных откройте вкладку *Запросы;*

– в открывшемся окне щелкните по кнопке «Создать»;

– из появившихся пунктов окна "Новый запрос" выберите *Простой запрос* и щелкните по кнопке «ОК»;

– в появившемся окне в строке *Таблицы/запросы* выберите таблицу (если других таблиц или запросов не было создано, она будет одна в открывающемся списке);

– в окне "Доступные поля" переведите выделение на параметр,

– с помощью кнопки  переведите необходимых полей в окно "Выбранные поля" (порядок важен - в таком порядке данные и будут выводиться);

– щелкните по кнопке. «Далее»;

– в строке параметра *Задайте имя запроса и* введите новое имя запроса;

– щелкните по кнопке «Готово». На экране появится таблица с результатами запроса.

2. Для сортировки данных:

– щелкните в любой строке поля;

– отсортируйте данные по убыванию. Для этого щелкните по кнопке  на панели инструментов или выполните команду *Записи, Сортировка, Сортировка по убыванию.*

3. Для сохранения запроса:

– щелкните по кнопке  или выполните команду *Файл, Сохранить;*

– закройте окно запроса.

4. Для создания запроса на выборку с параметром:

– создайте запрос на выборку. Запрос создавайте аналогично тому, как это делалось в п.1;

– задайте имя запроса;

– щелкните по кнопке «Готово». На экране появится таблица с результатами запроса;

– перейдите в режиме конструктора, щелкнув по кнопке или выполнив команду *Вид, Конструктор;*

– в строке параметра *Условия отбора* для необходимого поля введите фразу (скобки тоже вводить): [*Введите*]

– выполните запрос, щелкнув по кнопке  на панели инструментов или выполнив команду *Запрос, Запуск;*

– в появившемся окне введите слова для выборки и щелкните по кнопке «ОК». На экране появится таблица с записями соответствующая с введенным словом;

– сохраните запрос;

– закройте окно запроса.

5.5. На основе таблицы создайте отчет с группированием данных.

5.5.1. Технология работы создания отчета

– откройте вкладку *Отчеты* и щелкните по кнопке «Создать»;
– в открывшемся окне выберите пункт Мастер отчетов;
– щелкните по значку раскрывающегося списка в нижней части окна;
– выберите из появившегося списка необходимую таблицу;
– щелкните по кнопке «ОК», В появившемся окне выберите поля, которые будут присутствовать в отчете.
– щелкните по кнопке « Далее»;
– в появившемся окне присутствует перечень полей. Переведите выделение на необходимое поле;
– щелкните по кнопке  . Таким образом вы задаете группировку данных;
– щелкните по кнопке « Далее»;
– параметры появившегося окна (сортировка) оставим без изменений, поэтому щелкните по кнопке « Далее»;
– выберите макет – ступенчатый, « Далее»;
– в появившемся окне выберите стиль оформления отчета;
– щелкните по кнопке. «Далее»;
– в появившемся окне введите название отчета;
– щелкните по кнопке «Готово». На экране появится сформированный отчет;
– просмотрите, а затем закройте отчет.

ЗАКЛЮЧЕНИЕ

Курс представляет собой теоретическое и практическое введение в информационные системы. Подробное изучение работы основных локальных и удаленных базы данных осуществляется с использованием программных средств создания и управления системами базами данных и утилитами СУБД. Полученные теоретические сведения закрепляются путем применения их в процедурах разработки и создание информационные системы с помощью системами управления базами данных, а также в процедурах выполнения лабораторных работ.

В ходе изучения данного курса студенты приобретут основные навыки необходимые для работы с базами данных различного типа, а так же получат базовые знания по информационным системам.

Для выполнения данных лабораторных работ необходимо знакомство с принципами работы системами управления базами данных, а также минимальные практические навыки работы в среде Windows.

В результате выполнение лабораторных работ студенты:

- изучает базовые принципы построения БД. Основы создания БД с использованием программ Database Desktop и SQL Explorer. Усваивает главные понятия и определения.
- изучает базовые компоненты доступа к базам данных, имеющиеся в Delphi, научится создавать приложения, работающие с независимыми и связанными таблицами, использующими индексацию и различные средства фильтрации.
- изучает основы языка SQL и его использование в приложениях, а также компоненты доступа к базам данных.
- Осваивает настройки псевдонимов БД для доступа к Paradox, Dbase, Access и СУБД Interbase
- изучает программные средства, поставляемые вместе с Delphi и Interbase;
- изучает создать приложение, работающее с локальными БД и Interbase;
- Получить навыки работы с триггерами и хранимыми процедурами.

ЛАБОРАТОРНЫЕ РАБОТЫ

Каждый студент должен выполнять лабораторные работы согласно своего варианта, приведенного в вариантах заданий лабораторных работ (стр.110).

ЛАБОРАТОРНАЯ РАБОТА №1

ПРОЕКТИРОВАНИЕ И СОЗДАНИЕ БАЗЫ ДАННЫХ В СРЕДЕ MS ACCESS

Цель работы: Изучение методов создания баз данных в среде MS Access в режимах конструктора и мастера.

Для выполнения лабораторной работы студентам ставятся следующие задачи:

- Изучение основы построения баз данных в среде MS Access;
- Изучение создать многопользовательские базы данных MS Access;
- Изучение структуры таблицы: описание полей, ключ, индексы, ограничения на значения полей, пароли.
- Изучение создать таблицы в режиме конструктора
- Изучение создать таблицы с помощью мастера
- Изучение создать таблицы путем ввода данных
- Изучение создать запросов в режиме конструктора
- Изучение создать запросов с помощью мастера

Задания к лабораторной работе:

1. Создать базы данных в Access;
2. Создать таблицу базы данных;
3. Определить поля таблицы;
4. Определить ключи и индексы;
5. Установите ссылочную целостность между таблицами;
6. Сохранить созданную таблицу;
7. Создать запросы для баз данных;
8. Просмотр созданной таблицы.

ЛАБОРАТОРНАЯ РАБОТА №2

РАЗРАБОТКА ИНТЕРФЕЙСА ПРОГРАММЫ ДЛЯ РАБОТЫ С БАЗОЙ ДАННЫХ MS ACCESS

Цель работы: Изучение методов создания форм и отчетов с помощью конструктора и мастера для базы данных MS Access.

Для выполнения лабораторной работы студентам ставятся следующие задачи:

- Изучение создать мастер формы в MS Access;
- Изучение создать подчиненных форм в MS Access;
- Изучение создать формы в режиме конструктора;
- Изучение создать формы с помощью мастера;
- Изучение создать отчеты в режиме конструктора;
- Изучение создать отчеты с помощью мастера.
- Изучение создать макросы с помощью конструктора.

Задания к лабораторной работе:

1. Создать формы для работы с таблицами MS Access
2. Создать формы ввода данных в таблицу
3. Создать формы навигации данных таблицы
4. Создать формы редактирования данных
5. Создать формы отчетности (простой)
6. Создать формы отчетности (мастер/подчиненный)
7. Создать макросы для управления формами
8. Создать макросы для управления отчетами

ЛАБОРАТОРНАЯ РАБОТА №3

ПРОЕКТИРОВАНИЕ И СОЗДАНИЕ БАЗЫ ДАННЫХ В СРЕДАХ PARADOX, DBASE.

Цель работы: Изучение методов создания локальных баз данных в средах Paradox и Dbase.

Для выполнения лабораторной работы студентам ставятся следующие задачи:

- Изучение основы построения локальных баз данных;
- Изучение локальной архитектура базы данных;
- Изучение организации локальные баз данных;
- Изучение создать локальные БД Paradox, Dbase;
- Изучение ключи и индексы, связи между таблицами;
- Изучение структуры таблицы: описание полей, ключ, индексы, ограничения на значения полей, ограничения ссылочной целостности между таблицами, пароли.

Задания к лабораторной работе:

1. Создать базы данных (Paradox или dBase);
2. Создать таблицы, в которой будет храниться данные;
3. Определить структуру таблицы и различные типы полей;
4. Определить первичные и вторичные ключи и индексы;
5. Установите ссылочную целостность между таблицами;
6. Задать ограничений на значения полей;
7. Задать паролей и языкового драйвера (при необходимости);
8. Присваивать имен-алиасов для базы данных.

ЛАБОРАТОРНАЯ РАБОТА №4

РАЗРАБОТКА ИНТЕРФЕЙСА ПРОГРАММЫ ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ PARADOX, DBASE.

Цель работы: Изучение методов создания форм и отчетов с помощью конструктора и мастера в среде Delphi для баз данных Paradox и dBase.

Для выполнения лабораторной работы студентам ставятся следующие задачи:

- Изучение основных компонентов Delphi для работы с локальными базами данных: Table, Query, DataSource, DBEdit, DBGrid, DBNavigator, DBMemo, DBImage;
- Изучение использование Database Form Wizard и модули данных;
- Изучение средства для работы с базами данных Paradox и dBase: Database Desktop, BDE Configuration Utility, SQL Explorer, SQL Builder, BDE administrator;
- Изучение конфигурирования BDE для доступа локальной базы данных.

Задания к лабораторной работе:

1. Создать форму для работы с базы данных (Paradox и dBase);
2. Установить соединение с файлом базы данных с помощью алиаса или с указанием путь к файлу;
3. Создать форму ввода данных в базу данных;
4. Создать форму навигации данных хранящихся в базе данных;
5. Создать форму редактирования данных хранящихся в базе данных;
6. Создать форму подготовки отчетов по данным хранящихся в базе данных.

ЛАБОРАТОРНАЯ РАБОТА №5

ПРОЕКТИРОВАНИЕ И СОЗДАНИЕ БАЗЫ ДАННЫХ В СРЕДАХ INTERBASE, MS SQL.

Цель работы: Изучение основных методов проектирования и создание распределенных баз данных в средах InterBase и MSSQL.

Для выполнения лабораторной работы студентам ставятся следующие задачи:

- Изучение основы построения удаленных баз данных;
- Изучение принципы организации удаленных баз данных;
- Изучение "клиент-сервер" архитектуры приложений;
- Изучение создать удаленные базы данных;
- Изучение ключи и индексы, связи между таблицами;
- Изучение средства работы с удаленными базами данных: Database Desktop, InterBase Server Manager, Data Migration Expert, MS SQL Server Manager;
- Изучение организации данных: таблицы, индексы, ограничения, домены, просмотры, генераторы, триггеры, функции пользователя, хранимые процедуры, исключения, BLOB-фильтры, привилегии.

Задания к лабораторной работе:

1. Создание базы данных и ее компонентов (InterBase или MSSQL);
2. Создание и использование доменов;
3. Создание таблиц базы данных;
4. Создание триггеров базы данных;
5. Создание хранимых процедур базы данных;
6. Создание представлений базы данных;
7. Организация ссылочной целостности базы данных;

ЛАБОРАТОРНАЯ РАБОТА №6

РАЗРАБОТКА ИНТЕРФЕЙСА ПРОГРАММЫ ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ INTERBASE, MS SQL.

Цель работы: Изучение методов создания форм, отчетов и SQL запросов с помощью конструктора и мастера для баз данных InterBase и MS SQL.

Для выполнения лабораторной работы студентам ставятся следующие задачи:

- Изучение основы разработки программы для работы с базы данных (InterBase или MS SQL);
- Изучение компонентов Delphi для работы с базы данных: Database, Session, Query, StoredProc, DataSource, Table;
- Изучение средств для работы с сервером InterBase:, Database Explorer, Windows Interactive SQL, InterBase Communication Diagnostics Tool, BDE Configuration Utility;
- Изучение установить соединение с базой из программы IBConsole;
- Конфигурирование BDE для доступа к базам данных InterBase.

Задания к лабораторной работе:

1. Создать приложение для работы с БД (InterBase или MS SQL);
2. Создать алиас и установить соединение с базы данных;
3. Создать интерфейс ввода данных в таблицу базы данных;
4. Создать интерфейс просмотра данных хранящихся в таблице;
5. Создать интерфейс редактирования данных хранящихся в таблице;
6. Создать интерфейс для подготовки отчетов по данным хранящихся в таблице.

ВАРИАНТЫ ЗАДАНИЙ К ЛАБОРАТОРНЫМ РАБОТАМ

1. Автоматизированная информационная система «Библиотека»
2. Автоматизированная информационная система «ТСЖ»
3. Автоматизированная информационная система «Паспортный стол»
4. Автоматизированная информационная система «Провайдер»
5. Автоматизированная информационная система «Учебный отдел»
6. Автоматизированная информационная система «Деканат»
7. Автоматизированная информационная система «Кафедра»
8. Автоматизированная информационная система «Отдел кадр»
9. Автоматизированная информационная система «Мастерской»
10. Автоматизированная информационная система «Тест»
11. Автоматизированная информационная система «Абонент»
12. Автоматизированная информационная система «Радиотехника»
13. Автоматизированная информационная система «Аптека»
14. Автоматизированная информационная система «Бухгалтерия»
15. Автоматизированная информационная система «Декларант»
16. Автоматизированная информационная система «Склад»
17. Автоматизированная информационная система «Музей»
18. Автоматизированная информационная система «Больница»
19. Автоматизированная информационная система «Поликлиника»
20. Автоматизированная информационная система «Автопарка»
21. Автоматизированная информационная система «Аэропорта»
22. Автоматизированная информационная система «Аэрокасса»
23. Автоматизированная информационная система «ЖД станция»
24. Автоматизированная информационная система «ЖД касса»
25. Автоматизированная информационная система «Супермаркета»
26. Автоматизированная информационная система «ЗАГС»
27. Автоматизированная информационная система «Диспетчер»
28. Автоматизированная информационная система «Бюро находок»
29. Автоматизированная информационная система «Нотариус»
30. Автоматизированная информационная система «Прокат»

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Определение БД.
2. Особенности реляционной БД.
3. Назначение индексов.
4. Первичные и вторичные индексы.
5. Первичный и альтернативный ключ.
6. Назначение алиаса.
7. Свойства таблицы.
8. Сравнение возможностей по проверке правильности значений в СУБД Paradox и dBase.
9. Назначение ссылочной целостности.
10. Архитектура приложения БД в среде Delphi.
11. Основные компоненты для создания приложений БД. Их назначение, основные возможности.
12. Технология представления нескольких таблиц через один визуальный компонент.
13. Средства навигации по таблицы, технология их реализации в визуальных компонентах.
14. Способы доступа к БД.
15. Средства фильтрации данных, их сравнение.
16. Средства для работы с сервером InterBase
17. BDE Configuration Utility
18. Database Desktop
19. Database Explorer
20. Windows Interactive SQL
21. InterBase Communication Diagnostics Tool
22. InterBase Server Manager
23. Data Migration Expert
24. Конфигурирование BDE для доступа к базам данных InterBase.
25. Создание базы данных
26. Создание и использование доменов
27. Создание таблиц
28. Создание триггеров
29. Создание хранимых процедур
30. Создание представлений
31. Уничтожение компонентов базы данных
32. Организация ссылочной целостности базы данных
33. Массовое удаление и модификация данных
34. Использование компонента Query для работы с базами данных
35. Использование компонента StoredProc
36. Использование компонента DataSource
37. Использование компонента Database

СПИСОК ЛИТЕРАТУРЫ

1. К. Дейт. Введение в системы баз данных. 7-е изд. М.: СПб.: Вильямс, 2000.
2. Вендров А.М. CASE-технологии. Современные методы и средства проектирования информационных систем. М.: Финансы и статистика, 1998
3. Вендров А.М. Проектирование программного обеспечения экономических информационных систем. М.: Финансы и статистика, 2000
4. Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования. М.: Мир, 1999
5. Буч Г., Рамбо Д., Джекобсон А. Язык UML: руководство пользователя. М.: ДМК, 2000
6. М.Р. Когаловский. Энциклопедия технологий баз данных. М.: Финансы и статистика, 2002
7. Атре Ш. Структурный подход к организации баз данных. - М.: Финансы и статистика, 1983. - 320 с.
8. Бойко В.В., Савинков В.М. Проектирование баз данных информационных систем. - М.: Финансы и статистика, 1989. - 351 с.
9. Боуман Д, Эмерсон С., Дарновски М. Практическое руководство по SQL. - Киев: Диалектика, 1997.
10. Мартин Д. Планирование развития автоматизированных систем. - М.: Финансы и статистика, 1984. - 196 с.
11. Мейер М. Теория реляционных баз данных. - М.: Мир, 1987. - 608 с.
12. Нагао М., Катаяма Т., Уэмура С. Структуры и базы данных. - М.: Мир, 1986. - 197 с.
13. Хаббард Д. Автоматизированное проектирование баз данных. - М.: Мир, 1984. - 294 с.
14. Чаудхари С. Методы оптимизации запросов в реляционных системах //СУБД. - 1998. - №3. - С.22-36.
15. Гофман В., Хомоненко А. Delphi 6. – СПб.: БХВ-Петербург, - 2002. 489-901 стр.
16. http://www.opennet.ru/docs/RUS/rusql/
17. http://www.citforum.ru/database/dblearn/dblearn12.shtml