БОРИС ПАХОМОВ

Interbase и C++Builder

НА ПРИМЕРАХ

ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

ОСНОВЫ ЯЗЫКА SQL

ХРАНИМЫЕ ПРОЦЕДУРЫ, ТРИГГЕРЫ, ГЕНЕРАТОРЫ, ИСКЛЮЧЕНИЯ И ПРИВИЛЕГИИ

ПРОГРАММИРОВАНИЕ В СРЕДЕ BORLAND C++ BUILDER

ВЕДЕНИЕ УЧЕТНОЙ КАРТОЧКИ РАБОТНИКА

УЧЕТ ДВИЖЕНИЯ МАТЕРИАЛОВ НА СКЛАДАХ

РАСЧЕТ КРАТЧАЙШЕГО ПУТИ МЕЖДУ ДВУМЯ ТОЧКАМИ СЕТИ

+CD

bhv®

Борис Пахомов

Interbase *и* C++Builder
НА ПРИМЕРАХ

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.068
ББК 32.973.26-018.1
П12

Пахомов Б. И.

П12 Interbase и C++ Builder на примерах. — СПб.: БХВ-Петербург, 2006. — 288 с.: ил.

ISBN 5-94157-570-X

На практических примерах решения типичных задач по управлению кадрами, учету движения материалов на складах и нахождению оптимального пути между двумя пунктами показан процесс проектирования и программной реализации баз данных с использованием популярной СУБД Interbase и среды разработки Borland C++ Builder. Рассмотрены теоретические основы проектирования баз данных: модель базы данных, идентификация сущностей и атрибутов, создание индексов и набора правил при разработке таблиц и др. Дан обзор инструментальных средств Interbase. Описаны основные элементы СУБД Interbase: таблицы, триггеры, процедуры, исключения, привилегии и др. Уделено внимание составлению различных запросов на языке SQL. Рассмотрены основные компоненты среды Borland C++ Builder при разработке приложений баз данных.

На прилагаемом компакт-диске размещены учебные базы данных и исходные коды программ, описанных в книге.

Для начинающих программистов

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Елена Кашлакова*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.08.06.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 23,22.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.02.953.Д.006421.11.04
от 11.11.2004 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-570-X

Оглавление

Введение

Предлагаемая читателю книга посвящена проблемам знакомства с одной из современных передовых систем обработки информации — средой Borland C++Builder. В предыдущих книгах почти не рассматривался вопрос работы с системами управления базами данных (СУБД), и этот пробел автор попытался восполнить в данной работе. Кроме этой цели была и другая: показать начинающему, как на практике применить полученные знания. Для этого автором были разработаны программы в среде Borland C++Builder для решения реальных задач современного предприятия: создание и ведение картотеки учета персонала предприятия, на базе которой можно построить решение общей задачи управления кадрами, и создание и ведение компьютерного складского учета, на базе которого можно строить решение более общей задачи по учету движения материальных ценностей на предприятии. В основе обеих задач лежит использование средств современной СУБД Interbase, которая обеспечивает создание и ведение баз данных в сетевом варианте в режиме "клиент-сервер", имеет она и другие полезные возможности. Эта СУБД может использоваться во взаимодействии со средой Borland C++Builder через имеющиеся в ней специальные компоненты.

При создании программных продуктов использовался Borland C++Builder 6 и встроенная в него СУБД Interbase 6. Для нынешнего времени это несколько староватые версии, ведь в 2006 г. вышли в свет Borland C++Builder 2006 и Interbase 8, а Interbase 7 существует с 2002 г. Однако не станем огорчаться, ибо дела обстоят несколько иначе. Наша основная цель — обучить начинающего программиста работе с СУБД. Он должен узнать, что такое база данных, какова ее подробная структура, ее возможности, как строить ее элементы и как ими эффективно пользоваться при решении задач. На эти вопросы успешно отвечает версия 6. Если в версии 7 добавлен, например, новый тип данных BOOLEAN и удален оператор SET TERM, если седьмая версия обеспечивает многопроцессорную работу серверов и возможность получения статистической информации о списках пользователей, подсоединенных к базе данных, о том, сколько времени работает данный пользователь и какие запросы он выполняет, к каким таблицам обращается и т. п., то такие вопросы начинающего мало волнуют — это задачи администратора СУБД, а ему, начинающему, в первую очередь надо узнать, как построить свои базы данных, как их наполнять информацией и все остальное в этом же роде. Версия 6 Interbase срослась с Borland C++Builder 6, широко распространенным сегодня, который для начала вполне может быть удовлетворительным: из него взяты практически те же компоненты, которые использовались бы и в версии 2006 (третья книга автора как раз и посвящена этой

версии), которая только-только поступает на рынок программных продуктов. К тому же, Interbase 6 — продукт бесплатный. Кстати, все написанное работает и в СУБД Firebird, которая также является бесплатной.

Читающему эту книгу надо быть знакомым с основными компонентами Borland C++Builder. Здесь он познакомится только с Interbase: увидит и научится, на что автор надеется, как применять знания по СУБД и Borland C++Builder в решении практических задач предприятия.

Отметим еще один момент. В созданных и прилагаемых к настоящей книге комплексах программ (обработка карточки по учету кадров и обработка данных складского учета) имеются очень похожие обработчики событий, в частности, компонентов ComboBox. И довольно большие по объему. Многие из этих компонентов можно было бы заменить одной функцией с параметрами. Однако этого не сделано с определенной педагогической целью: пусть читатель постоянно натыкается на одни и те же малоразличимые блоки, привыкая к их содержимому. Для начинающего простые блоки, естественно, более понятны, чем усложненные функции, построенные как обобщающие элементы этих блоков. По мере привыкания читателя к повторяющимся блокам у него самого возникнет потребность их обобщить, и тогда он начнет искать способ, как это сделать: в программе по складскому учету в одном случае показано, как это делается.

Тексты обоих комплексов программ, снабженные в наиболее трудных местах комментарием, совместно с двумя базами данных для каждого из них прилагаются на компакт-диске.

И еще одно замечание. Для обслуживания результатов работы по вводу данных и расчету документов предусмотрены выводы результатов в табличном представлении, для чего использовались компоненты TDGrid. Некоторые из этих компонентов наделены свойствами причаливания (Docable). В частности, после щелчка на таком элементе он приобретает свойства перемещаемого и изменяющего свои размеры окна. Это сделано для таких объектов, как справочник материалов, справочник поставщиков-покупателей и др. Дело в том, что при проектировании таких справочников из-за большого количества имеющихся у них полей последними занимается почти вся площадь рабочего стола, и чтобы видеть всю картину в момент отладки, желательно наблюдать и поля ввода, и в целом всю таблицу через элемент TDBGrid. А для него остается небольшое пространство рабочего стола. При проектировании это еще можно пережить, но в реальной эксплуатации такое положение создает неудобства. Поэтому следует создать возможность перетаскивания мышью элемента TDBGrid в более удобное для пользователя место рабочего стола, одновременно давая возможность изменять размеры элемента.

Желаю успеха изучающим этот материал.

Глава 1

Базы данных. Interbase

Interbase — это система управления базой данных (СУБД), имеющая собственную структуру. Предоставляет пользователю возможность не только создавать собственно БД (структуру и элементы), но и наполнять ее данными, модифицировать ее, поддерживать в актуальном состоянии и пользоваться ее содержимым. У нее есть собственный алгоритмический язык. Этот язык — подмножество языка структурированных запросов — SQL. Interbase содержит в себе средства обеспечения собственной безопасности. Хотя Interbase может применяться в качестве автономного программного продукта при работе с базами данных, ее возможности можно использовать и при создании приложений в среде разработки Borland C++Builder, в которой имеется набор компонентов для работы с базами данных этого типа. Interbase обеспечивает работу в системе "клиент-сервер", давая возможность пользователям создавать свои серверы с базами данных, которые обслуживают сеть удаленных клиентов. Чтобы лучше понимать работу компонентов раздела Interbase среды Borland C++Builder, рассмотрим основные положения СУБД Interbase.

Обзор некоторых инструментальных элементов Interbase

IBConsole

Это графический пользовательский интерфейс, представленный отдельным программным продуктом. В его среде можно конфигурировать и поддерживать InterBase-сервер, создавать и вести базы данных на этом сервере, интерактивно работать на языке SQL, поддерживать пользователей и обеспечивать безопасность.

IBConsole запускается под Windows, однако может управлять базами данных на любом InterBase-сервере в локальной сети, в UNIX, Linux и NetWare-системах.

SQL

Interbase поддерживает язык структурированных запросов (SQL), с помощью которого можно работать с базами данных. Осуществлять эти запросы можно с помощью так называемого интерактивного SQL — isql, либо запуская из командной строки консольного режима (запуск интерактивного SQL из командной строки выполняется с помощью специальной утилиты с именем isql), после чего можно либо набирать сами операторы SQL, либо задать имя файла с операторами SQL, который будет исполняться, либо можно воспользоваться средствами IBConsole, в меню которого имеется режим выхода на работу с интерактивным SQL.

При создании базы данных встречаются с понятием диалекта SQL. Диалектов существует несколько. Дело в том, что некоторые элементы языка, такие как разделители объектов (двойные кавычки, например), данные точных типов (не с плавающей точкой, которые, по определению, не являются точными), такие как DECIMAL и NUMERIC, дата и время, по-разному интерпретируются в разных версиях Interbase. Чтобы добиться совместимости с предыдущими версиями, и было введено понятие диалекта SQL. В Interbase 6, который мы здесь рассматриваем, диалекты 1 и 2 обеспечивают совместимость с предыдущими версиями Interbase, диалект 3 — это версия SQL-92, интерпретируемая в Interbase 6.

Приведем перечень некоторых SQL-операторов, подробный смысл большинства из которых определен в последующем материале:

- ALTER DATABASE — добавляет вторичные файлы к текущей базе данных;
- ALTER DOMAIN — изменяет определение домена;
- ALTER EXCEPTION — изменяет текст сообщения, связанного с существующим исключением;
- ALTER INDEX — активирует или деактивирует индекс;
- ALTER PROCEDURE — изменяет определение существующей хранимой процедуры;
- ALTER TABLE — изменяет таблицу путем добавления, удаления или модификации колонок или ограничений на целостность данных;
- ALTER TRIGGER — изменяет существующий триггер;
- COMMIT — делает изменения транзакции неизменными и завершает транзакцию;

- CONNECT — соединяет с одной или большим количеством баз данных;
- CREATE DATABASE — создает новую базу данных;
- CREATE DOMAIN — задает определение колонки, которое становится глобальным в базе данных;
- CREATE EXCEPTION — создает определенную пользователем ошибку и связанное с ней сообщение об ошибке для использования в хранимых процедурах и триггерах;
- CREATE GENERATOR — создает генератор к базе данных;
- CREATE INDEX — создает индекс на одной и более колонках таблицы;
- CREATE PROCEDURE — создает хранимую процедуру, ее входные и выходные параметры и ее действия;
- CREATE ROLE — создает роль (пользователя или группу, обладающих определенными правами доступа. Например, роль "администратор");
- CREATE SHADOW — создает одну и более копий базы данных;
- CREATE TABLE — создает таблицу базы данных;
- CREATE TRIGGER — создает триггер базы данных;
- CREATE VIEW — создает вьюер базы данных;
- DECLARE EXTERNAL FUNCTION — объявляет функцию к базе данных (объявление будет храниться в базе данных), определенную пользователем (user-defined function — UDF): имя, где ее найти, входные параметры, единственное возвращаемое ею значение. Каждая UDF в библиотеке должна быть объявлена один раз для каждой базы данных, где она будет использована;
- DECLARE FILTER — объявляет существующий BLOB-фильтр к базе данных (объявление будет храниться в базе данных): где его найти, его имя, с какими типами данных он работает. BLOB-фильтр — это написанная пользователем программа, которая преобразует данные, хранящиеся в колонках данных типа BLOB (об этом и других типах будет сказано далее) в другой тип данных;
- DECLARE TABLE — объявляет таблицу базы данных;
- DELETE — удаляет строки из таблицы;
- DISCONNECT — отсоединяет приложение от базы данных;
- DROP DATABASE — удаляет базу данных;
- DROP DOMAIN — удаляет домен из базы данных;
- DROP EXCEPTION — удаляет исключение из базы данных;
- DROP EXTERNAL FUNCTION — удаляет объявление UDF из базы данных;

- DROP FILTER — удаляет объявление фильтра из базы данных;
- DROP INDEX — удаляет индекс из базы данных;
- DROP PROCEDURE — удаляет хранимую процедуру из базы данных;
- DROP ROLE — удаляет роль из базы данных;
- DROP SHADOW — удаляет набор копий базы данных;
- DROP TABLE — удаляет таблицу из базы данных;
- DROP TRIGGER — удаляет триггер из базы данных;
- DROP VIEW — удаляет вьюер из базы данных;
- EXECUTE PROCEDURE — вызывает хранимую процедуру;
- GRANT — назначает привилегии пользователям базы данных;
- INSERT — добавляет одну и более строк к таблице;
- REVOKE — отбирает привилегии у пользователей базы данных;
- ROLLBACK — восстанавливает базу данных в ее предыдущее (перед началом транзакции) состояние;
- SELECT — извлекает данные из одной и более таблиц базы данных;
- SET SQL DIALECT — задает диалект SQL для доступа к базе данных;
- SET STATISTICS — пересчитывает избирательность указанного индекса (избирательностью пользуется оптимизатор Interbase для планирования исполнения запроса на выборку);
- SET TRANSACTION — запускает транзакцию, определяя ее взаимодействие с базой данных и другими транзакциями;
- UPDATE — изменяет данные в строке таблицы или вьюера;
- WHENEVER — задает обработку ошибок при исполнении SQL-операторов.

Проектирование баз данных

С помощью базы данных описывают не только информацию о реальных предприятиях и организациях, но и обеспечивают ее обработку, символически представляя реальные объекты своими специальными структурами: таблицами, вьюерами, хранимыми процедурами и другими объектами. Поскольку информация в базе данных организована и хранится в виде определенных объектов, то к таким объектам может быть организован доступ с помощью приложений, создаваемых в различных программных средах и пользовательских интерфейсах, формируемых с помощью средств БД.

Одним из главных факторов в создании базы данных является ее правильное проектирование. Обычно перед помещением в базу данных информация представляется в виде прямоугольных таблиц, состоящих из строк и столбцов (колонок) и отражающих определенные сущности действительности. Например, процесс начисления и выдачи зарплаты работникам предприятия можно отобразить в виде такой прямоугольной таблицы, в которой в качестве столбцов будут характеристики работника и виды начислений и удержаний, даты оплаты, а в качестве строк — конкретные работники. Логическое проектирование БД — это интерактивный процесс, состоящий из расчленения на мелкие, элементарные данные больших структур информации. Этот процесс носит название нормализации (в философском смысле этот процесс носит название анализа). Цель нормализации состоит в определении природных связей между данными в будущей базе данных. Это делается путем разбиения конкретной таблицы на более мелкие таблицы с меньшим количеством столбцов. После такого разбиения лучше видно, каковы на самом деле связи между построенными таблицами (а этот обратный процесс в философии носит название синтеза).

Модель базы данных

При проектировании базы данных важно представлять различие между описанием базы данных (БД) и самой базой данных. Описание БД называют моделью данных. Она формируется на этапе проектирования БД. Модель — это шаблон для создания таблиц и их колонок. Он описывает логическую структуру БД, включая данные и их суть, типы данных, пользовательские операции, связи между объектами-данными, ограничения на целостность данных. Логические структуры, описывающие базу данных, не подвержены воздействиям со стороны соответствующих физических структур, хранимых в БД. *Реляционную базу данных* (базу данных, в которой отношения между данными определены) нетрудно переносить на различные платформы, потому что механизм доступа к базе данных определен моделью базы данных и остается неизменным, где бы база данных ни хранилась. Логическая структура БД также независима от взгляда на нее конечного пользователя. Логическая структура, если пользоваться строительной терминологией, это проект дома, а сама БД — дом, построенный по этому проекту. Поэтому, имея проект "дома", его можно построить и на земле (платформе) Windows, и на земле (платформе) UNIX, и т. д.

При создании базы данных ее можно настраивать на различные информационные потребности пользователя. Это делается с помощью так называемых *вьюеров* (viewers) — программ просмотра данных, которые выводят подмножества данных по заказам конкретных пользователей или их групп. Вьюеры могут использоваться, чтобы спрятать необходимые данные или отфильтровать данные, которые не требуются пользователю.

Цели проектирования

При проектировании базы данных разработчики преследуют определенные цели, которые, в основном, состоят в следующем:

- удовлетворить пользовательским требованиям к базе данных. До ее проектирования надо исследовать требования пользователей к БД, обеспечение пользовательского объема информации, выборки по запросам, расчет необходимых данных, безопасность информации и т. п.;
- обеспечить связность и целостность данных. Не должно быть такой ситуации, когда, например, строка из одной таблицы модифицируется или удаляется, а связанные с ней данные в другой таблице автоматически не претерпевают никаких изменений;
- обеспечить пользователя естественным, легко понимаемым структурированием информации. Хороший дизайн базы данных позволяет легче с ней общаться. Такая БД проще поддерживается и обновляется;
- удовлетворять пользовательским требованиям по эксплуатации БД. Хорошо спроектированная БД дает лучшие результаты по ее эксплуатации. Если таблицы слишком велики или у них слишком много или слишком мало индексов (об индексах и их свойствах — позднее), результатом будет большое время ожидания результатов. Если БД слишком велика и с высоким объемом транзакций (о них тоже поговорим позже), проблемы производительности во многом будут зависеть от качества проектирования (дизайна).

Структура проектирования БД

При проектировании базы данных выполняются следующие конкретные действия:

- определение требований к информации (объем, частота использования, безопасность и т. д.) путем опроса будущих пользователей;
- анализ реальных объектов, которые требуется смоделировать в БД. Перевод управления объектами в управление элементами БД и формирование списка элементов БД (синтез БД);
- решение вопросов идентификации элементов в БД;
- разработка набора правил доступа к каждой таблице (того, как каждая таблица станет наполняться и модифицироваться);
- установка отношений между объектами (таблицами и колонками);
- планирование безопасности БД.

Сбор и анализ данных

Перед проектированием основных объектов базы данных — таблиц и их колонок — необходимо провести анализ реальных данных на концептуальном уровне:

- главные функции и деятельность предприятия. Например, наем работников, морская перевозка продукции, управление запчастями, обработка платежных чеков и т. д.;
- объекты этой деятельности и функции (т. е. на что направлена деятельность). Соединение последовательности деловых операций или транзакций (групп операций, имеющих законченное экономическое действие — например, бухгалтерская проводка) в единую последовательность событий поможет определить все сущности и отношения в операционных актах. Например, когда вы наблюдаете за процессом найма рабочей силы, вы сможете сразу выделить и идентифицировать такие сущности этой деятельности, как должность (JOB), работник (EMPLOYEE) и подразделение (DEPARTMENT);
- характеристики этих объектов. Например, сущность EMPLOYEE может включать такую информацию, как табельный номер работника (EMPLOYEE_ID), имя (FIRST_NAME), фамилию (LAST_NAME), вид работы (JOB), зарплату (SALARY) и т. д.;
- отношения между объектами. Например, как такие сущности, как EMPLOYEE, JOB и DEPARTMENT, соотносятся между собой? Допустим, работник имеет одну должность и входит в одно подразделение, в то время как одно подразделение имеет множество работников и должностей.

Идентификация сущностей и атрибутов

Основываясь на требованиях заказчика (пользователя), которые вы собрали, можно определить сущности и их атрибуты. Сущность — это тип, объект или вещь, которые нуждаются, чтобы их описали в базе данных. Это может быть физический объект или компания, должность или проект. Каждая сущность имеет свойства, которые называют ее атрибутами. Допустим, что проектируется база данных, которая будет содержать сведения о работниках компании, об иерархии подразделений компании, о ее текущих проектах, о клиентах (покупателях) и продажах. Пример в табл. 1.1 показывает, как создать список сущностей и атрибутов для организации данных.

Таблица 1.1. *Сущности и атрибуты данных*

Сущность	Атрибуты сущности
EMPLOYEE	Employee Number (Табельный номер) Last Name (Фамилия) First Name (Имя) Department Number (Код подразделения) Job Code (Код должности) Phone Extension (Телефон) Salary (Зарплата)
DEPARTMENT	Department Number (Код подразделения) Department Name (Наименование) Department Head Name (Имя руководителя) Budget (Бюджет) Location (Месторасположение) Phone Number (Телефон)
PROJECT	Project ID (Код проекта) Project Name (Название проекта) Project Description (Описание проекта) Team Leader (Руководитель) Product (Изделие)

Составляя таким способом списки сущностей и соответствующих атрибутов, удается удалить избыточные записи. Могут быть таблицами сущности из составленного списка? Могут ли быть перемещены колонки из одной группы в другую? Появляется ли один и тот же атрибут в разных сущностях? На такие вопросы надо дать ответ. Каждый атрибут может появляться лишь однажды, и вы должны определить, какая сущность является первичным собственником этого атрибута. Например, атрибут DEPARTMENT HEAD NAME (имя руководителя) может быть удален, так как имя работника (FIRST NAME и LAST NAME) и его табельный номер уже существуют в сущности EMPLOYEE. А для доступа к руководителю подразделения можно воспользоваться табельным номером работника (руководитель в качестве работника имеет свой табельный номер).

Далее описывается, как отобразить составленный вами список в объекты базы данных: сущности — в таблицы, а атрибуты — в колонки.

Проектирование таблиц

В реляционной базе данных объектом базы данных, представляющим отдельную сущность, является таблица, которая является двумерной матрицей строк и столбцов. Каждый столбец матрицы представляет один атрибут. Каждая строка представляет специфический экземпляр сущности. Например, таблица "Работники" (EMPLOYEE) — это сущность. Ее атрибуты — код работника, фамилия, имя, контактный телефон и т. д. Специфические экземпляры сущности — это первый работник, второй работник и т. д. После определения сущностей и атрибутов создается модель данных, которая служит логическим проектом структуры вашей InterBase-базы данных. Модель данных является детальным описанием сущностей базы данных — таблиц, колонок, свойств колонок и отношений между таблицами и колонками. Пример в табл. 1.2 показывает, как сущность EMPLOYEE была преобразована из списка сущности/атрибуты в таблицу/колонки.

Таблица 1.2. *Таблица EMPLOYEE*

EMP_NO	LAST_NAME	FIRST_NAME	DEPT_NO	JOB_CODE	PHONE_EXT	SALARY
24	Smith	John	100	Eng	4968	64000
48	Carter	Catherine	900	Sales	4967	72500
36	Smith	Jane	600	Admin	4800	37500

Каждая строка таблицы EMPLOYEE представляет одного работника. EMP_NO, LAST_NAME, FIRST_NAME, DEPT_NO, JOB_CODE, PHONE_EXT и SALARY — это колонки, которые представляют атрибуты работника. Когда строка добавляется к таблице, элемент строки называется в таблице ячейкой. Вся строка записи называется в таблице просто записью. Колонки еще называют полями таблицы.

Определение неповторяющихся атрибутов

Одной из задач проектирования базы данных является обеспечение уникального определения каждого экземпляра (говоря простым языком, необходимо сделать так, чтобы система могла извлечь из таблицы единственную строку). Одну строку от другой отличают по значениям так называемого *первичного ключа*. Первичный ключ определяет неповторимость: значения, введенные в колонку первичного ключа или во множество колонок, составляющих первичный ключ (а он может состоять и из нескольких атрибутов), являются уникальными для каждой строки. Если вы попытаетесь ввести в первичный ключ значение, которое уже существует в другой строке той же колонки, InterBase прекратит операцию и выдаст ошибку. Например, в таб-

лице EMPLOYEE поле EMP_NO является уникальным атрибутом, который может использоваться в качестве идентификатора каждой строки таблицы. Следовательно, этот реквизит можно взять в качестве первичного ключа этой таблицы. Когда вы выбираете какой-то реквизит на роль первичного ключа, проверьте, действительно ли он уникален, т. е. не встречается более одного раза во всех строках таблицы.

Если не существует единственной колонки, обладающей уникальностью, то следует определить первичный ключ на основе двух или более колонок, которые вместе дают требуемую однозначность.

Вместе с первичным ключом таблицы (PRIMARY KEY) существует и *уникальный ключ* (UNIQUE KEY). В колонку, выбранную в качестве уникального ключа, вводятся уникальные значения. У таблицы может быть только один первичный ключ и любое количество уникальных. Если поле помечено как уникальное, то в него можно вводить для каждой строки только различные данные. Если в некоторую строку данного поля ввести данное, которое уже встречалось в какой-либо другой строке, система выдаст сообщение об ошибке. Так работает Interbase. Первичный ключ и уникальный ключ задаются при создании таблицы.

Создание набора правил при разработке таблицы

При разработке таблицы базы данных требуется создать набор правил для каждой таблицы и колонки, устанавливающих и обеспечивающих целостность данных (об этом мы говорили ранее: связанные данные должны и модифицироваться, и удаляться совместно). Эти правила включают:

- определение типов данных;
- выбор интернациональных наборов символов для интерпретации данных;
- создание колонок, основанных на доменах: например, в разных таблицах имеются колонки с одинаковыми характеристиками. Тогда создают элементы-домены с такими характеристиками и на их основе строят колонки таблиц;
- установка значений по умолчанию и по NULL-статусу;
- определение ограничений на целостность данных;
- определение CHECK-ограничений.
- Далее описывается суть каждой из определенных только что позиций.

Спецификация типов данных

Как только вы выбрали атрибут в качестве колонки таблицы, вы должны определить тип данного для атрибута. Тип данного автоматически определя-

ет, какие операции можно совершать с данными этой колонки и какое дисковое пространство может занимать элемент данного. Основные категории типов данных в SQL таковы:

- ❒ символьный тип;
- ❒ целое число;
- ❒ фиксированное десятичное число и десятичное число с плавающей точкой;
- ❒ типы данных даты и времени;
- ❒ типы BLOB-данных (графические данные, данные мультимедиа и т. п.).

> ***Примечание***
>
> В версии Interbase 7 добавлен тип BOOLEAN.

Выбор интернациональных наборов символов

При создании базы данных следует определить набор символов по умолчанию, который задает:

- ❒ какие символы должны использоваться в колонках с типами данных CHAR, VARCHAR и BLOB-text (текстовая информация типа BLOB);
- ❒ в каком порядке по умолчанию данные колонки будут сортироваться (по убыванию или по возрастанию). Элемент COLLATE оператора CREATE TABLE (Создать таблицу) позволяет пользователям указать свой порядок сортировки для колонок с типами данных CHAR и VARCHAR. Надо выбрать порядок сортировки, поддерживаемый заданным набором символов (например, символы английского алфавита упорядочиваются не так, как русского, потому что у них могут быть различные таблицы кодировки). Выбор набора символов по умолчанию — это шаг разработчика базы данных к ее международному использованию. Например, следующий оператор создает базу данных, которая использует набор символов ISO8859_1:

```
CREATE DATABASE 'employee.gdb'
DEFAULT CHARACTER SET iso8859_1;
```

Вы можете переопределить набор символов, заданный по умолчанию, создав другой набор, когда задаете тип данных колонки: спецификация типа данных для CHAR, VARCHAR или BLOB-text может включать утверждение CHARACTER SET для спецификации собственного набора символов для колонки. Если вы не указали свой набор для колонки, то для нее берется набор, принятый по умолчанию для базы данных. Если набор символов по умолчанию для базы данных позже будет изменен, все колонки, определенные после этого изменения, будут иметь новый набор символов, но существовавшие до изменения колонки новое изменение не затронет. Если вы не

задали набор символов по умолчанию во время создания базы данных, то набор будет установлен в значение NONE. Это означает, что не существует набора символов для колонок. Данные в таком случае хранятся и извлекаются в том порядке, в котором они были введены. Можно загрузить любой набор символов в колонку со статусом NONE, но нельзя загрузить те же данные в другую колонку, для которой был определен другой набор символов. Просто не будет выполнена транслитерация (отображение символов одной таблицы в другую) между исходным и новым набором символов.

Указание доменов

Когда некоторые таблицы в базе данных содержат колонки с одинаковыми характеристиками, то по ним можно создать шаблон элемента базы данных (домен) и хранить его в базе данных, а при создании таблиц делать ссылки на такой домен.

Установка значений по умолчанию и NULL-статуса

Когда определяется колонка таблицы, в операторе определения имеется утверждение DEFAULT, в котором задается значение колонки по умолчанию. Фактически это не значение всей колонки, а ее ячейки: при выполнении над таблицей операторов INSERT (вставить) или UPDATE (обновить) действия осуществляются над строками, а в рамках колонки — над ее ячейками. Если значение колонки во вставляемой строке не будет определено, то вместо него вставится то, которое указано в утверждении DEFAULT. Такой подход позволяет избежать многих неприятностей при работе с таблицей. Вот несколько примеров задания значений по умолчанию при определении колонок:

```
stringfld VARCHAR(10) DEFAULT 'abc'
integerfld INTEGER DEFAULT 1
numfld NUMERIC(15,4) DEFAULT 1.5
datefld1 DATE DEFAULT '5/5/2005'
datefld2 DATE DEFAULT 'TODAY'
userfld VARCHAR(12) DEFAULT USER
```

Если назначить по умолчанию NULL, то в колонку будет вставляться значение NULL (аналог значения "пусто" в C++), если пользователь не введет значение. Если назначить по умолчанию NOT NULL, это заставит систему потребовать от пользователя обязательного ввода значения в данное поле или же определить значение по умолчанию для колонки. Значение NOT NULL должно обязательно назначаться для первичного (PRIMARY KEY) и уникального (UNIQUE KEY) ключей-колонок.

Определение ограничений на целостность данных

Ограничения на целостность данных — это правила, управляющие связями колонка-таблица и таблица-таблица и проверкой данных на достоверность. Они охватывают все транзакции, которые имеют доступ к базе данных и автоматически поддерживаются системой. Ограничения на целостность данных могут быть применены как в целом к таблице, так и к отдельной колонке. Мы видели, что ограничения, налагаемые на первичный или уникальный ключ, гарантируют, что любые два значения в колонке или в наборе колонок не будут совпадать.

Значения данных, которые однозначно определяют строки (первичный ключ) в одной таблице, могут появляться в других таблицах. В этой связи вводится понятие *внешнего ключа* (FOREIGN KEY) — колонки или набора колонок таблицы, которые содержат значения, совпадающие с первичным ключом в другой таблице. С помощью утверждений языка SQL ON UPDATE и ON DELETE определяют, что произойдет с соответствующим внешним ключом, если первичный ключ изменится или будет удален. В листинге 1.1 — пример таблицы "Дополнительные данные" dop_dannie из приложения по обработке данных по персоналу предприятия (*см. главу 2*). Отметим, что здесь и в дальнейшем операторы SQL в листингах могут быть написаны в верхнем или нижнем регистрах, что не имеет существенного значения: компилятор в обоих случаях не выдает ошибки.

Листинг 1.1

```
create table dop_dannie
/*при удалении первичного ключа
таблицы osn_dannie автоматически будет удалена строка с таким же ключом
этой таблицы*/
(
       tn float not null,
       adr_e_mail varchar(25),
       alimenti varchar(100),
       primary key (tn),
             foreign key (tn) references osn_dannie (tn)
       on delete cascade
);
```

Задание CHECK-ограничений

Наряду с применением первичных и уникальных ключей, можно использовать и другой тип проверки данных на достоверность. Это применение

утверждения CHECK, которое действует на данное в момент его ввода. CHECK-ограничение запускает проверку условия, которое должно давать значение "истинно" (TRUE), когда происходит вставка в таблицу или колонку или их обновление. Например, при создании домена, который затем станет использоваться при создании колонки "Покупатель" (custno) в некоторой таблице, введен контроль на ввод значений в такую колонку (значения должны быть обязательно больше 1000) оператором:

```
create domain custno
as integer
check (value > 1000);
```

Установление связей между объектами

Связи между объектами (таблицами и колонками) в базе данных должны определяться на этапе проектирования. Например, связаны ли в компании работники и подразделения? Ответ: связаны. Каким образом? Ответ: один работник может входить только в одно подразделение (связь "один-к-одному"), но одно подразделение имеет много работников (связь "один-ко-многим"). Как связаны проекты и работники? Ответ: один работник может работать в одном и более проектах, и один проект может включать в себя некоторое количество работников (связь "многие-ко-многим"). Каждый из этих типов связей должен быть смоделирован в базе данных. Реляционная модель (модель со связями) представляет связи "один-ко-многим" с помощью пары первичный ключ/внешний ключ. Обратимся к рассмотренной уже таблице EMPLOYEE (см. табл. 1.2) и таблице проектов PROJECT (табл. 1.3).

Таблица 1.3. *Таблица PROJECT*

PROJ_ID	TEAM_LEADER	PROJ_NAME	PROJ_DESC	PRODUCT
DGPII	36	Automap	BLOB	data hardware
VBASE	48	Video database	BLOB	data software
HWRII	24	Translator upgrade	BLOB	data software

Проект может включать много работников, поэтому, чтобы избежать дублирования данных, таблица PROJECT должна ссылаться на информацию о работниках с помощью внешнего ключа. TEAM_LEADER — это внешний ключ, ссылающийся на первичный ключ EMP_NO в таблице EMPLOYEE. Теперь модификация табельного номера в таблице EMPLOYEE не пройдет бесследно для таблицы PROJECT, о чем говорится в следующем разделе.

Принудительное обеспечение целостности данных

Обычные соображения, лежащие в основе задания внешнего ключа, — быть уверенным, что станет поддерживаться целостность данных, когда более одной таблицы ссылается на одни и те же данные, так как строки в одной таблице всегда должны соответствовать строкам в ссылающихся на нее других. Например, обрабатывается движение товаров на складе. В вашей базе данных имеются таблицы "Поставщики", "Покупатели", содержащие коды товаров, и таблица "Ценник", которую сегодня модно называть "Прайс-лист" и которая содержит характеристики товаров: цену, единицу измерения и т. п. Все три таблицы связаны между собой кодом товара. Поэтому, например, неаккуратное изменение какой-либо строки в "Ценнике" может разрушить данные в остальных таблицах, а поддержание режима целостности данных обеспечивает их нормальное функционирование. InterBase принудительно обеспечивает ссылочную целостность следующими путями:

- до добавления внешнего ключа уникальный и первичные ключи, на которые ссылается внешний ключ, должны быть уже определены;
- если информация изменена в одном месте, она должна быть изменена во всех остальных местах, в которых она существует. Например, при создании таблицы БД внешний ключ задается такой последовательностью операторов:

```
foreign key (col [, col …]) references other_table (col [, col …])
[on delete {no action|cascade|set default|set null}]
[on update {no action|cascade|set default|set null}]
```

  Здесь `col` — это колонки, составляющие внешний ключ в данной таблице и первичный ключ в ссылочной таблице, `other_table` — это таблица, на которую ссылается данная таблица.

InterBase выполняет изменение информации автоматически, когда вы применяете режим ON UPDATE к утверждению REFERENCES при определении ограничений для таблицы или ее колонок. Вы можете задать:

- чтобы значение внешнего ключа заменялось на новое значение первичного ключа (режим CASCADE);
- чтобы оно установилось на значение колонки по умолчанию (режим SET DEFAULT);
- или в ноль (режим SET NULL).

Если вы выбираете режим NO ACTION в качестве режима ON UPDATE, вы должны вручную удостовериться, что внешний ключ обновлен после обновления первичного ключа. Например, чтобы изменить значение в колонке

EMP_NO (первичный ключ) таблицы EMPLOYEE, следует обязательно изменить колонку TEAM_LEADER (внешний ключ) таблицы PROJECT.

Нормализация таблицы

После того как таблица, ее колонки и ключи определены, надо посмотреть на проектирование как бы в целом и применить в нем правила нормализации таблицы, чтобы обнаружить логические ошибки. Нормализация — это разбиение больших таблиц на мелкие с целью объединить в группы данные, связанные между собой (детально методы нормализации здесь не рассматриваются). Такие связанные данные как раз и объединяются в таблицу. А польза от их объединения в общую таблицу очевидна: их легче обновлять и удалять (все происходит в одной и той же таблице, а не в разных), снижается вероятность дублирования и ввода несовместимых данных. В общем случае процесс нормализации включает в себя:

- удаление повторяющихся групп данных;
- удаление частично зависимых колонок;
- удаление транзитивно зависимых колонок.

Удаление повторяющихся групп данных

Если для какого-то значения первичного ключа существуют колонки, в которых имеется более одного значения, то такие значения образуют повторяющиеся группы. Посмотрим на таблицу DEPARTMENT в первом варианте (табл. 1.4). Ее первая строка, в которой DEPT_NO = 100, содержит повторяющуюся группу в колонке DEPT_LOCATIONS (Monterey, Santa Cruz, Salinas).

В таблице DEPARTMENT во втором варианте (табл. 1.5) видно, что для каждого значения первичного ключа, равного 100, все колонки, кроме колонки DEPT_LOCATION, содержат повторяющуюся информацию, т. е. имеют место повторяющиеся группы.

Таблица 1.4. *Первый вариант таблицы DEPARTMENT*

DEPT_NO	DEPARTMENT	HEAD_DEPT	BUDGET	DEPT_LOCATIONS
100	Sales	000	1000000	Monterey, Santa Cruz, Salinas
600	Engineering	120	1100000	San Francisco
900	Finance	000	400000	Monterey

Таблица 1.5. *Второй вариант таблицы DEPARTMENT*

DEPT_NO	DEPARTMENT	HEAD_DEPT	BUDGET	DEPT_LOCATION
100	Sales	000	1000000	Monterey
100	Sales	000	1000000	Santa Cruz
600	Engineering	120	1100000	San Francisco
100	Sales	000	1000000	Salinas

Чтобы нормализовать такую таблицу, мы должны удалить атрибут DEPT_LOCATION из таблицы и создать другую таблицу под именем DEPT_LOCATIONS (табл. 1.6). А затем мы можем создать первичный ключ из комбинации атрибутов DEPT_NO и DEPT_LOCATION.

Таблица 1.6. *Таблица DEPT_LOCATIONS*

DEPT_NO	DEPT_LOCATION
100	Monterey
100	Santa Cruz
600	San Francisco
100	Salinas

Теперь уже для каждого места размещения подразделения DEPT_LOCATION имеется отдельная строка, и повторяющиеся группы исчезли. У таблицы DEPARTMENT во втором варианте поля DEPT_NO и DEPT_LOCATION будут представлять собой внешний ключ, ссылающийся на первичный ключ таблицы DEPT_LOCATIONS. Любые изменения в таблице DEPT_LOCATIONS отразятся в таблице DEPARTMENT второго варианта.

Удаление частично зависимых колонок

Другим важным шагом в процессе нормализации является удаление любой неключевой колонки, которая зависит от некоторой части ключа. Такие колонки называют частично зависимыми. Неключевые колонки представляют информацию об объекте, но не определяют его однозначно. Предположим, например, что вам требуется отыскать некоего работника с помощью таблицы PROJECT (табл. 1.7) с составным первичным ключом по колонкам EMP_NO и PROJ_ID.

Таблица 1.7. *Таблица PROJECT (вначале)*

EMP_NO	PROJ_ID	LAST_NAME	PROJ_NAME	PROJ_DESC	PRODUCT
44	DGPII	Smith	Automap	BLOB data	hardware
47	VBASE	Jenner	Video	data- base BLOB data	software
24	HWRII	Stevens	Translator	up- grade BLOB data	software

Сразу возникнет проблема, так как поля PROJ_NAME, PROJ_DESC и PRODUCT — это атрибуты идентификатора проекта (PROJ_ID), но не табельного номера работника (EMP_NO), который таким образом частично зависит от первичного ключа EMP_NO + PROJ_ID. Чтобы нормализовать эту таблицу, надо будет удалить колонку LAST_NAME из таблицы PROJECT и создать другую таблицу под именем EMPLOYEE_PROJECT, которая в качестве первичного ключа имеет комбинацию EMP_NO + PROJ_ID (табл. 1.8).

Таблица 1.8. *Таблица EMPLOYEE_PROJECT*

EMP_NO	PROJ_ID	LAST_NAME
44	DGPII	Smith
47	VBASE	Jenner
24	HWRII	Stevens

А таблица PROJECT примет вид (табл. 1.9):

Таблица 1.9. *Таблица PROJECT (после преобразования)*

EMP_NO	PROJ_ID	PROJ_NAME	PROJ_DESC	PRODUCT
44	DGPII	Automap	BLOB data	hardware
47	VBASE	Video	database BLOB data	software
24	HWRII	Translator	upgrade BLOB data	software

Теперь для каждого работника в проекте будет существовать единственная строка, с ним связанная. Внешним ключом в таблице PROJECT надо объявить EMP_NO + PROJ_ID. Тогда при изменении данных в таблице EMPLOYEE_PROJECT автоматически изменятся данные в таблице PROJECT.

Удаление транзитивно зависимых колонок

Следующим шагом по нормализации таблицы является удаление любой неключевой колонки, которая зависит от другой неключевой колонки. Каждая такая неключевая колонка должна быть характеристикой ключевой колонки. Например, мы хотим добавить в таблицу PROJECT колонки TEAM_LEADER_ID и PHONE_EXT и сделать колонку PROJ_ID первичным ключом (табл. 1.10). Тогда PHONE_EXT становится неключевой колонкой, являясь характеристикой колонки TEAM_LEADER_ID (это ведь телефон, связанный с руководителем).

Таблица 1.10. *Таблица PROJECT ненормализованная*

PROJ_ID	TEAM_LEADER_ID	PHONE_EXT	PROJ_NAME	PROJ_DESC	PRODUCT
DGPII	48	4929	Automap	BLOB data	hardware
VBASE	36	4967	Video	database BLOB data	software
HWRII	24	4668	Translator	upgrade BLOB data	software

Чтобы нормализовать эту таблицу (табл. 1.11), мы должны удалить колонку PHONE_EXT, заменить TEAM_LEADER_ID на TEAM_LEADER и сделать TEAM_LEADER внешним ключом, ссылающимся на первичный ключ EMP_NO в таблице EMPLOYEE (табл. 1.12).

Таблица 1.11. *Таблица PROJECT нормализованная*

PROJ_ID	TEAM_LEADER	PROJ_NAME	PROJ_DESC	PRODUCT
DGPII	48	Automap	BLOB data	hardware
VBASE	36	Video	database BLOB data	software
HWRII	24	Translator	upgrade BLOB data	software

Таблица 1.12. *Таблица EMPLOYEE для ссылки по внешнему ключу*

EMP_NO	LAST_NAME	FIRST_NAME	DEPT_NO	JOB_CODE	PHONE_EXT	SALARY
24	Smith	John	100	Eng	4968	64000
48	Carter	Catherine	900	Sales	4967	72500
36	Smith	Jane	600	Admin	4800	37500

Выбор индексов

Индекс (указатель) — это механизм, используемый для ускорения извлечения записей таблицы в ответ на некоторый запрос при некоторых условиях поиска. Это как поиск по указателю в книге: индекс (указатель) содержит номера страниц, связанных с данным термином, что позволяет быстро найти нужную страницу. Индекс базы данных служит логическим указателем на физическое размещение (адрес) строки в таблице. Индекс хранит каждое значение проиндексированной колонки (колонки, по значениям которой предполагается вести поиск в таблице) или колонок с указателями на все дисковые блоки, содержащие строки с таким значением. Когда выполняется запрос, среда InterBase сначала проверяет, существует ли какой-либо индекс для данной таблицы. Затем определяется, будет ли более эффективным просмотреть всю таблицу или использовать существующий индекс, чтобы выполнить запрос. Если среда обработки решит, что надо использовать индекс, она ищет индекс, чтобы найти требуемые ключевые значения, по которым выбирает указатели для поиска строки таблицы, содержащей эти значения. Извлечение данных является быстрым, потому что значения индекса упорядочены и сами они относительно невелики. Это позволяет системе быстро отыскивать ключевые значения. Как только в индексе ключевое значение найдено, система выбирает связанный с этим значением указатель на физическое размещение данных. Использование индексов обычно требует меньшего количества страниц выборки, чем последовательный просмотр каждой строки таблицы. Индекс может быть определен на одной или нескольких колонках таблицы.

Стоит ли проводить индексацию и когда

Время последовательного просмотра строк таблицы для поиска нужной строки пропорционально количеству строк. Индексирование таблицы хотя бы по одной колонке значительно сокращает время поиска. Но есть и проблемы. Во-первых, главная помеха здесь в том, что индексы (а это отдельные файлы) потребляют дополнительное дисковое пространство. Во-вторых, вставка, удаление и обновление данных происходят дольше на индексированных, чем на неиндексированных колонках. Дело в том, что индекс должен каждый раз обновляться, когда изменяется данное в колонке, являющейся индексом, или когда строка таблицы меняет свои координаты: добавляется новая строка или удаляется существующая. Поэтому надо быть аккуратным, задавая сложные составные индексы. Однако преимущества индексов перевешивают их недостатки в процессе извлечения данных.

Создать индексированную колонку в таблице можно, когда:

- условия поиска часто используют эту колонку;
- JOIN-условия часто используют эту колонку (с помощью утверждения JOIN можно извлекать данные из двух и более таблиц при одном операторе SELECT);
- ORDER BY-утверждения оператора SELECT часто используют эту колонку для сортировки данных.

Нельзя создавать индексы для:

- колонок, на которые редко идет ссылка в условиях поиска;
- для часто обновляемых неключевых колонок;
- для колонок, имеющих незначительное количество возможных значений.

Создание индексов

Индексы создаются либо с использованием оператора CREATE INDEX, либо автоматически системой как часть оператора CREATE TABLE. InterBase допускает создание до 64 индексов в таблице. Чтобы создать индекс, надо иметь авторизованный доступ к таблице.

> ***Примечание***
>
> Чтобы увидеть все индексы, определенные в текущей базе данных, надо выполнить isql-команду SHOW INDEX. Чтобы увидеть все индексы, определенные в таблице, надо выполнить команду SHOW INDEX имя_таблицы. Чтобы увидеть информацию о конкретном индексе, надо выполнить команду SHOW INDEX имя_индекса. Но для выполнения всех перечисленных действий можно воспользоваться и утилитой IBConsole (об этом — далее).

InterBase автоматически генерирует индексы, когда таблицы определяются с использованием утверждений PRIMARY KEY, FOREIGN KEY и UNIQUE. Индексы по PRIMARY KEY и FOREIGN KEY сохраняют целостность информации.

Использование CREATE INDEX

Этот оператор создает индекс по одной и более колонкам таблицы. Индексирование одной колонки обеспечивает поиск по данным этой колонки, в то время как индексирование по совокупности колонок обеспечивает комплексный поиск по конъюнкции данных проиндексированных колонок.