Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информационные технологии»

Е. Г. Стародубцев

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

ПОСОБИЕ
по дисциплинам «Базы данных», «Технологии организации, хранения и обработки данных», «Разработка приложений баз данных для информационных систем» для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной и заочной форм обучения

Электронный аналог печатного издания

Гомель 2010

УДК 004.65(075.8)
ББК 30.2-5-05я73
С77

Рекомендовано к изданию научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 7 от 09.03.2009 г.*)

Рецензент: нач. сектора АСУ вычислительного центра ГГТУ им. П. О. Сухого
Н. С. Шестакова

Стародубцев, Е. Г.

С77 Системы управления базами данных : пособие по дисциплинам «Базы данных», «Технологии организации, хранения и обработки данных», «Разработка приложений баз данных для информационных систем» для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» днев. и заоч. форм обучения / Е. Г. Стародубцев. – Гомель : ГГТУ им. П. О. Сухого, 2010. – 30 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://lib.gstu.local. – Загл. с титул. экрана.

ISBN 978-985-420-897-8.

Изложены основы классификации, архитектуры, разработки и использования баз данных, систем управления базами данных (СУБД), информационных систем на основе баз данных. Рассмотрены основные этапы проектирования реляционных баз данных на основе принципов нормализации реляционных таблиц. Приведены примеры построения информационных моделей и разработки баз данных (на примере СУБД MS Access) для различных предметных областей.

Для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной и заочной форм обучения.

УДК 004.65(075.8)
ББК 30.2-5-05я73

ISBN 978-985-420-897-8

1. Информационные системы и базы данных

1.1. Основные понятия

В основе решения многих задач лежит обработка информации, для которой создаются информационные системы (ИС). *Автоматизированными* называют ИС, в которых применяют технические средства, в частности, ЭВМ. Большинство существующих ИС – автоматизированные ИС (АИС), поэтому для краткости будем называть их ИС. В широком понимании под определение ИС подпадает *любая система обработки информации*. Иногда используется более узкая трактовка понятия ИС как *совокупности аппаратно-программных средств, используемых для решения некоторой прикладной задачи*.

По области применения (*предметной области*) ИС делятся на системы, используемые в производстве, образовании, науке, торговле и других отраслях. ***По целевой функции*** ИС можно условно разделить на следующие основные категории: *управляющие*, *информационно-справочные*, *поддержки принятия решений*.

Банк данных – разновидность ИС, в которой реализованы функции *централизованного хранения и накопления обрабатываемой информации, организованной в одну или несколько баз данных*. Банк данных в общем случае состоит из следующих компонентов: *одной или нескольких баз данных*; *системы управления базами данных*; *словаря данных*; *администратора баз данных*; *вычислительной системы*; *обслуживающего персонала*. Кратко рассмотрим названные компоненты и некоторые связанные с ними важные понятия.

База данных (БД) – совокупность специальным образом организованных (*структурированных*) данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их связей в предметной области. Логическую структуру хранимых данных называют ***моделью представления данных*** (***моделью данных***). Основные модели данных: ***иерархическая***, ***сетевая***, ***реляционная*** (самая распространенная в настоящий момент), ***постреляционная***, ***многомерная***, ***объектно-ориентированная***.

Система управления базами данных (СУБД) – комплекс языковых и программных средств, предназначенный для создания, ведения и совместного использования БД многими пользователями.

Приложение – программа или комплекс программ, обеспечивающих автоматизацию обработки информации для прикладной зада-

чи. Далее будут рассматриваться приложения, использующие БД. Приложения могут создаваться как в среде, так и вне среды СУБД – с помощью систем программирования, использующих средства доступа к БД. Приложения, разработанные в среде СУБД, часто называют ***приложениями СУБД***, а приложения, разработанные вне СУБД, – ***внешними приложениями***. Для работы с БД часто достаточно средств СУБД и не нужно использовать приложения, создание которых требует программирования. Приложения разрабатывают главным образом в случаях, когда требуется обеспечить удобство работы с БД неквалифицированным пользователям или интерфейс СУБД не устраивает пользователей.

Словарь данных (СД) – подсистема банка данных, предназначенная для централизованного хранения информации о структурах данных, взаимосвязях файлов БД друг с другом, типах данных и форматах их представления, принадлежности данных пользователям. Термину «словарь данных» часто соответствует термин ***метаданные*** («данные о данных»), подчеркивающий назначение СД – описать «устройство БД» до занесения конкретных «просто данных» выбранной предметной области. Функционально СД присутствует во всех банках данных, но не всегда выполняющий эти функции компонент имеет именно такое название. Чаще всего функции СД выполняются СУБД и вызываются из основного меню системы или реализуются с помощью ее утилит.

Администратор баз данных (АБД) – лицо или группа лиц, отвечающих за выработку требований к БД, ее проектирование, создание, эффективное использование и сопровождение. В процессе эксплуатации АБД следит за функционированием ИС, обеспечивает защиту данных от несанкционированного доступа, контролирует правильность хранимой в БД информации. Для *однопользовательских ИС* функции АБД обычно возлагаются на лиц, непосредственно работающих с приложением БД.

Вычислительная система – взаимосвязанные и согласованно действующие ЭВМ или другие устройства, обеспечивающие автоматизацию процессов приема, обработки и выдачи информации потребителям. Вычислительная система должна иметь достаточный объем оперативной и внешней памяти, приемлемую мощность центральных процессоров для хранения и обработки данных ИС.

Обслуживающий персонал выполняет функции поддержания технических и программных средств банка данных в рабочем состоянии; проводит профилактические, регламентные, восстановительные и другие работы.

1.2. Предпосылки появления баз данных и систем управления базами данных

Массивы информации для хранения и обработки в ИС необходимо оптимально организовывать, обеспечивать правильность данных. Эффективное решение этих задач нельзя выполнить, используя только функции стандартных файловых систем.

К недостаткам файловой организации данных на внешних носителях при использовании ИС можно отнести:

• ***Изолированность и разделенность данных***. Операционная система контролирует и разграничивает доступ к данным, как правило, *на уровне файлов*. Поэтому несколько параллельно работающих приложений не смогут одновременно обновлять различные данные в одном и том же файле. Совместная обработка нескольких файлов также будет сложной.

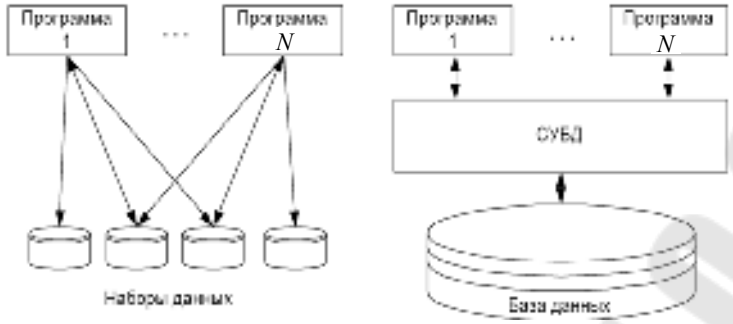
• ***Зависимость программ от данных***. Если описание структуры данных задается в прикладной программе, то любое внесение изменений в эту структуру требует, как минимум, перекомпиляции всех программ, использующих этот файл.

• ***Дублирование данных***. Если разные приложения используют данные, относящиеся к одному объекту, но хранят эти данные в своих независимых файлах, то возникает неконтролируемое *дублирование данных*. Это ведет к излишнему расходу памяти и возможной противоречивости данных: данные, измененные в одном файле, в другом файле могут остаться в прежнем виде.

• ***Отсутствие описаний данных***. В файлах операционной системы данные, обрабатываемые прикладными программами, хранятся без описания. Это создает сложности при документировании ИС, затрудняет поиск нужных данных и может вести к ошибкам.

Для устранения этих недостатков было предложено *отделить процесс хранения данных от процесса их обработки*, выделив специальную «программу-посредник» – СУБД, а сами данные организовать и хранить в виде определенной структуры – БД. Понятия «база данных» *(database – DB)* и «СУБД» (*Database Management System – DBMS*) появились в середине 1960-х гг., когда ЭВМ начали активно использовать в управлении производством. Программисты стали переходить от решения отдельных задач к комплексной автоматизации управления, *увязывая все задачи в единое целое общей целью*. Раньше приходилось *вводить и выводить одни и те же данные по многу раз*. Например, при управлении предприятием персональные сведения о работниках используются в разных задачах (разных отделах): кадрового учета, на-

числения зарплаты, планирования и т. д. Лучше ввести эти данные один раз, создав БД, и предоставить всем прикладным программам право получать необходимые сведения из этой БД (рис. 1).



а) *б*)

Рис. 1. Взаимодействие программ с данными
при независимой работе (*а*)
и при использовании СУБД (*б*)

Преимущества использования БД и СУБД: ***однократный ввод данных***, ведущий к сокращению затрат труда; ***независимость программ от данных***, что дает возможность независимо менять те и другие; ***сокращение затрат на программирование***; так как многие операции с данными стандартные (ввод, поиск, защита и т. д.), то вместо программирования их каждый раз заново всю работу с данными выносят в одну большую и сложную программу – СУБД.

2. Виды архитектуры информационных систем на основе баз данных. Физическая, логическая, концептуальная организация баз данных

Эффективность функционирования ИС во многом зависит от ее архитектуры. В настоящее время перспективной является архитектура *клиент-сервер*. В распространенном варианте она предполагает наличие компьютерной сети и ***распределенной базы данных***, включающей ***корпоративную базу данных*** (КБД) и ***персональные базы данных*** (ПБД). КБД размещается на компьютере-сервере, ПБД размещаются на компьютерах сотрудников подразделений, являющихся клиентами КБД.

Сервером определенного ресурса в компьютерной сети называется компьютер (или программа), управляющий этим ресурсом, **клиентом** – компьютер (программа), использующий этот ресурс. В качестве ресурса могут выступать, например, БД, файловые системы, службы печати, почтовые службы. Тип сервера определяется видом ресурса, ко-

торым он управляет. Например, если управляемым ресурсом является БД, то соответствующий сервер называется ***сервером БД***.

Достоинством организации ИС по архитектуре клиент-сервер является удачное сочетание *централизованного* хранения, обслуживания и *коллективного* доступа к общей корпоративной информации с *индивидуальной* работой пользователей над персональной информацией.

Архитектура клиент-сервер допускает различные варианты реализации. Исторически первыми появились *распределенные ИС с применением файл-сервера*. В таких ИС по запросам пользователей файлы БД передаются на персональные компьютеры (ПК) клиентов, где и производится их обработка. Недостаток такого варианта архитектуры – *высокая интенсивность передачи обрабатываемых данных*. Причем часто передаются *избыточные данные*: вне зависимости от объема данных, требуемых пользователю, файлы БД *передаются целиком*.

При архитектуре *клиент-сервер с использованием сервера БД* сервер БД выполняет основной объем обработки данных. Формируемые пользователем или приложением запросы поступают к серверу БД в виде инструкций специального языка запросов. Сервер БД выполняет поиск и извлечение нужных данных, которые затем передаются на компьютер пользователя. *Достоинствами* такого подхода в сравнении с предыдущим являются: меньший объем передаваемых данных и более быстрая обработка данных, так как основная обработка данных выполняется на более мощном компьютере-сервере (по сравнению с ПК клиента).

Для создания и управления ПБД и приложений, работающих с ними, используются различные СУБД, например, Access и Visual FoxPro фирмы Microsoft, Paradox фирмы Borland. КБД создается, поддерживается и функционирует под управлением сервера БД, например, Microsoft SQL Server или Oracle Server.

В зависимости от размеров организации и особенностей решаемых задач ИС может иметь одну из следующих конфигураций:

– ***компьютер-сервер, содержащий КБД и ПБД***;

– ***компьютер-сервер и ПК с ПБД***;

– ***несколько компьютеров-серверов и ПК с ПБД***.

Использование архитектуры клиент-сервер дает возможность постепенного наращивания ИС предприятия, во-первых, по мере развития предприятия, во-вторых, по мере развития самой ИС. Разделение общей БД на КБД и ПБД позволяет уменьшить сложность проектирования БД по сравнению с централизованным вариантом, а значит, снизить вероятность ошибок и стоимость проектирования.

Важнейшее достоинство применения БД в ИС – *обеспечение независимости данных от прикладных программ*. Это дает возможность пользователям не заниматься проблемами представления данных на физическом уровне: размещения данных в памяти, методов доступа к ним и т. д. Такая независимость достигается поддерживаемым СУБД **многоуровневым представлением данных** в БД на ***логическом*** (пользовательском) и ***физическом уровнях***. Благодаря СУБД и наличию логического уровня представления данных обеспечивается отделение **концептуальной** (***понятийной***) модели БД от ее ***физического представления*** в памяти ЭВМ.

Согласно требованиям стандартной архитектуры ANSI/SPARC, выделяются **три уровня архитектуры БД** (рис. 2).

Рис. 2. Три уровня архитектуры БД

• **Внутренний уровень** (также называемый **физическим**) наиболее близок к *физическому хранилищу информации*, т. е. связан со способами сохранения информации на физических устройствах.

• **Внешний уровень** (также называемый **пользовательским логическим**) наиболее близок к пользователям, т. е. связан со способами представления данных *для отдельных пользователей БД*; при этом *каждому пользователю* может соответствовать свое *внешнее представление данных*. На этом уровне выполняется описание БД в терминах принятой логической модели данных, зависящей от выбора СУБД.

• **Концептуальный уровень** (также называемый **общим логическим** или просто **логическим**) является «промежуточным» уровнем между двумя первыми и связан с *обобщенным представлением данных* для *всех пользователей БД*. Этот уровень характеризуется моделью предметной области (*инфологической моделью*), независимой от выбора СУБД и физических параметров среды хранения данных.

3. Классификация и функции систем управления базами данных

3.1. Признаки классификации и виды систем управления базами данных. Аппаратное и программное обеспечение систем управления базами данных

В качестве основных ***классификационных признаков СУБД*** можно использовать следующие: ***вид программы***, ***характер использования***, ***модель данных***. Эти признаки влияют на выбор СУБД и эффективность работы ИС. В общем случае под СУБД можно понимать любой программный продукт, поддерживающий процессы создания, ведения и использования БД. Рассмотрим, какие из программ имеют отношение к БД и в какой мере они связаны с БД.

К СУБД относятся следующие основные виды программ: ***полнофункциональные СУБД***; ***серверы БД***; ***клиенты БД***; ***средства разработки программ работы с БД***.

Полнофункциональные СУБД (ПФСУБД) – «традиционные» СУБД, которые сначала появились для больших машин, затем для мини-ЭВМ и ПЭВМ. Из всех СУБД современные ПФСУБД являются наиболее многочисленными и реализуют наиболее полный набор функций. К ПФСУБД относятся, например, Clarion Database Developer, DataEase, DataFlex, Microsoft Access, Microsoft FoxPro. Обычно ПФСУБД имеют *графический интерфейс*, позволяющий выполнять основные действия с БД: создавать (изменять) структуру таблиц, вводить данные, формировать запросы на обработку данных, разрабатывать экранные формы для работы пользователя ИС, отчеты по результатам обработки данных и т. п. Многие ПФСУБД включают также средства программирования. Некоторые системы имеют в качестве вспомогательных и дополнительные средства проектирования схем БД или CASE-подсистемы.

Серверы БД предназначены для организации центров обработки данных в сетях ЭВМ. Эта группа БД в настоящее время менее многочисленна, но их количество постепенно растет. Серверы БД реализуют функции управления БД, запрашиваемые другими (клиентскими) программами обычно с помощью операторов SQL. Примерами серверов БД являются: NetWare SQL (Novell), MS SQL Server (Microsoft), InterBase (Borland), Intelligent Database (Ingress). В роли ***клиентских программ*** для серверов БД в общем случае могут использоваться различные программы: ПФСУБД, электронные таблицы, текстовые процессоры и т. д.

Средства разработки программ работы с БД могут использоваться для создания разновидностей следующих программ:

– ***клиентских программ***;

– ***серверов БД и их отдельных компонентов***;

– ***пользовательских приложений***.

Программы первого и второго вида относительно малочисленны, т. к. предназначены, главным образом, для системных программистов. Пакетов третьего вида гораздо больше, но меньше, чем ПФСУБД.

К средствам разработки пользовательских приложений относятся системы программирования, например, система Clipper, библиотеки программ для различных языков программирования, пакеты автоматизации разработок (в том числе систем типа клиент-сервер). К наиболее распространенным можно отнести инструментальные системы: Delphi и Power Builder (Borland), Visual Basic (Microsoft), SILVERRUN (Computer Advisers Inc.), S-Designor (SDP и Powersoft). Кроме перечисленных средств для управления данными и организации обслуживания БД используются различные дополнительные средства, например *мониторы транзакций*.

По характеру использования СУБД относятся к **персональным** и **многопользовательским**.

Персональные СУБД обычно обеспечивают возможность создания ПБД и недорогих приложений, работающих с ними. Персональные СУБД или разработанные с их помощью приложения зачастую могут выступать в роли клиентской части многопользовательской СУБД. К персональным СУБД, например, относятся Visual FoxPro, Paradox, Clipper, Access и др.

Многопользовательские СУБД включают сервер БД и клиентскую часть и, как правило, могут работать в *неоднородной вычислительной среде* (с разными типами ЭВМ, операционными системами). К таким СУБД относятся, например, СУБД Oracle, Informix.

По используемой модели данных СУБД (как и БД) разделяют на иерархические, сетевые, реляционные и другие типы. Некоторые СУБД могут одновременно поддерживать несколько моделей данных.

3.2. Функции, выполняемые различными видами систем управления базами данных

СУБД обеспечивают выполнение ряда ***базовых функций***, к которым относятся следующие.

Описание логической структуры базы данных. Для описания данных в СУБД имеется специальный ***язык описания данных*** –

ЯОД (***Data Description Language – DDL***). При этом описание данных возможно *с двух точек зрения*:

– с точки зрения АБД, владеющего всей информацией обо всех данных, хранящихся в БД; это описание называется ***схемой БД***;

– с точки зрения *некоторой конкретной задачи*, решаемой над СУБД, и нуждающейся *только в части данных*; такое описание называется ***подсхемой***.

Манипулирование данными – выполнение всех операций с данными – ввод, проверка правильности, выборка, составление отчетов. Для описания операций с данными в СУБД имеется ***язык манипулирования данными – ЯМД*** (***Data Manipulating Language – DML***). Этот язык может быть реализован в двух вариантах:

– в виде расширения базового языка программирования (например, Кобола, Паскаля, Си) набором библиотечных функций;

– в виде самостоятельного ЯМД, который в этом случае называется *языком запросов*.

ЯОД и ЯМД в различных СУБД могут иметь отличия. Наибольшее распространение получили два стандартизованных языка: ***QBE*** (Query By Example) – язык запросов по образцу и ***SQL*** (Structured Query Language) – структурированный язык запросов. QBE в основном обладает свойствами ***языка манипулирования данными***, SQL сочетает в себе свойства языков обоих типов – ***описания*** и ***манипулирования данными***.

Обеспечение целостности БД. ***Целостность*** (непротиворечивость) ***данных*** – это способность данных правильно описывать объекты предметной области. Нарушения целостности могут быть из-за ошибок человека или из-за машинных сбоев. Обеспечение целостности данных – сложная задача. В частности, для защиты от машинных сбоев в хороших СУБД ведутся журналы учета всех обращений пользователей к БД, по которым можно восстановить данные при авариях.

Обеспечение многопользовательского доступа. Иногда с одной БД одновременно работают сотни и тысячи пользователей, например, в системах резервирования билетов, крупных банках. СУБД должна «навести порядок» в многочисленных обращениях, обеспечить в этих условиях сохранение целостности БД.

Защита данных от несанкционированного доступа. Могут защищаться отдельные поля, записи, блоки данных, для чего используются разнообразные ключи, пароли, шифры и т. п.

Перечисленные выше базовые функции СУБД, в свою очередь, используют следующие ***низкоуровневые функции***:

– ***управление данными во внешней памяти***;

– ***управление буферами оперативной памяти***;

– ***управление транзакциями***;

– ***ведение журнала изменений в БД***;

– ***обеспечение целостности и безопасности БД***.

Рассмотрим необходимость и особенности реализации низкоуровневых функций в современных СУБД различных видов.

Реализация функции ***управления данными во внешней памяти*** в разных СУБД может различаться и на уровне управления ресурсами (используя файловые системы или управление устройствами ПЭВМ), и по логике самих алгоритмов управления данными. В основном, методы и алгоритмы управления данными являются «внутренним делом» СУБД и прямого отношения к пользователю не имеют. Качество реализации этой функции наиболее сильно влияет на эффективность работы специфических ИС, например, с большими БД и объемами обработки данных, со сложными запросами.

Необходимость *буферизации данных* и, как следствие, реализации ***функции управления буферами оперативной памяти*** обусловлена тем, что объем оперативной памяти меньше объема внешней памяти. ***Буферы*** – области оперативной памяти, предназначенные для ускорения обмена между внешней и оперативной памятью. В буферах временно хранятся фрагменты БД, данные из которых предполагается использовать при обращении к СУБД или планируется записать в БД после обработки.

Механизм **транзакций** используется в СУБД для поддержания целостности данных в базе. **Транзакцией** называется некоторая неделимая последовательность операций над данными БД, которая отслеживается СУБД от начала и до завершения. Если по каким-либо причинам (сбои и отказы оборудования, ошибки в программном обеспечении, включая приложение) транзакция остается незавершенной, то она отменяется.

Транзакции имеют три основных свойства: ***атомарность*** (выполняются все входящие в транзакцию операции или ни одна из операций); ***сериализуемость*** (отсутствует взаимное влияние выполняемых в одно и то же время транзакций); ***долговечность*** (даже системные ошибки не должны приводить к утрате результатов зафиксированной транзакции).

Примером транзакции является операция перевода денег с одного счета на другой в банковской ИС. Здесь необходим, по крайней мере, двухшаговый процесс. Сначала снимают деньги с одного счета, затем добавляют их к другому счету. Если хотя бы одно из действий не выполнится успешно, результат операции окажется неверным и будет нарушен баланс между счетами.

Контроль транзакций важен в однопользовательских и в многопользовательских СУБД, где транзакции могут быть запущены параллельно. В последнем случае говорят о ***сериализуемости транзакций***. Под *сериализацией* параллельно выполняемых транзакций понимается составление такого плана их выполнения (*сериального плана*), при котором суммарный эффект реализации транзакций эквивалентен эффекту их последовательного выполнения.

При параллельном выполнении смеси транзакций возможны конфликты (***блокировки***), разрешение которых является функцией СУБД. При обнаружении таких случаев обычно производится «откат» путем отмены изменений, произведенных транзакциями.

Ведение журнала изменений в БД (*журнализация изменений*) выполняется СУБД для обеспечения надежности хранения данных в БД при наличии аппаратных сбоев и отказов, а также ошибок в программном обеспечении. Журнал СУБД – это особая БД или часть основной БД, непосредственно недоступная пользователю и используемая для записи информации обо всех изменениях БД. В различных СУБД в журнал могут заноситься разные объемы данных. Для эффективной реализации функции ведения журнала изменений в БД необходимо обеспечить повышенную надежность хранения и поддержания в рабочем состоянии самого журнала.

Обеспечение целостности БД составляет необходимое условие успешного функционирования БД, особенно для случая использования БД в сетях. Поддержание целостности БД включает проверку целостности и ее восстановление в случае обнаружения противоречий в БД. Целостное состояние БД описывается с помощью *ограничений целостности* в виде условий, которым должны удовлетворять хранимые данные. Примеры таких условий: ограничение диапазонов возможных значений данных; отсутствие повторяющихся записей в таблицах реляционных БД.

Обеспечение безопасности достигается в СУБД шифрованием прикладных программ, данных, защитой паролем, поддержкой уровней доступа к БД и к отдельным ее элементам (таблицам, формам и т. д.).

4. Разработчики и пользователи информационных систем на основе систем управления базами данных

Пользователей (включая разработчиков) БД можно разделить на три большие и отчасти перекрывающиеся группы.

1. ***Прикладные программисты***, которые отвечают за написание прикладных программ, использующих БД. Для этих целей применимы такие языки, как COBOL, PL/I, C++, Java или какой-нибудь высокоуровневый язык четвертого поколения. Прикладные программы получают доступ к БД посредством выдачи соответствующего запроса к СУБД (обычно это некоторый SQL-оператор). Подобные программы могут быть простыми пакетными приложениями или же интерактивными приложениями, предназначенными для поддержки работы конечных пользователей. В последнем случае они предоставляют пользователям непосредственный оперативный доступ к БД через рабочую станцию или терминал. Большинство современных приложений относится именно к этой категории.

2. ***Конечные пользователи***, работающие с СУБД непосредственно через рабочую станцию или терминал. Конечный пользователь может получать доступ к БД, применяя одно из интерактивных приложений или интерфейс, интегрированный в программное обеспечение самой СУБД. Большинство СУБД включает, по крайней мере, одно такое встроенное приложение, а именно – процессор языка запросов, позволяющий пользователю в диалоговом режиме вводить запросы к БД. Язык SQL – типичный пример языка запросов БД.

Кроме языка запросов в большинстве СУБД дополнительно предоставляются специализированные графические интерфейсы, в которых пользователь в явном виде не использует язык запросов. Работа с БД осуществляется за счет выбора пользователем необходимых команд меню или заполнения требуемых полей в предоставленных формах. Такие интерфейсы, основанные на меню и экранных формах, облегчают работу с БД непрофессиональных пользователей. Командный интерфейс, т. е. язык запросов, напротив, требует некоторого профессионального опыта (но не такого большого, какой необходим для написания прикладных программ на языке программирования). Однако командный интерфейс более гибок, чем некомандный, к тому же языки запросов обычно включают определенные функции, отсутствующие в некомандных интерфейсах.

3. ***Администратор данных*** (АД) и ***АБД***. АД – специалист, несущий основную ответственность за данные ИС. Считается, что АД

должен разбираться во всех данных предприятия и понимать нужды предприятия по отношению к данным *на уровне высшего управляющего звена* в руководстве предприятием и относится к этому звену (часто эта функция возложена на начальника или заместителя начальника вычислительного центра больших организаций). АД принимает решения о том, какие данные вносятся в БД в первую очередь, вырабатывает требования по сопровождению, обработке, защите данных после их занесения в БД.

АБД – *технические* специалисты, реализующие решения АД. АБД создают БД и организуют их сопровождение, отвечают за обеспечение быстродействия и технического обслуживания ИС на основе БД. Для больших организаций функции АБД выполняются несколькими специалистами, включая системных программистов, техников и т. д.

Кроме того, разработчиками БД предприятия (ИС на основе БД) могут быть внешние организации, разрабатывающие программное обеспечение и передающие его после внедрения на сопровождение АБД предприятия.

5. Принципы разработки и выполнения приложений при работе с базами данных

Современные СУБД позволяют решать широкий круг задач по работе с БД *без разработки приложения*. Иногда целесообразно *разработать приложение*. Например, если требуется автоматизация работы с данными, интерфейс СУБД недостаточно развит, либо стандартные функции СУБД по обработке информации не устраивают пользователя. Для разработки приложений СУБД должна иметь программный интерфейс, основу которого составляют функции и/или процедуры соответствующего языка программирования.

Существующие СУБД поддерживают следующие ***технологии*** (и их комбинации) ***разработки приложений***:

– ***ручное кодирование программ*** (например, Clipper, FoxPro, Paradox);

– ***создание текстов приложений с помощью генераторов*** (FoxApp в FoxPro, Personal Programmer в Paradox);

– ***автоматическая генерация готового приложения методами визуального программирования*** (Access, Paradox for Windows).

При *ручном кодировании* текст программ приложений набирается вручную, после чего выполняется отладка.

Использование генераторов упрощает разработку приложений, поскольку при этом можно получать программный код *без ручного набора*. Генераторы приложений облегчают разработку основных элементов приложений (меню, экранных форм и т. д.), но часто не исключают полностью ручное кодирование.

Средства визуального программирования приложений являются дальнейшим развитием идеи использования генераторов приложений. Приложение при этом «строится» из готовых «строительных блоков» с помощью удобной интегрированной среды. При необходимости разработчик легко может вставить в приложение свой код. Интегрированная среда имеет развитые средства создания, отладки и модификации приложений. Визуальное программирование позволяет быстрее создавать более эффективные приложения по сравнению с приложениями, полученными первыми двумя способами.

Разработанное приложение обычно состоит из одного или нескольких файлов операционной системы. Если основным файлом приложения является исполняемый файл (например, exe-файл), то это приложение, скорее всего, является ***независимым приложением***, выполняемым автономно от среды СУБД. Получение независимого приложения осуществляется путем ***компиляции*** исходных текстов программ, созданных разными способами: набором программ вручную, с помощью генератора приложения или среды визуального программирования.

Независимые приложения позволяют получать, например, СУБД FoxPro и система визуального программирования Delphi. Отметим, что с помощью средств Delphi обычно независимые приложения не разрабатывают, т. к. это достаточно трудоемкий процесс, а привлекают процессор баз данных BDE (Borland DataBase Engine), играющий роль ядра СУБД. Одним из первых средств разработки приложений для ПЭВМ является система Clipper, представляющая собой «чистый компилятор».

Во многих случаях приложение *не может исполняться без среды СУБД*. Выполнение приложения состоит в том, что СУБД анализирует файлы приложения (в частном случае – текст исходной программы) и автоматически строит необходимые исполняемые машинные команды. Другими словами, приложение выполняется ***методом интерпретации***.

Режим интерпретации реализован во многих современных СУБД, например Access, Visual FoxPro и Paradox, а также в СУБД недавнего прошлого, к примеру, FoxBase и FoxPro.

Кроме этого существуют системы, использующие промежуточный вариант между компиляцией и интерпретацией – так называемую ***псевдокомпиляцию***. В таких системах исходная программа путем компиляции преобразуется в промежуточный код (*псевдокод*) и записывается на диск. В этом виде ее в некоторых системах разрешается даже редактировать, но главная цель псевдокомпиляции – *преобразовать программу к виду, ускоряющему процесс ее интерпретации*. Такой прием широко применялся в СУБД, работающих под управлением DOS.

В СУБД, работающих под управлением Windows, псевдокод чаще используют для запрета модификации приложения. Это полезно для защиты от случайного или преднамеренного повреждения работающей программы. Например, такой прием применен в СУБД Paradox for Windows, где допускается разработанные экранные формы и отчеты преобразовывать в нередактируемые объекты.

Некоторые СУБД предоставляют пользователю возможность выбора варианта разработки приложения: как интерпретируемого СУБД программного кода или как независимой программы.

Преимущество применения независимых приложений – время выполнения машинной программы обычно меньше, чем при интерпретации. Такие приложения используют на слабых машинах и при установке систем «под ключ», когда приложение «закрывается» от доработок со стороны пользователей.

Преимущества применения интерпретируемых приложений:

• *Легкость их модификации*. Если готовая программа подвергается частым изменениям, то для их внесения нужна инструментальная система, т. е. СУБД или аналогичная среда. Для интерпретируемых приложений такой инструмент всегда под рукой, что очень удобно.

• *Наличие* (как правило) *мощных средств контроля целостности данных и защиты от несанкционированного доступа*, в отличие от систем компилирующего типа, где данные функции программируют вручную либо контролируются АБД.

При выборе средств для разработки приложения учитываются три основных фактора: ***ресурсы компьютера***; ***особенности приложения*** (потребность в модификации функций программы, время на разработку, необходимость контроля доступа и поддержание целостности информации); ***цель разработки*** (отчуждаемый программный продукт или система автоматизации своей повседневной деятельности).

Для пользователя современного ПК, создающего несложное приложение, больше подходит СУБД интерпретирующего типа. Такие сис-

темы достаточно мощны, удобны для разработки, отладки, сопровождения и модификации приложения. При использовании «слабого» ПК лучше выбрать систему со средствами разработки независимых приложений. При этом важно, что малейшее изменение в приложении вызывает повторение этапов программирования, компиляции и отладки программы. Как правило, разница в выполнении независимого приложения и приложения в режиме интерпретации – доли секунды в пользу независимого приложения. В то же время разница во времени подготовки приложения к его использованию обычно составляет величины порядка минуты-часы в пользу систем с интерпретацией.

6. Реляционная модель данных

6.1. Реляционные базы данных

Ядром любой БД является ***модель данных*** – *совокупность структур данных и операций их обработки*. Далее рассмотрим БД на основе одной из наиболее распространенных моделей – ***реляционной модели данных*** (РМД). В РМД данные организованы в виде *двумерных таблиц* (***реляционных таблиц***) или ***отношений*** (*relations*). РМД некоторой предметной области представляет собой *набор отношений, изменяющихся во времени*. При создании ИС совокупность отношений позволяет хранить данные об объектах предметной области и моделировать связи между ними.

Каждая реляционная таблица представляет собой двумерный массив и обладает следующими свойствами:

– *каждый элемент таблицы – один элемент данных*;

– *все столбцы в таблице однородные*, т. е. все элементы в столбце имеют одинаковый тип данных и длину;

– *каждый столбец имеет уникальное имя*;

– *порядок следования строк и столбцов может быть произвольным*.

Строки (записи) реляционной таблицы соответствуют ***кортежам***, а столбцы (поля) – ***атрибутам*** с наборами ***значений атрибутов***. Поле, каждое значение которого однозначно определяет (идентифицирует) соответствующую запись, называется ***простым ключом*** (ключевым полем). Если записи идентифицируются значениями нескольких полей, то таблица имеет ***составной ключ***.

Элементы РМД и формы их представления в СУБД приведены в табл. 1.

Таблица 1

Элементы реляционной модели

Элемент реляционной модели	Форма представления
Отношение	Таблица
Схема отношения	Строка заголовков столбцов таблицы (заголовок таблицы)
Кортеж	Строка таблицы
Сущность	Описание свойств объекта
Атрибут	Заголовок столбца таблицы
Домен	Множество допустимых значений атрибута
Значение атрибута	Значение поля в записи
Первичный ключ	Один или несколько атрибутов
Тип данных	Тип значений элементов таблицы

На рис. 3 дан пример представления отношения **СТУДЕНТ** в РМД.

Рис. 3. Представление отношения СТУДЕНТ в РМД

В РМД также используются следующие основные термины.

Сущность – объект любой природы, данные о котором хранятся в БД. Данные о сущности хранятся в отношении. ***Атрибуты*** – свойства, характеризующие сущность. В структуре таблицы каждый атрибут именуется и ему соответствует заголовок некоторого столбца таблицы. Математически отношение можно описать следующим образом. Пусть даны n множеств D_1, D_2, ..., D_n, тогда отношение R есть множество упорядоченных ***кортежей*** $<d_1, d_2, ..., d_n>$, где $d_k \in D_k$, d_k – ***атрибут***, а D_k – ***домен*** отношения R.

В общем случае порядок кортежей в отношении, как и в любом множестве, не определен. Однако в реляционных СУБД для удобства кортежи все же упорядочивают. Чаще всего для этого выбирают некоторый атрибут, по которому система автоматически сортирует кортежи по возрастанию или убыванию. Если пользователь не назначает ат-

рибута упорядочения, система автоматически присваивает номер кортежам в порядке их ввода. Формально, если переставить атрибуты в отношении (переставить столбцы в таблице, описывающей отношение), то получается новое отношение. Однако в реляционных БД *перестановка атрибутов не приводит к образованию нового отношения*.

Домен – множество всех возможных значений определенного атрибута отношения. Например, отношение **СТУДЕНТ** (рис. 3) включает 5 доменов. Домен 1 содержит номера всех студентов (например, номера личных дел, хранящихся в деканате), домен 2 – фамилии студентов, домен 3 – имена студентов, домен 4 – сокращенные наименования факультетов, домен 5 – даты рождения студентов. Каждый домен образует значения одного типа данных, например, числовые или символьные. Отношение **СТУДЕНТ** содержит 3 кортежа. Кортеж рассматриваемого отношения состоит из 5 элементов, каждый из которых выбирается из соответствующего домена. Каждому кортежу соответствует строка таблицы (рис. 3).

Схема отношения (***заголовок отношения***) – список имен атрибутов. Например, для приведенного примера схема отношения имеет вид

СТУДЕНТ (*Номер*, *Фамилия*, *Имя*, *Факультет*, *Дата_рождения*).

Первичным ключом (**ключом отношения**, **ключевым атрибутом**) называется атрибут отношения, идентифицирующий каждый из его кортежей. Например, в отношении **СТУДЕНТ** ключевым является атрибут *Номер* (если номера личных дел студентов не совпадают). Ключ может быть ***составным*** (***сложным***), т. е. состоять из нескольких атрибутов. Каждое отношение обязательно имеет комбинацию атрибутов, которая может служить ключом. Ее существование гарантируется тем, что отношение – это множество, которое не содержит одинаковых элементов – кортежей. Если в отношении нет повторяющихся кортежей, то, по крайней мере, вся совокупность атрибутов обладает свойством однозначной идентификации кортежей отношения. Во многих СУБД допускается создавать отношения, не определяя ключи.

Отношение может иметь несколько комбинаций атрибутов, каждая из которых идентифицирует все кортежи отношения. Все эти комбинации атрибутов являются ***возможными ключами*** отношения. Любой из возможных ключей может быть выбран как первичный. Если выбранный первичный ключ состоит из минимально необходимого набора атрибутов, говорят, что он является ***не избыточным***.

Ключи обычно используют для достижения следующих целей: 1) исключения дублирования значений в ключевых атрибутах (остальные атрибуты в расчет не принимаются); 2) упорядочения кортежей (возможны различные виды упорядочения); 3) ускорения работы с кортежами отношения; 4) организации связывания таблиц.

Пусть в отношении *R*1 имеется *неключевой* атрибут *A*, значения которого являются значениями ключевого атрибута *B* другого отношения *R*2. Тогда говорят, что атрибут *A* отношения *R*1 есть **внешний ключ**. С помощью внешних ключей устанавливаются связи между отношениями. Например, имеются отношения: **СТУДЕНТ** (рис. 3) и **ПРЕДМЕТ** (*Название*, *Семестр*, *Часы*), связанные отношением **СТУДЕНТ_ПРЕДМЕТ** (*Номер*, *Название*, *Оценка*) (рис. 4).

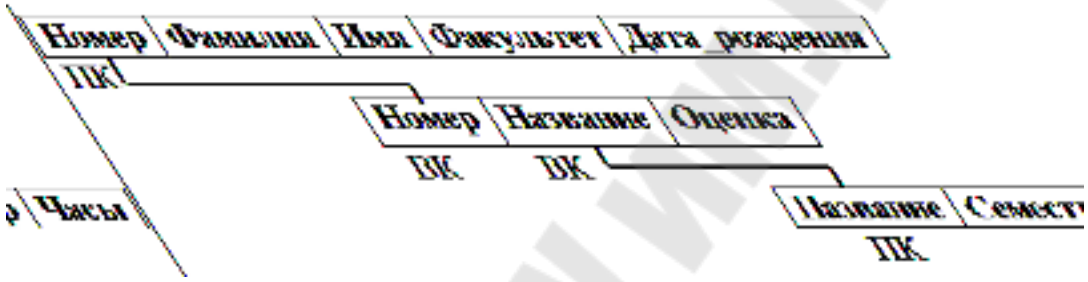


Рис. 4. Связь отношений

Атрибуты *Номер* и *Название* отношений **СТУДЕНТ** и **ПРЕДМЕТ** – первичные ключи (обозначены ПК на рис. 4), аналогичные атрибуты отношения **СТУДЕНТ_ПРЕДМЕТ** являются внешними ключами (обозначены ВК).

Иногда наряду с понятием ***сущность*** используют понятие ***информационный объект*** (ИО) – описание некоторого реального объекта, процесса набором логически связанных ***реквизитов*** (*информационных элементов*). Например, ИО **Студент** имеет реквизиты: Номер личного дела, Фамилия, Имя и т. д. ИО имеет множество реализаций – *экземпляров*, каждый из которых представлен набором конкретных значений реквизитов и определяется значением ключа (простого – один реквизит или составного – несколько реквизитов). ИО может иметь несколько ключей. Все ИО (сущности) из некоторой предметной области связаны между собой. Различают связи трех типов: **один к одному** (**1 : 1**); **один ко многим** (**1 : ∞** или **1 : M**); **многие ко многим** (**∞ : ∞** или **M : M**).

Связь **1 : 1** предполагает, что в каждый момент времени одному экземпляру ИО *A* соответствует не более одного экземпляра ИО *B* и наоборот. При этом каждая запись (строка) в таблице, характеризую-

щей объект *A*, соответствует только одной записи (строке) в таблице, характеризующей объект *B*. Пример: связь между ИО **Студент** и **Посещения занятий**, если каждый студент имеет определенный уникальный набор посещенных занятий по разным предметам за семестр.

При связи **1 : ∞** одному экземпляру ИО *A* соответствует 0, 1 или более экземпляров объекта *B*, но каждый экземпляр объекта В связан не более чем с одним экземпляром объекта *A*. При этом каждая запись в таблице, описывающей объект *A*, соответствует многим записям в таблице, описывающей объект *B*. Примером связи **1 : ∞** служит связь между ИО **Заработная плата** и **Сотрудники**, когда установленный размер заработной платы может повторяться многократно для различных сотрудников.

Связь **∞ : ∞** предполагает, что в каждый момент времени одному экземпляру ИО *A* соответствует 0, 1 или более экземпляров объекта *B* и наоборот. Пример такой связи – связь между ИО **Студент** и **Преподаватель**, когда один студент обучается у многих преподавателей, а один преподаватель обучает многих студентов. Связь **∞ : ∞** не реализуется в реляционных БД непосредственно (для двух таблиц). Для организации такой связи используется промежуточная (третья) таблица-связка для «замены» одной связи **∞:∞** на «сумму» двух связей **∞ : 1** и **1 : ∞**.

Связи между реляционными таблицами устанавливаются ***при помощи совпадающих значений связующих полей***. Например, таблицы, соответствующие отношениям **СТУДЕНТ** и **ПРЕДМЕТ** (рис. 4), связаны по полям *Номер* и *Название* с таблицей, характеризующей отношение **СТУДЕНТ_ПРЕДМЕТ**. При этом реализуются две связи типа **1 : ∞** (при наличии одинаковых значений номеров и названий предметов в соответствующих парах таблиц), т. к. одному студенту соответствует набор оценок по разным предметам, а одному предмету – группа студентов.

6.2. Нормализация реляционных таблиц

Одни и те же данные могут группироваться в таблицы различными способами. Группировка полей в таблицах должна быть рациональной, что означает сведение к минимуму дублирования данных и упрощение процедуры их обработки и обновления.

Нормализация таблиц – это формальный аппарат ограничений на формирование таблиц, который позволяет устранить дублирование данных, обеспечивает непротиворечивость хранимых данных, уменьшает трудозатраты на ведение БД. При практическом проектировании реляционных БД обычно используют *три нормальные формы таблиц*.

Таблица называется приведенной к ***первой нормальной форме*** (1НФ), если все ее поля ***простые*** (далее неделимы). Преобразование таблицы к 1НФ может привести к увеличению количества полей таблицы и изменению ключа. 1НФ предписывает, что все данные, содержащиеся в столбцах реляционной таблицы, должны быть ***атомарными*** или неделимыми. Согласно 1НФ, в каждой позиции (клетке таблицы), определяемой строкой и столбцом, должна быть только одна величина, но ***не массив или список величин***. Рассмотрим пример таблицы (рис. 5), *нарушающей* требования 1НФ (звездочкой будем обозначать ключевое поле).

Заказы1

КодЗаказа*	КодПокупателя	СодержаниеЗаказа
1	4	5 молотков, 3 дрели, 6 ключей
2	23	1 молоток, 4 ключа
3	15	2 ключа

Рис. 5

Таблица ***Заказы1*** служит для хранения данных о заказах на складе инструментов. Эта таблица имеет простой ключ, состоящий из одного поля *КодЗаказа*. Данные в столбце *СодержаниеЗаказа* таблицы ***Заказы1*** содержат ***списки величин*** и ***не являются атомарными***. Так как в столбце *СодержаниеЗаказа* находится слишком много информации, то получить упорядоченную информацию из этой таблицы будет трудно. Например, трудоемким окажется составление отчета о суммарных закупках инструментов различных видов. Таблицу ***Заказы1*** можно преобразовать (*без потери данных*) в таблицу ***Заказы2*** (рис. 6), удовлетворяющую требованиям 1НФ.

Заказы2

КодЗаказа*	КодСодержанияЗаказа*	КодПокупателя	Количество	СодержаниеЗаказа
1	1	4	5	молоток
1	2	4	3	дрель
1	3	4	6	ключ
2	1	23	1	молоток
2	3	23	4	ключ
3	3	15	2	ключ

Рис. 6

Эта таблица имеет составной ключ из двух полей: *КодЗаказа* и *КодСодержанияЗаказа*. Данные всех столбцов таблицы ***Заказы2*** являются атомарными.

Таблица находится во ***второй нормальной форме*** (2НФ), если она находится в 1НФ и каждое неключевое (*описательное*) поле *функционально* зависит от всего ключа (простого или составного).

Функциональная зависимость полей – это зависимость, при которой определенному значению ключа соответствует *только одно значение описательного поля*.

Каждый неключевой столбец таблицы в 2НФ должен ***полностью*** (т. е. ***функционально***) зависеть от ***всего первичного ключа*** (простого или *составного*). Если таблица имеет простой первичный ключ, состоящий только из одного столбца, то она *автоматически находится в 2НФ*. Если же первичный ключ *составной*, то таблица необязательно находится в 2НФ. В этом случае необходимо преобразовать таблицу (или разделить ее на несколько таблиц) так, чтобы первичный ключ однозначно определял значение в любом неключевом столбце. Одну и ту же таблицу можно привести к 2НФ разными способами, не теряя данных. Приведение таблицы к 2НФ позволяет *избежать повторения одних и тех же данных*, которое может появиться после приведения таблицы к 1НФ.

Например, таблица *Заказы2* не находится во 2НФ. Эта таблица имеет составной ключ, включающий 2 поля: *КодЗаказа* и *КодСодержанияЗаказа*. В то же время, неключевое поле *СодержаниеЗаказа* однозначно определяется только полем *КодСодержанияЗаказа*, т. е. *только одним* из двух полей, входящих в составной ключ. При заданной кодировке инструментов на складе для заполнения значения поля *СодержаниеЗаказа* достаточно знать только значение поля *КодСодержанияЗаказа* (1 – молоток, 2 – дрель и т. д.). Таким образом, для поля *СодержаниеЗаказа* отсутствует функциональная зависимость от *всего составного ключа* и таблица ***Заказы2*** не находится во 2НФ. Таблицу ***Заказы2*** можно преобразовать без потери данных в таблицу ***Заказы3*** (рис. 7), удовлетворяющую требованиям 2НФ.

Заказы3

КодЗаказа*	СодержаниеЗаказа*	КодПокупателя	Количество
1	молоток	4	5
1	дрель	4	3
1	ключ	4	6
2	молоток	23	1
2	ключ	23	4
3	ключ	15	2

Рис. 7

У таблицы ***Заказы3*** составной ключ содержит 2 поля: *КодЗаказа* и *СодержаниеЗаказа*, а неключевые или описательные поля *КодПокупателя* и *Количество* функционально зависят от *всего составного ключа*. Из данного примера также видно, как уменьшается повторение одинаковых данных при переходе от 1НФ ко 2НФ («исчезает» один столбец из двух, содержащих повторяющиеся данные в таблице **Заказы2**).

Таблица находится в ***третьей нормальной форме*** (3НФ), если она находится в 2НФ, и каждое неключевое поле *нетранзитивно зависит от первичного ключа*.

Транзитивная зависимость полей имеет место в том случае, если одно из двух описательных полей зависит от ключа, а другое описательное поле зависит от первого описательного поля. Таким образом, таблица соответствует 3НФ, если она соответствует 2НФ, и все неключевые столбцы *взаимно независимы*. Например, таблица ***Студенты2*** = (*Номер**, *Фамилия*, *Имя*, *Дата*, *Группа*, *Староста*) не находится в 3НФ, т. к. описательное поле *Староста* зависит от описательного поля *Группа* (определенное значение поля *Группа* требует соответствующего значения поля *Староста*, т. е. имеется транзитивная зависимость этих полей). Поэтому таблица ***Студенты2*** должна быть *расщеплена* на две связанные по полю *Группа* таблицы, каждая из которых находится в 3НФ:

Студенты = (*Номер* *, *Фамилия*, *Имя*, *Дата*, *Группа*),
Старосты = (*Группа**, *Староста*).

Транзитивные зависимости полей создают проблемы при *добавлении*, *обновлении* и *удалении* записей из таблицы. Например, если в какой-то группе поменялся староста, то для этой группы в каждую запись таблицы ***Студенты2*** (всего таких записей столько, сколько студентов в группе) нужно внести изменение в поле *Староста*. А для расщепленных таблиц Студенты, Старосты, удовлетворяющих 3НФ, в этом случае нужно изменить *только одну запись* в таблице ***Старосты***. Таблица *Заказы3* также находится в 3НФ: т. к. неключевые поля *КодПокупателя* и *Количество* взаимно независимы, то в этой таблице нет транзитивных зависимостей полей.

Таким образом, «плата» за нормализацию таблиц – добавление в таблицы новых полей, возможное увеличение количества таблиц в процессе нормализации. «Выгода» от нормализации – более простая и быстрая обработка данных; экономия памяти и машинного времени для хранения и обработки данных.

7. Примеры проектирования баз данных

Рассмотрим примеры проектирования реляционных БД с учетом требований нормализации средствами СУБД MS Access (ниже приведены окна создания схемы данных, вызываемые из основного меню MS Access командами **Сервис** / **Схема данных**).

Пример 1. Спроектировать БД «Каталог продукции производителей Республики Беларусь». БД должна содержать следующую информацию: сведения о продукции, сведения о предприятии-производителе, стоимость продукции на определенную дату.

Требуемая схема данных может иметь вид, представленный на рис. 8.

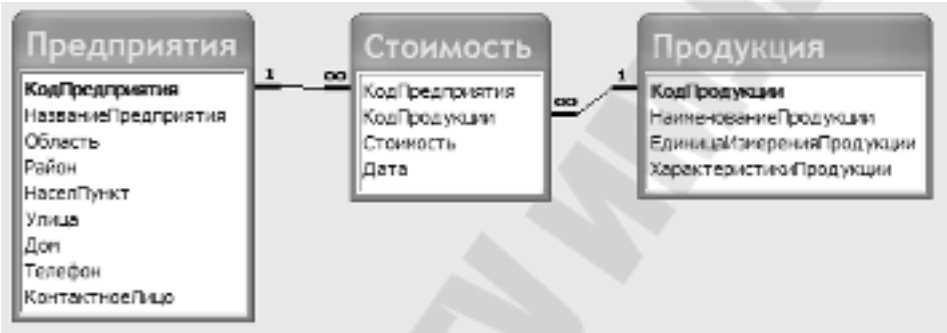


Рис. 8

Между таблицами используются связи типа «один ко многим» (**1 : ∞**). Поля на стороне отношения «один» (на схеме данных выделяются полужирным шрифтом) являются первичными ключами таблиц. Первичные ключи *КодПредприятия*, *КодПродукции* в таблицах ***Предприятия***, ***Продукция*** имеют тип данных *Счетчик*, а соответствующие поля на стороне отношения «многие» (внешние ключи) в таблице Стоимость имеют тип данных *Числовой* (при этом свойство *Размер поля* имеет значение *Длинное целое*). Остальные поля имеют типы данных *Текстовый*, *Числовой*, *Дата/время* в зависимости от заносимых данных. При этом задается ***минимально необходимое*** значение свойства *Размер поля* (например, нет необходимости для текстового поля *НазваниеПредприятия* задавать размер, больший 20–30 символов).

Отметим, что структуру реляционной БД, соответствующей требованиям нормализации, можно быстро создать, используя ***подход «справочных» и «оперативных» таблиц***. В данном примере к справочным таблицам (*справочникам*) относятся таблицы ***Предприятия***, ***Продукция*** (содержат по одной строке для каждого предприятия или вида продукции соответственно; данные в этих таблицах слабо зави-

сят от времени и изменяются относительно редко). *Оперативная таблица* ***Стоимость*** содержит много строк для каждого предприятия и вида продукции, ее данные сильно зависят от времени и быстро изменяются, т. к. стоимость продукции постоянно обновляется. Как правило, с течением времени объем данных оперативных таблиц становится существенно больше объема данных справочников. Между таблицами-справочниками и оперативными таблицами устанавливается связь **1 : ∞** (чаще всего используется в реляционных БД), для этого в таблицы нужно добавить соответствующие первичные и внешние ключи. В результате, разместив поля БД по справочным и оперативным таблицам (после анализа предметной области и выделения интересующих атрибутов отношений) и задав связи, получаем нормализованную БД, как правило, соответствующую 3НФ.

Пример 2. Разработать БД «Энергетик», в которой хранится и обрабатывается информация об энергетическом оборудовании предприятия (объединения предприятий). Для каждой единицы оборудования в БД должна храниться следующая информация:

1) инвентарный и серийный номера, марка, наименование;

2) данные о подразделении, где установлено оборудование: адрес (город, улица, дом, корпус); реквизиты главного энергетика;

3) организация – производитель оборудования (наименование, основные реквизиты, контактное лицо);

4) год выпуска, дата (год, месяц) ввода в эксплуатацию;

5) даты испытаний (год, месяц);

6) технические характеристики: масса (кг); номинальная теплопроизводительность (кВт, для соответствующего оборудования); габариты (длина × ширина × высота, м); гарантийный срок эксплуатации (месяцев); средняя наработка на отказ (ч); полный назначенный срок службы (лет);

7) лицо, ответственное за эксплуатацию оборудования (ФИО, подразделение, должность, телефон);

8) энергосберегающие мероприятия, связанные с оборудованием: наименование мероприятия, дата выполнения (год, квартал), затраты на внедрение (млн руб.); экономический эффект от внедрения мероприятия за квартал (млн руб.); для одной единицы оборудования могут периодически выполняться энергосберегающие мероприятия.

Требуемую информацию об оборудовании можно разными способами распределить по реляционным таблицам. Одну из возможностей проектирования БД иллюстрирует следующая схема данных (рис. 9).

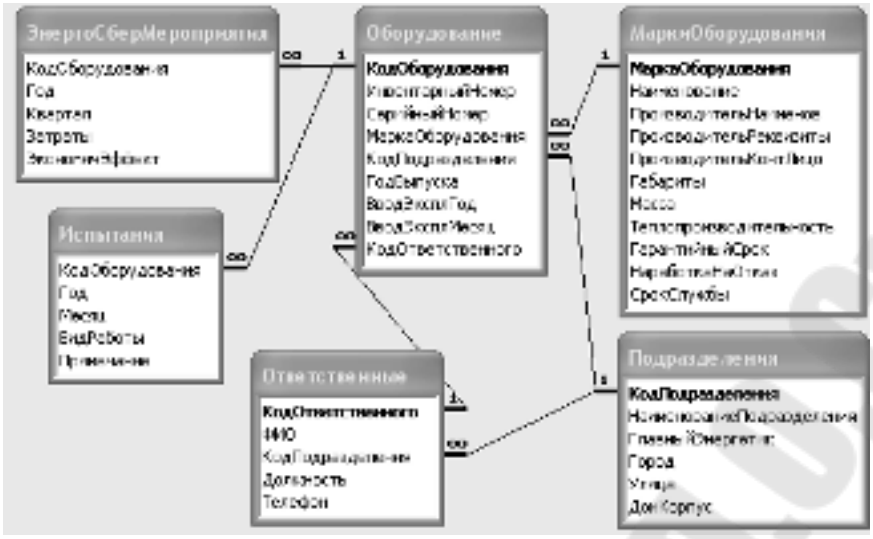


Рис. 9

Первичные ключи *КодОборудования*, *КодОтветственного*, *КодПодразделения* имеют тип данных *Счетчик*, а соответствующие поля на стороне отношения «многие» имеют тип данных *Числовой* (при этом свойство *Размер поля* имеет значение *Длинное целое*). Для таблиц ***Оборудование***, ***МаркиОборудования*** поле *МаркаОборудования* имеет тип данных *Текстовый*. Остальные поля имеют типы данных *Текстовый*, *Числовой* в зависимости от заносимых данных. При этом задается ***минимально необходимое*** значение свойства *Размер поля* (например, 10–15 символов для текстового поля *МаркаОборудования*).

Отметим, что в приведенных БД некоторые поля могут содержать много текстовых данных: поле *ХарактеристикиПродукции* в таблице ***Продукция*** – Пример 1; поле *ПроизводительРеквизиты* в таблице ***МаркиОборудования*** – Пример 2). С учетом анализа предметных областей эти данные атомарны – нас интересуют полные значения этих полей. Если бы требовалось хранить и обрабатывать по отдельности, например, каждую их характеристик вида продукции (масса, размеры, цвет и т. д.) или каждый из реквизитов производителя определенной марки оборудования (адрес, расчетный счет и т. д.), то указанные поля пришлось бы разбить на несколько полей согласно требованиям нормализации.

На приведенной схеме данных таблица ***Оборудование*** выступает в качестве оперативной таблицы по отношению к справочным таблицам ***МаркиОборудования***, ***Подразделения***, ***Ответственные***. Таблица ***Ответственные***, в свою очередь, является оперативной по отношению к таблице ***Подразделения***. Кроме этого, таблица ***Оборудование*** является справочной таблицей по отношению к таблицам ***ЭнергоСберМероприятия*** и ***Испытания***. Соответствующие первичные и внешние ключи обеспечивают связи **1 : ∞** между таблицами БД.

Литература

1. Хомоненко, А. Д. Базы данных : учеб. для высш. учеб. заведений / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев ; под ред. проф. А. Д. Хомоненко. – Москва : Бином-Пресс ; Санкт-Петербург : КОРОНА принт, 2006. – 736 с.

2. Дейт, К. Введение в системы баз данных / К. Дейт. – Москва : Вильямс, 2001. – 1072 с.

3. Кренке, Д. Теория и практика построения баз данных / Д. Кренке. – Санкт-Петербург : Питер, 2003. – 800 с.

4. Гетц, К. Разработка корпоративных приложений в Access 2002 / К. Гетц, М. Гунделой, П. Литвин. – Санкт-Петербург : Питер, 2003. – 400 с.

5. Информатика : учебник / под ред. проф. Н. В. Макаровой. – Москва : Финансы и статистика, 2006. – 768 с.

6. Гладких, Б. А. Информатика. Введение в специальность : учеб. пособие для вузов / Б. А. Гладких. – Томск : Изд-во науч.-техн. лит., 2002. – 350 с.

7. Исаченко, А. Н. Модели данных и системы управления базами данных / А. Н. Исаченко, С. П. Бондаренко. – Минск : БГУ, 2007. – 220 с.

Содержание

Учебное электронное издание комбинированного распространения

Учебное издание

Стародубцев Евгений Генрихович

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Пособие по дисциплинам «Базы данных», «Технологии организации, хранения и обработки данных», «Разработка приложений баз данных для информационных систем» для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной и заочной форм обучения

Электронный аналог печатного издания

Редактор *Н. В. Гладкова*
Компьютерная верстка *М. В. Аникеенко*

Подписано в печать 14.01.10.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 1,86. Уч.-изд. л. 1,81.
Изд. № 208.
E-mail: ic@gstu.by
http://www.gstu.by

Издатель и полиграфическое исполнение:
Издательский центр учреждения образования
«Гомельский государственный технический университет
имени П. О. Сухого».
ЛИ № 02330/0549424 от 08.04.2009 г.
246746, г. Гомель, пр. Октября, 48.