ФГБОУ ВПО «Воронежский государственный
технический университет»

Д.Г. Плотников

БАЗЫ ДАННЫХ
И ИХ БЕЗОПАСНОСТЬ

Утверждено Редакционно-издательским советом
университета в качестве учебного пособия

Воронеж 2015

УДК 004.05

Плотников Д. Г. Базы данных и их безопасность: учеб. пособие [Электронный ресурс]. – Электрон. текстовые, граф. данные (1,93 Мб) / Д. Г. Плотников. –Воронеж: ФГБОУ ВПО «Воронежский государственный технический университет», 2015. – 1 электрон. опт. диск (CD-ROM). – Систем. требования: ПК 500 и выше; 256 Мб ОЗУ; Windows XP; AdobeReader; 1024x768; CD-ROM; мышь. – Загл. с экрана.

В учебном пособии представлен теоретический и практический материал по основным вопросам организации и управления базами данных, рассмотрены основные приемы работы с ними при использовании языка запросов SQL.

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего профессионального образования по специальностям 090301 «Компьютерная безопасность», 090303 «Информационная безопасность автоматизированных систем», дисциплинам «Системы управления базами данных», «Основы построения защищенных СУБД», «Безопасность систем баз данных».

Табл. 53. Ил. 16. Библиогр.: 8 назв.

Рецензенты: ОАО «Концерн «Созвездие»
(канд. техн. наук, ведущий науч. сотрудник О. В. Поздышева);
д-р техн. наук, проф. А. Г. Остапенко

© Плотников Д. Г., 2015
© Оформление. ФГБОУ ВПО «Воронежский государственный технический университет», 2015

ВВЕДЕНИЕ

Цель данного учебного пособия по базам данных заключается в систематическом изложении теоретических основ построения баз данных, возможностей современных систем управления баз данных, технологии применения их для разработки и использования информационных систем, в том числе в сетях Интернет и интранет.

Индустрия исследований систем баз данных – это история развития приложений, достигших большой производительности и оказавших существенное влияние на экономику. Сейчас на исследованиях баз данных основана индустрия информационных услуг. Достижения в исследованиях баз данных стали основой фундаментальных разработок коммуникационных систем, транспорта, финансового менеджмента, систем баз знаний, методов доступа к научной литературе, а также большого количества гражданских и военных приложений. Они также послужили фундаментом значительного прогресса в ведущих областях науки – от информатики до биологии [3].

Можно утверждать, что появление баз данных стало важным достижением в области программного обеспечения. Базы данных лежат в основе информационных систем, и это коренным образом изменило характер работы многих организаций. С момента своего появления технология баз данных стала серьезной областью деятельности, а также катализатором многих значительных достижений в области программного обеспечения. При этом развитие систем баз данных еще не завершено. Приложения, которыми придется пользоваться в будущем, окажутся настолько сложными, что потребуется переосмыслить многие алгоритмы – например, используемые в настоящее время алгоритмы хранения файлов, а также доступа к ним.

Структурированный язык запросов (Structured Query Language – SQL или Sequel - результат) был разработан IBM в 1974-1979 гг. как язык взаимодействия с прототипом систем

управления реляционными базами данных System R. Первая коммерчески доступная СУБД, использующая SQL, была представлена в 1979 году Oracle Corporation. SQL давно стал стандартом, но его развитие продолжается. Многие существующие серверы реляционных баз данных поддерживают ту или иную версию или уровень стандарта ANSI/ISO SQL (ANSI – американский национальный институт стандартов, ISO – международная организация стандартов). Поддержка SQL позволяет пользователям и приложениям баз данных SQL – типа идентично работать со многими различными реляционными серверами баз данных.

Основы SQL изложены в большом числе монографий. Однако далеко не все из них одинаково доступны в настоящее время различным категориям обучаемых. Это является одной из причин написания данного пособия.

ОСНОВНЫЕ ТЕРМИНЫ

База данных – поименованная совокупность структурированных данных предметной области.

Система управления базами данных (СУБД) – комплекс программных средств для создания баз данных, хранения и поиска в них необходимой информации.

Структурирование данных – процесс группировки данных по определённым параметрам.

Таблица – это объект, предназначенный для хранения данных в виде записей (строк) и полей (столбцов).

Запрос – объект базы данных, позволяющий получить нужные данные из одной или нескольких таблиц.

Отчёт - объект базы данных, предназначенный для печати данных.

Форма - созданный пользователем графический интерфейс для ввода данных в базу.

Ключ – поле, которое однозначно определяет соответствующую запись (например, № личного дела ученика).

Запись – совокупность логически связанных полей, характеризующих типичные свойства реального объекта.

Поле – простейший объект базы данных, предназначенный для хранения значений одного параметра реального объекта или процесса.

1. ПОНЯТИЕ БАЗЫ ДАННЫХ. ФАЙЛОВЫЕ СИСТЕМЫ И БАЗЫ ДАННЫХ. КЛАССИФИКАЦИЯ ЗАДАЧ, РЕШАЕМЫХ С ИСПОЛЬЗОВАНИЕМ СУБД

В общем случае термины «Система с базами данных» и «Система управления базами данных» различны в том смысле, что второй термин – это подсистема управления системой с базами данных, которая является специализированным приложением. Но в некоторых контекстах оба этих термина, используются как имеющие одинаковый смысл.

Начнем с примеров некоторых приложений систем баз данных.

- Складской учет.

Интуитивно понятно, что на складе хранится очень много товаров. При этом товары могут быть однообразными или очень многообразными. Если это большой склад, то в штат слада должны входить не только охранники, работники, которые непосредственно отгружают товары, но и служащие, которые учитываю количество и качество имеющихся на складе товаров. Раньше для этой цели использовались специальные складские книги и множество стандартных приходно-расходных ордеров. Ясно, что работы с огромным количеством информации порождали столь же огромное количество ошибок.

Сейчас при использовании для складского учета баз данных весь объем данных о товарах находится в памяти ЭВМ. И не просто хранится в произвольном виде, а строго упорядочен. Использование баз данных дают возможность не только классифицировать товары, но и осуществлять к ним упорядоченный и оптимальный доступ. Когда покупатель приезжает на склад и делает запрос на покупку партии товара, служащий склада по названию товара очень быстро находит его характеристики (цена, срок хранения, вес, габариты, производителя, наличие необходимого количества товара на складе и т. п.). Далее весь процесс учета может быть автоматизирован. Если запасы товара на складе опустятся ниже некоторого загодя определенного уровня, то в таком случае

система сама сможет автоматически направить заказ на поставку дополнительного количества данного товара или известить об этом служащего. Если запрос поступает по телефону, то служащий склада также может проверить наличие товара, запустив специальное приложение базы данных.

- Использование кредитной карточки.

Если при покупках используется кредитная карточка, кассир должен проверить наличие кредитных средств. Это можно сделать или по телефону, или автоматически, с помощью специального считывающего устройства, связанного с компьютером. На основе номера кредитной карточки специальное приложение сверяет цену покупаемых в данный момент и купленных в течение этого месяца товаров с кредитным лимитом. После подтверждения допустимости такой покупки все сведения о приобретенных товарах вводятся в базу данных. Однако еще до получения подтверждения допустимости покупки приложение базы данных должно убедиться, что данная карточка не находится в списке украденных или утерянных. Кроме того, должно существовать еще одно самостоятельное приложение баз данных, которое будет оплачивать счета после получения суммы платежа, а также ежемесячно посылать каждому владельцу кредитной карточки полный отчет.

- Бухгалтерия учреждений.

Базы данных используются в бухгалтериях всех крупных предприятий. Каждый работник предприятия имеет свой уникальный номер, под которым данные о нем хранятся в банке данных предприятия. При начислении зарплаты используются данные о количестве отработанных дней, занимаемой должности, окладе, премиях, льготах по налогообложению и т. п. Каждый раз начисленная зарплата записывается в базу, чтобы в дальнейшем можно было пересчитать оплату ежегодного отпуска, выплаты за выслугу лет (она начисляется ежегодно), прогрессивный налог и т.п. Эти данные, не занимающие много места, хранятся долгие годы, и позволяют в дальнейшем составлять различные справки.

Каждый из вас, подумав, легко может привести пример использования (или необходимости использования при наличии компьютера дома) баз данных в повседневной жизни. От каталога любимых дисков до ведения домашней бухгалтерии и собственных налогов.

Широко распространенным предшественником баз данных являются **файловые системы**. Несмотря на то, что файловые системы давно устарели, все же существует несколько следующих причин, по которым с ними следует познакомиться:

- Понимание проблем, присущих файловым системам, может предотвратить их повторение в СУБД.
- Знать принципы работы файловых систем не только очень полезно, но и необходимо при выполнении перехода от файловой системы к системе баз данных.

Итак, **ФАЙЛОВЫЕ СИСТЕМЫ** – это набор программ, которые выполняют для пользователя некоторые операции, например, создание отчетов. Каждая программа определяет свои собственные данные и управляет ими.

Файловые системы были первой попыткой компьютиризировать известные ручные картотеки. Подобная картотека (или подшивка документов) в некоторой организации могла содержать всю внешнюю и внутреннюю документацию, связанную с каким-либо проектом, продуктом, задачей, клиентом или сотрудником. Обычно таких папок бывает очень много, они помечаются и хранятся в одном или нескольких шкафах. В целях безопасности шкафы могут закрываться на замок или находиться в охраняемых помещениях. У каждого из нас дома есть некое подобие такой картотеки, содержащее подшивки документов, представляющие собой счета, гарантийные талоны, рецепты, страховые и банковские документы и т. п. Если нам понадобится какая-то информация,

то потребуется просмотреть картотеку от начала до конца, чтобы найти искомые сведения. Более изощренный подход предусматривает использование в такой системе некоторого алгоритма индексирования, позволяющего ускорить поиск нужных сведений. Например, можно использовать специальные разделители или отдельные папки для различных логически связанных типов объектов, или использовать каталоги.

Ручные картотеки позволяют успешно справляться с поставленными задачами, если количество объектов невелико. Они также вполне годятся для работы с большим количеством объектов, которое нужно только хранить и извлекать. Однако они совершенно не подходят для тех случаев, когда нужно установить перекрестные связи или выполнить обработку сведений.

Файловые системы были разработаны в ответ на потребность в получении более эффективных способов доступа к данным, однако, вместо организации централизованного хранилища всех данных предприятия, был использован децентрализованный подход, при котором сотрудники каждого отдела при помощи специалистов по обработке данных работают со своими собственными данными и хранят их в своем отделе.

В файловых системах используются следующие понятия.

- Файл является простым набором ***записей***, которые содержат логически связанные данные.
- Каждая запись содержит логически связанный набор из одного или нескольких ***полей***, каждое из которых представляет некоторую характеристику моделируемого объекта.

Даже такого краткого описания файловых систем вполне достаточно для того, чтобы понять суть присущих им ограничений, которые перечислены ниже (рис. 1.1).

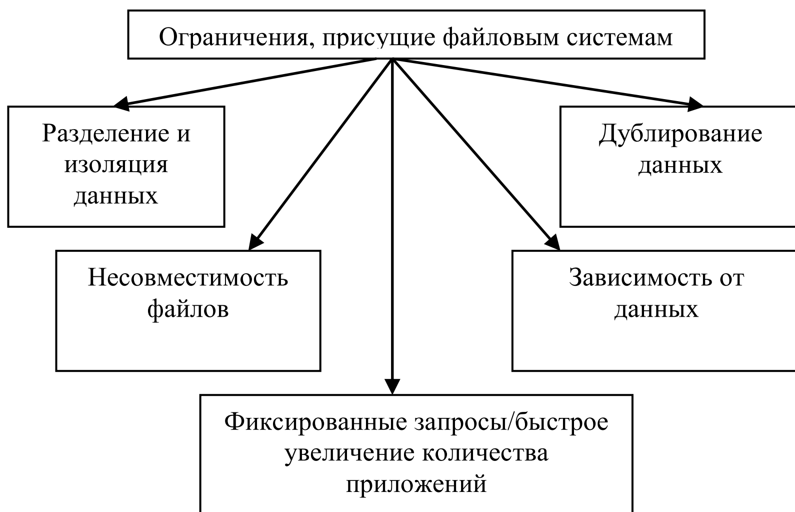


Рис. 1.1. Ограничения, присущие файловым системам

Под ***разделением и изоляцией данных*** понимается ситуация, когда данные изолированы в отдельных файлах, и при этом доступ к ним весьма затруднен. Например, для создания списка всех домов, отвечающих требованиям потенциальных арендаторов, предварительно нужно создать временный файл со списком арендаторов, желающих арендовать недвижимость типа «дом». Затем в файле всех объектов следует осуществить поиск объектов недвижимости типа «дом» с арендной платой ниже установленного арендатором максимума. Для извлечения соответствующей поставленным условиям информации программист должен организовать синхронную обработку двух файлов. Трудности существенно возрастают, когда необходимо извлечь данные более, чем из двух файлов.

Из-за децентрализованной работы с данными, проводимой в каждом подразделении независимо от других отделов, в файловой системе фактически поощряется бесконтрольное ***дублирование данных***, и это, в принципе, неизбежно. Бесконтрольное дублирование данных нежелательно по следующим причинам.

- Дублирование данных сопровождается неэкономным расходованием ресурсов, поскольку на ввод избыточных данных требуется затрачивать дополнительные время и деньги. Более того, для их хранения необходима дополнительная память. Во многих случаях дублирования данных можно избежать за счет совместного использования файлов.
- Гораздо более важен тот факт, что дублирование данных может привести к нарушению их целостности. Т.е. данные в разных отделах могут стать противоречивыми.

Зависимость данных. Физическая структура и способ хранения записей файлов данных жестко зафиксированы в коде программ приложений. Это означает, что изменить существующую структуру данных достаточно сложно. Если такие изменения все-таки происходят, то необходимо создавать специальные программы-конверторы, преобразующие старые формат в новый. Помимо этого, все обращающиеся к этому файлу приложения должны быть изменены в соответствии с новым форматом.

Под ***несовместимостью форматов файлов*** понимается следующее. Поскольку структура файлов определяется кодом приложения. Она также зависит от языка программирования этого приложения. Например, структура файла, созданного программой на языке FORTRAN, может совершенно отличаться от структуры файла, созданного программой на языке C. Прямая несовместимость таких файлов затрудняет процесс обработки.

Фиксированные запросы / быстрое увеличение количества приложений. С точки зрения пользователя возможности файловых систем намного превосходят возможности ручных картотек. Соответственно непрерывно возрастают их требования к реализации новых или модифицированию старых запросов. Однако, файловые системы во многом зависят от программиста, потому что все требуемые запросы и отчеты должны быть созданы именно им. Т. е. нет инструментов создания незапланированных или

произвольных запросов как с самим данным, так и к сведениям о том, какие типы данных доступны. Количество приложений в этом случае растет настолько быстро, что не хватает времени и исполнителей для разработки документации, и сопровождения приложений. В этом случае нагрузка на сотрудников отдела обработки данных настолько возрастала, что неизбежно наступал момент, когда программное обеспечение было неспособно адекватно отвечать запросам пользователей, эффективность его падала, а недостаточность документирования имела следствием дополнительное усложнение сопровождения программ. При этом часто игнорировались вопросы поддержки функциональности системы: не предусматривались меры по обеспечению безопасности или целостности данных; средства восстановления в случае сбоя аппаратного или программного обеспечения были крайне ограничены или вообще отсутствовали. Доступ к файлам часто ограничивался одним пользователем, т.е. не предусматривалось их совместное использование даже сотрудниками одного и того же отдела.

Все перечисленные выше ограничения файловых систем являются следствием двух факторов.

- Определение данных содержится внутри приложений, а не хранится отдельно и независимо от них.
- Помимо приложений не предусмотрено никаких других инструментов доступа к данным и их обработки.

Для повышения эффективности работы необходимо использовать новый подход, а именно ***базу данных*** и ***систему управления базами данных (СУБД)***.

БАЗА ДАННЫХ – это совместно используемый набор логически связанных данных (и описание этих данных), предназначенный для удовлетворения информационных потребностей группы пользователей.

Рассмотрим это определение более подробно. База данных – это единое, большое хранилище данных, которое однократно определяется, а затем используется одновременно многими пользователями. Вместо разрозненных файлов с избыточными данными, здесь все данные собраны вместе с минимальной долей избыточности. База данных уже не принадлежит какому-либо единственному отделу, а является общим корпоративным ресурсом. Причем база данных хранит не только рабочие данные, но и их описания. По этой причине базу данных еще называют *набором интегрированных записей с самоописанием*. В совокупности, описание данных называется *системным каталогом* или ***словарем данных***, а сами элементы описания принято называть ***метаданными***, т.е. данными о данных. Именно наличие самоописания данных в базе данных обеспечивает в ней ***независимость между программами и данными***.

Подход, основанный на применении баз данных, где определение данных отдельно от приложений, очень похож на подход, используемый при разработке современного программного обеспечения, когда наряду с внутренним определением объекта существует его внешнее определение.

Пользователи объекта видят только его внешнее определение и не заботятся о том, как он определяется и как функционирует. Одно из преимуществ такого подхода, а именно *абстрагирования данных*, заключается в том, что можно изменить внутреннее определение объекта без каких-либо последствий для его пользователей, при условии, что внешнее определение объекта остается неизменным. Аналогичным образом, в подходе с использованием баз данных, структура данных отделена от приложений и хранится в базе данных. Добавление новых структур данных или изменение существующих никак не влияет на приложения, при условии, что они не зависят непосредственно от изменяемых компонентов. Например, добавление нового поля в запись или создание нового файла никак не повлияет на работу имеющихся приложений. Однако удаление поля из используемого

приложением файла (конечно, если это поле используется) повлияет на это приложение, а потому его также потребуется соответствующим образом модифицировать.

Что такое «логически связанный»? При анализе информационных потребностей группы пользователей следует выделить сущности, атрибуты и связи (рис. 1.2). ***Сущностью*** называется отдельный тип объекта, который нужно представить в базе данных. ***Атрибутом*** называется свойство, которое описывает некоторую характеристику описываемого объекта; ***связь*** – это то, что объединяет несколько сущностей. Итак, база данных представляет сущность, атрибуты и логические связи между объектами. Иначе говоря, база данных содержит логически связанные данные.

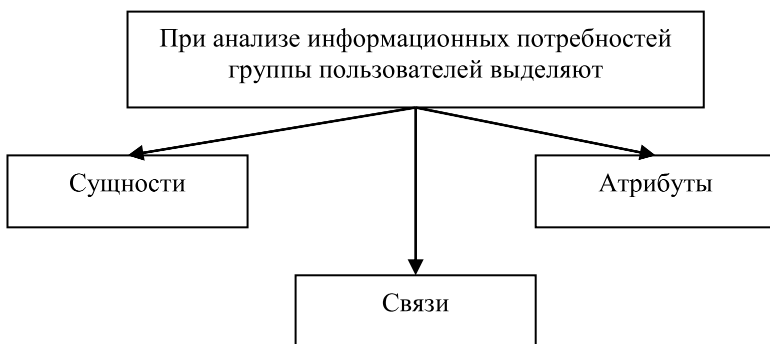


Рис. 1.2. Данные, выделяемые при анализе информационных потребностей

СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ (СУБД) – это программное обеспечение, с помощью которого пользователи могут определять, создавать и поддерживать базу данных, а также осуществлять к ней контролируемый доступ.

СУБД – это программное обеспечение, которое взаимодействует с прикладными программами пользователя и базой данных и обладает приведенными ниже возможностями.

- Позволяет определять базу данных, что обычно осуществляется с помощью *языка определения данных*. Этот язык предоставляет пользователям средства указания типа данных и их структуры, а также средства задания ограничений для информации, хранимой в базе данных.
- Позволяет вставлять, обновлять, удалять и извлекать информацию из базы данных, что обычно осуществляется с помощью *языка управления данными*. Наличие централизованного хранилища всех данных и их описаний позволяет использовать этот язык как общий инструмент организации запросов, который иногда называют ***языком запросов***. Наличие языка запросов позволяет устранить присущие файловым системам ограничения, при которых пользователям приходится иметь дело только с фиксированным набором запросов или постоянно возрастающим количеством программ.

Существует две разновидности языков запросов – ***процедурные*** и ***непроцедурные*** языки, которые отличаются между собой способом извлечения данных. основное отличие между ними заключается в том, что процедурные языки обычно обрабатывают информацию в базе данных последовательно, запись за записью, а непроцедурные оперируют сразу целыми наборами записей. Поэтому с помощью процедурных языков обычно указывается, как можно получить желаемый результат, тогда как непроцедурные языки используются для описания того, что следует получить. Наиболее распространенным типом непроцедурного языка является язык структурированных запросов (SQL), который в настоящее время определяется специальным стандартом и фактически является обязательным языком для любых реляционных СУБД.

- Предоставляет контролируемый доступ к базе данных с помощью перечисленных ниже средств:
  - ✓ системы обеспечения безопасности, предотвращающей несанкционированный доступ к базе данных со стороны пользователей;

- ✓ системы поддержки целостности данных, обеспечивающей непротиворечивое состояние хранимых данных;
- ✓ системы управления параллельной работой приложений, контролирующей процессы их совместного доступа к базе данных;
- ✓ системы восстановления, позволяющей восстановить базу данных до предыдущего непротиворечивого состояния, нарушенного в результате сбоя аппаратного или программного обеспечения;
- ✓ доступного пользователям каталога, содержащего описание хранимой в базе данных информации.

Отличие от файловой системы в случае с базой данных в том, что каждый пользователь по-прежнему имеет собственные приложения, но физическая структура и способ хранения данных контролируется с помощью СУБД.

Обладая указанными выше функциональными возможностями СУБД в общем случае предоставляет всем пользователям избыточную информацию. Для решения этой проблемы в СУБД предлагается другой механизм – создание ***представлений***, - который позволяет любому пользователю иметь свой собственный «взгляд» на базу данных. Язык определения данных включает средства определения представлений, каждое из которых является некоторым подмножеством базы данных.

Помимо упрощения работы за счет предоставления пользователям только действительно нужных им данных, представления обладают некоторыми другими достоинствами.

- Обеспечивают дополнительный уровень безопасности. Представления могут создаваться с целью исключения тех данных, которые не должны видеть некоторые пользователи.

- Предоставляют механизм настройки интерфейса базы данных.

- Позволяют сохранить внешний интерфейс базы данных непротиворечивым и неизменным даже при внесении

изменений в ее структуру – например, при добавлении или удалении полей, изменении связей, разбиении файлов, их реорганизации или переименовании. Если в файл добавляются или из него удаляются поля, не используемые в некотором представлении, то все эти изменения на данном представлении никак не отразятся. Таким образом, представление обеспечивает полную независимость программ от реальной структуры данных, то позволяет устранить важнейший недостаток файловых систем.

Все эти определения имеют несколько общий характер и справедливы для современных мощных многопользовательских СУБД, так как на самом деле реальный объем функциональных возможностей, предлагаемых в некоторой конкретной СУБД, отличается от продукта к продукту.

Например, СУБД для персонального компьютера может не поддерживать параллельный совместный доступ, а управление режимом безопасности, поддержанием целостности данных и восстановлением будет присутствовать только в очень ограниченной степени. Однако современные мощные многопользовательские СУБД представляют собой чрезвычайно сложное программное обеспечение, состоящее из миллионов строк кода и многих томов документации.

2. МОДЕЛИ ДАННЫХ. ОТОБРАЖЕНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ. СУЩНОСТИ И СВЯЗИ. МЕТОДЫ АБСТРАГИРОВАНИЯ ДАННЫХ. ИЕРАРХИЧЕСКАЯ, СЕТЕВАЯ, РЕЛЯЦИОННАЯ МОДЕЛИ ДАННЫХ

2.1. Сущности и связи между ними

Основная цель системы управления базами данных заключается в том, чтобы предложить пользователю абстрактное представление данных, скрыв конкретные особенности хранения и управления ими. Следовательно, отправной точкой при проектировании базы данных должно быть абстрактное и общее описание информационных потребностей организации, которые должны найти свое отражение в создаваемой базе данных.

При определении базы данных мы уже касались таких понятий как сущность, связь и атрибуты. Дадим более полное определение этих понятий

ТИПЫ СУЩНОСТЕЙ – это объект или концепция, которые характеризуются на данном предприятии как имеющие независимое существовании

Другими словами, ***тип сущности*** – это множество объектов реального мира с одинаковыми свойствами. Тип сущности характеризуется независимым существованием и может быть объектом с физическим существованием или объектом с концептуальным существованием. Каждый уникально идентифицируемый экземпляр типа сущности называется просто **сущностью (или экземпляром сущности**).

СУЩНОСТЬ – это экземпляр типа сущности, который может быть идентифицирован уникальным образом.

Каждый тип сущности идентифицируется именем и списком свойств. Несмотря на то, что тип сущности обладает уникальным набором атрибутов, каждая сущность имеет свои собственные значения для каждого атрибута. Тип сущности, существование которого зависит от какого-то другого типа, называется ***слабым типом сущности*** (в противоположность ***сильному типу сущности***, который ни от кого не зависит).

АТРИБУТ – это свойство типа сущности или типа связи

СВЯЗЬ – это ассоциация между сущностями, включающая по одной сущности из каждого участвующего в связи типа сущности.

ДОМЕН АТРИБУТА – набор значений, которые могут быть присвоены атрибуту

Каждый атрибут связан с набором значений, которые называются доменом. Различные атрибуты могут совместно использовать один и тот же домен.

Атрибуты делятся на ***простые*** и ***составные***, ***однозначные*** и ***многозначные***, а также ***производные***.

Простой атрибут – это атрибут, состоящий из одного компонента с независимым существованием

СОСТАВНОЙ АТРИБУТ –состоит из нескольких компонентов, каждый из которых характеризуется независимым существованием

ОДНОЗНАЧНЫЙ АТРИБУТ – атрибут, который содержит одно значение для одной сущности

МНОГОЗНАЧНЫЙ АТРИБУТ – атрибут, который содержит несколько значений для одной сущности

Например, у одного и того же отделения компании может быть несколько номеров телефонов

ПРОИЗВОДНЫЙ АТРИБУТ – атрибут, который представляет значение, производное от значения связанного с ним атрибута или некоторого множества атрибутов, принадлежащих некоторому (не обязательно данному) типу сущности.

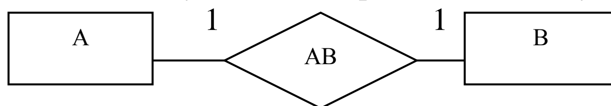
Например, возраст сотрудника вычисляется по его дате рождения, т.е. атрибут «возраст» производный от атрибута «дата рождения».

При построении моделей концептуального уровня можно использовать язык *ER-диаграмм* (от англ. Entity-Relationship, т. е. сущность-связь). В них тип сущности изображаются помеченными прямоугольниками (сильный тип сущности – с контуром внутри него), связь – помеченными ромбами или шестиугольниками, атрибуты – помеченными овалами, а связи между ними.

Ненаправленными ребрами, над которыми может проставляться степень связи (1 или буква M, заменяющая слово "много") и необходимое пояснение. Имя атрибута, который является первичным ключом, подчеркивается.

Между двумя сущностям, например, A и B возможны четыре вида связей.

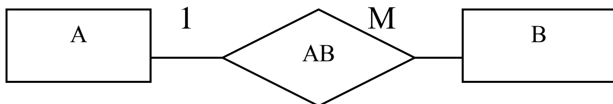
Первый тип – связь ОДИН-К-ОДНОМУ (1:1): в каждый момент времени каждому представителю (экземпляру) сущности A соответствует 1 или 0 представителей сущности B:



Пример:

Студент может не «заработать» стипендию, получить обычную или одну из повышенных стипендий.

Второй тип – связь ОДИН-КО-МНОГИМ (1:М): одному представителю сущности А соответствуют 0, 1 или несколько представителей сущности В.



Пример:

Квартира может пустовать, в ней может жить один или несколько жильцов.

Так как между двумя сущностями возможны связи в обоих направлениях, то существует еще два типа связи МНОГИЕ-К-ОДНОМУ (М:1) и МНОГИЕ-КО-МНОГИМ (М:N).

Пример: Если связь между сущностями МУЖЧИНЫ и ЖЕНЩИНЫ называется БРАК, то существует четыре возможных представления такой связи:

1. Традиционный брак

2. Многоженство

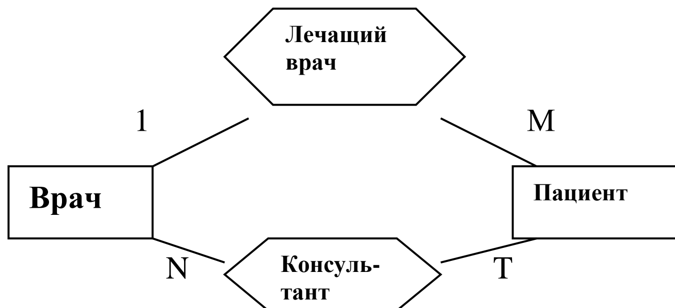
3. Многомужие

4. Групповой брак

Характер связей между сущностями не ограничивается перечисленными. Существуют и более сложные связи.

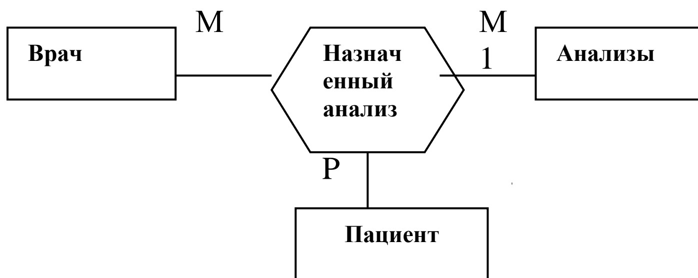
Пример:

- Множество связей между одними и теми же сущностями



(пациент, имея одного лечащего врача, может иметь также несколько врачей-консультантов; врач может быть лечащим врачом нескольких пациентов и может одновременно консультировать несколько других пациентов);

- Тренарные связи



Если экземпляры данной сущности должны участвовать в связи, то участие называется ***обязательным***, в противном случае – ***необязательным***. В первом случае этот факт отмечается маленьким черным кружком, помещенным в блок, смежный с блоком сущности, во втором случае кружок помещают вне блока.

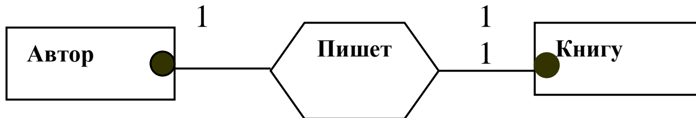
Пример:

а)

б)

в)

г)



а) каждый автор пишет не более одной книги, и каждая книга пишется не более, чем одним автором

б) каждый автор пишет одну и только одну книгу, и каждая книга пишется не более, чем одним автором

в) каждый автор пишет не более одной книги, но каждая книга пишется одним и только одним автором.

г) каждый автор пишет одну и только одну книгу, и каждая книга пишется одним и только одним автором.

Аналогичные схемы можно привести в случае связей один ко многим, многие к одному и многие ко многим.

При разработке концептуальной модели данных могут иметь место проблемы ER-моделирования. Эти проблемы

принято называть ***ловушками соединения (разветвления или разрыва***).

Ключи

Под ключом подразумевается элемент данных, который позволяет уникально идентифицировать отдельные экземпляры некоторого типа сущности.

ПОТЕНЦИАЛЬНЫЙ КЛЮЧ – это атрибут или набор атрибутов, который уникально идентифицирует каждый экземпляр сущности данного типа

Потенциальный ключ должен содержать значения, которые уникальны для каждого отдельного экземпляра сущности данного типа.

ПЕРВИЧНЫЙ КЛЮЧ – это потенциальный ключ, который выбран в качестве первичного ключа.

Сущность может иметь несколько потенциальных ключей. Выбор первичного ключа сущности осуществляется исходя из соображений суммарной длины атрибутов, минимального количества атрибутов в ключе, а также наличия гарантий уникальности его значений в текущий момент времени и обозримом будущем.

СОСТАВНОЙ КЛЮЧ – это потенциальный ключ, который состоит из двух или более атрибутов

Внешние ключи.

- Если сущность C связывает сущности A и B, то она должна включать внешние ключи, соответствующие первичным ключам сущностей A и B.
- Если сущность B обозначает сущность A, то она должна включать внешний ключ, соответствующий первичному ключу сущности A.

При рассмотрении проблемы выбора способа представления ассоциаций и обозначений в базе данных основной вопрос, на который следует получить ответ: "Каковы внешние ключи?". И далее, для каждого внешнего ключа необходимо решить три вопроса:

1. Может ли данный внешний ключ принимать неопределенные значения (NULL-значения)? Иначе говоря, может ли существовать некоторый экземпляр сущности данного типа, для которого неизвестна целевая сущность, указываемая внешним ключом? В случае поставок это, вероятно, невозможно – поставка, осуществляемая неизвестным поставщиком, или поставка неизвестного продукта не имеют смысла. Но в случае с сотрудниками такая ситуация, однако могла бы иметь смысл – вполне возможно, что какой-либо сотрудник в данный момент не зачислен вообще ни в какой отдел. Заметим, что ответ на данный вопрос не зависит от прихоти проектировщика базы данных, а определяется фактическим образом действий, принятым в той части реального мира, которая должна быть представлена в рассматриваемой базе данных. Подобные замечания имеют отношение и к вопросам, обсуждаемым ниже.

2. Что должно случиться при попытке УДАЛЕНИЯ целевой сущности, на которую ссылается внешний ключ? Например, при удалении поставщика, который осуществил по крайней мере одну поставку. Существует три возможности:

КАСКАДИРУЕТСЯ – операция удаления «каскадируется» с тем, чтобы удалить также поставки этого поставщика.

ОГРАНИЧИВАЕТСЯ– удаляются лишь те поставщики, которые еще не осуществляли поставок. Иначе операция удаления отвергается.

УСТАНАВЛИВАЕТСЯ – для всех поставок удаляемого поставщика NULL-значение внешний ключ устанавливается в неопределенное значение, а затем этот поставщик удаляется. Такая возможность, конечно, неприменима, если данный внешний ключ не должен содержать NULL-значений.

3. Что должно происходить при попытке ОБНОВЛЕНИЯ первичного ключа целевой сущности, на которую ссылается некоторый внешний ключ? Например, может быть предпринята попытка обновить номер такого поставщика, для которого

имеется по крайней мере одна соответствующая поставка. Этот случай для определенности снова рассмотрим подробнее. Имеются те же три возможности, как и при удалении:

КАСКАДИРУЕТСЯ – операция обновления «каскадируется» с тем, чтобы обновить также и внешний ключ в поставках этого поставщика.

ОГРАНИЧИВАЕТСЯ – обновляются первичные ключи лишь тех поставщиков, которые еще не осуществляли поставок. Иначе операция обновления отвергается.

УСТАНАВЛИВАЕТСЯ – для всех поставок такого поставщика NULL-значение внешний ключ устанавливается в неопределенное значение, а затем обновляется первичный ключ поставщика. Такая возможность, конечно, неприменима, если данный внешний ключ не должен содержать NULL-значений.

Таким образом, для каждого внешнего ключа в проекте проектировщик базы данных должен специфицировать не только поле или комбинацию полей, составляющих этот внешний ключ, и целевую таблицу, которая идентифицируется этим ключом, но также и ответы на указанные выше вопроса (три ограничения, которые относятся к этому внешнему ключу).

Ограничения целостности

Целостность (от англ. integrity – нетронутость, неприкосновенность, сохранность, целостность) – понимается как правильность данных в любой момент времени. Но эта цель может быть достигнута лишь в определенных пределах: СУБД не может контролировать правильность каждого отдельного значения, вводимого в базу данных (хотя каждое значение можно проверить на правдоподобность). Например, нельзя обнаружить, что вводимое значение 5 (представляющее номер дня недели) в действительности должно быть равно 3. С другой стороны, значение 9 явно будет ошибочным и СУБД должна его отвергнуть. Однако для этого ей следует сообщить, что номера дней недели должны принадлежать набору (1,2,3,4,5,6,7). Поддержание целостности базы данных может рассматриваться как защита данных от неверных изменений

или разрушений (не путать с незаконными изменениями и разрушениями, являющимися проблемой безопасности). Современные СУБД имеют ряд средств для обеспечения поддержания целостности (так же, как и средств обеспечения поддержания безопасности).

Выделяют три группы правил целостности:

1. Целостность по сущностям.
2. Целостность по ссылкам.
3. Целостность, определяемая пользователем.

Не допускается, чтобы какой-либо атрибут, участвующий в первичном ключе, принимал неопределенное значение.

Значение внешнего ключа должно либо:

- быть равным значению первичного ключа цели;
- быть полностью неопределенным, т.е. каждое значение атрибута, участвующего во внешнем ключе должно быть неопределенным.

Для любой конкретной базы данных существует ряд дополнительных специфических правил, которые относятся к ней одной и определяются разработчиком. Чаще всего контролируется:

- уникальность тех или иных атрибутов,
- диапазон значений (экзаменационная оценка от 2 до 5),
- принадлежность набору значений (пол «М» или «Ж»).

2.2. Модели данных

Существуют, так называемые, три уровня абстракции или три различных уровня описания элементов данных. Эти уровни формируют трехуровневую архитектуру, которая охватывает внешний, концептуальный и внутренний уровни. Цель трехуровневой архитектуры заключается в отделении пользовательского представления базы данных от ее

физического представления. Ниже перечислено несколько причин, по которым желательно выполнять такое разделение.

- Каждый пользователь должен иметь возможность обращаться к одним и тем же данным, используя свое собственное представление о них. Каждый пользователь должен иметь возможность изменять свое представление о данных, причем это изменение не должно оказывать влияние на других пользователей.
- Пользователи не должны непосредственно иметь дело с такими подробностями физического хранения данных в базе, как индексирование и хеширование. Иначе говоря, взаимодействие пользователя с базой не должно зависеть от особенностей хранения в ней данных.
- Администратор базы данных должен иметь возможность изменять структуру хранения данных в базе, не оказывая влияния не пользовательские представления.
- Внутренняя структура базы данных не должна зависеть от таких изменений физических аспектов хранения информации, как переключение на новое устройство хранения.
- Администратор базы данных должен иметь возможность изменять концептуальную или глобальную структуру базы данных без какого-либо влияния на всех пользователей.

Уровень, на котором воспринимают данные пользователи, называется ***внешним уровнем***, тогда как СУБД и операционная система воспринимают данные на ***внутреннем уровне***. ***Концептуальный уровень*** представления данных предназначен для отображения внешнего уровня на внутренний и обеспечения необходимой независимости друг от друга.

ВНЕШНИЙ УРОВЕНЬ – это представление базы данных с точки зрения пользователей. Этот уровень описывает ту часть базы данных, которая относится к каждому пользователю.

Каждый пользователь имеет дело с представлением «реального мира», выраженным в наиболее удобной для него форме. Внешнее представление содержит только те сущности и связи «реального мира», которые интересны пользователю. Другие сущности, атрибуты и связи, которые ему неинтересны, также могут быть представлены в базе данных, но пользователь может даже не подозревать об их существовании.

Помимо этого, различные представления могут по-разному отображать одни и те же данные.

КОНЦЕПТУАЛЬНЫЙ УРОВЕНЬ – это обобщающее представление базы данных. Этот уровень описывает то, какие данные хранятся в базе данных, а также связи, существующие между ними.

Концептуальный уровень в трехуровневой архитектуре является промежуточным уровнем. Этот уровень содержит логическую структуру всей базы данных. Фактически, это полное представление требований к данным со стороны организации, которое не зависит от любых соображений относительно способа их хранения. На концептуальном уровне представлены следующие компоненты:

- Все сущности, их атрибуты и связи;
- Накладываемые на данные ограничения;
- Семантическая информация о данных;
- Информация о мерах обеспечения безопасности и поддержки целостности данных.

Концептуальный уровень поддерживает каждое внешнее представление в том смысле, что любые доступные пользователю данные должны содержаться на этом уровне. Однако этот уровень не содержит никаких сведений о методах хранения данных. Например, описание сущности должно содержать сведения о типах данных атрибутов и их длине, но не должно включать сведений об организации хранения данных, например, об объеме занятого пространства в байтах.

ВНУТРЕННИЙ УРОВЕНЬ – это физическое представление базы данных в компьютерах. Этот уровень описывает, как информация хранится в базе данных.

Внутренний уровень описывает физическую реализацию базы данных и предназначен для достижения оптимальной производительности и обеспечения экономного использования дискового пространства. Он содержит описание структур данных и организации отдельных файлов, используемых для хранения данных в запоминающих устройствах. На этом уровне осуществляется взаимодействие СУБД с методами доступа операционной системы с целью размещения данных, создания индексов, извлечения данных и т.д.

На внутреннем уровне хранится следующая информация:

- Распределение дискового пространства для хранения данных и индексов,
- Описание подробностей сохранения записей,
- Сведения о размещении записей.
- Сведения о сжатии данных и выбранных методах их шифрования.

Ниже внутреннего уровня находится ***физический уровень***, который контролируется операционной системой, но под руководством СУБД. Однако, функции СУБД и операционной системы на физическом уровне не вполне четко распределены и могут варьироваться от системы к системе.

Общее описание базы данных называется ***схемой базы данных***. Существует три различных типа схем базы данных, которые определяются в соответствии с уровнями абстракции трехуровневой архитектуры. На самом высоком уровне имеется несколько ***внешних схем*** или ***подсхем***, которые соответствуют разным представлениям данных. На концептуальном уровне описание базы данных называют ***концептуальной схемой***, а на самом низком уровне абстракции – ***внутренней схемой***. СУБД отвечает за установление соответствия между этими тремя типами схем, а также за проверку их непротиворечивости. Концептуальная схема связана с внутренней схемой

посредством ***концептуального внутреннего отображения***. Оно позволяет СУБД найти физическую запись или набор записей на физическом уровне хранения, которые образуют ***логическую запись*** в концептуальной схеме, с учетом любых ограничений, установленных для выполняемых над данной логической записью операций. Оно позволяет обнаружить любые различия в именах объектов, именах атрибутов, порядке следования атрибутов, их типах данных и т.д. Наконец, каждая внешняя схема связана с концептуальной схемой с помощью ***внешне концептуального отображения***. С его помощью СУБД может отображать имена пользовательского представления на соответствующую часть концептуальной схемы.

Важно различать описание базы данных и саму базу данных. Описанием базы данных является схема базы данных.

Схема создается с помощью языка определения данных конкретной целевой СУБД. Это язык относительно низкого уровня, а необходимо описание схемы на некотором, более высоком уровне, которое будем называть ***моделью данных***.

МОДЕЛЬ ДАННЫХ – это интегрированный набор понятий для описания данных, связей между ними и ограничений, накладываемых на данные в некоторой организации.

Модель является представлением «реального мира» объектов и событий, а также существующих связей между ними. Это некоторая абстракция, в которой акцент делается на самых важных и неотъемлемых аспектах деятельности организации, а все второстепенные свойства игнорируются. Модель данных можно рассматривать как сочетание трех указанных ниже компонентов.

- Структурная часть, т.е. набор правил, по которым может быть построена база данных.
- Управляющая часть, определяющая типы допустимых операций с данными (операции обновления и

извлечения данных, а также операции изменения структуры базы данных).

- Набор ограничений поддержки целостности данных (необязательно), гарантирует корректность используемых данных.

Цель построения модели данных заключается в представлении данных в понятном виде.

Для отображения трехуровневой архитектуры можно идентифицировать следующие три связанные модели данных:

- Внешнюю модель данных, отображающую представления каждого существующего в организации типа пользователей, которую иногда называют ***предметной областью***.
- Концептуальную модель данных, отображающую логическое представление о данных, независимое от выбранной СУБД.
- Внутреннюю модель данных, отображающую концептуальную схему определенным образом, понятным выбранной целевой СУБД.

Существует много моделей данных. Они подразделяются на три категории:

- Объектные,
- Модели данных на основе записей,
- Физические модели данных.

Первые две используются для описания данных на концептуальном и внешнем уровнях, а последняя – на внутреннем уровне.

При построении **объектных моделей** используются такие понятия как сущность, атрибуты и связи.

Ниже перечислены некоторые наиболее общие типы объектных моделей данных:

- Модель типа «сущность-связь», или ER –модель
- Семантическая модель
- Функциональная модель
- Объектно-ориентированная модель.

Остановимся более подробно на моделях, основанных на записях.

В **модели на основе записей** база данных состоит из нескольких записей фиксированного формата, которые могут иметь разные типы. Каждый тип записи определяет фиксированное количество полей, каждое из которых имеет фиксированную длину. Существует три основных типа логических моделей данных на основе записей:

- реляционная модель данных,
- сетевая модель данных,
- иерархическая модель данных.

Иерархическая модель.

Иерархическая модель является ограниченным подтипом сетевой модели. В ней данные также представлены как коллекция записей, а связи – как наборы. Однако, в иерархической модели узел может иметь только одного родителя. Иерархическая модель может быть представлена как древовидный граф с записями в виде узлов и множествами в виде ребер.

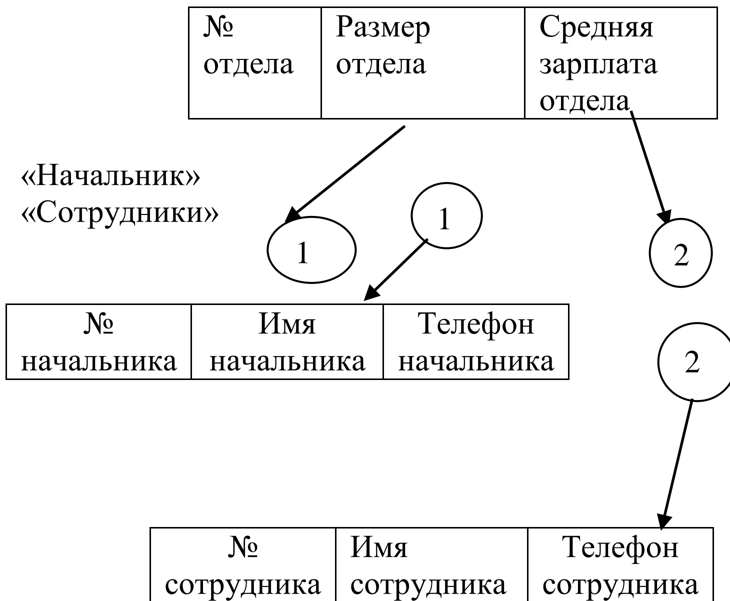
Большинство коммерческих систем основано на реляционной парадигме, тогда как самые первые системы баз данных строились на основе сетевой или иерархической модели. Если в реляционных системах для обработки информации в базе данных принят декларативный подход (т. е. они указывают, какие данные следует извлечь), то в сетевых и иерархических системах – навигационный подход (т. е. они указывают, как их следует извлечь). Таким образом, при работе с реляционной моделью независимость от данных обеспечивается в значительно большей степени.

Иерархическая БД состоит из упорядоченного набора деревьев; более точно, из упорядоченного набора нескольких экземпляров одного типа дерева.

Тип дерева состоит из одного «корневого» типа записи и упорядоченного набора из нуля или более типов поддеревьев

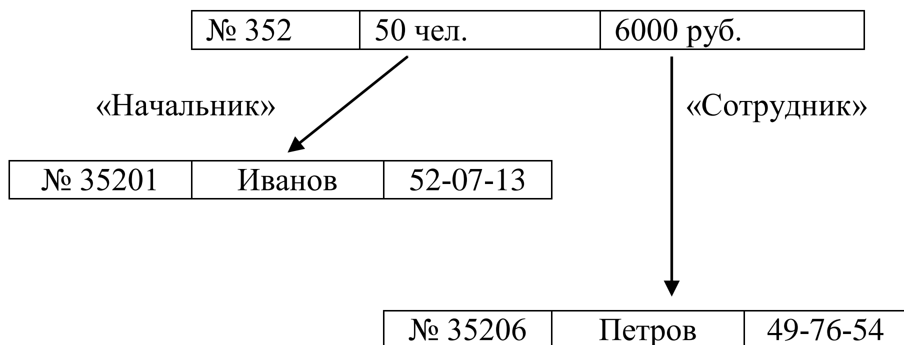
(каждое из которых является некоторым типом дерева). Тип дерева в целом представляет собой иерархически организованный набор типов записи.

Пример типа дерева (схемы иерархической БД):



Здесь запись «Отдел» является предком для записей «Начальник» и «Сотрудники», а записи «Начальник» и «Сотрудники» - потомки записи «Отдел». Между типами записи поддерживаются связи.

База данных с такой схемой могла бы выглядеть следующим образом (один экземпляр дерева):



Все экземпляры данного типа потомка с общим экземпляром типа предка называются близнецами. Для БД определен полный порядок обхода – сверху - вниз, слева - направо.

Манипулирование данными в иерархической модели

Примерами типичных операторов манипулирования иерархически организованными данными могут быть следующие:

- Найти указанное дерево БД (например, отдел 310);
- Перейти от одного дерева к другому;
- Перейти от одной записи к другой внутри дерева (например, от отдела - к первому сотруднику);
- Перейти от одной записи к другой в порядке обхода иерархии;
- Вставить новую запись в указанную позицию;
- Удалить текущую запись.

Ограничения целостности иерархической модели

Автоматически поддерживается целостность ссылок между предками и потомками. *Основное правило: никакой потомок не может существовать без своего родителя.* Заметим, что аналогичное поддержание целостности по

ссылкам между записями, не входящими в одну иерархию, не поддерживается.

Сетевая модель

В ***сетевой модели*** данные представлены в виде коллекции записей, а связи – в виде наборов. В отличие от реляционной модели, связи здесь явным образом моделируются наборами, которые реализуются с помощью указателей. Сетевую модель можно представить как граф с записями в виде узлов графа и наборами в виде его ребер.

Практически по всех СУБД, поддерживающих сетевые модели, между парой типов записей может быть объявлено несколько связей, направление и характер связи в сетевых моделях не являются очевидными, как в случае иерархической модели, поэтому имена и направления связей должны указываться как при графическом изображении БД, так и при ее описании.

Типичным представителем является Integrated Database Management System (IDMS) компании Cullinet Software, Inc., предназначенная для использования на машинах основного класса фирмы IBM под управлением большинства операционных систем. Архитектура системы основана на предложениях Data Base Task Group (DBTG) Комитета по языкам программирования Conference on Data Systems Languages (CODASYL), организации, ответственной за определение языка программирования Кобол. Отчет DBTG был опубликован в 1971 г., а в 70-х годах появилось несколько систем, среди которых IDMS.

Сетевой подход к организации данных является расширением иерархического. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре данных потомок может иметь любое число предков.

Сетевая БД состоит из набора записей и набора связей между этими записями, а если говорить более точно, из набора экземпляров каждого типа из заданного в схеме БД набора

типов записи и набора экземпляров каждого типа из заданного набора типов связи.

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка. Для данного типа связи L с типом записи предка P и типом записи потомка C должны выполняться следующие два условия:

- Каждый экземпляр типа P является предком только в одном экземпляре L;
- Каждый экземпляр C является потомком не более, чем в одном экземпляре L.

На формирование типов связи не накладываются особые ограничения; возможны, например, следующие ситуации:

а) Тип записи потомка в одном типе связи L1 может быть типом записи предка в другом типе связи L2 (как в иерархии).

б) Данный тип записи P может быть типом записи предка в любом числе типов связи.

в) Данный тип записи P может быть типом записи потомка в любом числе типов связи.

д) Может существовать любое число типов связи с одним и тем же типом записи предка и одним и тем же типом записи потомка; и если L1 и L2 - два типа связи с одним и тем же типом записи предка P и одним и тем же типом записи потомка C, то правила, по которым образуется родство, в разных связях могут различаться.

ж) Типы записи X и Y могут быть предком и потомком в одной связи и потомком и предком - в другой.

з) Предок и потомок могут быть одного типа записи.

Простой пример сетевой схемы БД:

Манипулирование данными в сетевой модели

Примерный набор операций может быть следующим:

- Найти конкретную запись в наборе однотипных записей (инженера Сидорова);
- Перейти от предка к первому потомку по некоторой связи (к первому сотруднику отдела 352);
- Перейти к следующему потомку в некоторой связи (от Сидорова к Иванову);
- Перейти от потомка к предку по некоторой связи (найти отдел Сидорова);
- Создать новую запись;
- Уничтожить запись;
- Модифицировать запись;
- Включить в связь;
- Исключить из связи;
- Переставить в другую связь и т.д.

Ограничения целостности в сетевой модели

В принципе их поддержание не требуется, но иногда требуют целостности по ссылкам (как в иерархической модели).

Реляционная модель

Реляционная модель данных основана на понятии математических отношений. В реляционной модели данные и

связи представлены в виде таблиц, каждая из которых имеет несколько столбцов с уникальными именами. В реляционной модели данных единственное требование состоит в том, что база данных с точки зрения пользователя выглядела как набор таблиц. Однако, такое восприятие относится только к логической структуре базы данных, т.е. ко внешнему и концептуальному уровням архитектуры. Оно не имеет отношения к физической структуре базы данных, которая может быть реализована с помощью разнообразных структур хранения.

Подробно реляционная модель рассматривается далее (риздел 3.1).

3. МАТЕМАТИЧЕСКИЕ ОСНОВЫ ПОСТРОЕНИЯ РЕЛЯЦИОННЫХ СУБД. РЕЛЯЦИОННАЯ АЛГЕБРА И БЕЗОПАСНЫЕ ВЫРАЖЕНИЯ. РЕЛЯЦИОННЫЕ ИСЧИСЛЕНИЯ, ПОСТРОЕННЫЕ НА ДОМЕНАХ И КОРТЕЖАХ

3.1. Реляционная модель

Остановимся более подробно на реляционной модели данных.

Реляционная модель впервые была предложена Э.Ф.Коддом в 1970 году. Коммерческие системы на основе реляционной модели данных начали появляться в конце 70-х – начале 80-х годов. В настоящее время существует несколько сотен типов различных реляционных СУБД. Примерами реляционных СУБД являются СУБД Access и FoxPro фирмы Microsoft, Paradox и Visul dBase фирмы Borland, а также R:Base фирмы Microrim.

Реляционная модель основана на математическом понятии отношения, физическим представлением которого является таблица.

Поясним используемую в реляционной модели терминологию, а также основные структурные понятия.

ОТНОШЕНИЕ – это плоская таблица, состоящая из столбцов и строк.

АТРИБУТ – это поименованный столбец отношения

ДОМЕН – это набор допустимых значений для одного или нескольких атрибутов.

КОРТЕЖ – это строка отношения.

Описание структуры отношения вместе со спецификацией доменов и любыми другими ограничениями возможных значений атрибутов иногда называют его **заголовком** (**содержанием**). Кортежи называются

расширением, **состоянием** или **телом отношения**, которое постоянно меняется.

СТЕПЕНЬ ОТНОШЕНИЯ определяется количеством атрибутов, которое оно содержит.

Отношение только с одним атрибутом имеет степень 1 и называется унарным отношением, с двумя атрибутами – бинарным.

КАРДИНАЛЬНОСТЬ - это количество кортежей, которое содержит отношение

Таким образом, можно дать определение реляционной базе данных следующим образом.

РЕЛЯЦИОННАЯ БАЗА ДАННЫХ – это набор нормализованных отношений.

Математические отношения

Для понимания истинного смысла термина **отношение** рассмотрим несколько математических понятий. Допустим, у нас есть два множества, *Д1* и *Д2*, где Д1 = {2,4} и *Д2 = {1,3,5}*. **Декартовым произведением** этих двух множеств *(Д1хД2)* называется набор из всех возможных пар, в которых первым идет элемент множества *Д1*, а вторым – элемент множества *Д2*. Альтернативный способ выражения этого произведения заключается в поиске всех комбинаций элементов, в которых первым идет элемент множества *Д1*, а вторым – множества *Д2*. В данном примере получим следующий результат:

Д1хД2 = {(2,1), (2,3), (2,5), (4,1), (4,3), (4,5)}.

Любое подмножество этого декартового произведения является отношением. Например, в нем можно выделить отношение R , показанное ниже.

$$R = \{(2,1), (4,1)\}.$$

Для определения тех возможных пар, которые будут входить в отношение, можно задать некоторые условия их выборки. Например, отношение R – это отношение, в котором второй элемент равен 1.

$$R = \{(x,y) \mid x\ \ Д1,\ y\ \ Д2 \text{ и } y=1\}.$$

Понятие отношения можно легко распространить и на три множества. Пусть имеется три множества: - *Д1, Д2, Д3*. Декартово произведение *Д1хД2хД3* этих трех множеств является набором, состоящим из всех возможных троек элементов, в которых первым идет элемент множества *Д1*, вторым – элемент множества *Д2*, а третьим – *Д3*. Любое подмножество этого декартового произведения является отношением.

Увеличивая количество множеств, можно дать обобщенное определение отношения на n доменах.

Пусть имеется n множеств *Д1, Д2, ..., Дn*. Декартово произведение для этих n множеств можно определить следующим образом:

$$Д1хД2х\ldots хДn = \{(d1\ \ D1,\ d2\ \ D2, \ldots ,\ dn\ \ Dn)\}.$$

Обычно это выражение записывают в таком символическом виде :

$$\mathop{X}_{i=1}^{n} Di$$

Любое множество n-арных кортежей этого декартового произведения является отношением n множеств. Для определения этих отношений необходимо указать множества, или **домены**, из которых выбираются значения.

Используя указанные концепции в контексте базы данных, мы получим следующее определение.

РЕЛЯЦИОННАЯ СХЕМА - это имя отношения, за которым следует множество пар имен атрибутов и доменов.

Каждый элемент n-арного кортежа состоит из атрибута и значения этого атрибута. Обычно при записи отношения в виде таблицы имена атрибутов перечисляются в заголовках столбцов, а кортежи образуют строки формата (*d1, d2,... , dn*), где каждое значение берется из соответствующего домена. Таким образом, в реляционной модели отношение можно представить, как произвольное подмножество декартового произведения доменов атрибутов, тогда как таблица – это всего лишь физическое представление такого отношения.

Отношение обладает следующими характеристиками.

- Отношение имеет имя, которое отличается от имен всех других отношений.
- Каждая ячейка отношения содержит только атомарное (неделимое) значение.
- Каждый атрибут имеет уникальное имя.
- Значения атрибута берутся из одного и того же домена.
- Порядок следования атрибутов не имеет никакого значения.
- Каждый кортеж является уникальным, т.е. дубликатов кортежей быть не может.
- Теоретически порядок следования кортежей в отношении не имеет никакого значения. (Однако, практически этот порядок может существенно повлиять на эффективность доступа к ним).

Большая часть свойств отношений происходит от свойств математических отношений:

- Поскольку отношение является множеством, то порядок элементов не имеет значения. Следовательно, порядок кортежей в отношении несущественен.

- В множестве нет повторяющихся элементов. Аналогично, отношение не может содержать кортежей-дубликатов.
- При вычислении декартового произведения множеств с простыми однозначными элементами (например, целочисленными значениями), каждый элемент в каждом кортеже имеет единственное значение. Аналогично, каждая ячейка отношения содержит только одно значение. Однако, математическое отношение не нуждается в нормализации.
- Набор возможных значений для данной позиции отношения определяется множеством, или доменом, на котором определяется эта позиция. В таблице все значения в каждом столбце должны происходить от одного и того же домена, определенного для данного атрибута.

Однако, в математическом отношении порядок следования элементов в кортеже имеет значение. Это утверждение неверно для отношения в реляционной модели, где специально оговаривается, что порядок атрибутов несущественен. Дело в том, что заголовки столбцов однозначно определяют, к какому именно атрибуту относится данное значение. Следствием этого факта является положение о том, что порядок следования заголовков столбцов в заголовке отношения несущественен.

Для реализации возможности уникальной идентификации каждого отдельного кортежа отношения по значениям его атрибутов используется понятие ***реляционных ключей***.

СУПЕРКЛЮЧ – это атрибут или множество атрибутов, которое единственным образом идентифицирует кортеж данного отношения.

Поскольку суперключ может содержать дополнительные атрибуты, которые необязательны для уникальной идентификации кортежа, интерес представляют суперключи,

состоящие только из тех атрибутов, которые действительно необходимы для уникальной идентификации кортежей.

ПОТЕНЦИАЛЬНЫЙ КЛЮЧ – это суперключ, который не содержит подмножества, также являющегося суперключом данного отношения.

Потенциальный ключ **К** для данного отношения **R** обладает двумя свойствами:

- ***Уникальность*** **–** в каждом кортеже отношения R значение ключа К единственным образом идентифицируют этот кортеж.
- ***Неприводимость*** – Никакое допустимое подмножество ключа К не обладает свойством уникальности.

Отношение может иметь несколько потенциальных ключей. Если ключ состоит из нескольких атрибутов, то он называется ***составным ключом.***

ПЕРВИЧНЫЙ КЛЮЧ – это потенциальный ключ, который выбран для уникальной идентификации кортежей внутри отношения

Поскольку отношение не содержит кортежей-дубликатов, всегда можно уникальным образом идентифицировать каждую его строку. Это значит, что отношение всегда имеет первичный ключ. В худшем случае все множество атрибутов может использоваться как первичный ключ, но обычно, чтобы различить кортежи, достаточно использовать несколько меньшее подмножество атрибутов. Потенциальные ключи, которые не выбраны в качестве первичного ключа, называются ***альтернативными ключами.***

ВНЕШНИЙ КЛЮЧ – это атрибут или множество атрибутов внутри отношения, которое соответствует потенциальному ключу некоторого (может быть, того же самого) отношения.

Если некий атрибут присутствует в нескольких отношениях, то его наличие обычно отражает определенную связь между кортежами этих отношений.

Модель данных кроме структурной части имеет управляющую часть, которая определяет типы допустимых операций с данными, и набор ограничений целостности, которые гарантируют корректность данных.

Поскольку каждый атрибут связан с некоторым доменом, для множества допустимых значений каждого атрибута отношения определяются так называемые ***ограничения домена.*** Помимо этого, задаются два важных ***правили целостности***, которые, по сути, являются ограничениями для всех допустимых состояний базы данных. Этих два основных правила реляционной модели называются ***целостностью сущностей и ссылочной целостностью***.

ОПРЕДЕЛИТЕЛЬ NULL – указывает, что значение атрибута в настоящий момент неизвестно или неприемлемо для этого кортежа.

Определитель NULL следует воспринимать как логическую величину «неизвестно». Другими словами, либо это значение не входит в область определения некоторого кортежа, либо никакое значение еще не задано. Однако, определитель NULL не следует понимать как нулевое значение или заполненную пробелами текстовую строку.

Первое ограничение целостности касается первичных ключей базовых отношений.

ЦЕЛОСТНОСТЬ СУЩНОСТЕЙ – в базовом отношении ни один атрибут первичного ключа не может содержать отсутствие значений, обозначаемых определителем NULL.

Здесь базовое отношение определяется как отношение, которое соответствует некоторой сущности в концептуальной схеме.

По определению, первичный ключ – это минимальный идентификатор, который используется для уникальной идентификации кортежей. Это значит, что никакое подмножество первичного ключа не может быть достаточным для уникальной идентификации кортежей, а, следовательно, первичный ключ не может содержать NULL.

Второе ограничение целостности качается внешних ключей.

ССЫЛОЧНАЯ ЦЕЛОСТНОСТЬ – если в отношении существует внешний ключ, то значение внешнего ключа должно либо соответствовать значению потенциального ключа некоторого кортежа в его базовом отношении, либо задаваться определителем NULL.

КОРПОРАТИВНЫЕ ОГРАНИЧЕНИЯ ЦЕЛОСТНОСТИ – это дополнительные правила поддержки целостности данных. определяемые пользователем или администраторами базы данных.

Одна из частей модели данных является управляющей, т.е. она определяет типы допустимых операций с данными, включая операции обновления и извлечения данных, а также операции изменения структуры базы данных. Для управления отношениями в реляционных СУБД используются самые разнообразные языки. Некоторые из являются *процедурными*, т.е. с их помощью пользователь точно указывает системе, как следует манипулировать данными. Другие – *непроцедурными,* т.е. пользователь указывает, какие данные ему нужны, а не как их следует извлекать.

Реляционная алгебра и реляционное исчисление являются основой для создания реляционных языков. Реляционную алгебру можно описать как (высокоуровневый) процедурный язык, т. е. тот, который может быть использован для того, чтобы сообщить СУБД о том, как следует построить требуемое отношение на базе одного или нескольких

существующих в базе данных отношений. Реляционное исчисление представляет собой непроцедурный язык, который можно использовать для определения того, каким будет некоторое отношение. Созданное на основе одного или нескольких других отношений базы данных. Однако, строго говоря, реляционная алгебра и реляционное исчисление эквивалентны друг другу, т. е. для каждого выражения алгебры существует эквивалентное выражение в реляционном исчислении (и наоборот).

Реляционная алгебра и реляционное исчисление представляют собой формальные, а не дружественные пользователю языки. В реляционных базах данных они использовались в качестве основы для разработки других языков управления данными, более высокого уровня. Для нас они представляют интерес потому, что иллюстрируют основные операции языков манипулирования данными, а также служат определенным критерием сравнения других реляционных языков.

3.2. Описание учебной базы данных

Для демонстрации операций реляционной алгебры и реляционного исчисления рассмотрим учебную базу данных «Дом – в наем».

Учебный проект

Проект для работы некоторой компании под названием «Дом-–в наем», которая занимается сдачей в аренду объектов недвижимости по поручению их владельцев. Компания предлагает полный комплект услуг владельцам, которые желают сдать в аренду свою меблированную недвижимость в местной или общенациональной прессе, опрос предполагаемых арендаторов, организацию просмотра сдаваемых в аренду объектов, а также составление договоров на аренду. После сдачи недвижимости в аренду на компанию возлагается

ответственность за нее, т. е. сотрудники компании должны регулярно инспектировать текущее состояние объектов.

Требования к данным

1. Компания состоит из нескольких отделений в разных городах (табл. 3.1).

2. Каждое отделение имеет собственный штат сотрудников. В каждом отделении имеется менеджер, отвечающий за работу этого отделения. Для каждого из них фиксируется дата вступления в должность. В зависимости от эффективности работы подразделения каждому менеджеру выплачивается ежемесячная премия и ежегодная компенсация транспортных расходов.

В штат каждого отделения входит несколько инспекторов (старших администраторов), которые отвечают за ежедневную деятельность отдельной группы сотрудников из 5 – 10 человек. В каждой такой группе имеется секретарь.

Каждый сотрудник имеет персональный (табельный) номер, который уникален для всех подразделений компании. О каждом сотруднике хранится информация из таблицы «Персонал» (табл. 3.2).

Таблица 3.1

«Отделения»

№ отделения	Улица	Район	Город	Индекс	Телефон	Факс
В5	Ленинградская	Левобережный	Воронеж	394052	490123	123456
В7	Театральная	Центральный	Липецк	395001	456789	546098
В3	Лесная	Северный	Тамбов	392023	879504	456378
В4	Московская	Северный	Курск	305007	567832	786954
В2	Пушкинская	Центральный	Ст.Оскол	387004	345163	765433

Таблица 3.2

«Персонал»

Личн. №	Имя, отч.	Фами-лия	Адрес	Теле-фон	Долж-ность	Пол	Дата рожд.	Оклад	№ страх	№ отделе ния
45	Юрий Ивано вич	Ивано в	Воронеж, Пр-т Революции, д. 8, кв. 23	56-87-88	менед жер	м	21.89.56	15000	4567990	В5
67	Ольга Сергее вна	Дашко ва	Липецк, ул. Ленина, в.98, кв. 567	33-45-89	секрет арь	ж	30.06.67	2400	4567899	В7

Продолжение табл. 3.2

Личн. №	Имя, отч.	Фами-лия	Адрес	Теле-фон	Долж-ность	Пол	Дата рожд.	Оклад	№ страх	№ отделения
32	Ирина Алексеевна	Берг	Курск, ул. Красина, д.56	54-78-98	инспектор	ж	12.12.57	1470	6543215	В4
78	Сергей Андреевич	Лосев	Воронеж, ул. Беговая, д.89	13-54-78	менеджер	м	09.06.70	2100	5678904	В5
01	Николай Николаевич	Петров	Воронеж, ул. Ленина, д.78, кв.13	56-89-56	менеджер	м	08.11.68	3500	4637892	В5
08	Иван Петрович	Смирнов	Воронеж, ул. Донская, д.156	13-98-99	менеджер	ж	10.05.61	2300	8765435	В5

О секретаре хранится дополнительная информация о скорости печати на пишущей машинке (табл. 3.3).

Таблица 3.3

«Секретари»

Личн. №	Имя, отч.	Фамилия	Скорость
67	Ольга Сергеевна	Дашкова	142
88	Ирина Андреевна	Осипова	100

Кроме того, хранятся сведения об одном ближайшем родственнике (наследнике) каждого сотрудника, включая их имена и фамилии, степени родства, адрес и телефон (табл. 3.4).

Таблица 3.4

«Родственники»

Личн. № сотрудника	Имя, отч.	Фамилия	Адрес	Телефон	Степень родства
45	Иван Иванович	Иванов			отец
67	Сергей Владимирович	Дашков			отец
32	Ирина Ивановна	Берг			мать
78	Сергей Сергеевич	Лосев			брат
01	Николай Николаевич	Петров			отец
08	Инна Юрьевна	Смирнова			жена

Здесь личный № - это личный номер того сотрудника, которому соответствует данный родственник.

3. Сдаваемые в аренду объекты недвижимости

Каждому объекту недвижимости присваивается номер, уникальный для всех отделений (таблица «Объекты» представлена в табл. 3.5) Каждый объект закрепляется за определенным сотрудником компании. Для одного сотрудника – не более 10 объектов одновременно.

При исключении объекта из списка требуется сохранять данные об объектах в течении 3-х лет.

4. Владельцы недвижимости

Компания управляет недвижимостью частных лиц. О каждом частном владельце хранится следующая информация (табл. 3.6): ФИО, адрес, телефон.

5. Клиенты-арендаторы

Когда клиент впервые обращается в отделение фирмы, о нем записываются сведения, представленные в табл. 3.7.

Каждый клиент получает личный номер, уникальный для всех отделений компании.

6. Осмотр недвижимости

Потенциальный арендатор осматривает сдаваемые объекты недвижимости. сведения о таком просмотре включают (табл. 3.8).

Таблица 3.5

«Объекты»

№ объекта	Индекс	Город	Район	Улица	Тип объекта	Кол-во комнат	Арендная плата	№ сотр. за кот. закр. объект	№ владельца
01	394001	Воронеж	Центральный	Плехановская	Квартира	4	15000	01	
09	394029	Воронеж	Левобережный	Ленинградская	Квартира	2	30000	08	
12	305007	Курск	Центральный	Ленина	Квартира	3	10000	32	
11	394001	Воронеж	Центральный	Ленина	Дом	5	20000	01	

Таблица 3.6

«Владелец-лицо»

№ объекта	ФИО	Адрес	Телефон
01	Андреев А.И.	Воронеж, Донская, 56	43-78-66
09	Андреев А.И.	Воронеж, Донская, 56	43-78-66
12	Иванов А.А.	Курск, Ленина, 145, 87	44-77-88
13	Петров И.И.	Липецк, Ранняя, 136	12-88-99
11	Андреев А.И.	Воронеж, Донская, 56	43-78-66
02	Аксенов П.Р.	Воронеж, Плезановская, 35, 123	78-98-67

Таблица 3.7

«Арендатор»

Личн. №	ФИО	Адрес	Телефон	Предп. тип жилья	Max арендная плата	Дата беседы	ФИО сотрудника, провод. собеседов.	Комментарии
01	Сидоров Ю.К.	Воронеж, пр. Революции, 45, 13	56-45-76	Квартира	10000	01.01.03	Лосев	2-й этаж
04	Рачков П.Л.	Воронеж, Ленина, 45	56-98-78	Дом	80000	02.01.03	Лосев	Не менее 3-х комнат
07	Плетнева Р.С.	Курск, Майская, 56, 76	87-87-99	Квартира	30000	13.03.03	Берг	1 комната
12	Воронецева Н.Н.	Воронеж, Матросова, 13, 2	31-99-34	Квартира	70000	08.02.03	Смирнов	Не первый этаж

Таблица 3.8

«Осмотр»

Личн. № аренд.	№ объекта	Дата осмотра	Комментарии
01	01	02.01.03	Нет балкона
01	09	03.02.03	Грязный подъезд
04	11	12.03.03	Требуется ремонт

7. Реклама недвижимости

По каждому сделанному рекламному объявлению сохраняются следующие сведения (табл. 3.9):

Таблица 3.9

«Реклама»

№ рекламир. объекта	Дата рекламы	Стоимость	Название газеты
01	12.12.02	500	«Комммуна»
09	03.07.02	700	«Камелот»
23	23.09.02	150	«Камелот»

8. Договора об аренде

Запись по каждому заключенному договору включает сведения из табл. 3.10.

Номер договора об аренде уникален для всех отделений компании. Клиенты могут арендовать как один, так и сразу несколько объектов недвижимости.

По завершении договора об аренде сведения о нем должны храниться в течение 3-х лет.

9. Инспекция арендованного объекта

Каждый объект проверяется не реже, чем раз в полгода. Инспектируются только арендованные или сдаваемые в аренду объекты. После каждой инспекции фиксируются сведения о проверенном объекте (табл. 3.11).

Таблица 3.10

«Договор»

№ договора	ФИО арендатора	арендная плата	сумма задатка	дата уплаты задатка	дата начала аренды	дата окончания аренды	срок аренды	ФИО сотрудника, состав. договор
01	Сидоров	6000	1000	05.01.03	06.01.03	06.01.04	1 год	Смирнов
02	Рачков	3000	1500	05.12.02	06.12.02	05.06.03	6 мес.	Смирнов

Таблица 3.11

«Инспекция»

№ объекта	ФИО сотрудника-инспектора	Дата инспекции	Комментарии
01	Лосев	15.01.03	Все хорошо
09	Лосев	11.04.03	Сломан диван
11	Смирнов	07.03.03	Не застал жильца

3.3. Операции реляционной алгебры

Реляционная алгебра – это теоретический язык операций, который на основе одного или нескольких отношений позволяют создавать другое отношение без изменений самих исходных отношений. Таким образом, оба операнда и результат являются отношеними, а поэтому результаты одной операции могут стать исходными данными для другой операции. Это позволяет создавать вложенные выражения реляционной алгебры, точно так же, как создаются вложенные арифметические выражения. Реляционная алгебра является языком последовательного использования отношений, в котором все кортежи, возможно, взятые даже из разных отношений, обрабатываются одной командой, без организации циклов.

Существует несколько вариантов выбора операций, которые включаются в реляционную алгебру. Пять основных операций реляционной алгебры, а именно *выборка, проекция, декартово произведение, объединение и разность,* выполняют большинство операций извлечения данных. На основании пяти основных операций можно вывести дополнительные операции, такие как операции *соединения, пересечения и деления*.

Операции выборки и проекции являются **унарными,** поскольку они работают с одним отношением. Другие отношения работают с парами отношений, и поэтому их называют **бинарными** операциями.

В приведенных ниже определениях R и S – это два отношения, определенные над атрибутами A = {a1, a2, …, ak} и B = {в1, в2, …, вk} соответственно.

Выборка (или ограничение)

σ (R) – предикат

Операция выборки работает с одним отношением R и определяет результирующее отношение, которое содержит только те кортежи (строки) отношения R, которые удовлетворяют заданному условию (предикату).

Пример. Составить список сотрудников с окладом, превышающим 3000 рублей.

$$\sigma_{\text{оклад}>3000}(\text{«Персонал»})$$

Результатом операции будет отношение «Персонал» (табл. 3.12).

Проекция

П (R) - атр1, …атр. n

Операция проекции работает с одним отношением R и определяет новое отношение, содержащее вертикальное подмножество отношений R, создаваемое посредством извлечения значений, указанных атрибутов и исключения из результата строк - дубликатов.

Пример.

Создать ведомость зарплаты всех сотрудников компании с указанием следующих атрибутов: Фамилия, Имя, Зарплата.

$$\Pi_{\text{фам, имя, з/п}}(\text{Персонал})$$

В этом примере операция проекции определяет новое отношение, которое будет содержать только атрибуты Фамилия, Имя, Зарплата отношения «Персонал» (табл. 3.13), размещенные в указанном порядке.

Таблица 3.12

Персонал

Личн. №	Имя, отч.	Фамилия	Адрес	Телефон	Должность	Пол	Дата рожд.	Зарп-лата	№ страх	№ отделения
45	Юрий Иванович	Иванов	Воронеж, Пр-т Революции, д. 8, кв. 23	56-87-88	менеджер	м	21.89.56	15000	4567990	В5
01	Николай Николаевич	Петров	Воронеж, ул. Ленина, д.78, кв.13	56-89-56	менеджер	м	08.11.68	3500	4637892	В5

Таблица 3.13

Персонал

Имя, отч.	Фамилия	Зарплата
Юрий Иванович	Иванов	15000
Ольга Сергеевна	Дашкова	2400
Ирина Алекссевна	Берг	1470
Сергей Андреевич	Лосев	2100
Николай Николаевич	Петров	3500
Иван Петрович	Смирнов	2300

Декартово произведение

RxS – операция декартового произведения определяет новое отношение, которое является результатом конкатенации (сцепления) каждого кортежа из отношения R с каждым кортежем из отношения S.

Операторы выборки и проекции извлекают информацию только из одного отношения. Очевидно, возможно возникновение таких ситуаций, когда нужна некоторая комбинация данных из нескольких отношений. Оператор декартового произведения умножает два отношения, что в результате приводит к созданию другого отношения, состоящего из всех возможных пар кортежей обоих отношений.

Пример.

Создать список всех арендаторов, которые осматривали объекты недвижимости. С указанием сделанных ими комментариев.

(П «Арендатор»)) X **(П («Осмотр»)**)**личн.№, ФИО №аренд.,№ объекта, коммент.**

Здесь перемножаются все кортежи двух отношений: первый – список арендаторов, второй – осмотр.

Результатом будет отношение «Арендатор» (табл. 3.14)

Таблица 3.14

Арендатор

Личн. №	ФИО	Личн. № аренд	№ объекта	Комментарии
01	Сидоров Ю.К.	01	01	Нет балкона
01	Сидоров Ю.К.	01	09	Грязный подъезд
01	Сидоров Ю.К.	04	11	Требуется ремонт
04	Рачков П.Л.	01	01	Нет балкона

Продолжение таблицы 3.14

Личн. №	ФИО	Личн. № аренд	№ объекта	Комментарии
04	Рачков П.Л.	01	09	Грязный подъезд
04	Рачков П.Л.	04	11	Требуется ремонт
07	Плетнева Р.С.	01	01	Нет балкона
07	Плетнева Р.С.	01	09	Грязный подъезд
07	Плетнева Р.С.	04	11	Требуется ремонт
12	Воронецева Н.Н.	01	01	Нет балкона
12	Воронецева Н.Н.	01	09	Грязный подъезд
12	Воронецева Н.Н.	04	11	Требуется ремонт

В результате произведения получается отношение, которое содержит больше информации, чем необходимо. Для получения искомого списка необходимо для этого отношения произвести операцию выборки с извлечением тех кортежей, для которых выполняется условие: данный арендатор осматривал данную недвижимость.

Результатом этой операции будет являться отношение «Арендатор» (табл. 3.15)

Таблица 3.15

Арендатор

Личн. №	ФИО	Личн. № аренд	№ объекта	Комментарии
01	Сидоров Ю.К.	01	01	Нет балкона
01	Сидоров Ю.К.	01	09	Грязный подъезд
04	Рачков П.Л.	04	11	Требуется ремон

Как мы увидим позднее, комбинация декартового произведения и выборки может быть сведена к одной операции соединения.

Объединение

R *U* S – объединение отношений R и S с кортежами I и J соответственно можно получить в результате их конкатенации с образованием одного отношения с максимальным количеством кортежей (I + J), если кортежи-дубликаты исключены. При этом отношения R и S должны быть совместимы по объединению.

Объединение отношений возможно только в том случае, если совпадают их схемы, т.е. если они имеют одинаковое количество атрибутов с совпадающими доменами. Иначе говоря, отношения должны быть *совместимы по объединению*. Отметим, что в некоторых случаях для получения двух совместимых по объединению отношений может быть использована операция проекции.

Пример.

Создать список всех городов, в которых имеется отделение компании или объект недвижимости

(П «Отделение»)) ***U*** **(П («Объекты»)**)
город **город**

Результатом объединения будет отношение «Город» (табл. 3.16).

Таблица 3.16

Город

Город
Воронеж
Липецк
Тамбов
Курск
Ст.Оскол

Разность

R – S - разность двух отношений R и S состоит из кортежей, которые имеются в отношении R, но отсутствуют в отношении S. Причем отношения R и S должны быть совместимы по объединению.

Пример.

Создать список всех городов, в которых есть отделения компании, но нет объектов недвижимости, сдаваемых в аренду

$$(\Pi_{\text{город}}\ («Отделение»)) - (\Pi_{\text{город}}\ («Объекты»))$$

Результатом разности будет отношение «Отделение» (табюл. 3.17).

Таблица 3.17

Отделение

Город
Липецк
Тамбов
Ст.Оскол

Операции соединения

Как правило, пользователя интересует лишь некоторая часть всех комбинаций кортежей декартового произведения, которая удовлетворяет заданному условию. Поэтому вместо декартового произведения обычно используется одна из самых важных операций реляционной алгебры – операция соединения. В результате этой операции создается новое отношение. Операция соединения является производной от операции декартового произведения двух операндов – отношений тех кортежей, которые удовлетворяют условию, указанному в предикате соединения в качестве формулы выборки. С точки зрения эффективной реализации в реляционных СУБД, эта операция является одной из самых трудных и часто оказывается

одной из основных причин, вызывающих проблемы с производительностью, свойственной всем реляционным системам.

Существует несколько типов операций соединения:

- Тета-соединения (Θ-join)
- Соединение по эквивалентности (equi-join), которое является частным видом тета-соединения.
- Естественное соединение (natural-join)
- Внешнее соединение (outer-join)
- Полусоединение (semi - join)

Тета-соединение

$\mathbf{R} \bowtie_{F} \mathbf{S}$ –

Операция тета-соединения определяет отношение, которое содержит кортежи из декартового произведения отношений R и S, удовлетворяющие предикату F. Предикат F имеет вид R.a1 Θ S.b1, где вместо Θ может быть указан один из операндов сравнения (<, >, <=, >=, = или ¬=).

Также, как и в случае с декартовым произведением, степенью тета-соединения называется сумма степеней операндов-отношений R и S. Если предикат F содержит только оператор равенства, то соединение называется *соединением по эквивалентности*.

Пример (операция соединения по эквивалентности)

Создать список всех арендаторов, которые осматривали объект недвижимости, с указанием их имен и сделанных ими комментариев.

(П **(«Осмотр»)**)
личн.№, ФИО **Персонал.личн. №= Осмотр.личн.№**
№аренд.,№ объекта, коммент. **аренд.** **аренд.**

Ранее для получения этого списка использовались операции декартового произведения и выборки. однако тот же

самый результат можно получить с помощью операции соединения по эквивалентности.

Результатом этой операции будет являться следующее отношение (табл. 3.18).

Таблица 3.18

Арендатор

Личн. № аренд.	ФИО	Личн. № аренд	№ объекта	Комментарии
01	Сидоров Ю.К.	01	01	Нет балкона
01	Сидоров Ю.К.	01	09	Грязный подъезд
04	Рачков П.Л.	04	11	Требуется ремонт

Естественное соединение

R $\bowtie$ **S** Естественным соединением называется соединение по эквивалентности двух отношений R и S, выполненное по всем общим атрибутам X , из результатов которого исключается по одному экземпляру каждого общего атрибута.

Степенью естественного соединения называется сумма операндов-отношений R и S минус количество атрибутов x.

Пример.

Создать список всех арендаторов, которые осматривали объекты недвижимости, с указанием их имен и сделанных ими комментариев.

В предыдущем примере для составления этого списка использовалось соединение по эквивалентности, но в нем присутствовало два одинаковых атрибута «личный № арендатора».

Для удаления одного из этих атрибутов можно воспользоваться операцией естественного соединения.

$$(\Pi_{\text{личн.№, ФИО}}(\text{«Арендатор»})) \bowtie (\Pi_{\text{№аренд.,№ объекта, коммент.}}(\text{«Осмотр»}))$$

Внешнее соединение

Зачастую при соединении двух отношений кортеж из одного отношения не находит соответствующего кортежа в другом отношении. Иначе говоря, в столбцах соединения оказываются несовпадающие значения. Может потребоваться, чтобы строка из одного отношения была представлена в результате соединения, даже если в другом отношении не совпадающего значения. Эта цель может быть достигнута с помощью внешнего соединения.

R ⊃◁ **S** Левым внешним соединением называется соединение, при котором кортежи отношения R, не имеющие совпадающих значений в общих столбцах отношения S, также включаются в результирующее отношение.

Для обозначения отсутствующих значений во втором отношении используется определитель NULL. Внешнее соединение становится все более распространенным в реляционных СУБД, к тому же в настоящее время оно является оператором. Включенным в новый стандарт SQL. Преимуществом внешнего соединения является то, что при таком соединении сохраняется исходная информация, т.е. внешнее соединение сохраняет кортежи, которые были бы утрачены при использовании других типов соединения.

Пример (Левое внешнее соединение)

Создать отчет о ходе проведении осмотров объектов недвижимости.

В данном случае необходимо создать отношение, состоящее как из перечня осмотренных клиентами объектов недвижимости (с приведением их комментариев по этому поводу), так и перечня объектов недвижимости, которыееще не осматривались. Это можно сделать с помощью следующей операции

(П («Объекты»)) ⊃◁ («Осмотр») № объекта, город, улица

Результатом этой операции будет являться следующее отношение (табл. 3.19).

Таблица 3.19

Объекты

№ объекта	Город	Улица	Личн. № аренд.	Дата осмотра	Комментарии
01	Воронеж	Плехановская	01	02.01.03	Нет балкона
09	Воронеж	Ленинградская	01	03.02.03	Грязный подъезд
12	Курск	Ленина	04	12.03.03	Требуется ремон
11	Воронеж	Ленина	NULL	NULL	NULL

Существует также **правое внешнее соединение**, называемое так потому, что в результирующем отношении содержатся все кортежи правого отношения. Кроме того, существует и **полное внешнее соединение**, в результирующем отношении которого помещаются все кортежи из обоих отношений и в котором для обозначения несовпадающих значений кортежей используются определители NULL.

Полусоединение

R $\triangleright_F$ **S** –

Операция полусоединения определяет отношение, которое содержит те кортежи отношения R, которые входят в соединение отношений R и S.

Преимущество полусоединения заключается в том, что оно позволяет сократить количество кортежей, которое нужно обработать для получения соединения. Это особенно полезно при вычислении соединений в распределенных системах.

Пример.

Создать отчет, содержащий полную информацию обо всех сотрудниках, работающих в отделении компании, расположенном в городе Воронеже.

«Отделения» ▷ «Персонал»
Отделения.личн.№ = Персонал.личн. № and город = «Воронеж»

Результатом этой операции будет являться отношение представленное в табл. 3.20.

Пересечение

R ∩ S Операция пересечения определяет отношение, которое содержит кортежи, присутствующие как в отношении R, так и в отношении S. Отношения R и S должны быть совместимы по объединению

Пример.

Создать список всех городов, в которых имеется отделение компании и объект недвижимости

(П «Отделение»)) ***U*** **(П («Объекты»))**
город **город**

Результатом пересечения будет отношение «Отделение» (табл. 3.21).

Таблица 3.20

Сотрудники

Личн. №	Имя, отч.	Фамилия	Адрес	Телефон	Долж-ность	Пол	Дата рожд.	Зарплата	№ страх	№ отделения
45	Юрий Иванович	Иванов	Воронеж, Пр-т Революции, д. 8, кв. 23	56-87-88	менеджер	м	21.89.56	15000	4567990	В5
78	Сергей Андреевич	Лосев	Воронеж, ул. Беговая, д.89	13-54-78	менеджер	м	09.06.70	2100	5678904	В5
01	Николай Николаевич	Петров	Воронеж, ул. Ленина, д.78, кв.13	56-89-56	менеджер	м	08.11.68	3500	4637892	В5
08	Иван Петрович	Смирнов	Воронеж, ул. Донская, д.156	13-98-99	менеджер	ж	10.05.61	2300	8765435	В5

Таблица 3.21

Отделение

Город
Воронеж
Курск

Деление

Оператор деления может быть полезен в случае запросов особого типа, котрые довольно часто встречаются в приложениях баз данных.

Если A – множество атрибутов отношения R, а B – отношения S, причем B является подмножеством A и C = A - B.

R/S Результатом оператора деления является набор кортежей отношения R, определенных на множестве атрибутов C, которые соответствуют комбинации всех кортежей отношения S.

Этот оператор можно сформулировать на основе других основных операторов

T1 = **П**c (R)

T2 = **П**c ((SxT1) – R)

T = T1 – T2

Правила преобразования операций реляционной алгебры

Пусть есть три отношения:

R, определенное на множестве атрибутов A={A1,...,AN}

S-B={B1,...,BM}

T – C=(C1,…., CP)

p, q, r – предикаты

L, L1, L2, M, M1, M2, N – наборы атрибутов

1. Операция выборки с конъюнктивным предикатом может быть преобразована в последовательность операций выборки по членам конъюнкции (и наоборот).

$$\sigma_{p \wedge q \wedge r}(R) = \sigma_p(\sigma_q(\sigma_r(R)))$$

Этот метод преобразования иногда называют *каскадной выборкой.* Например

$$\sigma_{№=3 \wedge оклад=1500}(Персонал) = \sigma_{оклад=1500}(\sigma_{№=3}(Персонал))$$

2. Правило коммутативности операций выборки

$$\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$$

3. В последовательности операций проекции необходима только последняя из этих операций

$$\Pi_L \Pi_M \ldots \Pi_N(R) = \Pi(R)$$

4. Правило коммутативности операций выборки и проекции

Если предикат p включает только атрибуты, входящие в список проекции, то операции выборки и проекции будут обладать свойством коммутативности.

$$\Pi_{Ai...Am}((\sigma_p(R)) =(\sigma_p(\Pi_{Ai...Am} R))$$

где p {A1, A2,......, Am}

5. Правило коммутативности операции тета-соединения (и декартового произведения)

$$R \bowtie_P S = S \bowtie_P R$$

$$R \times S = S \times R$$

Операции соединения по эквивалентности и естественного соединения являются особыми случаями операции тета-соединения, поэтому приведенное выше правило применимо и к этим двум операциям.

6. Правило коммутативности операций выборки и тета-соединения (или декартового произведения).

Если предикат операции выборки включает атрибуты только одного из соединяемых отношений, то операции выборки и соединения (или декартового произведения) будут обладать свойством коммутативности.

$$\sigma_p(R \bowtie_r S) = \sigma_p(R) \bowtie_r S$$

$$\sigma_p(R \times S) = (\sigma_p(R)) \times S$$

где p {A1, A2,......, An}

В альтернативном случае, когда предикат операции выборки представляет собой конъюнкцию предикатов вида ($p \wedge q$), где предикат p включает в себя атрибуты только отношения R, а предикат q включает атрибуты только отношения S, операция выборки и тета-соединения будут обладать следующим вариантом свойств коммутативности:

$$\sigma_{p \wedge q}(R \bowtie_r S) = \sigma_p(R) \bowtie_r \sigma_q(S)$$

$$\sigma_{p \wedge q}(R \times S) = (\sigma_p(R)) \times (\sigma_q(S))$$

7. Правило коммутативности операций проекции и тета-соединения (или декартового произведения)

Если список атрибутов операции проекции имеет вид

L = L1 U L2 , где в подсписок L1 входят атрибуты только отношения R, а в подсписок L2 - атрибуты только отношения S, то в том случае, когда условие соединения содержит только атрибуты отношения L, операции проекции и тета-соединения будут обладать следующим вариантом свойств коммутативности.

$$\Pi_{L1 \cup L2}(R \bowtie_r S) = \Pi_{L1}(R) \bowtie_r \Pi_{L2}(S)$$

Если условие соединения содержит дополнительные атрибуты, не входящие в отношение L, например, атрибуты

M = M1U M2, где список M1 содержит атрибуты только отношение R, а список M2 – только атрибуты отношения S, то дополнительно потребуется выполнить завершающую операцию проекции:

$$\Pi_{L1 \cup L2}(R \bowtie_r S) = \Pi_{L1 \cup L2}((\Pi_{L1 \cup M1}(R)) \bowtie_r (\Pi_{L2 \cup M2}(S)))$$

8. Правило коммутативности операций объединения и пересечения (но не разности)

$$R \cup S = S \cup R$$

$$R \cap S = S \cap R$$

9. Правило коммутативности операции выборки и операций над отношениями (объединения, пересечения, разности)

$$\sigma_p(R \cup S) = \sigma_p(S) \cup \sigma_p(R)$$

$$\sigma_p(R \cap S) = \sigma_p(S) \cap \sigma_p(R)$$

$$\sigma_p(R - S) = \sigma_p(S) - \sigma_p(R)$$

10. Правило коммутативности операций проекции и объединения.

$$\Pi_L(R \cup S) = \Pi_L(S) \cup \Pi_L(R)$$

11. Правило ассоциативности операций тета-соединения (и декартового произведения).

Операции декартового произведения всегда ассоциативны.

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

$$(R \times S) \times T = R \times (S \times T)$$

12. Правило ассоциативности операций объединения и пересечения (но не операции разности).

$$(R \cup S) \cup T = S \cup (R \cup T)$$

$$(R \cap S) \cap T = S \cap (R \cap T)$$

3.4. Реляционное исчисление

В выражениях реляционной алгебры всегда явно задается некий порядок, а также подразумевается некая стратегия оценки запроса. В реляционном исчислении не существует никакого описания оценки запроса, поскольку в запросе реляционного исчисления указывается, *что* следует извлечь, а не *как*.

Название «Реляционное исчисление» произошло от части символьной логики, которая называется *исчислением предикатов*. В контексте баз данных оно существует в двух формах: в форме *реляционного исчисления кортежей* и в форме *реляционного исчисления доменов*.

В логике первого порядка или теории исчисления предикатов под ***предикатом*** подразумевается истинностная функция с аргументами. При подстановке вместо аргументов значений функция становится выражением, называемым ***суждением,*** которое может быть истинным или ложным.

Если предикат содержит переменную, то у этой переменной должна быть соответствующая область определения.

Если P - предикат, то множество всех значений переменной x, при которых суждение P становится истинным, можно символически записать следующим образом:

{x | P(x)}

Предикаты могут соединяться с помощью логических операторов AND, OR, NOT с образованием составных предикатов.

Реляционное исчисление кортежей

В реляционном исчислении кортежей задача состоит в нахождении таких кортежей, для которых предикат является истинным. Это исчисление основано на ***переменных кортежа***. Переменными кортежа являются такие переменные. Областью определения которых является указанное отношение – т.е. переменные, для которых допустимые значения могут быть только кортежи данного отношения. Понятие «область определения» в данном случае относится не к используемому диапазону значений, а к домену, на котором определены эти значения.

Например, для указания отношения Staff в качестве области определения переменной кортежа S используется следующая форма записи:

RANG OF S IS Saff

Кроме того. Запрос «найти множество всех кортежей S, для которых P(S) является истинным» можно записать следующим образом:

(S | P(S))

Здесь предикат P называется *формулой*.

Для указания количества экземпляров, к которым должен быть применен предикат, в формулах могут использоваться два типа ***кванторов***. **Квантор существования - ∃ и квантор общности - ∀.**

Переменные кортежа называются **свободными переменными,** если они не квалифицируются кванторами ∃ или ∀, в противном случае они называются **связанными переменными**.

Не каждая последовательность формул является допустимой. Допустимыми формулами могут быть только недвусмысленные и небессмысленные последовательности. Формула (или правильно построенная формула) в исчислении предикатов определяется следующими правилами.

- Если P является n-арной формулой, а t1, t2,...,tn – это константы или переменные, то выражение P(t1,t2,...,tn) является правильно построенной формулой.
- Если t1 и t2 являются константами или переменными из одного домена, а Θ представляет собой один из операторов сравнения, то выражение t1 Θ t2 является правильно построенной формулой.
- Если выражения F1 и F2 являются формулами, то их конъюнкция обозначается как F1 ^ F2, дизъюнкция – F1 V F2 и отрицание – как – F1.
- Если выражение F1 является формулой со свободной переменной X, то выражение ∃(X) и ∀F(X) также являются формулами.

Примеры. Реляционное исчисление кортежей

1. Создать список всех менеджеров, зарплата которых превышает 3000 руб

RANG of S is «Персонал»

{S.ФИО | S.должность = «Менеджер» ^ S.оклад > 3000 }

2. Создать список всех сотрудников, которые отвечают за работу с объектами недвижимости в Воронеже.

RANG of S is «Персонал»

RANG of P is «Объекты»

{S | ∃ P (P.личн№ = S.№сотрудника) ^ P.город = «Воронеж»}

Атрибут «№ сотрудника» в отношении «Объекты» содержит номер того сотрудника, который отвечает за работу с данным объектом недвижимости. Этот запрос можно сформулировать иначе: «Для всех сотрудников, данные о которых нужно привести в списке, в отношении «Объекты» имеются кортежи, соответствующие этим сотрудникам, причем значение атрибута «Город» в каждом таком кортеже равно «Воронеж».

3. Создайте список всех сотрудников, которые в данный момент не работают с объектами недвижимости.

RANG of S is «Персонал»
RANG of P is «Объекты»
{S.ФИО | -(∃ P (P.личн№ = S.№сотрудника))}

4. Создать список всех клиентов, осматривавших объекты недвижимости в городе Воронеже, с указанием их имен и комментариев, сделанных по поводу проведенного осмотра

RANG of R is «Арендатор»
RANG of P is «Объекты»
RANG of V is «Осмотр»

{R.ФИО, V.комментарии | ∃ **V** (R.№ = V.№арендатора) ^ ∃ (**V.**№объекта = P.№объекта ^ P.город = «Воронеж»)}

Реляционное исчисление доменов

В реляционном исчислении доменов используются переменные, значения которых берутся из доменов, а не из кортежей отношений. Если P(d1, d2,...,dn) обозначает предикат с переменными d1, d2,...,dn , то множество всех переменных домена d1, d2,...,dn, для которых предикат или формула P(d1, d2,...,dn) истинны, обозначается следующим выражением:

{d1, d2,...,dn | P(d1, d2,...,dn) }

В реляционном исчислении доменов зачастую требуется проверить **условие принадлежности**, чтобы определить, принадлежит ли значение указанному отношению. Выражение R(x, y) считается истинным тогда и только тогда, когда в отношении R имеется кортеж со значениями x и y в двух его атрибутах.

Пример. Реляционное исчисление доменов

1. Найдите имена всех менеджеров, оклад которых превышает 3000 руб.

{фамилия, имя, отчество | ∃ должность, оклад (Персонал(фамилия, должность, оклад) ^ должность = «Менеджер» ^ оклад > 3000)}

2. Создать список всех сотрудников, которые отвечают за работу с объектами недвижимости в Воронеже.

{фамилия, имя, отчество | ∃ личный № Персонал(личный № , фамилия, имя, отчество) ^ ∃ город (Объекты (№ объекта, личный №) ^ город = «Воронеж»)}

Эти запросы безопасны, и, если реляционное исчисление доменов ограничивается только такими выражениями, оно эквивалентно реляционному исчислению кортежей, ограниченному только безопасными выражениями, которое, в свою очередь, эквивалентно реляционной алгебре. Это означает, что для каждого выражения реляционной алгебры существует эквивалентное выражение реляционного исчисления, а для каждого выражения реляционного исчисления доменов или кортежей существует эквивалентное выражение реляционной алгебры.

3.5. Задания для самостоятельной работы

Задание 1.

Даны отношения, моделирующие работу банка и его филиалов. Клиент может иметь несколько счетов, при этом они могут быть размещены как в одном, так и в разных филиалах банка. В отношении R_1 содержится информация обо всех клиентах и их счетах в филиалах нашего банка (табл. 3.22). Каждый клиент, в соответствии со своим счетом, может рассчитывать на некоторый кредит от нашего банка, сумма допустимого кредита также зафиксирована.

Таблица 3.22

Клиент

R1				
ФИО клиента	№ филиала	№ счета	Остаток	Кредит

Использованием языка реляционной алгебры составить запросы, позволяющие не выбрать:

1) Филиалы, клиенты которых имеют счета с остатком, превышающим $1000.
2) Клиентов, которые имеют счета во всех филиалах данного банка.
3) Клиентов, которые имеют только по одному счету в разных филиалах банка. То есть в общем у этих клиентов может быть несколько счетов, но в одном филиале не более одного счета.
4) Клиенты, которые имеют счета в нескольких филиалах банка, расположенных только в одном районе.
5) Филиалы, которые не имени ни одного клиента.
6) Филиалы, которые имеют клиентов с остатком на счету 0 (ноль).
7) Филиалы, у которых есть клиенты с кредитом, превышающим остаток на счету в 2 раза.

Задание 2.

Дины отношения, моделирующие работу международной фирмы, имеющей несколько филиалов. Филиалы фирмы могут быть расположены в разных странах, это отряжено в отношении R_1 (табл. 3.23). Клиенты фирмы также могут быть из разных стран, и это отражено в отношении R_4 (табл. 3.26). По каждому конкретному заказу клиент мог заказать несколько разных товаров (табл. 3.24, 3.25).

Таблица 3.23

«Филиал»

R1	
Филиал	Страна

Таблица 3.24

«Филиал»

R2		
Филиал	Заказчик	№ заказа

Таблица 3.25

«Заказ»

R3		
№ заказа	Товар	Количество

Таблица 3.26

«Заказчик»

R4	
Заказчик	Страна

С использованием реляционной алгебры составить запросы, позволяющие выбрать:

1) Заказчиков, которые работают со всеми филиалами фирмы, но покупают только один товар.

2) Филиалы фирмы, которые торгуют всеми товарами.

3) Товары, которые работают с филиалами фирмы, которые расположены только в одной стране.

4) Заказчиков, которые работают с филиалами фирмы, которые расположены только в одной стране.

5) Филиалы, с которыми не работает ни один заказчик.

6) Заказчиков, которые работают только с филиалами, расположенными в той же стране, что и заказчик.

7) Заказчиков, которые покупают все товары, представленные в отношении R_3.

Задание 3.

Даны отношения, моделирующие работу фирмы, занимающейся разработкой программных систем (табл. 3.27-3.29). Каждый сотрудник административно закреплен только за одним отделом. Файлы хранятся на разных серверах. На разных серверах файлы могут иметь одинаковые имена. Создатель файла является его владельцем, поэтому у каждого файла только один владелец, но владелец файла может разрешить пользоваться файлом другим сотрудникам. Существует множество системного программного обеспечения, каждая программа может работать с одним или с несколькими файлами, расположенными на одном или нескольких серверах.

Таблица 3.27

«Сотрудник»

R1	
Название файла	Имя владельца файла

R4	
Сотрудник	Отдел

Таблица 3.28

«Программа»

R2		
Название программы	Название файла	Сервер

Таблица 3.29

«Файл»

R3	
Название файла	Название сервера

С использованием реляционной алгебры составить запросы, позволяющие выбрать:

1. Файлы, которые имеют нескольких пользователей из разных отделов.
2. Программы, которые работают только с одним файлом.
3. Файлы, которые имеют одно и тоже имя, но расположены на различных серверах и используются сотрудниками разных отделов.
4. Файлы, с которыми работают сотрудники всех отделов.
5. Файлы, пользователями которых являются сотрудники только одного отдела.
6. Программы, которые работают со всеми серверами.
7. Отделы, сотрудники которых не работают ни с одним файлом. То есть отделы, в которых нет ни одного сотрудника, работающего с каким-нибудь файлом.
8. Отделы, сотрудники которых работают со всеми серверами.
9. Серверы, с которыми работают сотрудники только одного отдела.

4. ЗАДАЧИ И ЭТАПЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ. ИСПОЛЬЗОВАНИЕ НОРМАЛЬНЫХ ФОРМ ПРИ ПРОЕКТИРОВАНИИ ПРИЛОЖЕНИЙ В РЕЛЯЦИОННЫХ СУБД. ЭТАПЫ НОРМАЛИЗАЦИИ ОТНОШЕНИЙ. МЕТОДОЛОГИИ ПРОЕКТИРОВАНИЯ

4.1. Этапы проектирования баз данных

Проектирование реляционных БД на основе принципов нормализации.

Что такое проект? Это схема - эскиз некоторого устройства, который в дальнейшем будет воплощен в реальность. Что такое проект реляционной базы данных? Это набор взаимосвязанных отношений, в которых определены все атрибуты, заданы первичные ключи отношений и заданы еще некоторые дополнительные свойства отношений, которые относятся к принципам поддержки целостности. Почему именно взаимосвязанных отношений? Потому что при выполнении запросов мы производим объединение отношений, и одни и те же значения должны в разных отношениях-таблицах обозначаться одинаково. Действительно, если мы в одной таблице оценки будем обозначать цифрами, а в другой словами «отлично», «хорошо» и т. д., то мы не сможем объединить эти таблицы по столбцу Оценка, хотя по смыслу это для нас одно и то же, но то, что интуитивно понятно человеку, совсем не понятно «умному» компьютеру. Это проблема систем с искусственным интеллектом, которые могут решать весьма сложные интеллектуальные задачи, трудные для рядового инженера, но иногда пасуют перед простейшими интуитивными ассоциациями, понятными любому школьнику. И это необходимо учитывать. Поэтому проект базы данных должен быть очень точен и выверен. Фактически проект базы данных – это фундамент будущего программного комплекса, который будет использоваться достаточно долго и многими пользователями. И как в любом здании, можно достраивать мансарды, переделывать крышу, можно даже менять окна, но

заменить фундамент, не разрушив всего здания, невозможно. Этапы жизненного цикла базы данных изображены на рис. 4.1. Они аналогичны, в основном, развитию любой программной системы, однако в них есть определенная специфика, касающаяся только баз данных. Более подробно мы будем рассматривать этапы жизненного цикла БД в следующих разделах учебного пособия.

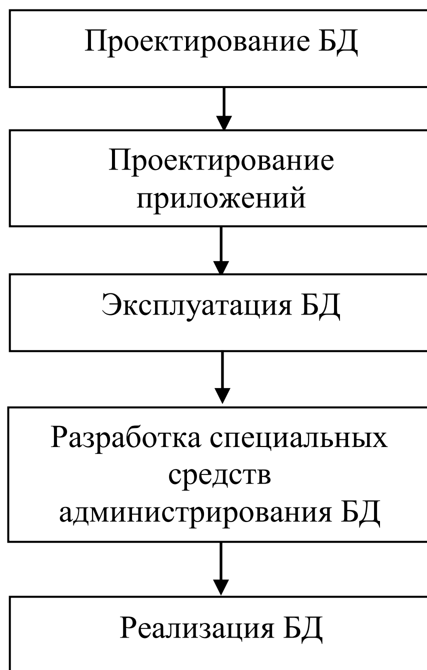


Рис. 4.1. Этапы жизненного цикла БД

Процесс проектирования БД представляет собой последовательность переходов от неформального словесного описания информационной структуры предметной области к формализованному описаний объектов предметной области в терминах некоторой модели. В общем случае можно выделить следующие этапы проектирования:

1. Системный анализ и словесное описание информационных объектов предметной области.

2. Проектирование инфологической модели предметной области - частично формализованное описание объектов предметной области в терминах некоторой семантической модели, например, в терминах Е-модели.

3. Даталогическое или логическое проектирование БД, то есть описание- БД в терминах принятой даталогической модели данных.

4. Физическое проектирование БД, то есть выбор эффективного размещении БД на внешних носителях для обеспечения наиболее эффективной работы приложения.

Если мы учтем, что между вторым и третьим этапами необходимо принять, решение, с использованием какой стандартной СУБД будет реализовываться наш проект, то условно, процесс проектирования БД можно представить последовательностью выполнения пяти соответствующих этапов. Рассмотрим более подробно этапы проектирования БД. (рис. 4.2).

Системный анализ предметной области

С точки зрения проектирования БД в рамках системного анализа, необходимо осуществить первый этап, то есть провести подробное словесное описание объектов предметной области и реальных связей, которые присутствуют между описываемыми объектами. Желательно, чтобы данное описание позволяло корректно определить все взаимосвязи между объектами предметной области.

В общем случае существуют два подхода к выбору состава и структуры предметной области:

Функциональный подход - он реализует принцип движения «от задач» и применяется тогда, когда заранее известны функции некоторой группы лиц и комплексов задач, для обслуживания информационных потребностей которых создается рассматриваемая БД. В этом случае мы можем четко

выделить минимальный необходимый набор объектов предметной области, которые должны быть описаны.

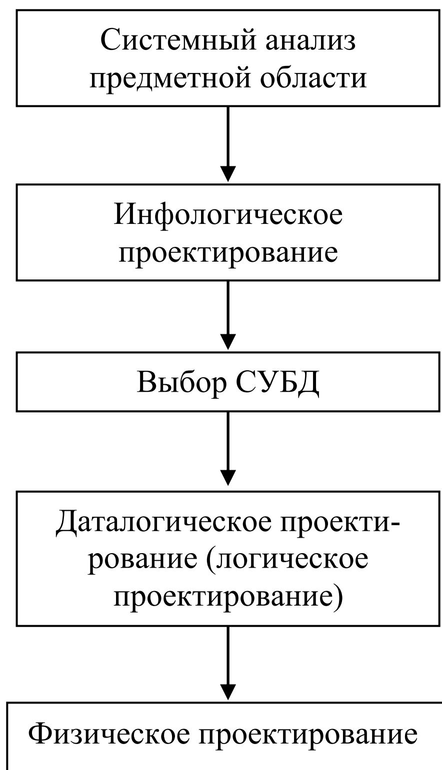


Рис. 4.2. Этапы проектирования базы данных

Предметный подход - когда информационные потребности будущих пользователей БД жестко не фиксируются. Они могут быть многоаспектными и весьма динамичными. Мы не можем точно выделить минимальный набор объектов предметной области, которые необходимо описывать. В описание предметной области в этом случае включаются такие объекты и взаимосвязи, которые наиболее характерны и наиболее существенны для нее. БД, конструируемая при этом, называется предметной, то есть она

может быть использована при решении множеств разнообразных, заранее не определенных задач. Конструирование предметной БД и некотором смысле кажется гораздо более заманчивым, однако трудность всеобщего охвата предметной области с невозможностью конкретизации потребностей пользователей может принести к избыточно сложной схеме БД, которая для конкретных задач будет неэффективной.

Чаще всего на практике рекомендуется использовать некоторый компромиссный вариант, который, с одной стороны, ориентирован на конкретные задачи или функциональные потребности пользователей, а с другой стороны, учитывает возможность наращивания новых приложений.

Системный анализ должен заканчиваться подробным описанием информации об объектах предметной области, которая требуется для решения конкретных задач, и которая должна храниться в БД, формулировкой конкретных задач, которые будут решаться с использованием данной БД с кратким описанием алгоритмов их решения, описанием выходных документов, которые должны генерироваться в системе, описанием входных документов, которые служат основанием для заполнения данными БД.

Пример описания предметной области

Пусть требуется разработать информационную систему для автоматизации учета получения и выдачи книг в библиотеке. Система должна предусматривать режимы ведения системного каталога, отражающего перечень областей знаний, по которым имеются книги в библиотеке. Внутри библиотеки области знаний в систематическом каталоге могут иметь уникальный внутренний номер и полное наименование. Каждая книга может содержать сведения из нескольких областей знаний. Каждая книга в библиотеке может присутствовать в нескольких экземплярах. Каждая книга, хранящаяся в библиотеке, характеризуется следующими параметрами:

- уникальный шифр;

- название;
- фамилии авторов (могут отсутствовать);
- место издания (город);
- издательство;
- год издания;
- количество страниц;
- стоимость книги;
- количество экземпляров книги в библиотеке.

Книги могут иметь одинаковые названия, но они различаются по своему уникальному шифру (ISBN).

В библиотеке ведется картотека читателей.

На каждого читателя в картотеку заносятся следующие сведения:

- фамилия, имя, отчество,
- домашний адрес;
- телефон (будем считать, что у нас два телефона - рабочий и домашний);
- дата рождения.

Каждому читателю присваивается уникальный номер читательского билета.

Каждый читатель может одновременно держать на руках не более 5 книг. Читатель не должен одновременно держать более одного экземпляра книги одного названия.

Каждая книга в библиотеке может присутствовать в нескольких экземплярах. Каждый экземпляр имеет следующие характеристики:

- уникальный инвентарный номер;
- шифр книги, который совпадает с уникальным шифром из описания книг;
- место размещения в библиотеке.

В случае выдачи экземпляра, книги читателю в библиотеке хранится специальный вкладыш, в котором должны быть записаны следующие сведения:

- номер билета читателя, который взял книгу;
- дата выдачи книги;
- дата возврата.

Предусмотреть следующие ограничения на информацию в системе:

1. Книга может не иметь ни одного автора.
2. В библиотеке должны быть записаны читатели не моложе 17 лет.
3. В библиотеке присутствуют книги, изданные начиная с 1960 по текущий год.
4. Каждый читатель может держать на руках не более 5 книг.
5. Каждый читатель при регистрации в библиотеке должен дать телефон для связи: он может быть рабочим или домашним.
6. Каждая область знаний может содержать ссылки на множество книг, но каждая книга может относиться к различным областям знаний.

С данной информационной системой должны работать следующие группы пользователей:

- библиотекари;
- читатели;
- администрация библиотеки.

При работе с системой библиотекарь должен иметь возможность решать следующие задачи:

1. Принимать новые книги и регистрировать их в библиотеке.
2. Относить книги к одной пли к нескольким областям знаний.
3. Проводить каталогизацию книг, то есть назначение новых инвентарных номеров, примятым книгам, и, помещая их

на полки библиотеки, запоминать место размещения каждого экземпляра.

4. Проводить дополнительную каталогизацию, если поступило несколько экземпляров книги, которая уже есть в библиотеке, при этом информации о книге в предметный каталог не вносится, а каждому новому экземпляру присваивается новый инвентарный номер и для него определяется место на полке библиотеки.

5. Проводить списание старых и не пользующихся спросом книг. Списывать можно только книги, ни один экземпляр которых не находится у читателей. Списание проводится по специальному акту списания, который утверждается администрацией библиотеки.

6. Вести учет выданных книг читателям, при этом предполагается два режима работы: выдача книг читателю и прием от него возвращаемых им книг обратно в библиотеку. При выдаче книг фиксируется, когда и какой экземпляр книги был выдан данному читателю и к какому сроку читатель должен вернуть этот экземпляр книги. При выдаче книг наличие свободного экземпляра и его конкретный номер могут определяться по заданному уникальному шифру книги или инвентарный номер может быть известен заранее. Не требуется вести «историю» чтения книг, то есть требуется отражать только текущее состояние библиотеки. При приеме книги, возвращаемой читателем, проверяется соответствие возвращаемого инвентарного номера книги выданному инвентарному номеру, и она ставится на свое старое место на полку библиотеки,

7. Проводить списание утерянных читателем книг по специальному акту списания или замены, подписанному администрацией библиотеки.

8. Проводить закрытие абонемента читателя, то есть уничтожение данных о нем, если читатель хочет выписаться из библиотеки и не является ее должником, то есть за ним не числится ни одной библиотечной книги.

Читатель должен иметь возможность решать следующие задачи:

1. Просматривать системный каталог, то есть перечень всех областей знаний, книги по которым есть в библиотеке.

2. По выбранной области знаний получить полный перечень книг, которые числятся в библиотеке.

3. Для выбранной книги получить инвентарный номер свободного экземпляра книги или сообщение о том, что свободных экземпляров книги нет. В случае отсутствия свободных экземпляров книги читатель должен иметь возможность узнать дату ближайшего предполагаемого возврата экземпляра данной книги. Читатель не может узнать данные о том, у кого в настоящий момент экземпляры данной книги находятся на руках (в целях обеспечения личной безопасности держателей требуемой книги).

4. Для выбранного автора получить список книг, которые числятся в библиотеке.

Права администрации библиотеки представленны на рис. 4.3.

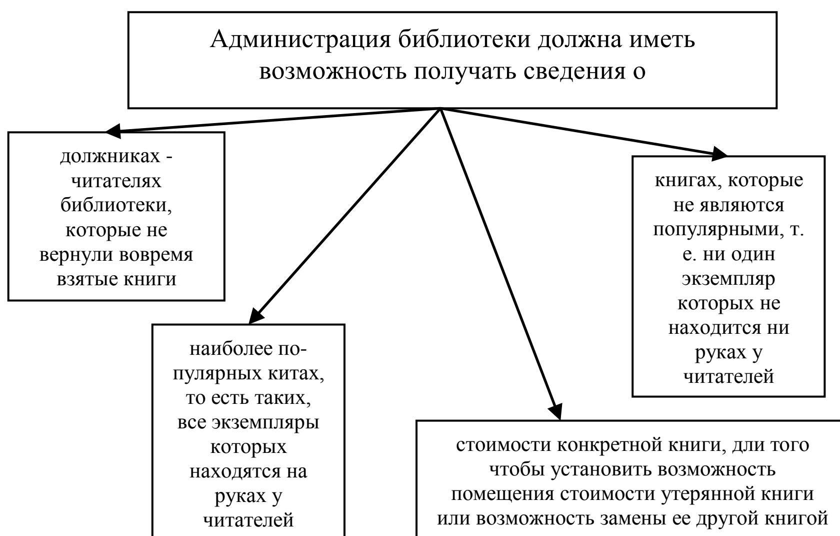


Рис. 4.3. Права администрации библиотеки

Этот совсем небольшой пример показывает, что перед началом разработки необходимо иметь точное представление о том, что же должно выполняться в нашей системе, какие пользователи в ней будут работать, какие задачи будет решать каждый пользователь. И это правильно, ведь когда мы строим здание, мы тоже заранее предполагаем: для каких целей оно предназначено, в каком климате оно будет стоять, на какой почве, и в зависимости от этого проектировщики могут предложить нам тот или иной проект. Но, к сожалению, очень часто по отношению к базам данных считается, что все можно определить потом, когда проект системы уже создан. Отсутствие четких целей создания БД может свести на нет все усилия разработчиков, и проект БД получится «плохим», неудобным, не соответствующим ни реально моделируемому объекту, ни задачам, которые должны решаться с использованием данной БД.

Отложим на время рассмотрение этапа инфологического моделирования предметной области - этому серьезному вопросу будет посвящена следующая, седьмая глава, а мы пойдем классическим путем и рассмотрим сначала этап даталогического проектирования. Напомним, что этап даталогического проектирования происходит уже после выбора конкретной модели данных. И мы рассматриваем диалогическое проектирование для реляционной модели данных.

Даталогическое проектирование

В реляционных БД даталогическое или логическое проектирование приводит к разработке схемы БД, то есть совокупности схем отношений, которые адекватно моделируют абстрактные объекты предметной области и семантические связи между этими объектами. Основой анализа корректности схемы являются так называемые функциональные зависимости между атрибутами БД. Некоторые зависимости между атрибутами отношений являются нежелательными из-за побочных эффектов и аномалий, которые они вызывают при

модификации БД. При этом под процессом модификации БД мы понимаем внесение новых данных в БД или удаление некоторых данных из БД, а также обновление значений некоторых атрибутов.

Однако этап логического или даталогического проектирования не заканчивается проектированием схемы отношений. В общем случае в результате выполнения этого этапа должны быть получены следующие результирующие документы:

Описание концептуальной схемы БД в терминах выбранной СУБД.

Описание внешних моделей в терминах выбранной СУБД.

Описание декларативных правил поддержки целостности базы данных.

Разработка процедур поддержки семантической целостности базы данных.

Однако перед тем ты описывать построенную схему в терминах выбранной СУБД, ним надо выстроить эту схему, именно этому процессу и посвящен данный раздел.

Мы должны построить корректную схему БД, ориентируясь на реляционную модель данных.

Корректной назовем схему БД, и которой отсутствуют нежелательные зависимости между атрибутами отношений.

Процесс разработки корректной схемы реляционной БД называется логическим проектированием БД

Проектирование схемы БД может быть выполнено двумя путями:

путем декомпозиции (разбиения), когда исходное множество отношений, исходящих в схему БД заменяется другим множеством отношений (число их при этом возрастает), являющихся проекциями исходных отношений;

путем синтеза, то есть путем компоновки из заданных исходных элементарных зависимостей между объектами предметной области схемы БД.

4.2. Нормализация отношений

Классическая технология проектирования реляционных баз данных связана с теорией нормализации, основанной на анализе функциональных зависимостей между атрибутами отношений. Понятие функциональной зависимости является фундаментальным в теории нормализации реляционных баз данных. Мы определим его далее, а пока коснемся смысла этого понятия. Функциональные зависимости определяют устойчивые отношения между объектами и их свойствами в рассматриваемой предметной области. Именно поэтому процесс поддержки функциональных зависимостей, характерных для данной предметной области, является базовым для процесса проектирования.

Процесс проектирования с использованием декомпозиции представляет собой процесс последовательной нормализации схем отношений, при этом каждая последующая итерация соответствует нормальной форме более высокого уровня и обладает лучшими свойствами по сравнению с предыдущей.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

В теории реляционных БД обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);

- пятая нормальная форма, или форма проекции-соединения (5NF или PJNF)

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле улучшает свойства предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

В основе классического процесса проектирования лежит последовательность переходов от предыдущей нормальной формы к последующей. Однако в процессе декомпозиции мы сталкиваемся с проблемой *обратимости*, то есть возможности восстановления исходной схемы. Таким образом, декомпозиция должна сохранять *эквивалентность* схем БД при замене одной схемы на другую.

Схемы БД называются ***эквивалентными***, если содержание исходной БД может быть получено путем естественного соединения отношений, входящих в результирующую схему, и при этом не появляется новых кортежей в исходной БД.

При выполнении эквивалентных преобразовании сохраняется множество исходных фундаментальных функциональных зависимостей между атрибутами отношений.

Функциональные зависимости определяют не текущее состояние БД, а все возможные ее состояния, то есть они отражают те связи между атрибутами, которые присущи реальному объекту, который моделируется с помощью БД. Поэтому определить функциональные зависимости по текущему состоянию БД можно только в том случае, если экземпляр БД содержит абсолютно полную информацию (то есть никаких добавлений и модификации БД не предполагает-ся), Б реальной жизни это требование невыполнимо, поэтому

набор функциональных зависимостей задает разработчик, системный аналитик, исходя из глубокого системного анализа предметной области.

Приведем ряд основных определений.

Функциональной зависимостью набора атрибутов B отношения R от набора атрибутов A того же отношения, обозначаемой как

$$R.A \text{ -> } R.B \text{ или } A \text{ -> } B$$

называется такое соотношение проекций R[A] и R[B], при котором в каждый момент времени любому элементу проекции R[A] соответствует только один элемент проекции R[B], входящий вместе с ним в какой-либо кортеж отношения R.

Функциональная зависимость R.A -> R.B называется *полной*, если набор атрибутов B функционально зависит от A и не зависит функционально от любого подмножества A, то есть

R. A -> R.B называется *полной*, если:

$$\forall A1 \subseteq A \Rightarrow R.A \text{ -/-> } R.B,$$

что читается следующим образом:

для любого A1, являющегося подмножеством A, R.B функционально не зависит от R. A1, в противном случае зависимость R.A -> R.B называется неполной.

Функциональная зависимость R.A -> R.B называется транзитивной, если существует набор атрибутов C такой, что:

1. C не является подмножеством A.
2. C не включает в себя B.
3. Существует функциональная зависимость R.A-> R.C.
4. Не существует функциональной зависимости R.C-> R.A.
5. Существует функциональная зависимость R.C -> R.B.

Тогда уже знакомое нам определение потенциального ключа можно дать следующим образом: потенциальным

ключем называется набор атрибутов отношения, который полностью и однозначно (функционально полно) определяет значения всех остальных атрибутов отношения, то есть возможный ключ - это набор атрибутов, однозначно определяющий кортеж отношения, и при этом при удалении любого атрибута из этого набора его свойство однозначной идентификации кортежа теряется.

Взаимно-независимые атрибуты - это такие атрибуты, которые не зависят функционально один от другого.

Если в отношении существует несколько функциональных зависимостей, то каждый атрибут или набор атрибутов, от которого зависит другой атрибут, называется *детерминантом* отношения.

Для функциональных зависимостей как фундаментальной основы проекта БД были проведены исследования, позволяющие избежать избыточного их представления. Ряд зависимостей могут быть выведены из других путем применения правил, названных аксиомами Армстронга, по имени исследователя, впервые сформулировавшего их. Это три основных аксиомы:

Рефлексивность: если B является подмножеством A, то A->B.

Дополнение: если A->B , то AC->BC.

Транзитивность: если A->B B->C , то A->C.

Доказано, что данные правила являются полными и исчерпывающими, то есть, применяя их, из заданного множества функциональных зависимостей можно вывести все возможные функциональные зависимости.

Множество всех возможных функциональных зависимостей, выводимое из заданного набора исходных функциональных зависимостей, называется его замыканием.

Отношение находится в **первой нормальной форме** тогда и только тогда, когда на пересечении каждого столбца и каждой строки находятся только элементарные значения атрибутов.

В некотором смысле это определение избыточно, потому что собственно оно определяет само отношение в теории реляционных баз данных. Однако в силу исторически сложившихся обстоятельств и для преемственности такое определение первой нормальной формы существует и мы должны с ним согласиться. Отношения, находящиеся в первой нормальной форме, часто называют просто нормализованными отношениями. Соответственно, ненормализованные отношения могут интерпретироваться как таблицы с неравномерным заполнением, например таблица «Расписание» (табл. 4.1).

Здесь на пересечении одной строки и одного столбца находится целый набор элементарных значений, соответствующих набору дней, перечню пар, набору дисциплин, по которым проводит занятия один преподаватель.

Для приведения отношения «Расписание» к первой нормальной форме необходимо дополнить каждую строку фамилией преподавателя (табл. 4.2).

Отношение находится во **второй нормальной форме** тогда и только тогда, когда оно находится в первой нормальной форме и не содержит неполных функциональных зависимостей непервичных атрибутов от атрибутов первичного ключа.

Таблица 4.1

«Расписание»

Преподаватель	День недели	Номер пары	Название дисциплины	Тип занятий	Группа
Петров В. И.	Понед. Вторник Вторник	1 1 2	Теор. выч. проц. Комп. графика Комп. графика	Лекция Лаб. раб. Лаб. раб.	4906 4907 4906
Киров В. А.	Понед. Вторник Вторник	2 3 4	Теор. информ. Пр-е на C++ Пр-е на C++	Лекция Лаб. раб. Лаб. раб.	4906 4907 4906
Серов А. А.	Понед. Среда Четверг	3 3 4	Защита информ. Пр-е на VB Пр-е на VB	Лекция Лаб. раб. Лаб. раб.	4944 4942 4922

Таблица 4.2

«Расписание»

Преподаватель	День недели	Номер пары	Название дисциплины	Тип занятий	Группа
Петров В. И.	Понед.	1	Теор. выч. проц.	Лекция	4906
Петров В. И.	Вторник	1	Комп. графика	Лаб. раб.	4907
Петров В. И.	Вторник	2	Комп. графика	Лаб. раб.	4906
Киров В. А.	Понед.	2	Теор. информ.	Лекция	4906
Киров В. А.	Вторник	3	Пр-е на C++	Лаб. раб.	4907
Киров В. А.	Вторник	4	Пр-е на C++	Лаб. раб.	4906
Серов А. А.	Понед.	3	Защита информ.	Лекция	4944
Серов А. А.	Среда	3	Пр-е на VB	Лаб. раб.	4942
Серов А. А.	Четверг	4	Пр-е на VB	Лаб. раб.	4922

Рассмотрим отношение, моделирующее сдачу студентами текущей сессии. Структура этого отношения определяется следующим набором атрибутов:

(ФИО. Номер зач. кн. Группа. Дисциплина. Оценка)

Так как каждый студент сдаст целый набор дисциплин в процессе сессии, то первичным ключом отношения может быть (Номер, зач.кн., Дисциплина), который однозначно определяет каждую стоку отношения. С другой стороны, атрибуты ФИО и Группа зависят только от части первичного ключа - от значения атрибута Номер зач. кн., поэтому мы должны констатировать наличие неполных функциональных зависимостей в данном отношении. Для приведения данного отношения ко второй нормальной форме следует разбить его на проекции, при этом должно быть соблюдено условие восстановления исходного отношении без потерь (рис. 4.4).

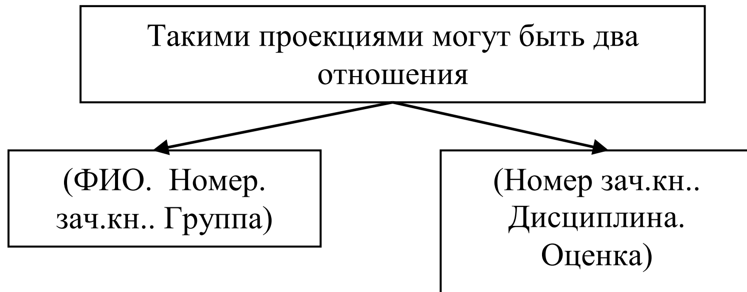


Рис. 4.4. Проекции из набора отношений

Этот набор отношений не содержит неполных функциональных зависимостей, и поэтому эти отношения находятся во второй нормальной форме.

А почему надо приводить отношения ко второй нормальной форме? Иначе говоря, какие аномалии или неудобства могут возникнуть, если мы оставим исходное отношение и не будем его разбивать на два? Давайте

рассмотрим ситуацию, когда студент переведен из одной группы в другую. Тогда в первом случае (если мы не разбивали исходное отношение на два) мы должны найти все записи с данным студентом и в них изменить значение атрибута Группа на новое. Во втором же случае меняется только один кортеж в первом отношении. И конечно, опасность нарушения корректности (непротиворечивости содержании) БД в первом случае выше. Может получиться так, что часть кортежей поменяет значение атрибута Группа, а часть по причине сбоя в работе аппаратуры останется в старом состоянии. И тогда наша БД будет содержать записи, которые относят одного студента одновременно к разным группам. Чтобы это не произошло, мы должны принимать дополнительные непростые меры, например, организовывать процесс согласованного изменения с использование сложного механизма транзакций, который мы будем рассматривать, в главах, посвященных вопросам распределенного доступа к БД. Если же мы перешли ко второй нормальной форме, то мы меняем только один кортеж. Кроме того, если у нас есть студенты, которые еще не сдавали экзамены, то в исходном отношении мы вообще не можем хранить о них информацию, а во второй схеме информация о студентах и их принадлежности к конкретной группе хранится отдельно от информации, которая связана со сдачей экзаменов, и поэтому мы можем в этом случае отдельно работать со студентами и отдельно хранить и обрабатывать информацию об успеваемости и сдаче экзаменов, что в действительности и происходит.

Отношение находится в **третьей нормальной форме** тогда и только тогда, когда оно находится во второй нормальной форме и не содержит транзитивных зависимостей.

Рассмотрим отношение, связывающее студентов с группами, факультетами и специальностями, на которых он учится.

(ФИО. Номер зач.кн.. Группа, Факультет. Специальность. Выпускающая кафедра)

Первичным ключом отношения является Номер зач.кн., однако рассмотрим остальные функциональные зависимости. Группа, в которой учится студент, однозначно определяет факультет, на котором он учится, а также специальность выпускающую кафедру. Кроме того, выпускающая кафедра однозначно определяет факультет, на котором обучаются студенты, выпускаемые по данной кафедре. Но если мы предположим, что одну специальность могут выпускать несколько кафедр, то специальность не определяет выпускающую кафедру. В этом случае у нас есть следующие функциональные зависимости:

Номер зач.кн. -> ФИО
Номер зач.кн. -> Группа
Номер зач.кн. -> Факультет
Номер зач.кн. -> Специальность
Номер зач.кн. -> Выпускающая кафедра/
Группа -> Факультет
Группа -> Специальность
Группа -> Выпускающая кафедра
Выпускающая кафедра -> Факультет

И эти зависимости образуют транзитивные группы. Для того чтобы избежать этого, мы можем предложить следующий набор отношений:

(Номер. зач.кн., ФИО. Специальность. Группа)
(Группа. Выпускающая кафедра)
(Выпускающая кафедра. Факультет)

Первичные ключи отношений выделены.

Теперь необходимо удостовериться, что при естественном соединении мы не потеряем ни одной строки не

получим лишних кортежей. И это упражнение и предлагаю выполнить и самостоятельно.

Полученный набор отношений находится в третьей нормальной форме.

Отношение находится в **нормальной форме Бойса - Кодда**, если оно находится в третьей нормальном форме и каждый детерминант отношения является потенциальным ключом отношении.

Рассмотрим отношение, моделирующее сдачу студентом текущих экзаменов. Предположим, что студент может сдавать экзамен по одной дисциплине несколько раз, если он получил удовлетворительную оценку. Допустим, что во избежание возможных полных однофамильцев мы можем однозначно идентифицировать студента номером его зачетной книги, но, с другой стороны, у нас ведется электронный учет текущей успеваемости студентов, поэтому каждому студенту присваивается в период его обучения в вузе уникальный номер-идентификатор. Отношение, которое моделирует сдачу текущей сессии, имеет следующую структуру:

(Номер зач.кн.. Иденфикатор_студента. Дисциплина. Дата, Оценка)

Потенциальными ключами отношения являются Номер_зач.кн, Дисциплина, Дата и Идентификатор_студента, Дисциплина, Дата.

Какие функциональные зависимости у нас имеются?

Номер_зач.кн. Дисциплина. Дата -> Оценка;
Идентификатор_студента. Дисциплина. Дата -> Оценка;
Номер зач.кн. -> Идентификатор_студента;
Идентификатор_студента; -> Номер зач.кн.

Откуда взялись две последние функциональные зависимости? Но ведь мы предварительно описали, что каждому студенту ставится в соответствие один номер зачетной книжки и один Идентификатор_студента, поэтому по значению Номер зач.кн. можно однозначно определить Идентификатор_студента (это третьи зависимость) и обратно (и это четвертая зависимость). Оценим это отношение.

Это отношение ладится в третьей нормальной форме, потому что неполных функциональных зависимостей непервичных атрибутов от атрибутов возможного ключа здесь не присутствует и нет транзитивных зависимостей. А как же третья и четвертая зависимости, разве они не являются неполными? Нет, потому что зависимым не является непервичный атрибут, то есть, атрибут, не входили ни в один возможный ключ. Поэтому придраться к этому мы не можем. Но вот под четвертую нормальную форму наше отношение не подходит, потому что у нас есть два детерминанта Номер зач.кн. и Идентификатор_студев, которые не являются возможными ключами отношения. Для приведения отношения к нормальной форме Бойса-Кодда надо разделить отношение, например, на два со следующими схемами:

(Идентификатор_студента, Дисциплина, Дата, Оценка)
(Номер эач.кн,. Идентификатор_студента)

или наоборот:

(Номер зач.кн., Дисциплина. Дата, Оценка)
(Номер зач.кн.. Идентификатор_студента)

Эти схемы равнозначны с точки зрения теории нормализации, поэтому выбирать проектировщикам следует исходя из некоторых дополнительных рассуждений. Ну, например, если учесть, что зачетные книжки могут теряться, то как они будут восстанавливаться: если с тем же самым

номером, то нет разницы, но если с новым номером, то тогда первая схема предпочтительней.

В большинстве случаев достижение третьей нормальной формы или даже формы Бойса-Кодда считается достаточным для реальных проектов баз данных, однако в теории нормализации существуют нормальные формы высших порядков, которые уже связаны не с функциональными зависимостями между атрибутами отношений, а отражают более тонкие вопросы семантики предметной области и связаны с другими видами зависимостей. Прежде чем перейти к рассмотрению нормальных форм высших порядков, дадим еще несколько определений.

В отношении R (A, B, C) существует **многозначная зависимость (multi valid dependence, MVD)** R.A ->> R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от C.

Когда мы рассматривали функциональные зависимости, то каждому значению детерминанта соответствовало только одно значение зависимого от него атрибута. При рассмотрении многозначных зависимостей мы выделяем случаи, когда одному значению некоторого атрибута соответствует устойчиво постоянное множество значений другого атрибута. Когда это может быть? Рассмотрим конкретную ситуацию, понятную всем студентам. Пусть дано отношение, которое моделирует предстоящую сдачу экзаменов на сессии. Допустим, оно имеет вид:

(Номер зач.кн.. Группа. Дисциплина)

Перечень дисциплин, которые должен сдавать студент, однозначно определяется не его фамилией, а номером группы (то есть специальностью, на которой он учится).

В данном отношении существуют следующие две многозначные зависимости:

Группа ->> Дисциплина Группа ->> Номер зач.кн.

Это означает, что каждой группе однозначно соответствует перечень дисциплин по учебному плану и номер группы определяет список студентов, которые в этой группе учатся.

Если мы будем работать с исходным отношением, то мы не сможем хранить информацию о новой группе в ее учебном плане - перечне дисциплин, которые должна пройти группа до тех пор, пока в нее не будут зачислены студенты. При изменении перечни дисциплин по учебному плану, например, при добавлении новой дисциплины, внести эти изменении в отношение для всех студенток, донимающихся в данной группе, весьма затруднительно. С другой стороны, если мы добавляем студента в уже существующую группу, то мы должны добавить множество кортежей, соответствующих перечню дисциплин для данной группы. Эти аномалии модификации отношения как раз и связаны с наличием двух многозначных зависимостей.

В теории реляционных баз данных доказывается, что в общем случае и отношении R (A, B, C) существует многозначная зависимость R.A ->>R. B в том и только в том случае, когда существует многозначная зависимость R.A ->>R.C.

Дальнейшая нормализация отношений, подобных нашему, основывается на теореме Фейджина.

Отношение R (A, B, C) можно спроецировать без потерь в отношения R1 (A, B) и R2 (A, C) в том и только в том случае, когда существует MVD A ->> B | C (что равнозначно наличию двух зависимостей A ->> B и A ->> C).

Под проецированием без потерь понимается такой способ декомпозиции отношения путем применения операции проекции, при котором исходное отношение полностью и без избыточности восстанавливается путем естественного соединения полученных отношений. Практически теорема доказывает наличие эквивалентной схемы для отношения, в котором существует несколько многозначных зависимостей.

Отношение R находится в **четвертой нормальной форме** (4NF) в том и только и том <случае, если в случае существования многозначной зависимости A ->> B все остальные атрибуты R функционально зависят от A

В нашем примере можно произвести декомпозицию исходного отношения в два отношения:

(Номер зач.кн.. Группа)
(Группа. Дисциплина)

Оба эти отношения находятся в 4NF и свободны от отмеченных аномалий. Действительно, обе операции модификации теперь упрощаются: добавление нового студента связано с добавлением всего одного кортежа в первое отношение, а добавление новой дисциплины выливается в добавление одного кортежа во второе отношение, кроме того, во втором отношении мы можем хранить любое количество групп с определенным перечнем дисциплин, в которые пока еще не зачислены студенты.

Последней нормальной формой является **пятая нормальная форма** 5NF, которая связана с анализом нового вида зависимостей, *зависимостей «проекции соединения» (project-join зависимости,* обозначаемые как *PJ-зависимости)*. Этот вид зависимостей является в некотором роде обобщением многозначных зависимостей.

Отношение R (X, Y,..., Z) удовлетворяет зависимости соединения (X, Y,..., Z) в том и только в том случае, когда R восстанавливается без потерь путем соединения своих проекций на X, Y,..., Z. Здесь X, Y,..., Z - наборы атрибутов отношения R.

Наличие PJ-зависимости в отношении делает его в некотором роде избыточным и затрудняет операции модификации.

Отношение R находится в *пятой нормальной форме* (нормальной форме проекции-соединения - PJ/NF) в том и только в том случае, когда любая зависимость соединения в R следует из существования некоторого возможного ключа в R.

Рассмотрим отношение R1:

R1 (Преподаватель. Кафедра. Дисциплина)

Предположим, что каждый преподаватель может работать на нескольких кафедрах и на каждой кафедре может вести несколько дисциплин. В этом случае ключом отношения является полный набор из трех атрибутов. В отношении отсутствуют многозначные зависимости, и поэтому отношение находится в 4NF.

Введем следующие обозначения наборов атрибутов:

ПК (Преподаватель. Кафедра)
ПД (Преподаватель, Дисциплина)
КД (Кафедра, Дисциплина)

Допустим, что отношение R1 удовлетворяет зависимости проекции соединения (ПК, ПД, КД). Тогда отношение R1 не находится в NF/PJ, потому что единственным ключом его является полный набор атрибутов, а наличие зависимости PJ связано с наборами атрибутов, которые не составляют

возможные ключи отношения R1. Для того чтобы привести это отношение к NF/PJ, его надо представить в виде трех отношений:

R2 (Преподаватель. Кафедра)
R3 (Преподаватель. Дисциплина)
R4 (Кафедра. Дисциплина)

Пятая нормальная форма редко используется на практике. В большей степени она является теоретическим исследованием. Очень тяжело определить само наличие зависимостей «проекции-соединения», потому что утверждение о наличии такой зависимости делается для всех возможных состояний БД, а не только для текущего экземпляра отношения R1. Однако знание о возможном наличии подобных зависимостей, даже теоретическое, нам все же необходимо.

5. ЗАЩИТА БАЗЫ ДАННЫХ

В этом разделе мы познакомимся со всем диапазоном проблем, охватываемых понятием защиты баз данных, и выясним, почему организации должны относиться к потенциальным угрозам их компьютерным системам со всей серьезностью (рис. 5.1). Кроме того, мы уточним все типы опасностей, которые могут угрожать самим компьютерам и различным компьютерным системам.

Под защитой базы данных понимают обеспечение защищенности базы данных против любых предумышленных или непредумышленных угроз с помощью различных компьютерных и некомпьютерных средств.

Понятие защиты применимо не только к сохраняемым в базе данных. Бреши в системе защиты могут возникать и в других частях системы, что, в свою очередь, подвергает опасности и собственно базу данных. Следовательно, защита базы данных должна охватывать используемое оборудование, программное обеспечение, персонал и собственно данные. Для эффективной реализации защиты необходимы соответствующие средства контроля, которые определяются конкретными требованиями, вытекающими из особенностей эксплуатируемой системы. Необходимость защищать данные, которую раньше очень часто отвергали или которой просто пренебрегали, в настоящее время все яснее осознается различными организациями. Повод для такой перемены настроений заключается в участившихся случаях разрушения компьютерных хранилищ корпоративных данных, а также в осознании того, что потеря или просто временная недоступность этих данных может послужить причиной настоящей катастрофы. Более подробное обсуждение проблем защиты компьютерных систем заинтересованный читатель может найти в работах Пфлегера (Pfleger, 1997).

База данных представляет собой важнейший корпоративный ресурс, который должен быть надлежащим образом защищен с помощью соответствующих средств контроля.

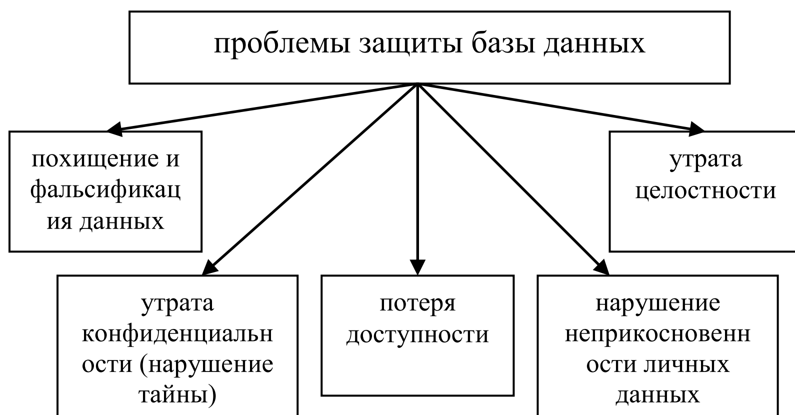


Рис. 5.1. Проблемы защиты баз данных

Указанные ситуации отмечают основные направления, в которых руководство должно принимать меры, снимающие степень риска, т. е. потенциальную возможность потери или повреждения данных. В некоторых ситуациях все отмеченные аспекты повреждения данных тесно связаны между собой, так что действия, направленные на нарушение неприкосновенности личных данных или фальсификация информации — могут возникнуть вследствие как намеренных, так и непреднамеренных действий и вовсе необязательно сопровождаться какими-либо изменениями в базе данных или системе, которые можно будет обнаружить тем или иным образом.

Похищение и фальсификация данных

Похищение и фальсификация данных могу происходить не только в среде базы данных — вся организация так или иначе подвержена этому риску. Однако действия по

похищению или фальсификации информации всегда совершаются людьми, поэтому основное внимание должно быть сосредоточено на сокращении общего количества удобных ситуаций для выполнения подобных действий. Например, рекомендуется строго ограничить доступ к данным о выплачиваемой зарплате, точно фиксировать количество отпечатанных платежных поручений, а также организовать четкий учет и автоматическое уничтожение всех файлов, используемых в несанкционированных попытках печати платежных документов. Похищения и фальсификация вовсе не обязательно связаны с изменением каких-либо данных, что справедливо и в отношении потери конфиденциальности или нарушения неприкосновенности личных данных.

Потеря конфиденциальности и нарушение Неприкосновенности личных данных

Понятие конфиденциальности означает необходимости сохранения данных в тайне. Как правило, конфиденциальными считаются те данные, которые являются критичными для всей организации, тогда как понятие неприкосновенности данных касается требования защиты информации об отдельных работниках. Следствием нарушения в системе защиты, вызвавшего потерю конфиденциальности данных, может быть утрата позиций в конкретной борьбе, тогда как следствием нарушения неприкосновенности личных данных будут юридические меры, принятые в отношении организации.

Утрата целостности и потеря доступности данных

Утрата целостности приводит к искажению или разрушению данных, что может иметь самые серьезные последствия для дальнейшей работы организации. В настоящее время множество организаций функционирует в непрерывном режиме, представляя свои услуги клиентам по 24 часа в сутки и по 7 дней в неделю. Потеря доступности данных будет означать, что либо данные, либо система, и другие одновременно окажутся недоступными пользователям, что

может подвергнуть опасности само дальнейшее существование организации. В некоторых случаях те события, которые послужили причиной перехода системы в недоступное состояние, могут одновременно вызвать и разрушение данных в базе.

Защита базы данных имеет целью минимизировать потери, вызванные заранее предусмотренными событиями. Принимаемые решения должны обеспечивать эффективное использование понесенных затрат и исключать излишнее ограничение представляемых пользователям возможностей. В последнее время уровень компьютерной преступности существенно возрос, причем прогнозы на будущее неутешительны и предвещают продолжение его роста и в новом столетии. Компьютерные преступления могут угрожать любой части системы, поэтому наличие необходимых мер защиты является жизненно важным.

Типы опасностей

Под **опасностью** будем понимать любую ситуацию или событие, намеренное или непреднамеренное, которое способно неблагоприятно повлиять на систему, а следовательно, и на всю организацию.

Угроза может быть вызвана ситуацией или событием, способным принести вред организации, причиной которого может служить человек, происшествие или стечение обстоятельств. Вред может быть очевидным (например, потеря оборудования, программного обеспечения или данных) или неочевидным (например, потеря доверия партнеров или клиентов). Перед каждой организацией стоит проблема выяснения всех возможных опасностей, что в некоторых случаях является совсем непростой задачей. Поэтому на выявление хотя бы важнейших угроз может потребоваться достаточно много времени и усилий, что следует учитывать. В

предыдущем разделе мы указали основные области, в которых следует ожидать потерь в случае намеренного (заранее задуманного) или ненамеренного (случайного) происшествия. Преднамеренные угрозы всегда осуществляются людьми и могут быть совершены как авторизированными, так и неавторизированными пользователями, причем последние могут не принадлежать организации.

Любая опасность должна рассматриваться как потенциальная возможность нарушения системы защиты, которая в случае своей реализации может оказать то или иное негативное влияние. Например, следствием просмотра и раскрытия засекреченных данных могут стать похищения и фальсификация, утрата конфиденциальности и нарушение неприкосновенности личных данных в организации. Хотя некоторые типы опасностей могут быть как намеренными, так и непреднамеренными, результаты в любом случае будут одинаковы.

Уровень потерь, понесенных организацией в результате реализации некоторой угрозы, зависит от многих факторов, включая наличие заранее продуманных контрмер и планов преодоления непредвиденных обстоятельств. Например, если отказ оборудования вызвал разрушение вторичных хранилищ данных, вся работа в системе должна быть свернута до полного устранения последствий аварии.

- Имеется ли в системе резервное оборудование с развернутым программным обеспечением.
- Когда в последний раз выполнялось резервное копирование системы.
- Время, необходимое на восстановление системы.
- Существует ли возможность восстановления и ввода в эксплуатацию разрушенных данных.

Любая организация должна установить типы возможных опасностей, которым может подвергнуться ее компьютерная

система, после чего разработать соответствующие планы и требуемые контрмеры, с оценкой уровня затрат, необходимых для их реализации. Безусловно, затраты времени, усилий и денег едва ли окажутся эффективными, если они будут касаться потенциальных опасностей, способных причинить организации лишь незначительный ущерб. Деловые процессы организации могут быть подвержены таким опасностям, которые непременно следует учитывать, однако часть из них может иметь место в исключительно редких ситуациях.

Тем не менее, даже столь маловероятные обстоятельства должны быть приняты во внимание, особенно если их внимание может оказаться весьма существенным.

Существующие примеры брешей, обнаружившихся в защите компьютерных систем, демонстрируют тот факт, что даже высочайшего уровня защищенности самой системы может оказаться недостаточно, если вся деловая среда не будет иметь необходимого уровня защиты. Целью является достижение баланса между обоснованным уровнем реализации защитных механизмов, функционирование которых не вызывает излишних ограничений в работе пользователей, и издержками на их поддержание.

Некоторые из угроз могут носить случайный характер. По всей видимости, именно случайные опасности являются причиной основной части потерь в большинстве организаций. Любой случайный инцидент, послуживший причиной нарушения системы защиты, должен быть зафиксирован в документации, с указанием сведений о персонах, связанных с его появлением. Время от времени подобные записи должны анализироваться с целью установления частоты возникновения сходных инцидентов связанных с одними и теми же людьми. Полученные сведения следует использовать для уточнения существующих процедур и установленных ограничений. Например, повторяющееся рассоединение или обрыв кабелей должен послужить причиной пересмотра способа укладки или маршрута их проведения. Аналогично, повторяющееся внесение в систему вирусов и другого нежелательного

программного обеспечения должно быть расценено как серьезная угроза, однако здесь могут иметь место трудности с оценкой или преднамеренности их появления. Тем не менее, могут быть разработаны процедуры, предназначенные для проверки всего нового программного обеспечения и носителей, поступающих в распоряжение организации. Они могут быть дополнены ограничениями, запрещающими работникам использовать их собственное программное обеспечение. После принятия подобных мер появление в системе вирусов едва ли можно будет расценивать как случайное.

Типы возможных угроз лежат в широком диапазоне — от пожаров и наводнений, непосредственно воздействующих на все элементы системы, до некоторых специфических инцидентов, воздействующих лишь на одну из ее частей. Например, случайный обрыв кабеля может блокировать работу только одного пользователя и не оказать влияния на состояние программного обеспечения или данных. Как бы там ни было, хотя, на первый взгляд, некоторые из событий не оказывают заметного влияния на систему, их звуки в последствии могут быть весьма существенными. Поэтому при анализе отдельных опасностей оценка их возможного влияния должна производится с разных точек зрения. Например, если обратиться к случаю отказа оборудования, послужившего причиной разрушения устройств внешней памяти, то полезно будет дать ответы на все приведенные ниже вопросы.

- Существуют ли резервное оборудование, которое может быть использовано немедленно?
- Защищено ли резервное оборудование надлежащим образом?
- Может ли используемое программное обеспечение нормально работать на резервном оборудовании?
- Если резервных устройств не существует, насколько быстро удастся устранить возникшую проблему?

- Когда в последний раз создавалась резервная копия базы данных и файла журнала?
- Находятся ли носители с резервной копией в пожарозащищенном месте или вне расположения организации?
- Если большая часть базы данных подлежит восстановлению с резервной копии и из файлов журнала, то сколько времени потребуется на эту процедуру?
- Каков объем потерь в результатах обработки данных и имеется ли возможность восстановить утраченные данные?
- Сможет ли организация продолжать нормально функционировать, пока система будет неактивна, и если да, то как долго?
- Окажет ли инцидент какое-либо прямое воздействие на клиентов организации?
- Если система будет восстановлена, то сохраниться ли угроза повторения данного инцидента, если для его предотвращения не будут приняты какие- либо специальные меры?
- Можно ли использовать данный инцидент для доработки существующих планов устранения непредвиденных обстоятельств?

Если при анализе последствий только одного события потребовалось ответить на столько вопросов, то для исчерпывающего анализа всех возможных опасностей понадобится ответить на гораздо большее их число. Ответы на вопросы должны позволить оценить вероятный эффект события, выраженный в уровне потери данных, времени, средств и других менее очевидных показателях — например, степени доверия клиентов. Для получения полной оценки всех возможных опасностей следует выполнить процедуру оценки риска. Существуют различные методики выполнения данной оценки, как ручные, так и с помощью специализированных программных пакетов.

Контрмеры — компьютерные средства контроля

В отношении опасностей, угрожающих компьютерным системам, могут быть приняты контрмеры самых различных типов, начиная от физического наблюдения и заканчивая административно-организационными процедурами.

Несмотря на широкий диапазон компьютерных средств контроля, доступных в настоящее время на рынке, общий уровень защищенности СУБД определяется возможностями используемой операционной системы, поскольку работа этих двух компонентов тесно связанна между собой.

В этом разделе мы обсудим различные компьютерные средства контроля, доступные в многопользовательской среде. Обычно для платформы IBM PC применяются не все перечисленные на рис. 5.2 типы средств контроля.

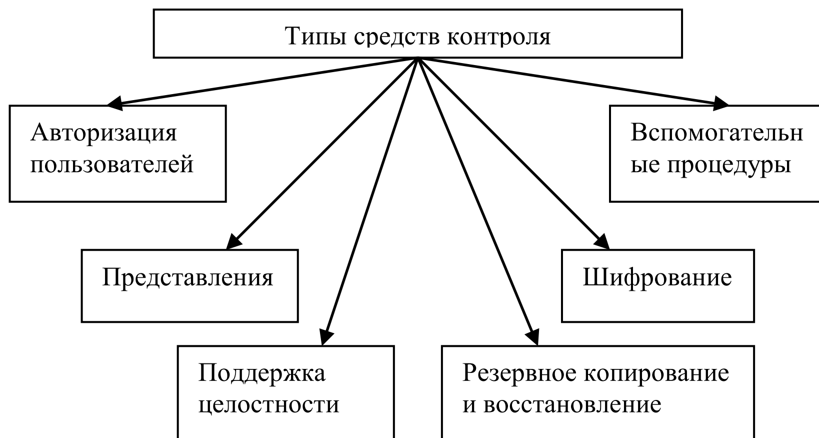


Рис. 5.2. Типы средств контроля

Авторизация пользователей

Авторизация – это предоставление прав (или привилегий), позволяющих их владельцу иметь законный доступ к системе или к ее объектам.

Средства автоматизации пользователей могут быть встроены непосредственно в программное обеспечение и управлять не только предоставленными пользователям правами доступа к системе или объектам, но и набором операций, которые пользователи могут выполнять с каждым доступным ему объектом. По этой причине механизм авторизации часто иначе называют подсистемой управления доступом. Термин «владелец» в определении может представлять пользователя-человека или программу. Термин «объект» может представлять таблицу данных, представление, приложение, процедуру или любой другой объект, который может быть создан в рамках системы. В некоторых системах все предоставляемые субъекту права должны быть явно указаны для каждого из объектов, к которым этот субъект должен обращаться. Процесс авторизации включает аутентификацию субъектов, требующих получения доступа к объектам.

Аутентификация – это механизм определения того, является ли пользователь тем, за кого себя выдает.

За представление пользователям доступа к компьютерной системе обычно отвечает системный администратор. Каждому пользователю присваивается уникальный идентификатор, который используется операционной системой для определения того, кто есть, кто. С каждым идентификатором связывается определенный пароль, выбираемый пользователем и известный операционной системе. При регистрации пользователь должен предоставлять системе свой пароль для выполнения проверки (аутентификации) того, является ли он тем, за кого себя выдает.

Подобная процедура позволяет организовать контролируемый доступ к компьютерной системе, но не обязательно предоставляет право доступа к СУБД или иной прикладной программе. Для получения пользователем права доступа к СУБД может использоваться отдельная подобная

процедура. Ответственность за предоставление прав доступа к СУБД обычно несет администратор базы данных (АБД), в обязанности которого входит создание индивидуальных идентификаторов пользователей, но на этот раз уже в среде самой СУБД. Каждый из идентификаторов пользователей СУБД так же связывается с паролем, который должен быть известен только данному пользователю. Этот пароль будет использоваться подпрограммами СУБД для идентификации данного пользователя.

Некоторые СУБД поддерживают списки разрешенных идентификаторов пользователей и связанных с ними паролей, отличающиеся от аналогичного списка, поддерживаемого операционной системой. Другие типы СУБД поддерживают списки, элементы которых приведены в соответствие существующим спискам пользователей операционной системы и выполняют регистрацию исходя из текущего идентификатора пользователя, указанного им при регистрации в системе. Это предотвращает попытки пользователей зарегистрироваться в среде СУБД под идентификатором, отличным от того, который они использовали при регистрации в системе.

Использование паролей является наиболее распространенным методом аутентификации пользователей. Однако этот подход не дает абсолютной гарантии, что данный пользователь является именно тем, за кого себя выдает. Ниже в этой главе мы обсудим методы, позволяющие сократить подобный риск.

Привилегии

Как только пользователь получит право доступа к СУБД, ему могут автоматически предоставляться различные другие привилегии, связанные с его идентификатором. В частности, эти привилегии могут включать разрешение на доступ к определенным базам данных, таблицам, представлениям и индексам или же право запуска различных утилит СУБД. Некоторые типы СУБД функционируют как закрытые системы, поэтому пользователям помимо разрешения на доступ к самой

СУБД потребуется иметь отдельные разрешения и на доступ к конкретным ее объектам. Эти разрешения выдаются либо АБД, либо владельцами определенных объектов системы. В противоположность этому, открытые системы по умолчанию предоставляют авторизированным пользователям полный доступ ко всем объектам базы данных. В этом случае привилегии устанавливаются посредством явной отмены тех или иных прав конкретных пользователей. Типы привилегий, которые могут быть предоставлены авторизированным субъектам, включают, например, право доступа к указанным базам данных, право выборки данных, право создания таблиц и других объектов.

Право владения и привилегии

Некоторыми объектами в среде СУБД владеет сама СУБД. Обычно это владение организуется посредством использования специального идентификатора особого суперпользователя, например, с именем Database Administrator. Как правило, владение некоторым объектом предоставляет его владельцу весь возможный набор привилегий в отношении этого объекта. Это правило применяется ко всем авторизированным пользователям, получающим права владения определенными объектами. Любой вновь созданный объект автоматически передается во владения его создателю, который и получает весь возможный набор привилегий для данного объекта. Тем не менее, хотя пользователь может быть владельцем некоторого представления, единственной привилегией, которая будет представлена ему в отношении этого объекта, может оказаться право выборки данных из этого представления. Причиной подобных ограничений состоит в том, что данный пользователь имеет столь ограниченный набор прав в отношении базовых таблиц созданного им представления. Принадлежащие владельцу привилегии могут быть переданы им другим авторизированным пользователям. Например, владелец нескольких таблиц базы данных может предоставить другим пользователям право выборки

информации из этих таблиц, но не позволить им вносить в эти таблицы какие-либо изменения. В некоторых типах СУБД, всякий раз, когда пользователю предоставляется определенная привилегия, дополнительно может указываться, передается ли ему право предоставлять эту привилегию другим пользователям (уже от имени этого пользователя). Естественно, что в этом случае СУБД должна контролировать всю цепочку предоставления привилегий пользователям, с указанием того, кто именно ее предоставил, что позволит поддерживать корректность всего набора установленных в системе привилегий. В частности, эта информация будет необходима в случае отмены предоставленных ранее привилегий для организации каскадного распространения вносимых изменений среди цепочки пользователей.

Если СУБД поддерживает несколько различных типов идентификаторов авторизации, с каждым из существующих типов могут быть связаны различные приоритеты. В частности, если СУБД поддерживает использование идентификаторов как отдельных пользователей, так и их групп, то, как правило, идентификатор пользователя будет иметь более высокий приоритет, чем идентификатор группы. Пример определения идентификаторов пользователей и групп в подобной СУБД приведен в табл. 5.1.

Таблица 5.1

Идентификаторы пользователей и групп

Идентификатор пользователя	Тип	Группа	Идентификатор члена группы
SG37	Пользователь	Sales	
SG14	Пользователь	Sales	
SG5	Пользователь		
Sales	Группа		SG37, SG14

В столбцах *Идентификатор* пользователя и *Тип* приведены определенные в системе идентификаторы и указывается их тип – отдельный пользователь и группа пользователей. Столбцы с заголовками *Группа* и *Идентификатор члена группы* содержат сведения о группе, которой принадлежит пользователь, и его идентификаторе в этой группе. С каждым конкретным идентификатором может быть связан определенный набор привилегий, определяющий тип доступа к отдельным объектам базы данных (например, для чтения (Read), обновления (Update), удаления (Delete) или все типы сразу (All)). С каждым типом привилегии связывается некоторое двоичное значение:

READ	UPDATE	INSERT	DELETE	ALL
0001	0010	0100	1000	1111

Отдельные двоичные значения суммируются, и полученная сумма однозначно характеризует, какие именно привилегии (если они есть) имеет каждый конкретный пользователь или группа в отношении определенного объекта базы данных. В табл. 5.2 приведен пример матрицы управления доступом, определяющей набор привилегий, которые пользователи SG37, SG5 и группа Sales имеют в отношении указанных столбцов таблицы Property_for_Rent.

Таблица 5.2

«Матрица управления доступом»

Идентификатор пользователя	Столбец Pno	Столбец Type	Столбец Price	Столбец Ono	Столбец Sno	Столбец Bno	Лимит Выбираемых строк
SG37	0101	0101	0111	0101	0111	0000	100
SG5	1111	1111	1111	1111	1111	1111	нет
Sales	0001	0001	0001	0000	0000	0000	15

Содержимое матрицы показывает, что группе пользователей с идентификатором Sales предоставлена только привилегия Read (код 0001) в отношении атрибутов Pno, Type и Price. Кроме того, для нее установлен максимальный размер результирующего набора данных запроса, равный 15 строкам. Пользователь SG14 (его реальное имя David Ford) является членом этой группы и не имеет каких-либо дополнительных привилегий доступа, поэтому он пользуется только теми правами, которые предоставлены данной группе. Пользователь SG37 (Ann Beech) имеет собственные привилегии Read и Insert (определяются значением 0001+0100=0101) в отношении атрибутов Pno, Type и Ono, а также привилегии Read, Update и Insert (значение 0001+0010+0100=0111) в отношении атрибутов Price и Sno. Кроме того, для этого пользователя максимальный размер результирующего набора данных запроса установлен равным 100 строкам. Пользователю SG5 (Susan Brand) предоставлены привилегии Read, Update, Insert и Delete (значение 0001+0010+0100+1000=1111), т. е. привилегия All для доступа ко всем атрибутам таблицы, а размер результирующих наборов данных запросов не ограничивается.

Обычно для реализации механизмов контроля доступа СУБД используются подобные матрицы, хотя отдельные детали в разных системах могут отличаться. В некоторых СУБД пользователю разрешается указывать, под каким идентификатором он намерен работать далее – это целесообразно в тех случаях, когда один и тот же пользователь может являться членом сразу нескольких групп. Очень важно освоить все механизмы авторизации и другие средства защиты, предоставляемые целевой СУБД. Особенно это важно для тех систем, в которых существуют различные типы идентификаторов и допускается передача права присвоения привилегий. Это позволит корректно выбирать типы привилегий, предоставляемых отдельным пользователям, исходя из используемых ими обязанностей и набора используемых прикладных программ.

Представление (подсхема)

Представление – это динамический результат одной или нескольких реляционных операций с базовыми отношениями с целью создания некоторого иного отношения.

Представление является виртуальным отношением, которого реально в базе данных не существует, но которое создается по требованию отдельного пользователя в момент поступления этого требования.

Механизм представления представляет собой мощный и гибкий инструмент организации защиты данных, позволяющий скрыть от определенных пользователей некоторые части базы данных. В результате пользователи не будут иметь никаких сведений о существовании любых атрибутов или строк данных, которые недоступны через представления, находящиеся в их распоряжении. Представление может быть определено на базе нескольких таблиц, после чего пользователю будет предоставлены необходимые привилегии доступа к этому представлению, но не к базовым таблицам. В данном случае использование представления являет собой более жесткий механизм контроля доступа, чем обычное представление пользователю тех или иных прав доступа к базовым таблицам.

Резервное копирование и восстановление

Резервное копирование – это периодически выполняемая процедура получения копии базы данных и ее файла журнала (а также, возможно, программ) на носителе, сохраняемом отдельно от системы.

Любая современная СУБД должна предоставлять средства резервного копирования, позволяющие восстанавливать базу данных в случае ее разрушения. Кроме того, рекомендуется создавать резервные копии базы данных и ее файла журнала с некоторой установленной периодичностью,

а также организовывать хранение созданных копий в местах, обеспеченных необходимой защитой. В случае отказа, в результате которого база данных становится непригодной для дальнейшей эксплуатации, резервная копия и зафиксированная в файле журнала оперативная информация используются для восстановления базы данных до последнего согласованного состояния.

Ведение журнала – это процедура создания и обслуживания файла журнала, содержащего сведения обо всех изменениях, внесенных в базу данных с момента создания последней резервной копии, и предназначенного для обеспечения эффективного восстановления системы в случае ее отказа.

СУБД должна представлять средства ведения системного журнала, в котором будут фиксироваться сведения обо всех изменениях состояния базы данных и ходе выполнения текущих транзакций, что необходимо для эффективного восстановления базы данных в случае отказа. Преимущества использования подобного журнала заключаются в том, что в случае нарушения работы или отказа СУБД базу данных можно будет восстановить до последнего известного согласованного состояния, воспользовавшись последней созданной резервной копией базы данных и оперативной информацией, содержащейся в файле журнала. Если в отказавшей системе функция ведения системного журнала не использовалась, базу данных можно будет восстановить до того состояния, которое было зафиксировано в последней созданной резервной копии. Все изменения, которые были внесены в базу данных после создания последней резервной копии, окажутся потерянными.

Контрольная точка – это момент синхронизации между состоянием базы данных и состоянием журнала выполнения транзакций. В этот момент все буфера принудительно выгружаются на устройства вторичной памяти.

Современные СУБД, как правило, предоставляют средства создания контрольных точек, позволяющих зафиксировать в базе данных серию последних выполненных изменений. Механизм создания контрольных точек может использоваться совместно с ведением системного журнала, что позволит повысить эффективность процесса восстановления. В момент создания контрольной точки СУБД выполняет действия, обеспечивающие запись на диск всех данных, хранившихся в основной памяти машины, а также помещение в файл журнала специальной записи контрольной точки.

Поддержка целостности

Средства поддержки целостности данных также вносят определенный вклад в общую защищенность базы данных, поскольку их назначением является предотвращение перехода данных в несогласованное состояние, а значит, и предотвращение угрозы получения ошибочных или некорректных результатов расчета.

Шифрование

Шифрование – это кодирование данных с использованием специального алгоритма, в результате чего данные становятся недоступными для чтения любой программы, не имеющей ключа дешифрования.

Если в системе с базой данных содержится весьма важная конфиденциальная информация, то имеет смысл закодировать ее с целью предупреждения возможной угрозы несанкционированного доступа с внешней стороны (по отношению к СУБД). Некоторые СУБД включают средства шифрования, предназначенные для использования в подобных целях. Подпрограммы таких СУБД обеспечивают санкционированный доступ к данным (после их декодирования), хотя это будет связанно с некоторым

снижением производительности, вызванным необходимостью перекодировки. Шифрование также может использоваться для защиты данных при их передаче по линиям связи. Существует множество различных технологий кодирования данных с целью сокрытия передаваемой информации, причем одни из них называют необратимыми, а другие обратимыми. Необратимые методы, как и следует из их названия, не позволяют установить исходных данных, хотя последние могут использоваться для сбора достоверной статистической информации. Обратимые технологии используются чаще. Компоненты системы шифрования представлены на рис. 5.3.

Рис. 5.3. Компоненты системы шифрования

Некоторые системы шифрования, называемые симметричными, используют один и тот же ключ как для шифрования, так и для дешифрования, при этом предполагается

наличие защищенных линий связи, предназначенных для обмена ключами. Однако большинство пользователей не имеют доступа к защищенным линиям связи, поэтому для получения надежной защиты длина ключа должна быть не меньше длины самого сообщения (Leiss,1982). Тем не менее большинство эксплуатируемых систем построено на использовании ключей, которые короче самих сообщений. Одна из распространенных систем шифрования называется DES (Data Encryption Standard) — в ней используется стандартный алгоритм шифрования, разработанный фирмой IBM. В этой системе для шифрования и дешифрования используется один и тот же ключ, который должен храниться в секрете, хотя сам алгоритм шифрования не является секретным. Этот алгоритм предусматривает преобразование каждого 64-битового блока обычного текста с использованием 56-битового ключа шифрования. Система шифрования DES расценивается как достаточно надежная далеко не всеми — некоторые разработчики полагают, что следовало бы использовать более длинное значение ключа. Так в системе шифрования PGP (Pretty Good Privacy) используется 128-битовый симметричный алгоритм, применяемый для шифрования блоков отсылаемых данных.

В настоящее время ключи длиной до 64 бит раскрываются с достаточной степенью вероятности правительственными службами развитых стран, для чего используется специальное оборудование, хотя и достаточно дорогое. Тем не менее, в течение ближайших лет эта технология может попасть в руки организованной преступности, крупных организаций и правительств других государств. Хотя предполагается, что ключи длиной до 80 бит также окажутся раскрываемыми в ближайшем будущем, есть уверенность, что ключи длиной 128 бит в обозримом будущем останутся надежным средством шифрования. Термин «сильное шифрование» и «слабое шифрование» иногда используются для подчеркивания различий между алгоритмами, которые не могут быть раскрыты с помощью существующих в настоящее время

технологий и теоретических методов (сильные), и теми, которые допускают подобные раскрытия (слабые).

Другой тип систем шифрования предусматривает использование для шифровки и дешифровки сообщений различных ключей — подобные системы принято называть несимметричными. Примером такой системы является система с открытым ключом, предусматривающая использование двух ключей, один из которых является открытым, а другой хранится в секрете. Алгоритм шифрования также может быть открытым, поэтому любой пользователь, желающий направить владельцу ключей зашифрованное сообщение, может использовать его открытый ключ и соответствующий алгоритм шифрования. Однако дешифровать данное сообщение сможет только тот, кто знает парный закрытый ключ шифрования. Системы шифрования с открытым ключом могут также использоваться для отправки вместе с сообщением “цифровой подписи”, подтверждающей, что данное сообщение было действительно отправлено владельцем открытого ключа. Наиболее популярной несимметричной системой шифрования является RSA (это инициалы трех разработчиков данного алгоритма). Как правило, симметричные алгоритмы являются более быстродействующими, чем несимметричные, однако на практике обе схемы часто применяются совместно, когда алгоритм с открытым ключом используется для шифрования случайным образом сгенерированного ключа шифрования, а уже этот случайный ключ для шифровки самого сообщения с применением некоторого симметричного алгоритма.

Вспомогательные процедуры

Хотя выше уже были описаны различные механизмы, которые могут использоваться для защиты данных в среде СУБД, сами по себе они не гарантируют необходимого уровня защищенности и могут оказаться неэффективными в случае неправильного применения или управления. По этой причине в данном разделе мы обсудим различные вспомогательные

процедуры, которые должны использоваться совместно с описанными выше механизмами защиты.

Авторизация и аутентификация

Выше в этой главе уже описывался механизм паролей, являющийся самым распространенным методом подтверждения личности пользователей. С точки зрения обеспечения необходимого уровня защиты очень важно, чтобы все используемые пароли держались пользователями в секрете и регулярно обновлялись через некоторый установленный интервал времени. В ходе процедуры регистрации в системе значение пароля не должно отражаться на экране, а списки идентификаторов пользователей и их паролей должны сохраняться в системе в зашифрованном виде. Кроме того, в организации должен использоваться некоторый стандарт для выбора допустимых значений пароля, например, все пароли должны быть не меньше установленной длины, обязательно содержать цифры и служебные символы, а также подлежать замене через установленный интервал времени (скажем, пять недель). Для выявления слабых паролей в системе следует использовать специальное программное обеспечение (например, в качестве пароля могут использоваться реальные имена или адреса пользователей, что недопустимо). Кроме того, специализированные программы могут применяться для выявления устаревших паролей.

Еще одним аспектом авторизации является разработка процедур, посредством которых конкретным пользователям предоставляются права доступа к различным объектам базы данных. Очень важно хранить историческую информацию о предоставлении прав пользователям, особенно если служебные обязанности пользователей меняются, и они перестают нуждаться в доступе к определенным массивам данных. В частности, если пользователь увольняется, чрезвычайно важно немедленно удалить его учетную запись и аннулировать все предоставленные ему права доступа, что позволит

своевременно пресечь любые возможные попытки нарушения защиты системы.

Копирование

Процедуры, регламентирующие процессы создания резервных копий, определяются типом и размерами эксплуатируемой базы данных, а также тем набором соответствующих инструментов, который предоставляется используемой СУБД. Эти процедуры должны включать необходимые этапы, на которых будет непосредственно выполняться создание резервной копии. Как уже указывалось выше, крупные базы данных могут полностью копироваться раз в неделю или даже раз в месяц, но при этом следует организовать обязательное инкрементное копирование, выполняемые с более высокой частотой. День и час выполнения резервного копирования должен устанавливаться ответственными лицами.

В процедурах копирования также может указываться, какие еще части системы (например, прикладные программы), помимо самих данных, должны подлежать копированию. В зависимости от частоты внесения в систему изменений, в течение одних суток может выполняться несколько копирований; созданные копии должны помещаться в безопасное место. Место хранения последних копий должно быть оборудовано как минимум несгораемыми шкафами. Кроме того, желательно использовать некоторое внешнее хранилище, в которое будет помещаться второй экземпляр созданных копий. Все упомянутые детали должны найти свое четкое отражение в разработанных процедурах резервного копирования, которые должны неукоснительно выполняться обслуживающим персоналом.

Восстановление

Как и процедуры копирования, процедуры восстановления также должны быть тщательно продуманны и проработаны. То, какие именно процедуры восстановления

будут выполняться, должно определяться типом имевшего места отказа (разрушение носителя, отказ программного обеспечения или отказ оборудования системы). Кроме того, процедурами восстановления должны учитываться особенности методов восстановления, принятых в используемой СУБД. В любом случае разработанные процедуры восстановления должны быть тщательно протестированы, поскольку необходимо получить полную гарантию, что они работают правильно, еще до того, как произойдет реальный отказ системы. В идеале, процедуры восстановления должны регулярно тестироваться с некоторым установленным интервалом.

Аудит

Одно из назначений процедуры аудита состоит в проверке того, все ли предусмотренные средства управления задействованы и соответствует ли уровень защищенности установленным требованиям. В ходе выполнения инспекции аудиторы могут ознакомиться с используемыми ручными процедурами, обследовать компьютерные системы и проверить состояние всей имеющейся документации на данную систему.

Все упомянутые процедуры и средства контроля (рис. 5.4) должны быть достаточно эффективными, в противном случае они должны быть подвергнуты пересмотру. Для определения активности использования базы данных анализируются ее файлы журнала. Эти же источники могут использоваться для выявления любых необычных действий в системе. Регулярное проведение аудиторских проверок, дополненное постоянным контролем содержимого файлов журнала с целью выявления ненормальной активности в системе, очень часто позволяет своевременно обнаружить и пресечь любые попытки нарушения защиты.

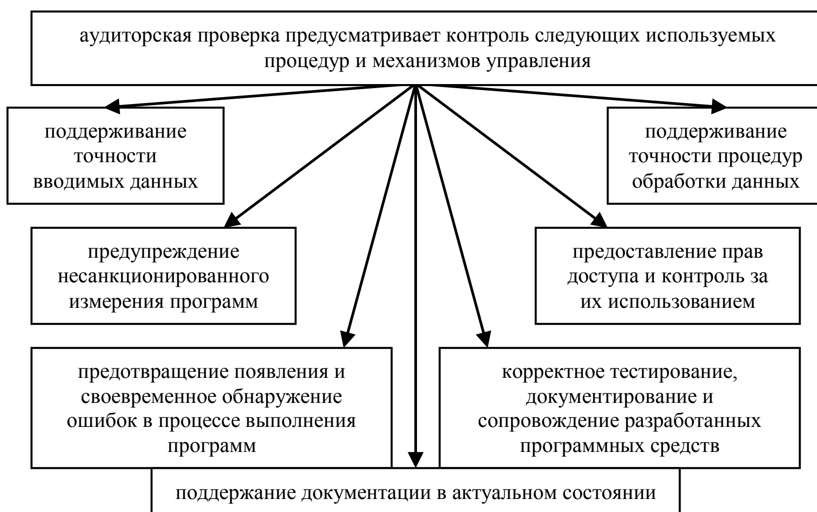


Рис. 5.4. Процедуры и механизмы управления аудита

Установка нового прикладного программного обеспечения

Новые приложения, разработанные собственными силами или сторонними организациями, обязательно следует тщательно протестировать, прежде чем принимать решения об их развертывании и передаче в эксплуатацию. Если уровень тестирования будет недостаточным, существенно возрастает риск разрушения базы данных. Следует считать хорошей практикой выполнение резервного копирования базы данных непосредственно перед сдачей нового программного обеспечения в эксплуатацию. Кроме того, первый период эксплуатации нового приложения обязательно следует организовать тщательное наблюдение за функционированием системы. Отдельным вопросом, который должен быть согласован со сторонними разработчиками программ, является право собственности на разработанное ими программное обеспечение. Данная проблема должна быть решена еще до начала разработки, причем это особенно важно в тех случаях, когда существует вероятность, что впоследствии организации

обязательно потребуется вносить изменения в создаваемые приложения. Риск, связанный с подобной ситуацией, состоит в том, что организация не будет иметь юридического права использовать созданное программное обеспечение или модернизировать его. Потенциально подобная ситуация угрожает организации серьезными потерями.

Установки или модернизация системного программного обеспечения

В обязанности АБД входит выполнение модернизации программного обеспечения СУБД при поступлении от разработчика очередных пакетов изменений. В некоторых случаях вносимые изменения оказываются совсем незначительными и касаются только небольшой части модулей системы. Однако возможны ситуации, когда потребуется полная ревизия всей установленной системы. Как правило, каждый поступающий пакет изменений сопровождается печатной или интерактивной документацией, содержащей подобные сведения о сути изменений и их назначении. Многие склонны полагать, что после установки любого из пакетов модернизации продолжение нормального функционирования существующих баз данных и прикладного программного обеспечения автоматически гарантируется. С ростом интенсивности использования базы данных и увеличением объемом сопроводительной документации к пакету эта убежденность укрепляется. Однако обеспечение защиты данных и приложений имеет превалирующую важность, и это следует учитывать. Никакие изменения и модернизация ни в коем случае не должны вносится в систему без предварительной оценки их возможного влияния на имеющиеся данные и программное обеспечение.

Результатом ознакомления с сопроводительной документацией пакета должен стать план действий по его установке. В этом плане должны быть отражены любые изменения, которые могут повлиять на базу данных и приложения, а также предложены решения по их реализации. В

некоторых случаях может потребоваться, чтобы программисты выполнили поиск в тексте программ определенных конструкций или осуществили какие-либо иные подготовительные действия. Иногда требуемые изменения могут оказаться минимальными — после модернизации системы достаточно будет просто перекомпилировать некоторые из программ. В других случаях могут потребоваться более существенные подготовительные действия — например, изменение типа используемых данных. Однако, какие бы изменения не потребовались, все они должны быть учтены и для каждого из них должна быть дана оценка времени выполнения с учетом объема всего имеющегося программного обеспечения. Главная задача АБД — обеспечить плавный и безболезненный переход от старой версии системы к новой.

В функционирующей системе, которая должна быть постоянно доступна на протяжении всего рабочего времени, установки любых пакетов модернизации обычно выполняются во внерабочее время — например, в выходные дни. Непосредственно перед выполнением модернизации должна быть сделана полная резервная копия существующей системы — на случай возможного отказа. Затем выполняется установка пакета модернизации, а в данные и программы вносятся все необходимые изменения, сопровождаемые требуемыми процедурами тестирования. Только после полного завершения указанных процедур система может быть запущена в работу с использованием реальных данных.

Изменения, вносимые в программы СУБД в результате модернизации, могут оказать влияние на любые используемые прикладные программы, созданные различными программистами. Поэтому список необходимых изменений должен быть либо разослан всем заинтересованным лицам, либо открыт для всеобщего доступа. Некоторые из вносимых изменений могут представлять собой интерес, если они связаны, например, с исправлением замеченных ранее ошибок и позволяют удалить использовавшиеся до этого искусственные способы их обхода. Кроме того, желательно, чтобы

сопроводительная документация включала список известных ошибок с указанием использовавшихся путей их обхода («заплат»). Эти сведения также должны быть разосланы всем заинтересованным лицам или сделаны общедоступными.

Контрмеры — некомпьютерные средства контроля

Некомпьютерные средства контроля (рис. 5.5) включают такие методы, как выработку ограничений, соглашений и других административных мер, не связанных с компьютерной поддержкой. Мы рассмотрим такие некомпьютерные средства контроля, как:

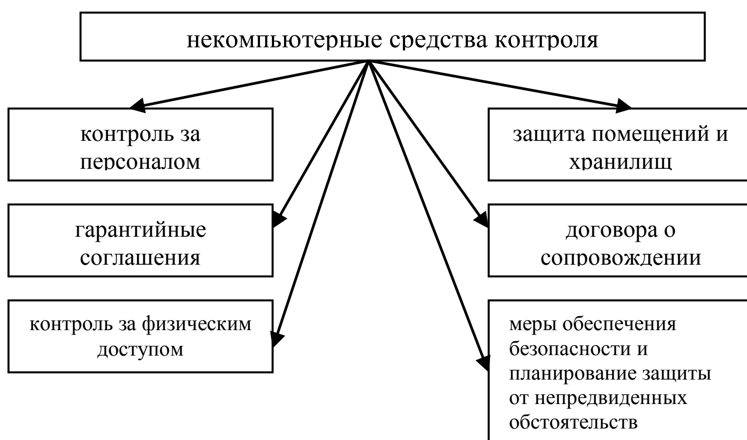


Рис. 5.5. Некомпьютеные средства контроля

Меры обеспечения безопасности и планирование защиты от непредвиденных обстоятельств

Понятия выработки мер обеспечения безопасности и планирования защиты от непреднамеренных обстоятельств существенно отличаются друг от друга. Первое предполагает исчерпывающее определение средств, посредством которых будет обеспечиваться защита вычислительной системы данной организации. Второе состояние в определении методов, с помощью которых будет поддерживаться функционирование

организации в случае возникновения любой аварийной ситуации. Каждая организация должна подготовить и реализовать как перечень мер обеспечения безопасности, так и план защиты от непредвиденных обстоятельств.

Меры обеспечения безопасности

В документе по мерам обеспечения безопасности должно быть определено следующее:

- область деловых процессов организации, для которой они устанавливаются;
- ответственность и обязанности отдельных работников;
- дисциплинарные меры, принимаемые в случае обнаружения нарушения установленных ограничений;
- процедуры, которые должны обязательно выполняться.

Процедуры, определяемые как меры обеспечения безопасности, могут потребовать разработки других процедур, определение которых выходит за рамки данного документа. Например, одно из установленных в системе ограничений может заключаться в требовании, чтобы доступ к прикладным приложениям системы могли получать только авторизированные пользователи, однако способ реализации этого требования в данном документе не конкретизируется. Для исчерпывающего определения методов авторизации различных пользователей потребуется разработать отдельный набор процедур. Кроме того, дополнительно может быть подготовлен стандарт выполнения процедуры авторизации, устанавливающий, например, принятый формат паролей (если они используются в системе). В подобном случае документ с определением мер обеспечения безопасности постоянно сохраняет свою значимость, хотя может потребоваться его периодический пересмотр, тогда как выполняемые процедуры реализации этих требований должны модифицироваться при каждом внесении изменений в систему или модернизации используемой технологии. Пример перечня мер обеспечения безопасности можно найти в работе Герберта (Herbert, 1990).

Планирование защиты от непредвиденных обстоятельств

План защиты от непредвиденных обстоятельств разрабатывается с целью подробного определения последовательности действий, необходимых для выхода из различных ситуаций, непредусмотренных процедурами нормального функционирования системы — например, в случае пожара или диверсии. В системе может существовать единый план защиты от непредвиденных обстоятельств, а может быть несколько — каждый по отдельному направлению. Типичный план защиты от непредвиденных обстоятельств должен включать такие элементы, как:

- сведения о том, кто является главным ответственным лицом и как можно установить с ним контакт;
- информация о том, кто и на каком основании принимает решение о возникновении необычной ситуации;
- технические требования к передаче управления резервным службам, которые могут включать следующее:
  - ♦ сведения о расположении альтернативных сайтов;
  - ♦ сведения о необходимом дополнительном оборудовании;
  - ♦ сведения о необходимости дополнительных линий связи.
- организационные требования в отношении персонала, который осуществляет передачу управления резервным службам;
- сведения о любых внешних источниках, в которых можно будет получить помощь,— например, о поставщиках оборудования;
- сведения о наличии страховки на случай данной конкретной ситуации.

Любой разработанный план защиты от непредвиденных обстоятельств должен периодически пересматриваться и тестироваться на предмет его осуществимости.

Контроль за персоналом

Создатели коммерческих СУБД возлагают всю ответственность за эффективность управления системой на ее пользователей. Поэтому, с точки зрения защиты системы, исключительно важную роль играют отношение к делу и действия людей, непосредственно вовлеченных в эти процессы. Важность этого замечания подчеркивается тем, что основной риск в любой организации связан с действиями, производимыми сотрудниками самой организации, а не с возможными внешними угрозами. Отсюда следует, что обеспечение необходимого уровня контроля за персоналом позволяет минимизировать возможный риск.

Защита помещений и хранилищ

Основное оборудование системы, включая принтеры, если они используются для печати конфиденциальной информации, должно размещаться в запираемом помещении с ограниченным доступом, который должен быть разрешен только основным специалистам. Все остальное оборудование, особенно переносное, должно быть надежно закреплено в месте размещения и снабжено сигнализацией. Однако условие держать помещение постоянно запертым может оказаться нежелательным или неосуществимым, поскольку работникам может требоваться постоянный доступ к установленному там оборудованию.

Выше, при обсуждении вопросов резервного копирования, уже упоминалось о необходимости иметь защищенное помещение, предназначенное для хранения носителей информации. Для любой организации жизненно необходимо иметь подобное надежно защищенное место, в котором будут храниться копии программ, резервные копии системы, прочие архивные материалы и документация. Предпочтительно, чтобы это помещение находилось на площадке, отличной от места расположения основного оборудования системы. Все носители информации, включая диски и магнитные ленты, должны храниться в несгораемых

сейфах. Аналогичным образом должна сохраняться и документация. Все, что хранится в подобном помещении, должно быть зарегистрировано в специальном каталоге, с указанием даты помещения носителя или документа на хранение. Периодичность, с которой новые материалы будут помещаться в подобный архив, должна регламентироваться разработанными процедурами копирования. Кроме того, организации могут использовать и внешние хранилища, периодически доставляя туда вновь созданные резервные копии и другие архивные материалы, причем частота доставки также должна определяться установленными нормами и процедурами. Существуют независимые компании, специализирующиеся на организации внешних хранилищ информации. Они предоставляют свои услуги многочисленным компаниям-клиентам.

Гарантийные договора

Гарантийные договора представляют собой юридические соглашения в отношении программного обеспечения, заключаемые между разработчиками программ и их клиентами, на основании которых некоторая третья фирма обеспечивает хранение исходного текста программ приложения, разработанного для клиента. Это одна из форм страховки клиентов на случай, если компания-разработчик отойдет от дел. В этом случае клиент получит право забрать исходные тексты программ у третьей фирмы, вместо того чтобы остаться с приложением, лишенным всякого сопровождения. Данная область юриспруденции считается одной из тех, в которых очень часто допускаются ошибки и различные недооценки (рис. 5.6). По мнению некоторых авторов, (Revella, 1993), 95% всех гарантийных договоров по программному обеспечению не достигает своей изначальной цели.

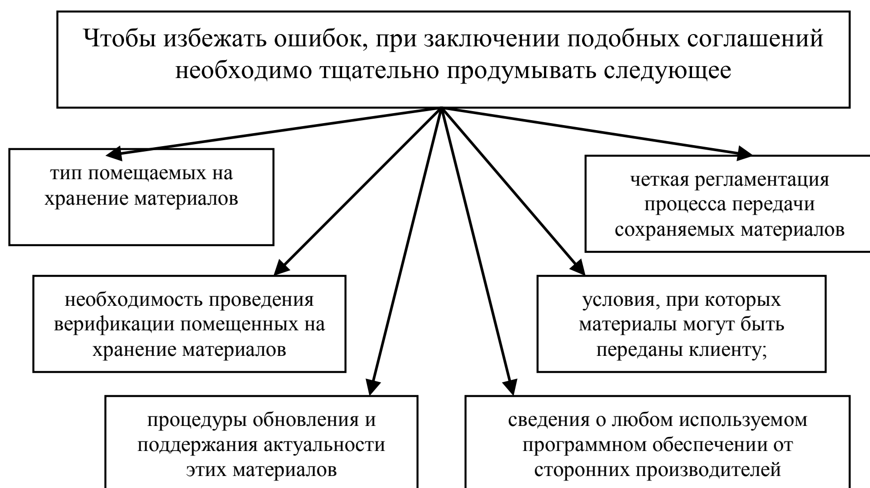


Рис. 5.6. Меры избегания ошибок

Договора о сопровождении

Для всего используемого организацией оборудования и программного обеспечения сторонней разработки или изготовления обязательно должны быть заключены соответствующие договора о сопровождении. Установленный интервал ожидания ответа в случае любого отказа или ошибки зависит от важности отказавшего элемента для нормального функционирования всей системы. Например, если произойдет отказ сервера базы данных, желательна немедленная реакция обслуживающей организации с целью скорейшего возвращения системы в работоспособное состояние. Однако в случае отказа принтера договор вполне может предусматривать устранение неисправности в течение рабочего дня или даже двух — предполагается, что всегда найдется подходящий резервный принтер, который можно будет использовать до устранения аварии. В некоторых случаях договором может быть предусмотрена временная замена неисправного оборудования на период ремонта отказавших технических средств.

Контроль за физическим доступом

В этом разделе мы кратко рассмотрим, какие методы могут быть использованы для организации контроля за физическим доступом к оборудованию и т.д. В целом, все методы контроля могут быть разделены на внешние и внутренние.

Внутренний контроль

Этот метод контроля используется внутри отдельных зданий и предназначен для управления тем, кто будет иметь доступ в определенные помещения этих зданий. Например, наиболее «ответственные» помещения (скажем, те, в которых установлены основные компьютеры) могут быть оборудованы системами входного контроля. В подобных системах могут использоваться самые различные принципы — например, специальные ключи, карточки, кодовые замки или средства указания пароля. Наиболее сложные комплексы предполагают использование отпечатков пальцев, снимков радужной оболочки глаз, фиксацию особенностей голоса или почерка. Однако в настоящее время очень не многие коммерческие организации применяют столь изощренные методы контроля — в основном из-за их высокой стоимости.

Внешний контроль

Данный метод контроля применяется вне строений и предназначен для ограничения доступа на площадку или в отдельные здания. Для наблюдения за территорией может использоваться специальная охрана, в обязанности которой входит контроль входа/выхода персонала и посетителей организации. Следует отметить, что основной задачей любых механизмов контроля за физическим доступом является обеспечение их эффективности, однако требуемая эффективность должна достигаться без создания дополнительных препятствий персоналу в выполнении их служебных обязанностей. В противном случае возрастает

опасность поиска персоналом всевозможных путей обхода установленных требований.

Защита ПК

В отличии от больших компьютерных систем, защита вычислительных комплексов, состоящих из обычных персональных компьютеров, может представлять собой серьезную проблему, поскольку их перемещение не для кого не составит труда. Общепринятые меры контроля доступа, применяемые для мейнфреймов и миникомпьютеров, мало подходят для систем, состоящих из ПК. Основное затруднение состоит в том, что подобные компьютеры обычно располагаются непосредственно на рабочих местах сотрудников. Поэтому какой-либо специальный контроль за физическим доступом к этим устройствам, как правило, отсутствует — за исключением мер, принимаемых для защиты всего здания или отдельных его помещений.

В настоящее время большинство персональных компьютеров оборудуется замками, блокирующих несанкционированный доступ к клавиатуре. Этот способ не обеспечивает высокой степени защиты компьютера, однако вполне подходит для предотвращения случайного доступа к ПК. Более высокую степень защищенности предоставляет организация доступа к системе с использованием индивидуальных идентификаторов пользователей и соответствующих личных паролей. Однако в этом случае необходимо, чтобы пользователи не оставляли компьютер в активном состоянии на сколько-нибудь продолжительный период времени.

Если обрабатываемые данные сохраняются не в самом компьютере, а на дисках, то, используя специальные хранилища, можно обеспечить необходимую степень их защиты. В этом случае каждый пользователь будет лично отвечать за создание любых необходимых копий данных и программного обеспечения, а также за организацию их защищенного хранения. Следовательно, очень важно, чтобы

весь работающий с ПК персонал знал методы и процедуры, которые должны использоваться для организации защиты компьютерного оборудования и данных.

Защита от компьютерных вирусов

Важнейшей проблемой, особенно актуальной для среды ПК, является риск занесения в систему нежелательных и зачастую просто опасных программ, называемых компьютерными вирусами. Беспечное и небрежное отношение к методам использования оборудования часто приводит к поражению систем различными вирусами — например, в тех случаях, когда персонал приносит на рабочие места и запускает на своих компьютерах различные игровые программы. Распространению вирусов в системе способствует бесконтрольный обмен дисками, хотя некоторые вирусы способны самостоятельно распространяться в сетевой среде.

Установленные требования к защите системы должны содержать четкие указания о процедурах, которые должны применяться к любому программному обеспечению, прежде чем оно будет допущено к установке в системе — даже в тех случаях, когда оно поступило непосредственно от разработчика или другого надежного источника. Кроме того, в этих требованиях должно явно указываться, что в системе может использоваться только программное обеспечение, прошедшее необходимый контроль. Надежной защитой от данного типа угрозы могут также служить надлежащие процедуры поверки и тестирования нового и модернизированного программного обеспечения.

СУБД и защита в Web

Меры защиты, связанные с использованием СУБД представлены на рис. 5.7.

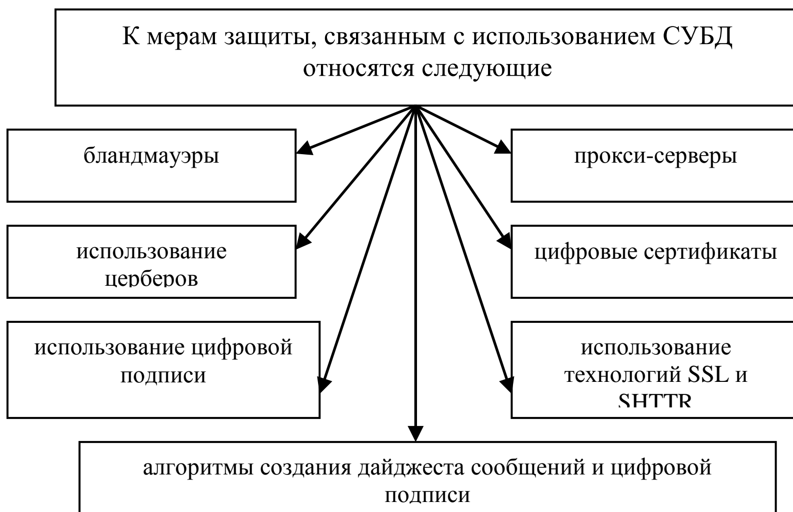


Рис. 5.7. Меры защиты, связанные с использованием СУБД

Защита статистических баз данных

Обычно статистические базы данных используются для генерации статистической информации, например, для определения усредненных и сводных показателей различных наборов данных. Однако информация отдельных записей в статистической базе данных должна сохраняться конфиденциально и не должна быть доступна пользователям. Основная проблема при работе с подобными типами баз данных состоит в возможности в возможности использования ответов на допустимые запросы для получения ответов на запрещенные запросы. Для разрешения этой проблемы могут использоваться различные стратегии.

- Предотвращение выполнения запросов, работающих с незначительным количеством записей базы данных.
- Случайное добавление дополнительных строк к основному набору данных, обрабатываемому запросу. В результате ответ будет содержатьопределенную ошибку и позволит лишь оценить правильное значение.
- Использование для ответов на запросы только случайных наборов исходных данных.

- Сохранение исторических сведений о результатах выполнения запросов и отклонение любых запросов, использующих значительное количество исходных данных, обработанных предыдущим запросом.

Типичными примерами статистических баз данных являются хранилища данных (warehouse) и магазины данных (data mart).

Оценка риска

В этом разделе содержится краткий обзор основных этапов процедуры оценки риска. Полный перечень этих этапов приведен ниже.

- Определение состава группы защиты.
- Определение анализируемой области и сбор информации о системе.
- Выявление всех существующих средств защиты.
- Выявление и оценка всех имеющихся активов.
- Выявление и оценка всех существующих опасностей и угроз.
- Подбор необходимых контрмер, проведение сравнительного анализа требуемых затрат и достигаемого эффекта, сравнение с уже существующими средствами защиты.
- Подготовка рекомендаций.
- Тестирование системы защиты.

Определение состава группы защиты

Любая организация, нуждающаяся в обеспечении защиты своей вычислительной среды, должна выделить группу людей, ответственных за проведение соответствующих мер. Хотя размер и состав группы должны определяться, прежде всего масштабом и сложностью структуры самой организации, весьма вероятными кандидатами на участие в подобной группе являются представители отдела информационных технологий, отдела кадров, юридической службы, ревизионной группы и

службы, отвечающих за эксплуатацию строений. Цели, стоящие перед группой защиты:

- разработка требований защиты, включая подготовку необходимых стандартов и требуемых процедур;
- проведение анализа риска;
- подбор, рекомендация и эффективная реализация необходимых мер защиты;
- контроль функционирования и сопровождение системы защиты;
- поддержание оптимального баланса между защищенностью системы и эффективностью ее использования.

Определение анализируемой области и сбор информации о системе

За исключением совсем небольших организаций, рекомендуется последовательно сосредотачивать внимание на одной определенной области, например, на отдельной компьютерной подсистеме. Все особенности функционирования этой подсистемы должны быть зафиксированы в документации, что позволит ясно представить ее функциональное назначение.

Выявление всех существующих средств защиты

Необходимо выделить все существующие средства противодействия возможным угрозам. Все они последовательно должны сравниваться с любыми предлагаемыми новыми средствами с целью создания оптимального набора необходимых контрмер, отвечающего уровню подверженности системы той или иной опасности.

Выявление и оценка всех имеющихся активов

Каждый связанный с системой актив должен быть выявлен и оценен. Оценке подлежит все оборудование системы, все используемое в ней программное обеспечение (как прикладное, так и системное), все необходимые и используемые данные, задействованный персонал, строения и другие

элементы, например, энергия, документация и расходные материалы. При определении цены обязательно потребуется выполнить некоторые расчеты. Для таких активов, как оборудование, отправной точкой расчетов является их стоимость. Однако задача оценки стоимости данных является более сложной, если только данные не были куплены в некотором внешнем источнике в этом случае отправной точкой расчетов также является их цена.

Обычно данные могут подвергаться опасности потери различных уровней полному уничтожению, временной недоступности или просто несанкционированному раскрытию. Может быть установлена стоимость любого элемента данных на каждый возможный случай их потери. Помимо всего прочего, она будет зависеть от таких требований, как сохранение неприкосновенности личных данных, коммерческая тайна и др. Максимальная стоимость устанавливается на случай максимально возможных потерь. После выполнения независимой оценки отдельных активов следует установить взаимосвязи между ними, например, сохранение данных зависит от определенного оборудования и программного обеспечения. Полученные результаты сопоставляются и согласовываются, причем самая большая стоимость устанавливается на случай самого тяжелого из всех существующих вариантов потерь.

Выявление и оценка всех существующих опасностей и угроз

Каждая потенциально важная опасность должна быть зафиксирована, начиная от самых очевидных, которые могут происходить очень часто, и заканчивая абсолютно исключительными ситуациями, которые возникают крайне редко, но все же потенциально возможны. Каждая опасность рассматривается в свете потерь, к которым она может привести. Для каждого типа актива системы определяется показатель в отношении каждой потенциальной угрозы, вызывающей каждый тип потери. Это значение должно учитывать

вероятность появления данной угрозы. Аналогичным образом устанавливается значение, определяющее вероятность реализации угрозы на практике (уязвимость). Все эти цифры затем обобщаются и согласуются с целью определения риска, создаваемого каждым типом опасности.

Подбор необходимых контрмер, проведение сравнительного анализа требуемых затрат и достигаемого эффекта, сравнение с уже существующими средствами защиты.

После завершения оценки риска можно выбрать контрмеры, необходимые для его уменьшения, руководствуясь соотношением стоимости и эффективности. В каждом случае может быть доступно несколько различных вариантов; целью данного этапа является достижение максимальной эффективности за минимальную цену, что и будет характеризовать общие выгоды от реализации той или иной контрмеры. Обязательно следует провести сравнение с уже существующими мерами обеспечения безопасности, поскольку они вполне могут обеспечивать достаточно высокий уровень защищенности той или иной части системы.

Подготовка рекомендаций

Завершив подбор всех необходимых контрмер, группа защиты должна подготовить отчет, содержащий подробные рекомендации, сделанные на базе выполненного обследования (если оно необходимо). Помимо подачи рекомендаций, в обязанности группы защиты входит наблюдение за их практической реализацией.

Тестирование системы защиты

Очень полезной практикой следует считать периодическое проведение тестирования всех реализованных средств защиты, что позволяет получить гарантии их реальной способности противостоять различным опасностям и угрозам. Подобное тестирование позволяет выявить и те средства, которые потенциальные злоумышленники могут разработать в ответ на установленные меры защиты, что, в свою очередь,

послужит основанием для дальнейшего совершенствования и укрепления защиты системы.

Законодательство по защите и неприкосновенности данных

Законодательство по защите и гарантии неприкосновенности данных касается личных данных граждан и правам каждого человека на неприкосновенность его личных данных. Этот тип законодательства служит, с одной стороны, для защиты личных данных граждан от различных злоупотреблений, а с другой стороны, для предоставления организациям (как общественным, так и частным) прав, необходимых для выполнения предусмотренных законом действий или обязанностей.

Неприкосновенность личных данных определяет право каждого человека на защиту его личных данных от сбора, хранения и опубликования (как в отношении лично его, так и неперсонально, в общем потоке).

В последние годы существенно возросла общая озабоченность по поводу сохранения неприкосновенности личных данных. Эта озабоченность вызвана и связана в основном с разработкой новых компьютерных технологий, появившихся в результате быстрого роста объема сохраняемых в компьютерах данных и предусматривающих определенные изменения в методах хранения и использования информации. Многие из этих методов включают обмен или совместную обработку информации, поступающей из различных компьютерных систем (возможно, даже расположенных в различных странах), и далеко не всегда регулируются каким-либо законодательством.

В результате во многих странах был поднят вопрос об охране личных данных граждан как внутри, так и за пределами их государственных границ, что потребовало создания определенной международной кооперации. Результатами этой кооперации стало приведение существующих национальных

законодательств в некоторое соответствие. Однако все еще существуют определенные различия, как в самом законодательстве, так и в методах его разработки. Тем не менее, именно всеобщая озабоченность по поводу неприкосновенности личных данных вызвала к жизни разработку законов о защите личных данных.

Защита личных данных - законодательство, предусматривающее защиту личных данных от незаконного сбора, хранения и опубликования, а также предоставляющее необходимые гарантии недопустимости уничтожения или искажения данных, сохраняемых с разрешения законов.

Приведенное выше определение защиты личных данных достаточно широко и не содержит конкретных требований, например, к способам хранения данных. Хотя в настоящее время некоторые законодательные акты о защите личных данных касаются только тех данных, которые сохраняются в компьютерах, этот аспект законодательства пересматривается и в будущем весьма вероятно проявление законов, имеющих более широкую область применения. Именно по этой причине мы исключили из определения замечания по поводу методов хранения данных.

В общем случае, как в Северной Америке, так и в Европе, действующее законодательство представляет отдельным лицам некоторый набор основных прав, например, право знать, что их данные собираются и хранятся, а также право требовать изменения некорректных сведений. Безусловно, существуют и исключения, например, области, связанные с проблемами национальной безопасности, или случаи, когда разглашение сведений может помешать проведению юридических расследований. Кроме того, существуют отличия и между национальными законодательствами. Например, в некоторых случаях законодательство определяет отношение к любым данным, как

собираемым вручную, так и накапливаемым в компьютерах, или же имеет несколько различных вариантов в административных единицах федеральных государств, например, в Германии и США. Как бы там ни было, в связи с продолжающимся ростом обмена данными между отдельными странами и необходимостью защиты этих данных от недобросовестного использования потребуется дальнейшее согласование основных юридических положений в этой области юриспруденции.

Резюме

- **Защита баз данных** предусматривает предотвращение любых преднамеренных и непреднамеренных угроз с использованием компьютерных и некомпьютерных средств контроля.
- Защите подлежат не только собственно данные, сохраняемые в базе. Нарушения защиты могут повлиять на другие части системы, в результате чего пострадает и сама база данных. Поэтому защита баз данных охватывает оборудование, программное обеспечение, персонал и собственно данные.
- Целью организации защиты базы данных является предотвращение таких нарушений, как похищение и фальсификация, утрата конфиденциальности (тайности), нарушение неприкосновенности личных данных, утрата целостности данных и потеря доступности данных.
- **Угрозой** считается любая ситуация или событие, как внутреннее, так и внешнее, которое способно нежелательным образом воздействовать на систему, а значит, и на всю организацию в целом.
- **Компьютерные средства контроля** в многопользовательской вычислительной среде включают следующее: авторизацию пользователей, представления, средства копирования/восстановления, инструменты поддержания целостности данных, шифрование и вспомогательные процедуры.

- **Авторизация** пользователей заключается в предоставлении им необходимых прав (или привилегий), позволяющих их владельцу получать санкционированный доступ к системе или ее объектам.
- **Аутентификация** представляет собой механизм определения того, является ли данный пользователь именно тем, за кого себя выдает.
- **Представление** является динамическим результатом одной или более реляционных операций, выполняемых над базовыми отношениями с целью создания нового отношения. Представление является виртуальным отношением, которого реально в базе данных не существует, но которое воспроизводится по каждому запросу пользователя в момент поступления этого запроса. Механизм представлений является мощным и весьма гибким инструментом организации защиты, позволяющим скрывать от определенных пользователей некоторую часть базы данных.
- **Резервное копирование** представляет собой процесс периодического создания копии базы данных и ее файла журнала (а также, возможно, программного обеспечения), помещаемых на внешние по отношению к системе носители информации. **Ведение журнала** базы данных представляет собой процесс создания и обработки файла журнала, в котором фиксируются все изменения, внесенные в базу данных с момента создания ее последней резервной копии, что позволяет эффективно восстанавливать систему в случае ее отказа. **Контрольная точка** обозначает момент синхронизации между состоянием базы данных и журналом регистрации транзакций. При создании контрольной точки содержимое всех системных буферов принудительно выводится на диск.
- **Средства поддержания целостности данных** также вносят свой вклад в защиту базы данных, поскольку предотвращают переход данных в несогласованное состояние, а значит, и получение ошибочных или некорректных результатов расчетов.

- **Шифрование** представляет собой процедуру кодирования данных с использованием специальных алгоритмов, которые делают данные непригодными для чтения с помощью любой программы, не имеющей ключа шифрования.
- **Некомпьютерные средства контроля** включают разработку мер обеспечения безопасности, планирование защиты от непредвиденных обстоятельств, контроль за персоналом, защиту мест размещения оборудования, заключение гарантийных договоров и договоров на сопровождение, а также контроль за физическим доступом.
- **Статистические базы** данных используются для генерирования статистической информации — например, усреднения и суммирования информации в различных наборах данных. Однако сведения об отдельных записях статистической базе данных должны сохраняться конфиденциальными и недоступными для пользователей. Основная проблема при работе с подобными базами данных заключается в исключении возможности с помощью ответов на допустимые запросы получить информацию об ответах на недопустимые запросы.
- **Неприкосновенность личных данных** определяет право каждого человека на защиту его личных данных от сбора, хранения и опубликования (как в отношении лично его, так и неперсонально, в общем потоке).
- **Защита личных данных** состоит в защите личных данных каждого человека от незаконного сбора, хранения и опубликования, а также в предоставлении необходимых гарантий недопустимости уничтожения или искажения данных, сохраняемых с разрешения закона.

6. СРЕДСТВА ПОДДЕРЖАНИЯ ЦЕЛОСТНОСТИ БАЗЫ ДАННЫХ

6.1. Основные понятия. Транзакции и их свойства

СУБД должна иметь доступный конечному пользователю системный каталог или словарь данных.

СЛОВАРЬ (СИСТЕМНЫЙ КАТАЛОГ) – это хранилище данных, которые описывают сохраняемую в базе данных информацию, т.е. метаданные, или «данные о данных»

Системный каталог СУБД является одним из фундаментальных компонентов системы. Многие программные компоненты СУБД строятся на использовании данных, хранящихся в системном каталоге. Например, модуль контроля прав доступа использует системный каталог для проверки наличия у пользователя полномочий, необходимых для выполнения запрошенных им операций. Для проведения подобных проверок системный каталог должен включать следующие компоненты:

- имена пользователей, для которых разрешен доступ к базе данных;
- имена элементов данных в базе данных
- элементы данных, к которым каждый пользователь имеет право доступа, и разрешенные типы доступа к ним – для вставки, обновления, удаления или чтения.

Другим примером могут служить средства проверки целостности данных, которые используют словарь для проверки того, удовлетворяет ли запрошенная операция всем установленным ограничениям поддержки целостности данных. Для выполнения этой проверки в системном каталоге должны храниться такие сведения:

- имена элементов данных из базы данных
- типы и размеры элементов данных

- ограничения, установленные для каждого из элементов

Термин «словарь данных» часто используется для программного обеспечения более общего типа, чем просто каталог СУБД. Система словаря данных может быть либо активной, либо пассивной. *Активная* система всегда согласуется со структурой базы данных, поскольку она автоматически поддерживается этой системой. *Пассивная* система может противоречить состоянию базы данных из-за инициируемых пользователями изменений. Если словарь данных является частью базы данных, то он называется *интегрированным* словарем данных. *Изолированный* словарь данных обладает своей собственной специализированной СУБД. Его предпочтительно использовать на начальных этапах проектирования базы данных для некоторой организации, когда требуется отложить на какое-то время привязку к конкретной СУБД. Однако, недостаток этого подхода заключается в том, что после выбора СУБД и воплощения базы данных изолированный словарь данных значительно труднее поддерживать в согласии с состоянием БД.

Недавно была предпринята попытка стандартизировать интерфейс словарей данных для достижения большей доступности и упрощения их совместного использования. Ее результатом стала разработка службы словаря информационных ресурсов – IRDS. Определения IRDS были приняты в качестве стандарта Международной организацией стандартизации.

Стандарты IRDS определяют набор правил хранения информации в словаре данных и доступа к ней, преследуя при этом три следующих цели:

- расширяемость данных;
- целостность данных;
- контролируемый доступ к данным.

Требования поддержки целостности данных

Назначение требований поддержки целостности данных состоит в поддержании постоянной внутренней согласованности информации БД. Существуют следующие требования поддержки целостности данных:

- Обязательные данные. Необходимо установить, какие из атрибутов должны содержать одно из допустимых значений. другими словами, нас интересуют атрибуты, которые всегда должны иметь конкретные значения, отличные от NUUL.
- Ограничения для доменов атрибутов. Домен атрибутов устанавливает набор допустимых значений, которые могут присваиваться этому атрибуту.

Целостность сущностей. Атрибут первичного ключа сущности не может иметь значение NULL.

- Ссылочная целостность. Связи между сущностями моделируются посредством помещения в дочернее отношение копии первичного ключа родительского отношения. Понятие ссылочной целостности означает, что если внешний ключ дочернего отношения содержит некоторое значение, то это значение должно ссылаться на существующее и конкретное значение ключа в родительском отношении. Поддержка ссылочной целостности организуется посредством задания необходимых ограничений для значений первичных и внешних ключей. Эти ограничения определяют условия, которые должны соблюдаться при обновлении или удалении значений первичного ключа, а также при вставке или обновлении значений внешнего ключа. При этом вставка нового значения первичного ключа или удаление значений внешнего ключа не вызывает каких-либо проблем со ссылочной целостностью.

Для каждого внешнего ключа отношения следует указать условия, которые должны выполняться при обновлении или удалении соответствующего значения первичного ключа.

- Требования данного предприятия. Эти требования иначе называются бизнес-правилами и определяются теми методами и ограничениями, которые приняты на данном предприятии в отношении выполнения некоторых операций.

- Документирование всех ограничений целостности данных. Все установленные требования поддержки целостности данных для локальной модели должны быть детально отображены в документации.

Механизм транзакций

Все современные СУБД имеют средства для выполнения трех важнейших функций:

- механизм поддержания транзакций;
- службы управления параллельностью;
- средства восстановления баз данных.

Эти функции находятся во взаимной зависимости и предназначены для защиты баз данных от потери данных или их перехода в несогласованное состояние. Рассмотрим их по отдельности.

Транзакция – это действие или серия действий, выполняемых одним пользователем или прикладной программой, которые осуществляют доступ или изменение содержимого базы данных.

Транзакция является логической единицей работы, выполняемой в базе данных. Она может быть представлена отдельной программой, являться частью алгоритма программы или даже отдельной командой и включать произвольное количество операций, выполняемых в БД. С точки зрения БД, выполнение программы некоторого приложения может расцениваться как серия транзакций, в промежутках между которыми выполняется некоторая обработка данных, осуществляемая вне среды БД.

Любая транзакция всегда должна переводить БД из одного согласованного состояния в другое, хотя допускается, что согласованность состояния БД нарушается в ходе выполнения транзакции.

Любая транзакция завершается одним из двух возможных способов. В случае успешного завершения результаты транзакции фиксируются в БД, и последняя переходит в новой согласованное состояние. Если выполнение транзакции не увенчалось успехом, она отменяется. В этом случае в базе данных должно быть восстановлено то согласованное состояние, в котором она находилась до начала данной транзакции. Этот процесс называется ***откатом***. Зафиксированная транзакция не может быть отменена. Следует отметить, что отмененная транзакция может быть еще раз запущена позже и, в зависимости от причин предыдущего отказа, вполне успешно завершена и зафиксирована в БД.

Никакая СУБД не обладает внутренней возможностью установить, какие именно изменения должны быть восприняты как единое целое, образующие одну логическую транзакцию. Следовательно, должен существовать метод, позволяющий указывать границы каждой из транзакций извне, со стороны пользователя. В большинстве языков манипулирования данными для указания границ отдельных транзакций используются операторы Begin Transaction, Commit и RollBack (или их эквиваленты).

Свойства транзакций

Существуют некоторые свойства, которыми должна обладать любая из транзакций:

- Атомарность – любая транзакция представляет собой неделимую единицу работы, которая может быть либо выполнена вся целиком, либо не выполнена вовсе.
- Согласованность – каждая транзакция должна переводить БД из одного согласованного состояния в другое согласованное состояние.
- Изолированность – все транзакции выполняются независимо одна от другой. Другими словами, промежуточные результаты незавершенной транзакции не должны быть доступны тругим транзакциям.

- Продолжительность – результаты успешно завершенной транзакции должны сохраняться в базе данных постоянно и не должны быть утеряны в результате последующих сбоев.

Менеджер транзакций осуществляет координацию работы транзакций, выполняемых прикладными программами. Он взаимодействует с *планировщиком*, отвечающим за реализацию выбранной стратегии управления параллельностью. В некоторых случаях планировщик называют *менеджером блокировки*, если используемых протокол управления параллельностью строится на основе блокировки. Цель работы планировщика состоит в достижении максимально возможного уровня параллельности, при условии исключения влияния параллельно выполняющихся транзакций друг на друга, поскольку это может послужить источником нарушения согласованности базы данных.

Если в процессе выполнения транзакций происходит отказ, то база данных может оказаться в несогласованном состоянии. Задачей *менеджера восстановления* является предоставление гарантий того, что в подобном случае база данных будет автоматически возвращена в то состояние, в котором она находилась до начала данной транзакции, следовательно, останется согласованной.

Менеджер буферов отвечает за передачу данных между основной памятью компьютера и вторичной дисковой памятью.

6.2. Понятие управление параллельностью

Управление параллельностью – это процесс организации одновременного выполнения в базе данных различных операций, гарантирующий исключение их взаимного влияния друг на друга.

Если пользователи только читают данные, то в этом случае работа каждого из них не оказывает никакого влияния на работу остальных пользователей.

Однако, если несколько пользователей одновременно обращаются к базе данных и хотя бы один из них имеет целью обновить хранимую в базе информацию, возможно взаимное влияние процессов друг на друга, способное привести к несогласованности данных.

Данная задача подобна задачам, стоящим перед любой многопользовательской системой. Выполнение приложений в ней чередуется, таким образом достигается их параллельное выполнение.

Однако, несмотря на то, что каждая из транзакций может сама по себе выполняться вполне корректно, подобное чередование операций способно приводить к неверным результатам, из-за чего целостность и согласованность базы данных

Рассмотрим три примера потенциальных проблем, которые могут иметь мести при параллельном выполнении транзакций – проблему потерянного обновления, проблему зависимости от нефиксированных результатов и проблему несогласованной обработки.

Проблема потерянного обновления

Результаты вполне успешной завершенной операции обновления одной транзакции могут быть перекрыты результатами выполнения другой транзакции. Эта аномалия известна как проблема потерянного обновления (табл. 6.1).

Таблица 6.1

Пример проблемы потерянного времени

Время	Транзакция T1	Транзакция T2	Поле a1
t1		начало	100
t2	начало	чтение a1	100
t3	чтение a1	a1=a1+100	100
t4	a1=a1-10	запись a1	200
t5	запись a1	commit	90
t6	commit		90

Например, транзакция Т2 выполняется параллельно с транзакцией Т1.Транзакция Т1 заключается в снятии 10 тысяч рублей со счета, на котором исходно находится 100 тыс. руб., транзакция Т2 предполагает помещение 100 тыс. руб. на этот же счет. Если обе транзакции выполняются последовательно, то в результате должно оказаться 190 тыс. руб. на счете.

Пусть они выполняются параллельно и начинаются практически одновременно Т2 увеличивает сумму на 100 т.р. и записывает результат, равный 200 т.р. Тем временем Т1 уменьшает значение своей копии исходной суммы на 10 т.р. и записывает результат 90 т.р., перекрывая результат предыдущего обновления. Избежать потери результатов выполнения транзакции Т2 можно, запретив транзакции Т1 считывать исходное значение на счету вплоть до завершения выполнения транзакции Т2.

Проблема зависимости от нефиксированных результатов

Проблема зависимости от нефиксированных результатов возникает в том случае, если одна из транзакций получит доступ к промежуточным результатам выполнения другой транзакции до того, как они будут зафиксированы в БД. Пусть, например, пусть Т4 увеличивает значение суммы на счете до 200 т.р., после чего выполнение транзакции отменяется, поэтому СУБД должна выполнить откат с восстановлением исходного состояния 100 т.р. Предположим, однако, что транзакция Т3 уже успела считать измененное значение счета и использовать это значение для операции снятия денег со счета. Тогда на счете останется не 90 т.р., а 190 т.р. Причина выполнения отката транзакции Т4 несущественна – допустим, что при ее выполнении была обнаружена некоторая ошибка. Источник ошибки заключается в предположении транзакции Т3, что выполненное в транзакции Т4 изменение будет успешно зафиксировано в БД, хотя на самом деле имел место откат этой транзакции. Проблему можно устранить, запретив транзакции

Т3 считывать значение счета до тех пор, пока транзакция Т4 не будет либо зафиксирована, либо отменена.

Пример проблемы зависимости от нефиксированных результатов представлен в табл. 6.2.

Таблица 6.2

Пример проблемы зависимости от нефиксированных результатов

Время	Транзакция Т3	Транзакция Т4	Поле a1
t1		начало	100
t2		чтение a1	100
t3		a1=a1+100	100
t4	начало	запись a1	200
t5	чтение a1	…	200
t6	a1=a1-10	откат	100
t7	запись a1		190
t8	commit		190

Проблема несогласованной обработки

Транзакции, которые только считывают информацию из БД, также могут давать неверные результаты, если им будут доступны для чтения промежуточные результаты одновременно выполняющихся и еще не завершенных транзакций, обновляющих информацию в базе. В некоторых случаях эту проблему называют ***чтением мусора*** или ***неповторяемостью чтения***.

Проблема несогласованной обработки возникает в тех случаях, когда транзакция считывает несколько значений из БД, после чего вторая транзакция обновляет некоторые из этих значений непосредственно во время выполнения первой транзакции.

Например, Т6 – вычисляет сумму остатков на 3-х счетах, а Т5 – переносит 10 т.р. с одного счета на другой. Тогда Т1 взяла остаток на 1-м и 2-м счетах, а Т2 перенесла 10 т.р. с 1-го счета на 3-й. Итог окажется на 10 т.р. больше, чем нужно. Эту проблему можно устранить, запретив транзакции Т6 считывать

значения на счетах, до тех пор, пока транзакция T5 не зафиксирует выполненные ею обновления.

Еще один пример несогласной обработки представлен в табл. 6.3.

Таблица 6.3

Пример несогласной обработки

Время	Транзакция T5	Транзакция T6	Поле a1	Поле в1	Поле c1	Поле Sum
t1		начало	100	50	25	0
t2	начало	Sum=0	100	50	25	0
t3	чтение a1	чтение a1	100	50	25	0
t4	a1=a1-10	Sum= Sum+a1	100	50	25	100
t5	запись a1	чтение в1	90	50	25	100
t6	чтение c1	Sum= Sum+в1	90	50	25	150
t7	c1=c1+10		90	50	25	150
t8	запись c1		90	50	35	150
t9	commit	чтение c1	90	50	35	150
t10		Sum= Sum+c1	90	50	35	185
t11		commit	90	50	35	185

Упорядочиваемость и восстанавливаемость

Назначение протоколов управления параллельностью состоит в подготовке такого графика выполнения транзакций, который исключит возможность их влияния на результаты работы друг друга. Одно из очевидных решений состоит в выполнении в каждый момент времени только одной транзакции. Однако, назначение многопользовательских СУБД состоит в обеспечении максимальной степени параллельности выполнения транзакций пользователей, поэтому те транзакции, которые не оказывают влияния на работу друг друга, вполне могут выполняться одновременно.

Познакомимся с понятием упорядочиваемости как со средством, способным помочь в выявлении тех транзакций,

которые ***гарантированно*** не вызовут проблем нарушения согласованности данных при одновременном выполнении.

График – это последовательность запуска операций множества параллельно выполняемых транзакций, сохраняющая очередность выполнения операций в каждой отдельной транзакции.

Последовательный график – график, в котором операции каждой из транзакций выполняются строго последовательно и не могут чередоваться с операциями, выполняемыми в других транзакциях.

Непоследовательный график - график, в котором чередуются операции из некоторого набора одновременно выполняемых транзакций.

Суть **упорядочивания** состоит в отыскании таких непоследовательных графиков, которые позволят транзакциям выполняться параллельно, но без оказания взаимного влияния друг на друга, и, привести БД в состояние, которое может быть достигнуто при использовании последовательного графика.

В отношении достижения упорядоченности весьма важен порядок выполнения операций чтения и записи данных.

- Если две транзакции считывают некоторый элемент данных, они не будут конфликтовать между собой и порядок их выполнения не имеет значения.
- Если две транзакции считывают или записывают совершенно независимые элементы данных, они не будут конфликтовать между собой и порядок их выполнения не имеет значения.
- Если одна транзакция записывает элемент данных, а другая транзакция этот же элемент данных считывает или записывает, порядок их выполнения имеет существенное значение.

Пример эквивалентных графиков (табл. 6.4):

Вариант А – непоследовательный график S1

Вариант Б – непоследовательный график S2, эквивалентный графику S1

Вариант В – последовательный график S3, эквивалентный графикам S1 и S2.

Таблица 6.4

Пример эквивалентность графиков

А		Б		В	
T7	T8	T7	T8	T7	T8
начало		начало		начало	
чтение a1		чтение a1		чтение a1	
запись a1		запись a1		запись a1	
	начало		начало	чтение в1	
	чтение a1		чтение a1	запись в1	
	запись a1	чтение в1		commit	
чтение в1			запись a1		начало
запись в1		запись в1			чтение a1
commit		commit			запись a1
	чтение в1		чтение в1		чтение в1
	запись в1		запись в1		запись в1
	commit		commit		commit

Поскольку операции записи на счет a1 в T8 не конфликтует с последующей операцией чтения в1 в T7, можно изменить порядку выполнения этих операций и получить эквивалентный график S2.

Если поменять порядок выполнения и следующих не конфликтующих между собой операций, то мы получим эквивалентный последовательный график S3.

Если в результате упорядочивания можно получить последовательный график, то этот тип упорядочивания принято называть ***конфликтным упорядочиванием***. В конфликтно упорядоченном графике порядок выполнения любых конфликтующих операций соответствует размещению в последовательном графике.

Для проверки конфликтной упорядоченности можно использовать ***граф предшествования***, который состоит из следующих элементов:

- вершин, соответствующих каждой из транзакций;

- направленных ребер Ti->Tj, где транзакция Ti считывает значение элемента, записанного транзакцией Tj;
- направленных ребер Ti->Tj, где транзакция Tj записывает значение в элемент данных после того, как он был считан транзакцией Ti.

Если граф предшествования содержит петли, то соответствующий ему график не является конфликтно упорядоченным.

Пример графика не являющегося конфликтно упорядоченным

Рассмотрим две транзакции, график выполнения которых представлен в табл. 6.5.

Таблица 6.5

Пример неконфликтно упорядорченных графиков

Транзакция T1	Транзакция T2
начало	
чтение a1	
a1=a1+100	
запись a1	начало
	чтение a1
	a1=a1*1.1
	запись
	в1=в1*1.1
	запись в1
чтение в1	commit
в1=в1-100	
запись в1	
commit	

Здесь в транзакции T1 100 рублей передается со счета в1 на счет a1. Транзакция T2 увеличивает текущее значение каждого из этих счетов на 10%. Граф предшествования для данного графика представлен на рис. 6.1.

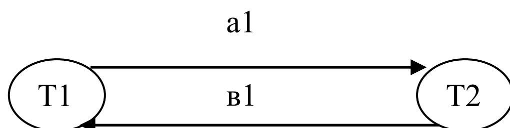


Рис. 6.1. Граф предшествования

Граф содержит петлю, следовательно, график не является конфликтно упорядоченным.

Упорядочивание по просмотру

Существует и несколько других типов упорядочивания, которые выдвигают менее строгое определение эквивалентности графиков, чем то, которое дается в случае конфликтной упорядоченности. Одно из этих определений называют **упорядочиванием по просмотру.** Два графика С1 и С2, состоящих из одних и тех же операций, входящих в состав n транзакций Т1, Т2….Тn, являются **эквивалентными по просмотру**, если выполняются следующие три условия.

- Для каждого элемента данных а1: если транзакция Т1 прочла исходное значение а1 в графике С1, эта же транзакция Т1 должна прочесть исходное значение а1 в графике С2.
- Для каждой операции чтения элемента данных а1 транзакцией Тi в графике С1: если считанное значение элемента а1 было записано транзакцией Тj, то и в графике С2 транзакция Тi должна считывать значение элемента а1, записанное транзакцией Тj.
- Для каждого элемента данных а1: если в графике С1 последняя операция записи значения а была выполнена транзакцией Тj, эта же самая транзакция должна выполнять последнюю запись значения элемента данных а1 и в графике С2.

График является **упорядоченным по просмотру**, если он **эквивалентен по просмотру** некоторому последовательному графику.

Каждый конфликтно упорядоченный график в то же время является упорядоченным по просмотру, однако обратное утверждение неверно.

Пример упорядоченного по просмотру графика, который не является конфликтно упорядоченным

В примере (табл. 6.6) для транзакций Т4 и Т5 не соблюдается правило вынужденной записи (здесь слепая запись).

Таблица 6.6

Пример упорядоченного графика, не являющийся конфликтно упорядоченным

Транзакция Т3	Транзакция Т4	Транзакция Т5
начало		
чтение a1		
	начало	
	запись a1	
	commit	
запись a1		
commit		
		начало
		запись a1
		commit

Восстанавливаемость

Упорядоченными называются такие графики, которые позволяют сохранить согласованность базы данных в предположении, что ни одна из транзакций этого графика не будет отменена. Противоположный подход анализирует **восстанавливаемость** транзакций, входящих в данный график.

В табл. 6.7. представлен пример восстанавливаемости.

Вместо commit в Т1 делается откат. Транзакция Т2 уже считала измененное значение счета a1, записанное транзакций Т1, выполнила обновление и зафиксировала результаты в БД. Строго говоря, следовало бы отменить результаты выполнения

T2, поскольку она использовала значение, которое должно быть отменено. Т. е. этот график не обладает свойством восстанавливаемости и поэтому является некорректным.

Таблица 6.7

Пример восстанавливаемости

Транзакция Т1	Транзакция Т2
начало	
чтение а1	
а1=а1+100	
запись а1	начало
	чтение а1
	а1=а1*1.1
	запись
	в1=в1*1.1
	запись в1
чтение в1	commit
в1=в1-100	
запись в1	
откат	

Восстанавливаемый график – это график, в котором для каждой пары транзакций Ti и Tj выполняется следующее правило: если транзакция Tj считывает элемент данных, предварительно записанный транзакцией Ti, то фиксация результатов транзакции Ti должна выполняться до фиксации результатов транзакции Tj.

6.3. Методы управления параллельностью

Существует два основных метода управления параллельностью, позволяющих организовать одновременное безопасное выполнение транзакций при соблюдении определенных ограничений: метод блокировки и метод временных меток.

По своей сути, и блокировка, и использование временных меток, являются **консервативными** (или **пессимистическими**) подходами, поскольку они откладывают выполнение транзакций, способных в будущем в тот или иной момент времени войти в конфликт с другими транзакциями. Оптимистические методы строятся на предположении, что вероятность конфликта невысока, поэтому они допускают асинхронное выполнение транзакций, а проверка на наличие конфликта откладывается на момент их завершения и фиксации в БД.

Блокировка – это процедура, используемая для управления параллельным доступом к данным. Когда некоторая транзакция получает доступ к БД, механизм блокировки позволяет (с целью исключения получения некорректных результатов) отклонить попытки получения доступа к этим данным со стороны других транзакций.

Существует несколько различных вариантов этого механизма, однако, все они построены на одном и том же фундаментальном принципе: транзакция должна потребовать выполнить блокировку **для чтения** или **для записи** некоторого элемента данных перед тем, как она сможет выполнить в базе данных соответствующую операцию чтения или записи. Установленный **блок** препятствует модификации элемента данных другими транзакциями или даже считыванию его, если этот блок был установлен для записи. Реально блокировка может осуществляться посредством установки некоторого бита в соответствующий элемент данных, означающего, что этот фрагмент базы данных является заблокированным.

Блокировка для чтения – если транзакция установила блокировку элемента данных для чтения, она сможет считать его, но не сможет обновить.

Блокировка для записи – если транзакция остановила блокировку элемента данных для записи, она может как читать, так и обновлять этот элемент.

До тех пор, пока транзакция будет удерживать некоторый элемент заблокированным для записи, никакая другая транзакция не сможет ни считать, ни обновить его.

Помимо этих правил, в некоторых системах транзакциям разрешается устанавливать блокировку для чтения, которая позже может расширяться и преобразовываться в блокировку для записи. Такой подход повышает эффективность работы, позволяя транзакциям вначале проанализировать данные, а затем принять решение, следует ли их изменять. По этой же причине в некоторых системах транзакциям разрешается устанавливать блокировку элемента данных для записи, с последующим сужением ее до уровня блокировки для чтения.

Использование в транзакциях блокировок само по себе не гарантирует упорядоченности получаемых графиков, что может быть продемонстрировано на том же примере.

Пример неверного графика с использованием блокировок

Допустимый график, построенный на основе описанных выше правил, может иметь следующий вид.

S={блокировка записи(T1, a1), чтение (T1, a1), запись(T1, a1), снятие блокировки (T1, a1),

блокировка записи (T2, a1), чтение (T2, a1), запись (T2, a1), снятие блокировки (T2, a1),

блокировка записи (T2, в1), чтение (T2, в1), запись (T2, в1), снятие блокировки (T2, в1), commit (T2),

блокировка записи (T1, в1), чтение (T1, в1), запись (T1, в1), снятие блокировки (T1, в1), commit (T1) }

Результат выполнения графика может быть различным:

если T1 выполняется до T2, то a1 =220, в1 = 400

если T2 выполняется до T1, то a1 =210, в1 = 340.

Отсюда видно, что график не является упорядоченным.

В этом примере (табл. 6.8) проблема состоит в том, что в графике установленная транзакциями блокировка снимается, как только соответствующая операция чтения/записи будет выполнена и доступ к блокируемому элементу данных уже

больше не потребуется. однако, сама транзакция продолжает блокировать другие элементы данных и после того, как блокировка элемента будет отменена. Хотя подобные действия внешне способствуют повышению уровня параллельности обработки в системе, они позволяют транзакциям оказывать влияние на работу друг друга, что может послужить причиной потери полной изолированности и атомарности транзакций.

Таблица 6.8

Пример неверного графика с использованием блокировок

Транзакция Т1	Транзакция Т2
начало	
чтение а1	
а1=а1+100	
запись а1	начало
	чтение а1
	а1=а1*1.1
	запись
	в1=в1*1.1
	запись в1
чтение в1	commit
в1=в1-100	
запись в1	
commit	

Для обеспечения упорядоченности следует использовать дополнительный протокол, определяющий моменты установки и снятия блокировки для каждой из транзакций. Самым известным из таких протоколов является метод двухфазной блокировки.

Двухфазная блокировка – транзакция выполняется по протоколу двухфазной блокировки, если в ней все операции блокирования предшествуют первой операции разблокирования.

В соответствии с основным правилом этого протокола, каждая транзакция может быть разделена на две фазы: *фазу*

нарастания, в которой выполняются все необходимые блокировки и не освобождается ни одного из элементов данных; и *фазу сжатия*, в которой освобождаются все выполненные ранее блокировки и не может быть затребовано ни одной новой. Как правило, транзакция устанавливает некоторые блокировки. выполняет, определенную обработку, после чего может затребовать установку дополнительных необходимых ей блокировок. Однако, она не может освободить ни одного из блоков, пока не достигнет той стадии, на которой ей уже не потребуется установка новых блокировок.

Если СУБД поддерживает операции расширения уровня блокировки, то их выполнение допускается только на фазе нарастания. Подобные действия могут перевести транзакцию в состояние ожидания на то время, пока другие транзакции отменят установленные ими блокировки для чтения данного элемента. Снижение уровня блокировки допускается только в фазе сжатия.

Пример устранения проблемы потерянного обновления (табл. 6.9)

Таблица 6.9

Пример устранения проблемы потерянного обновления

Время	Транзакция Т3	Транзакция Т4	Поле a1
t1		начало	100
t2	начало	блокировка записи a1	100
t3	блокировка записи a1	чтение a1	100
t4	Ожидание	a1=a1+100	100
t5	Ожидание	запись a1	200
t6	Ожидание	rollback/unlock(a1) (повторный прогон)	200
t7	чтение a1		200
t8	a1=a1-100		200
t9	запись a1		190
t10	commit/ unlock(a1)		190

Чтобы избежать потери выполненного обновления, транзакция Т2 должна предварительно установить блокировку счета а1 для записи. В момент запуска Т1 также потребует установить блокировку а1 для записи. Однако, поскольку этот элемент уже будет заблокирован, запрос Т1 удовлетворить не удастся, поэтому данная транзакция будет переведена в состояние ожидания освобождения Т2 необходимого ей элемента. Однако это произойдет только после фиксации результатов транзакции Т2 в БД.

Пример использования протокола двухфазной блокировки для устранения проблемы зависимости от нефиксированных результатов

Пример использования протокола двухфазной блокировки для устранения проблемы зависимости от нефиксированных результатов представлен в табл. 6.10.

Таблица 6.10

Пример использования протокола двухфазной блокировки для устранения проблемы зависимости от нефиксированных результатов

Время	Транзакция Т3	Транзакция Т4	Поле а1
t1		начало	100
t2		блокировка записи а1	100
t3		чтение а1	100
t4	начало	a1=a1+100	100
t5	блокировка записи а1	запись а1	200
t6	Ожидание	rollback/unlock(a1) (повторный прогон, откат)	100
t7	чтение а1		100
t8	a1=a1-100		100
t9	запись а1		90
t10	commit		90

Во избежании возникновения данной ошибки Т4 должна предварительно установить блокировку а1 для записи. После выполнения отката этой транзакции выполненное ею обновление а1 будет отменено и этому элементу данных будет возвращено прежнее значение (100). В момент начала Т3 она тоже потребует блокировку для записи, но ее можно будет выполнить только после снятия блокировки Т4, поэтому Т3 переводится в состояние ожидания.

Пример использования протокола двухфазной блокировки для устранения проблемы несогласованной обработки

Пример использования протокола двухфазной блокировки для устранения проблемы несогласованной обработки представлен в табл. 6.11.

Таблица 6.11

Пример использования протокола двухфазной блокировки для устранения проблемы несогласованной обработки

Время	Транзакция Т5	Транзакция Т6	Поле а1	Поле в1	Поле с1	Поле Sum
t1		начало	100	50	25	0
t2	начало	Sum=0	100	50	25	0
t3	блокировка записи а1		100	50	25	0
t4	чтение а1	блокировка чтения а1	100	50	25	0
t5	а1=а1-100	Ожидание	100	50	25	0
t6	запись а1	Ожидание	90	50	25	0
t7	блокировка записи с1	Ожидание	90	50	25	0
t8	чтение с1	Ожидание	90	50	25	0
t9	с1=с1+10	Ожидание	90	50	25	0
t10	запись с1	Ожидание	90	50	35	0
t11	commit/unlock(a1, c1)	Ожидание	90	50	35	0

Продолжение табл. 6.11

Время	Транзакция T5	Транзакция T6	Поле a1	Поле в1	Поле c1	Поле Sum
t12		чтение a1	90	50	35	0
t13		Sum = Sum +a1	90	50	35	90
t14		блокировка чтения в1	90	50	35	90
t15		чтение в1	90	50	35	90
t16		Sum = Sum +в1	90	50	35	140
t17		блокировка чтения c1	90	50	35	140
t18		чтение c1	90	50	35	140
t19		Sum = Sum +c1	90	50	35	175
t20		commit/unlock(a1, в1,c1)	90	50	35	175

Для устранения этой проблемы в T5 операциям чтения должна предшествовать установка блокировки соответствующих элементов данных для записи, тогда как в T6 операциям чтения должна предшествовать установка блокировки считываемых элементов данных для чтения.

Можно доказать, что если все транзакции в графике следуют двухфазному протоколу блокировки, этот график гарантированно будет конфликтно упорядоченным.

Каскадный откат

Однако, несмотря на то, что двухфазный протокол гарантирует упорядоченность, могут иметь место проблемы с интерпретацией допустимого момента выполнения отмены блокировок.

В табл. 6.12 транзакция T14 выполняет блокировку счета a1 для записи, после чего суммирует его текущее значение с текущим значением в1, для которого устанавливается блокировка для чтения. Полученное новое значение заносится в

БД на счет а1, после чего блокировка а1 отменяется. Затем Т15 устанавливает блокировку а1 для записи, считывает его текущее значение, изменяет и записывает новое значение, после чего отменяет блокировку а1. Наконец, Т16 устанавливает блокировку счета а1 для чтения и считывает его текущее значение. В этот момент происходит отказ в работе Т14, в результате которого происходит ее откат. Однако, поскольку Т15 зависит от результатов Т14, для нее также необходимо выполнить откат. Аналогично - для Т16.

Таблица 6.12

Пример каскадного отката

Время	Транзакция Т14	Транзакция Т15	Транзакция Т16
t1	начало		
t2	блокировка записи а1		
t3	чтение а1		
t4	блокировка чтения в1		
t5	чтение в1		
t6	а1=в1+а1		
t7	запись а1		
t8	снятие блокировки а1	начало	
t9	…	блокировка записи а1	
t10	…	чтение а1	
t11	…	а1=а1+100	
t12	….	запись а1	
t13	…	снятие блокировки а1	
t14	…	…	
t15	откат	…	
t16		…	начало
t17		….	блокировка чтения а1
t18		откат	….
t19			откат

Подобная ситуация, в которой отмена единственной транзакции приводит к целой серии откатов зависящих от нее транзакций, называется **каскадным откатом**.

Каскадные откаты – явление нежелательное, поскольку потенциально они способны привести к потере большого объема выполненной работы. Очевидно, что было бы очень полезно разработать протокол, исключающий возникновение каскадных откатов.

Одним из возможных вариантов является дополнение обычного двухфазного протокола блокировки требованием откладывать выполнение отмены всех установленных блокировок до конца транзакции – как в предыдущих примерах. В подобном случае продемонстрированная в последнем примере проблема никогда не возникнет, поскольку T15 не сможет установить требуемую ей блокировку для записи, пока T14 не завершит свою работу тем или иным образом. Этот вариант протокола называется ***строгим 2PL***. Может быть доказано, что при использовании строгого протокола двухфазной блокировки транзакции могут быть расположены в том порядке, в котором они завершают свою работу.

Другой вариант протокола 2PL, называемый ограниченным 2PL, предусматривает откладывание до конца транзакции освобождение только блокировок для записи. В большинстве СУБД реализуется один из этих двух вариантов протокола 2PL.

Взаимная блокировка

Еще одна проблема, связанная с двухфазной блокировкой, может иметь место при любых схемах освобождения заблокированных элементов. Эта проблема носит название ***взаимной блокировкой*** и является следствием того факта, что любая транзакция может быть переведена в состояние ожидания освобождения требуемого ей элемента данных.

Кроме того, транзакции могут входить в состояние ***бесконечного ожидания*** (самоблокировки), не имея

возможности установить требуемую им новую блокировку, хотя СУБД не будет фиксировать состояние взаимной блокировки. Эта ситуация возможна в случаях, когда алгоритм перевода транзакций в состояние ожидания недоработан и не принимает во внимание время, на протяжении которого транзакция уже находится в состоянии ожидания. Для исключения самоблокировок может использоваться система приоритетов, в которой приоритет транзакции тем выше, чем дольше она находится в состоянии ожидания. Альтернативным вариантом является использование для ожидающих транзакций очередей, построенной по схеме FIFO.

Взаимная блокировка – это тупиковая ситуация, которая может возникнуть, когда две (или более) транзакции находятся во взаимном ожидании освобождения блокировок, удерживаемых каждой из них (табл. 6.13).

Таблица 6.13

Пример взаимной блокировки двух транзакций

Время	Транзакция Т17	ТранзакцияТ18
t1	начало	
t2	блокировка записи а1	начало
t3	чтение а1	блокировка записи в1
t4	а1=а1-10	чтение в1
t5	запись а1	в1=в1+100
t6	блокировка записи в1	запись в1
t7	Ожидание	блокировка записи а1
t8	Ожидание	Ожидание
t9	Ожидание	Ожидание
t10	…	Ожидание
t11	…	…

Взаимная блокировка возникает, поскольку каждая из двух транзакций входит в состояние ожидания освобождения

требуемого ресурса другой транзакцией. В момент времени t2 транзакция T17 запрашивает и осуществляет блокировку для записи элемента данных balx, а момент времени t3 транзакция T18 выполняет блокировку для записи элемента данных baly. Ни одна из транзакций не в состоянии продолжать работу, поскольку каждая ожидает завершения работы другой. Если в системе возникает состояние взаимной блокировки, вовлеченные в него приложения не смогут разрешить данную проблему собственными силами. Ответственность за обнаружение взаимных блокировок и выхода тем или иным образом из этой тупиковой ситуации должна быть возложена на СУБД.

К сожалению, существует только один способ нарушить состояние взаимной блокировки – выполнение одной или более транзакций должно быть отменено. Подобное действие будет сопровождаться откатом всех изменении, внесенных отмененными транзакциями.

Существует два общих метода обработки взаимных блокировок: предупреждение этих ситуаций и выявление взаимных блокировок с последующим их устранением.

При использовании метода предупреждения взаимных блокировок СУБД заранее определяет ситуации, в которых транзакция сможет вызвать появление и устранение взаимных блокировок. При использовании второго метода, СУБД допускает появление подобных ситуаций в системе, однако затем распознает их появление и организует выход из сложившейся тупиковой ситуации. Второй метод проще первого, поэтому он чаще употребим.

Выявление взаимных блокировок

Выявление взаимных блокировок обычно производится с помощью конструкции, называемой **графом ожидания**. Этот граф отражает зависимость транзакций друг от друга. Транзакция Ti зависит от Tj в том случае, если транз. Tj заблокировала элемент данных, необходимый для продолжения работы тр. Ti. Граф ожидания строится следующим образом.

- Для каждой выполняющейся транзакции создается отдельная вершина.
- Отдельное направленное ребро Ti->Tj ___создается для каждого случая, когда транзакция Ti ожидает освобождения элемента данных, заблокированного транзакцией Tj.

Взаимная блокировка имеет место в том и только в том случае, если граф ожидания содержит петлю.

Граф ожидания для последнего примера представлен на рис. 6.2.

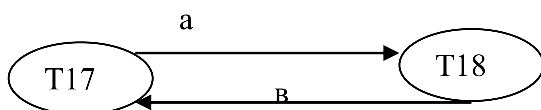


Рис. 6.2. Граф ожидания

На рис. 6.2 показан граф ожиданий для транзакций предыдущего примера. Этот граф содержит петлю (T17-> T18-> T17), поэтому можно сделать заключение, что в системе имеет место ситуация взаимной блокировки.

Поскольку наличие петли в графе ожиданий является необходимым и достаточным условием существования в системе ситуаций взаимной блокировки, алгоритм выявления этих ошибочных ситуаций предполагает генерирование через регулярные интервалы времени графа текущих ожиданий в системе и анализ его на наличие петель. При этом выбор интервала времени между последовательными генерациями графа имеет важное значение. Если он слишком мал – выявление взаимных блокировок создаст в системе заметную перегрузку, если слишком велик – наличие взаимной блокировки может оставаться незамеченным достаточно продолжительное время.

6.4. Предупреждение взаимных блокировок

Один из возможных подходов предупреждения взаимных блокировок состоит в установлении порядка выполнения транзакций на основе использования временных отметок. Были предложены два возможных алгоритма.

Первый, получивший название «ожидание-отмена», требует, чтобы более старые транзакции ожидали завершения более новых. В противном случае транзакция отменяется и перезапускается с той же самой временной отметкой. Однако рано или поздно она станет самой старой из активных транзакций и уже не будет отменена.

Второй алгоритм, «отмена-ожидание», использует диаметрально противоположный подход: только более новые транзакции могут ожидать завершения более старой транзакции. Если более старая транзакция потребует выполнения блокировки элемента данных, уже заблокированного более новой транзакцией, последняя будет отменена.

Использование временных отметок

Использование методов блокировки в сочетании с двухфазным протоколом гарантирует упорядоченность графиков. Порядок следования транзакций в эквивалентном последовательном графике основывается на той очередности, в которой транзакции выполняют блокировку требуемых им элементов данных. Если транзакции требуется элемент, который уже заблокирован другой транзакцией, она переводится в состояние ожидания вплоть до освобождения требуемого ресурса. Другой подход, который тоже гарантирует достижение упорядоченности, для определения очередности выполнения транзакций в эквивалентном последовательном графике использует временные отметки транзакций

Методы использования временных отметок совершенно отличаются от методов с использованием блокировок. Эти методы не предусматривают какого-либо ожидания –

вовлеченные в конфликт транзакции просто отменяются, после чего запускаются заново.

Временная отметка – уникальный идентификатор, создаваемый СУБД с целью обозначения относительного момента времени запуска транзакции.

Метод использования временных отметок – это протокол управления параллельностью, основная цель которой состоит в установлении глобальной очередности выполнения транзакций, при которой боле старые транзакции (транзакции с меньшим значением временной отметки) имеют более высокий приоритет при разрешении возникающих конфликтов.

При использовании протокола временных отметок, если транзакция предпринимает попытку чтения или записи элемента данных, операция чтения или записи выполняется только в том случае, если *последнее обновление требуемого элемента данных* было выполнено более старой транзакцией. В противном случае транзакция, запросившая операцию чтения/записи, отменяется и перезапускается с присвоением ей новой временной отметки. Без получения новой временной отметки транзакция с более старой временной отметкой не сможет завершить свою работу, поскольку более новая транзакция уже успела зафиксировать свои результаты в базе данных.

Помимо временных отметок для транзакций, в системе должны использоваться временные отметки для элементов данных. Каждый элемент данных должен иметь ***временную отметку чтения***, содержащую временную отметку последней из транзакций, выполнявших его чтение, и ***временную отметку записи***, содержащую временную отметку последней транзакции, записавшей содержимое этого элемента данных.

Согласно протоколу, с упорядочиванием по временным отметкам, обработка транзакции Т, имеющей временную отметку (Т), выполняется следующим образом.

Вариант 1. Транзакция Т выдает команду «чтение (х)»

- Транзакция Т запрашивает операцию чтения элемента данных (х), который уже обновлялся более новой (поздней) транзакцией:

Если (Т) <запись (х) – то

более старая транзакция опоздала и уже не может считать предыдущее, устаревшее значение, поэтому весьма вероятно, что любые другие считанные ею значения данных могут быть не согласованы с измененным значением. В этом случае выполнение транзакции Т должно быть прекращено, после чего ее следует перезапустить с новой временной отметкой.

- В противном случае (Т)> запись (х) – операция чтения может быть выполнена.

Вариант 2. Транзакция Т выдает команду «запись (х)»

- Транзакция Т запрашивает операцию записи элемента данных (х), значение которого уже было считано более новой (поздней) транзакцией:

Если (Т) <чтение (х) – то

более новая транзакция уже использовала текущее значение элемента данных и в результате его обновления данной транзакцией может возникнуть ошибка. подобная ситуация возникает в тех случаях, когда транзакция опоздала с выполнением записи и более новая транзакция уже считала жлемент данных или даже перезаписала его.

В этом случае следует выполнить откат транзакции Т и перезапустить ее с использованием новой временной отметки.

- Транзакция Т запрашивает операцию записи элемента данных (х), значение которого уже было перезаписано более новой (поздней) транзакцией:

Если (Т) <запись (х)

Это означает, что транзакция Т пытается поместить в элемент данных (х) устаревшее значение.

В этом случае выполнение транзакции Т должно быть прекращено, после чего ее следует перезапустить с новой временной отметкой.

- В противном случае операция записи может быть выполнена, а временной отметке записи обновляемого элемента данных должно быть присвоено новое значение:

запись (x) = (T)

Эта схема, называемая базовым протоколом упорядочивания по временным отметкам, гарантирует, что график выполнения транзакций будет конфликтно упорядоченным, а результаты его выполнения будут эквиваленты последовательному графику, в котором транзакции выполняются в хронологическом порядке присваивания им временных отметок.

Однако базовый протокол упорядочивания по временным отметкам не обеспечивает восстанавливаемости графиков.

Правило записи Томаса

Для обеспечения повышенного уровня параллельности может использоваться модернизированный базовый протокол упорядочивания по временным отметкам, который позволяет достичь менее жесткой упорядоченности за счет отмены устаревших операций записи. Это расширение известно, как ***правило записи Томаса.***

- Транзакция Т запрашивает операцию записи элемента данных (x), значение которого уже было считано более новой (поздней) транзакцией:

Если (T) <чтение (x) - то

В этом случае транзакция Т откатывается и перезапускается с использованием новой временной отметки.

- Транзакция Т запрашивает операцию записи элемента данных (x), значение которого уже было перезаписано более новой (поздней) транзакцией:

Если (T) <запись (x)

Это означает, что более новая транзакция уже перезаписала значение этого элемента данных и значение, которое более старая транзакция собирается записать в данный элемент, было вычислено на основе устаревшего исходного

значения данного элемента. В подобном случае операция записи вполне безопасно может быть проигнорирована. Этот метод иногда называется **правилом игнорирования устаревшей записи.** Его применение позволяет повысить уровень параллельности обработки в системе.

- В противном случае операция записи может быть выполнена, а временной отметке записи обновляемого элемента данных должно быть присвоено новое значение:

запись (x) = (T)

Применение правила Томаса позволяет генерировать графики, которые невозможно создать при использовании каких-либо других протоколов управления параллельностью.

В табл. 6.14 представлен пример базового протокола упорядочивания по временным отметкам.

Таблица 6.14

Пример базового протокола упорядочивания по временным отметкам

Время	Операция	Транзакция T19	Транзакция T20	Транзакция T21
t1		начало		
t2	чтение a1	чтение a1		
t3	a1=a1+10	a1=a1+10		
t4	запись a1	запись a1	начало	
t5	чтение в1		чтение в1	
t6	в1=в1+20		в1=в1+20	начало
t7	чтение в1			чтение в1
t8	запись в1		запись в1*	
t9	в1=в1+30			в1=в1+30
t10	запись в1			запись в1
t11	c1=100			c1=100
t12	запись c1			запись c1
t13	c1=50	c1=50		commit
t14	запись c1		начало	

Продолжение табл. 6.14

Время	Операция	Транзакция T19	Транзакция T20	Транзакция T21
t15		запись с1**		
t16	чтение в1	commit	чтение в1	
t17	в1=в1+20		в1=в1+20	
t18	запись в1		запись в1	
t19			commit	

T19 имеет временную отметку (T19), T20 – (T20), T21 – (T21)

(T19) <(T20) <(T21)

*- в момент времени t8 операция записи в T20 нарушает первое правило записи, приведенное в описании протокола с использованием временных отметок. Поэтому она отменяется и вновь запускается в момент времени t14.

**- в момент времени t14 операция записи в T19 может быть безопасно проигнорирована с использованием правила игнорирования устаревших операций записи, поскольку T21 уже поместила новое значение в этот элемент данных в момент t12.

Оптимистические технологии

В некоторых типах вычислительных систем конфликты между транзакциями происходят очень редко, поэтому сложные технологии оказываются совершенно излишними.

Оптимистические технологии основываются на предположении, что конфликты возможны нечасто, поэтому эффективнее будет организовать выполнение транзакций, исключив все задержки, связанные с достижением гарантированной упорядоченности. Перед завершением работы транзакции выполняется проверка с целью определения, имел ли место конфликт. Если это так, транзакция откатывается и перезапускается. Подобная технология потенциально позволяет

достичь существенно более высокого уровня параллельности по сравнению с традиционными протоколами, поскольку не требует использования механизма блокировок.

Оптимистический протокол управления параллельностью включает до трех фаз, в зависимости от того, выполняется ли в данной транзакции только чтение или еще и обновление информации.

- *Фаза чтения.* Она охватывает транзакцию от ее начала вплоть до момента непосредственного выполнения фиксации результатов. Транзакция считывает значения всех необходимых ей элементов данных и помещает их в локальные переменные. Любые обновления применяются только к локальной копии данных, но не к информации, сохраняемой в самой БД.
- *Фаза проверки*. Эта фаза следует за фазой чтения. Выполняются поверки, необходимые для получения гарантий отсутствия нарушения упорядоченности в случае переноса в БД изменений, выполненных транзакцией. Для транзакций, включающих только чтение, проверка состоит в подтверждении того, что использованные транзакцией значения по-прежнему остаются текущими значениями соответствующих элементов данных. Если нарушение не отмечено, транзакция завершает свое выполнение. Если найдены измененные значения, транзакция отменяется и перезапускается. Если в транзакции выполняется обновление данных, то проверка включает выполнение контроля, сохранится ли база данных в согласованном состоянии после внесения в нее результатов данной транзакции, а также не будут ли нарушены условия упорядоченности. В случае обнаружения ошибка транзакция отменяется и перезапускается.
- *Фаза записи*. Эта фаза выполняется после успешного завершения фазы проверки, но только в отношении транзакций, включающих операции обновления. В этой фазе все изменения, внесенные в локальные копии данных, переносятся собственно в БД.

В фазе проверки анализируются входящие в транзакцию операции чтения и записи, способные оказать влияние на выполнение других транзакций. Каждой транзакции в начале ее выполнения присваивается некоторая временная отметка. Дополнительные временные отметки присваиваются транзакции в начале фазы проверки и в момент завершения ее выполнения – включая и фазу записи, если таковая имеется. Для того, чтобы все проверки были удовлетворены, должно выполняться одно из следующих условий.

1. Все транзакции M с более старыми временными отметками должны быть уже завершены до начала выполнения транзакции T:

конец (M) = начало (T)

2. Если транзакция T стартовала до завершения какой-либо из транзакций M, то

а) множество элементов данных, записанных стартовавшей ранее транзакцией, не должно включать ни одного из элементов данных, прочитанных данной транзакцией;

б) фаза записи стартовавшей ранее транзакции должна быть завершена до прихода текущей транзакции в фазу проверки:

начало (T)<конец (M)<проверка (T)

Уровень детализации блокируемых элементов данных

Уровень детализации – это размер элементов данных, выбранных в качестве защищаемой единицы для протокола управления параллельностью.

Как правило, в качестве элемента данных выбирается один из перечисленных ниже объектов, размеры которых варьируются от очень крупных до мельчайших.

- Вся БД
- Отдельный файл
- Отдельная страница данных (иногда называемая областью или блоком базы данных – сектор на физическом диске, используемом для хранения таблиц)

- Отдельная запись
- Отдельное поле

Размер, или уровень детализации, элемента данных, который может быть заблокирован отдельной операцией транзакции, оказывает сильнейшее влияние на общую производительность системы и эффективность работы протокола управления параллельностью. Однако, можно достичь нескольких компромиссов, что следует учитывать при выборе размера элемента данных в системе.

Очевидно, что чем крупнее размер элемента данных, тем ниже возможный уровень параллельности в системе. С другой стороны, чем мельче размер элемента данных, тем больший объем информации о выполненных блокировках придется хранить. Оптимальный размер элемента данных зависит от природы выполняемых транзакций если типичная транзакция обрабатывает незначительное количество записей, целесообразно установить уровень детализации блокировки равным отдельной записи. В то же время, если типичной транзакции требуется доступ ко множеству записей одного и того же файла, более выгодным решением будет установить уровень детализации равным отдельному блоку или даже файлу. В этом случае для получения доступа к требуемым данным транзакции потребуется заблокировать лишь один (или несколько) элементов данных.

В идеале СУБД должна поддерживать сменный уровень детализации, позволяющий блокировать отдельные записи, страницы и целые файлы. Некоторые системы автоматически повышают уровень детализации, если определенная транзакция блокирует больше установленного процента записей или страниц некоторого файла.

Иерархия уровней детализации.

Уровень 0	БД
Уровень 1	Файл
Уровень 2	Страница
Уровень 3	Запись
Уровень 4	Поле

Существующие уровни детализации блокируемых элементов можно представить в виде иерархической структуры, в которой каждый узел будет представлять элемент данных определенного размера. Если какой-то из элементов данных блокируется, то автоматически оказываются заблокированными и все его узлы-потомки. Если некоторая транзакция потребует установить блокировку любого из потомков уже заблокированного узла, СУБД, прежде чем принять решение о возможности удовлетворения этого запроса, потребуется обследовать иерархический путь от корня схемы до того узла, который представляет запрошенный элемент, - с целью проверки, является ли заблокированным какой-либо из предков требуемого узла. Если это так, то выполнение запроса будет отклонено.

Кроме того, транзакция может потребовать установить блокировку узла, который имеет заблокированный узел-потомок, т.е. повести анализ всего дерева иерархии.

Для сокращения объемов поисков, необходимых для выявления блокировок, СУБД может использовать иную стратегию выполнения блокировки, называемую **многоуровневой блокировкой**. В ней используется новый тип блокировки – **блокировка намерения**. Когда блокируется любой из узлов схемы, блокировка намерения устанавливается на все узлы – предки данного узла.

Блокировка намерений может быть либо разделяемой – для чтения, - любо эксклюзивной – для записи. Разделяемая блокировка намерения конфликтует только с эксклюзивной блокировкой, тогда как эксклюзивная блокировка намерения конфликтует как с разделяемой, так и с эксклюзивной блокировкой.

Для гарантированного получения упорядоченных графиков в случае применения нескольких уровней блокировки двухфазный протокол блокировки должен функционировать следующим образом:

- Ни один элемент данных не может быть заблокирован, пока не будет освобожден представляющий его узел.
- Ни один из узлов не может быть освобожден, ока его родительский узел не будет освобожден от блокировки намерения.
- Ни один узел не может быть освобожден, пока не будут освобождены все его потомки.

В этом случае блокировка устанавливается, начиная с корня иерархии вниз, с использованием блокировок намерения вплоть до достижения узла, представляющего элемент данных, блокируемый для записи или для чтения. Освобождение блоков выполняется снизу-вверх.

6.5. Восстановление данных

Восстановление данных – это процесс возвращения базы данных в корректное состояние, утраченное в результате сбоя или отказа.

Концепцию восстановления данных можно представить, как службу СУБД, гарантирующую надежное сохранение БД в согласованном состоянии даже при наличии в системе отказов. Как подобная служба может быть организована?

Для хранения данных в общем случае может быть использовано четыре различных типа носителей: оперативная память, магнитный диск, магнитная лента, оптический диск.

Существует множество различных типов отказов, которые могут влиять не только на содержимое ОЗУ, но и вторичной памяти:

- Аварийное прекращение работы, вызванное ошибкой оборудования или программного обеспечения, приведшей к разрушению содержимого ОЗУ;
- Отказ носителей информации;
- Ошибка прикладных программ;

- Стихийные бедствия – пожары, отказы в сети электропитания;
- Небрежное или легкомысленное обращение персонала;
- Диверсии и преднамеренное разрушение или уничтожение данных, оборудования или ПО.

Какова бы ни была причина отказа, может быть два принципиально разных следствия – утрата содержимого ОЗУ и утрата копии БД на дисках.

Менеджер восстановления должен гарантировать, что при восстановлении после сбоя для каждой отдельной транзакции в БД будут постоянно фиксироваться либо все внесенные ею изменения, либо ни одного из них. Ситуация осложняется тем фактом, что запись в БД не представляет собой атомарного действия (выполняемого за один шаг), и поэтому существует вероятность, что, когда выполнение транзакции будет завершено, внесенные ею изменения не будут реально отражены в БД по той причине, что еще не достигли файлов БД.

Буфера СУБД занимают определенную часть оперативной памяти и используются для обмена информацией данными со вторичной памятью. Только после того как соответствующий буфер выгружен во вторичную память, можно считать, что выполненные операции обновления приобрели постоянный характер.

Если отказ системы произойдет между записью данных в буфер и выгрузкой буфера во вторичную память, менеджер восстановления должен уточнить состояние транзакции, выполнявшей запись в момент аварии. Если транзакция уже выдала команду завершения (выгрузки во вторичную память), то менеджер восстановления должен повторно ее выполнить.

Если на момент отказа системы транзакция еще не была завершена, менеджер восстановления должен отменить любые е результаты (откатить). Если требуется откатить только одну транзакцию, то это – *частичный откат*. Кроме того, транзакция может отменяться по внутренним причинам – по

требованию пользователя или в результате возникновения исключительной ситуации в прикладной программе. Если требуется выполнить откат всех активных транзакций, то это – *глобальный откат*.

Функции восстановления

Типичная СУБД должна предоставлять такие функции восстановления как:

- механизм резервного копирования, предназначенного для периодического создания копий БД;
- средства ведения журнала, в котором фиксируются текущее состояние транзакций и вносимые в БД изменения;
- функции создания контрольных точек, обеспечивающих перенос выполняемых в БД изменений во вторичную память с целью сделать их постоянными;
- менеджер восстановления, обеспечивающий восстановление согласованного состояния БД, нарушенного в результате отказа.

Механизм резервного копирования. Любая СУБД должна предоставлять механизм, позволяющий создавать резервную копию БД и ее файла журнала через установленные промежутки времени и без необходимости останавливать систему. Резервное копирование может выполняться для Бд в целом или в инкрементном режиме. В последнем случае в копию помещаются сведения только об изменениях. Как правило резервные копии создаются на автономных носителях – например, на магнитных лентах.

Файл журнала. Для фиксации хода выполнения транзакций в БД СУБД использует специальный файл, который называется журналом. В него может помещаться след. информация:

- записи о транзакциях, включающие:
  - идентификатор транзакции;

- тип записи журнала (начало транзакции, операции вставки, обновления или удаления, отмены или фиксации транзакции);
- идентификатор элемента данных, вовлеченного в операцию обработки БД,
- копию элемента данных до операции;
- копию элемента данных после операции;
- служебную информацию файла журнала, включающую указатели на предыдущую и последующую записи журнала для этой транзакции;

- записи контрольных точек.

Часто журнал используется и для других целей, отличных от целей восстановления. В этом случае он может содержать дополнительную информацию (например, сведения о пользователях, об операциях, о завершении сеанса пользователя).

По причине его важности журнал может создаваться в двух и даже в трех экземплярах, которые автоматически поддерживаются системой.

Один из подходов к автономной обработке файла журнала состоит в разделении оперативного файла на две независимые части, организованные в виде файлов с произвольным доступом. Записи журнала помещаются в первый файл до тех пор, пока он не оказывается заполненным до установленного уровня (например, на 70%). Затем открывается второй файл и все записи журнала для *новых* транзакций помещаются уже в него. Сведения о *старых* транзакциях помещаются в первый файл до тех пор, пока обработка всех старых транзакций не будет завершена. В этот момент первый файл закрывается и переводится в автономное состояние. Подобный подход упрощает восстановление отдельных транзакций, т.к. записи о каждой транзакции всегда содержатся в одном фрагменте файла журнала – либо в оперативном, либо в автономном. Следует отметить, что файл журнала потенциально является узким местом с точки зрения производительности любых систем, поэтому скорость записи

информации в файл журнала может оказаться одним из важнейших факторов, определяющих общую производительность системы с БД.

Создание контрольных точек

Контрольная точка – это момент синхронизации между базой данных и журналом регистрации транзакций. Все буфера системы принудительно записываются во вторичную память системы.

Контрольные точки организуются через установленный интервал времени и включают выполнение следующих действий.

- Запись всех имеющихся в ОЗУ записей журнала во вторичную память;
- Запись всех модифицированных блоков в буфера БД во вторичную память;
- Помещение в файл журнала записи контрольной точки. Эта запись содержит идентификаторы всех транзакций, которые были активны в момент создания этой контрольной точки.

Если произошел отказ, то просматриваются все транзакции, выполнявшиеся после последней контрольной точки. Транзакция, которая выполнялась в момент отказа должна быть отменена, а остальные повторены.

Методы восстановления

Тип процедуры, которая будет использоваться для восстановления БД, зависит от размера повреждений, которые были нанесены этой базе в результате отказа. Рассмотрим два варианта:

- Если базе данных нанесены обширные повреждения (например, разрушилась магнитная головка диска), то потребуется восстановить ее последнюю резервную копию, после чего повторить в ней все выполненные транзакции, сведения о которых присутствуют в журнале регистрации. Безусловно, предполагается, что файл журнала поврежден не

был. Рекомендуется, чтобы во всех случаях, когда это возможно, файл журнала создавался на дисковых носителях, отличных от тех, на которых размещены основные файлы базы данных. Подобное решение снижает риск одновременной потери как файлов базы данных, так и файла ее журнала.

- Если база данных не получила физических повреждений, но лишь утратила согласованность размещенных в ней данных (например, из-за аварийного останова системы в процессе обработки транзакций), то достаточно будет отменить те изменения, которые вызвали переход базы данных в несогласованное состояние. Кроме того, возможно потребуется повторно прогнать некоторые транзакции, чтобы иметь уверенность в том, что внесенные в них изменения действительно зафиксированы во вторичной памяти. В данном случае нет необходимости обращаться к резервной копии базы данных, поскольку вернуть БД в согласованное состояние можно с помощью информации о содержимом полей до и после модификации, сохраняемой в файле журнала. Далее будем рассматривать только 2-й случай.

Метод восстановления с использованием отложенного обновления

При использовании этого протокола обновления не заносятся в БД до тех пор, пока транзакция не выдаст команду фиксации выполненных изменений. Если выполнение транзакции будет прекращено до достижения этой точки, никаких изменений в БД выполнено не будет, поэтому не потребуется и их отмена. Однако в данном случае может потребоваться повторный прогон уже завершившихся транзакций, поскольку их результаты могли еще не достичь вторичной памяти. При применении данного метода файл журнала используется с целью восстановления следующим образом.

- При запуске транзакции в журнал помещается запись *Начало транзакции.*

- При выполнении любой операции записи помещаемая в файл журнала строка содержит все указанные выше данные (за исключением значения элементов данных до обновления). Реально запись изменений в буфере СУБД или саму БД не производится.
- Когда транзакция достигает своей конечной точки, в журнал помещается запись *Транзакция завершена*. Все записи журнала по данной транзакции выводятся на диск, после чего выполняется фиксация внесенных транзакцией изменений. Для внесения действительных изменений в БД используется информация, помещенная в файл журнала.
- В случае отмены выполнения транзакции записи журнала по данной транзакции аннулируются и не выводятся на диск.
- Любые транзакции, для которых в файл журнала присутствуют записи *Начало транзакции* и *Транзакция завершена* должны быть выполнены повторно. Процедура повторного прогона транзакций выполняет все операции записи в БД, используя информацию о состоянии элемента данных *после обновления*, содержащуюся в записях журнала по данной транзакции, причем в том порядке, в каком они были записаны в файл журнала. Если эти операции записи уже были успешно завершены до возникновения отказа, это не окажет никакого влияния на состояния элементов данных, поскольку они не могут быть испорчены, если будут записаны еще раз. Однако, этот метод гарантирует, что будут обновлены любые элементы данных, которые не были корректно обновлены до момента отказа.
- Любая транзакция, для которой в файле журнала присутствуют записи *Начало транзакции* и *Отмена транзакции*, просто игнорируются, поскольку никаких реальных обновлений информации в БД по ней не выполнялось, а значит, не требуется и реального выполнения их отката.

Метод восстановления с использованием немедленного обновления

При использовании этого протокола все изменения вносятся в БД сразу же после из выполнения в транзакции, не дожидаясь ее завершения. Помимо необходимости повторного прогона изменений, выполненных транзакциями, закончившимися до появления сбоя, в данном случае может потребоваться выполнить откат изменений, внесенных транзакциями, которые не были завершены к этому моменту. При применении данного метода файл журнала используется с целью восстановления следующим образом.

- При запуске транзакции в журнал помещается запись *Начало транзакции.*
- При выполнении любой операции записи помещаемая в файл журнала строка содержит все указанные выше данные.
- Как только упомянутая выше запись будет помещена в файл журнала, все выполненные обновления вносятся в буфера БД.
- В собственно файлы базы данных изменения будут внесены при очередной разгрузке буферов БД во вторичную память.
- Когда транзакция завершает свое выполнение, в файл журнала заносится запись *Транзакция завершена*.

Очень важно, чтобы в файл журнала все записи (или хотя бы определенная их часть) помещалась до внесения соответствующих изменений в БД. Это требование известно, как ***протокол предварительной записи журнала***. При использовании этого протокола менеджер восстановления всегда сможет безопасно предположить, что если для определенной транзакции в файле журнала отсутствует запись *Транзакция завершена,* то это значит, что эта транзакция была активна в момент возникновения отказа, и, следовательно, должна быть отменена.

Если выполнение транзакции было прекращено, то для отмены выполненных ею изменений может быть использован

файл журнала, так как в нем сохранены сведения об исходных значениях всех измененных элементов данных.

На случай отказа системы процедурой восстановления предусмотрено использование файла журнала для повторного прогона или отката транзакций.

Метод теневых страниц

Этот метод предусматривает организацию на время выполнения транзакции двух таблиц страниц – текущую и теневую. Когда транзакция начинает работать, обе таблицы идентичны. Теневая таблица страниц никогда не изменяется и может быть использована для восстановления БД в случае отказа системы. В ходе выполнения транзакции текущая таблица страниц используется для записи в БД всех вносимых изменений. После завершения транзакции текущая таблица страниц становится теневой таблицей. Этот метод имеет ряд преимуществ перед предыдущими методами:

- исключается нагрузка, связанная с ведением журнала транзакций;
- процесс восстановления происходит значительно быстрее, поскольку нет необходимости в повторном прогоне или откате операций.

Однако ему свойственны и определенные недостатки: фрагментация данных и необходимость периодического выполнения процедуры сборки мусора для возврщения в систему неиспользуемых блоков памяти.

7. ЭКСПЛУАТАЦИЯ БАЗ ДАННЫХ

Типичная СУБД обычно предоставляет различные утилиты администрирования БД, включая утилиты загрузки данных и контроля за функционированием системы.

БД и СУБД являются корпоративными ресурсами, которыми следует управлять так же, как и любыми другими ресурсами. Управление данными и БД предусматривает управление и контроль за СУБД и помещенными в нее данными. **Администратор данных** (**АД**) отвечает за управление данными, включая планирование базы данных, разработку и сопровождение стандартов, бизнес - правил и деловых процедур, а также за концептуальное и логическое проектирование базы данных. АД консультирует и дает свои рекомендации руководству высшего звена, контролируя соответствие общего направления развития базы данных, установленным корпоративными целями.

Администратор базы данных (**АБД**) отвечает за физическую реализацию базы данных, включая физическое проектирование и воплощение проекта, за обеспечение безопасности и целостности данных, за сопровождение операционной системы, а также за обеспечение максимальной производительности приложений и пользователей. По сравнению с АД, обязанности АБД носят более технический характер, и для него необходимо знание конкретной СУБД и системного окружения. В одних организациях между этими ролями не делятся различий, а в других важность корпоративных ресурсов отражена именно в выделении отдельных групп персонала с указанным кругом обязанностей.

В проектировании больших БД участвуют два разных типа разработчиков: разработчики логической базы данных и разработчики физической базы данных. разработчик логической БД занимается идентификацией данных (т.е. сущностей и их атрибутов), связей между данными и устанавливает ограничения, накладываемые на хранимые данные. Он должен обладать всесторонним и полным пониманием структуры

данных организации и ее бизнес - правил. Бизнес правила описывают основные характеристики данных с точки зрения организации. Например, «Любой сотрудник не может отвечать одновременно более чем за десять сдаваемых в аренду или продаваемых объектов недвижимости».

Для более эффективной работы разработчик логической БД должен как можно раньше вовлечь всех предполагаемых пользователей БД в процесс создания модели данных. Работа **разработчика логической БД** делится на два этапа:

- Концептуальное проектирование БД, которое совершенно не зависит от таких деталей ее воплощения, как конкретная целевая СУБД, приложения, языки программирования или любые другие физические характеристики.
- Логическое проектирование БД, которое проводится с учетом особенностей выбранной модели данных: реляционной, сетевой, иерархической или объектно-ориентированной.

Разработчик физической БД получает готовую логическую модель данных, занимается ее физической реализацией, в том числе:

- преобразование логической модели данных в набор таблиц и ограничений целостности данных;
- выбором конкретных структур хранения и методов доступа к данным, обеспечивающих необходимый уровень производительности при работе с БД;
- проектированием любых требуемых мер защиты данных.

Многие этапы физического проектирования БД в значительной степени зависят от выбранной целевой СУБД.

Разработчик физической БД должен уметь выбрать наиболее подходящую стратегию хранения данных с учетом всех существующих особенностей их использования.

Сразу после создания БД следует приступить к разработке приложений. Эту работу выполняют **прикладные программисты**.

Пользователи являются клиентами БД – она проектируется, создается и поддерживается для того, чтобы обслуживать их информационные потребности. Пользователей можно классифицировать по способу использования ими системы.

- Наивные пользователи – обычно даже не подозревают о наличии СУБД. Они пользуются разработанными приложениями.
- Опытные пользователи – они знакомы со структурой БД и возможностями СУБД. Для выполнения требуемых им операций они могут использовать язык запросов (например, SQL).

Обзор жизненного цикла информационных систем

Информационная система – ресурсы, которые позволяются выполнять сбор, корректировку и распространение информации внутри организации.

Начиная с 70-х годов системы БД стали постепенно заменять файловые системы. Параллельно с этим росло признание того факта, что данные являются важным корпоративным ресурсом, к которому нужно относиться так же уважительно, как и к другим ресурсам организации. Это привело к тому, что во многих организациях появились функциональные подразделения, занимавшиеся администрированием данных и администрированием БД. Они отвечали за обработку и управление корпоративными данными и корпоративными БД.

Типичная компьютеризированная информационная система включает такие компоненты, как:

- БД
- программное обеспечение БД
- прикладное программное обеспечение
- аппаратное обеспечение, в том числе устройства хранения
- персонал, использующий и разрабатывающий эту систему.

Жизненный цикл информационной системы обычно состоит из нескольких этапов: планирование, сбор и анализ требований, проектирование (включая проектирование БД), создание прототипа, реализация, тестирование, преобразование данных и сопровождение.

Нас будет интересовать жизненный цикл приложения БД. Этапы жизненного цикла приложения БД перечислены ниже. Эти этапы не являются строго последовательными, а включают некоторое количество повторов предыдущих шагов в виде *циклов обратной связи*.

- Планирование разработки БД – планирование самого эффективного способа реализации этапов жизненного цикла системы.
- Определение требований к системе – определение диапазона действия и границ приложения БД, состава его пользователей и областей применения.
- Сбор и анализ требований пользователей – на этом этапе выполняется сбор и анализ требований пользователей из всех возможных областей применения.
- Проектирование БД – полный цикл разработки включает концептуальное, логическое и физическое проектирование БД.
- Выбор целевой СУБД
- Разработка приложений – определение пользовательского интерфейса и прикладных программ, которые используют и обрабатывают базу данных.
- Создание прототипов (необязательно) – создается рабочая модель приложения БД, которая позволяет разработчикам или пользователям представить и оценить окончательный вид и способы функционирования системы.
- Реализация – создание внешнего, концептуального и внутреннего определений БД и прикладных программ.
- Конвертирование и загрузка данных – преобразование и загрузка данных (и прикладных программ) из старой системы в новую.

- Тестирование.
- Эксплуатация и сопровождение – на этом этапе приложение БД считается полностью разработанным и реализованным.

Остановимся подробнее на этих этапах.

Планирование разработки БД – это подготовительные действия, позволяющие с максимально возможной эффективностью реализовать этапы жизненного цикла приложений баз данных.

Этот этап состоит из определения трех основных компонентов: требуемого объема работы, необходимых ресурсов и общей стоимости проекта. Планирование разработки БД должно быть связано с общей стратегией построения информационной системы организации. Суть этой стратегии заключается в решении таких основных задач, как:

- определение бизнес-планов и целей организации с последующим выделением ее потребностей в информационных технологиях;
- оценка показателей уже существующих информационных систем с целью выявления их сильных и слабых сторон;
- оценка возможностей использования информационных технологий для достижения конкурентно способного преимущества.

Для поддержки планирования разработки БД может быть создана корпоративная модель данных, отображающая наиболее важные данные и связи между ними, а также их отношение к различным функциональным сферам организации. Благодаря этому можно будет понять, в каких случаях потребуется совместный доступ к данным. Обычно корпоративная модель данных имеет вид упрощенной ER-диаграммы.

Планирование разработки БД также должно включать разработку стандартов, которые определяют, как будет осуществляться сбор данных, каким будет их формат, какая потребуется документация и как будет выполняться

проектирование и реализация приложений. Например, специальные правила могут определять, как присваиваются имена элементам данных, описываемых в словаре данных, что, в свою очередь, позволит предотвратить их избыточность и противоречивость. Кроме того, необходимо строго документировать любые требования к данным.

Этот этап может также предусматривать выбор наиболее подходящих инструментов автоматизированного проектирования и создания программ, которые принято называть CASE-инструментами. В самом широком смысле слова термин «CASE-инструмент» применим к любым средствам автоматизированного проектирования и создания программ. CASE-инструменты могут включать следующие компоненты:

- словарь данных, предназначенный для хранения информации о данных, используемых в создаваемом приложении;
- инструменты проектирования, обеспечивающие проведение анализа данных;
- инструменты разработки корпоративной модели данных, а также концептуальных и логических моделей данных;
- инструменты, позволяющие создавать прототипы приложений.

CASE-инструменты можно разбить на три категории: CASE-инструменты высокого уровня, CASE-инструменты низкого уровня и интегрированные CASE-инструменты. *CASE-инструменты высокого уровня* применяются на начальных стадиях жизненного цикла разработки БД, а *CASE-инструменты низкого уровня* – на более поздних, начиная со сдатии реализации, в ходе тестирования и на протяжении всего процесса сопровождения функционирующей системы. *Интегрированные CASE-инструменты* применяются на всех стадиях жизненного цикла системы, а поэтому они должны поддерживать все функции CASE-инструментов как высокого, так и низкого уровней.

Использование CASE-инструментов способствует повышению производительности труда разработчиков за счет приведенных ниже преимуществ:

- *Стандарты*. CASE-инструменты способствуют расширению использования стандартов как в ходе разработки программного проекта, так и в работе самой организации. Они позволяют создавать стандартные тестовые компоненты.
- *Интеграция*. CASE-инструменты позволяют сохранять всю генерируемую информацию в специальном хранилище или в словаре данных. Собранные данные могут быть скомпонованы между собой, что будет гарантировать успешность интеграции всех частей системы.
- *Поддержка стандартных методов*. CASE-инструменты существенно упрощают построение и использование диаграмм, что позволяет создавать корректную и актуальную документацию.
- *Непротиворечивость*. CASE-инструменты способны обеспечить автоматическую проверку непротиворечивости данных.
- *Автоматизация.* Некоторые CASE-инструменты позволяют автоматически преобразовывать фрагменты спецификаций проекта в выполняемый код.

Определение требований к системе – это определение диапазона действия и границ приложения базы данных, состава его пользователей и областей применения.

Прежде, чем приступить к проектированию приложения БД, важно установить границы исследуемой области и способы взаимодействия приложения с другими частями информационной системы организации. Эти границы должны охватывать не только текущих пользователей и области применения, но и будущих пользователей и возможные области применения.

Сбор и анализ требований пользователей – это сбор и анализ информации о той части организации, работа которой будет поддерживаться с помощью создаваемого приложения

БД, а также использование этой информации для определения требований к создаваемой системе.

Необходимая информация может быть собрана следующими способами:

- посредством опроса отдельных сотрудников предприятия, особенно специалистов в наиболее важных областях ее деятельности;
- с помощью наблюдений за деятельностью предприятия;
- посредством изучения документов,
- с помощью анкетирования,
- за счет использования опыта подобных систем.

Собранные сведения о каждой важной области должны включать:

- используемую и создаваемую документацию,
- подробные сведения о выполняемых транзакциях,
- список требований с указанием их приоритетов.

Определение набора требуемых функциональных возможностей приложения БД является критически важным действием, поскольку системы с неадекватной или неполной функциональностью будут лишь раздражать пользователей. Однако, чрезмерное увеличение набора функциональных возможностей также может стать источником проблем, поскольку это вызовет чрезмерное усложнение системы.

Проектирование БД

Проектирование БД – это процесс создания проекта БД, предназначенной для поддержки функционирования предприятия и способствующей достижения его целей.

Основными целями проектирования БД являются:

- представление данных и связей между ними, необходимых для всех основных областей применения данного приложения и любых существующих групп его пользователей;
- создание модели данных, способной поддерживать выполнение любых требуемых транзакций обработки данных;

- разработка предварительного варианта проекта, структура которого позволяет удовлетворить все основные требования, предъявляемые к производительности системы.

Существую два подхода к проектированию систем БД: «нисходящий» и «восходящий». Восходящий подход предполагает начало работы с определения атрибутов, анализа связей между ними, определение сущностей и связей между сущностями. Этот подход лучше всего подходит для проектирования простых БД с относительно небольшим количеством атрибутов. Процесс нормализации – вариант всходящего подхода.

Для проектирования сложных БД более подходит нисходящий подход. Он начинается с разработки моделей данных, которые содержат несколько высокоуровневых сущностей и связей, затем работа продолжается в виде серии нисходящих уточнений низкоуровневых сущностей, связей и относящихся к ним атрибутов. Нисходящий подход демонстрируется в концепции «сущность-связь».

Помимо этих подходов, для проектирования БД могут применяться другие подходы, например, подход типа «изнутри наружу» или «смешанная стратегия проектирования».

Выбор целевой СУБД

Это выбор СУБД подходящего типа, предназначенной для поддержки создаваемого приложения БД. Цель данного этапа заключается в выборе системы, удовлетворяющей как текущим, так и будущим требованиям организации, при оптимальном уровне затрат, включающем расходы на приобретение СУБД и дополнительного аппаратного и программного обеспечения, а также расходы, связанные с переходом к новой системе и необходимостью обучения персонала.

Основные этапы процедуры выбора СУБД:

- Определение области компетенции проводимого изучения. На этом этапе определяются те критерии, которые будут использоваться для оценки возможностей СУБД, предварительный список анализируемых продуктов, а также все

прочие необходимые условия проведения изучения, включая сведения об установленным для него временных рамках.

- Сокращение списка претендентов до двух-трех продуктов. Решение о включении некоторой СУБД в этот список может зависеть от установленного бюджета, уровня поддержки продукта со стороны производителя, совместимости с другими программными продуктами, а также требованиями к используемому оборудованию.
- Оценка продуктов. Ниже представлены возможные параметры оценки СУБД.

Определение данных: расширенная поддержка первичных ключей;

- определение внешних ключей;
- предусмотренные типы данных;
- расширяемость типов данных;
- определение доменов;
- простота реструктуризации;
- средства поддержки целостности данных;
- реализация механизма представлений;
- поддержка словаря данных;
- независимость от данных;
- тип базовой модели организации данных;
- поддержка эволюции схемы.

Физические параметры: предусмотренные файловые структуры;

- поддержка определения файловых структур;
- простота организации;
- средства индексирования;
- поля/записи с переменной длиной;
- сжатие данных;
- возможности шифрования;
- требования к памяти;
- требования к устройствам хранения данных;

Доступность: язык запросов – совместимость со стандартами SQL-92/SQL3

- интерфейс для других систем;
- интерфейс для языков третьего поколения;
- многопользовательский доступ;
- защита БД;
- управление доступом к данным;
- поддержка механизма авторизации;

Обработка транзакций:

- процедуры резервного копирования и восстановления;
- поддержка контрольных точек;
- средства ведения системного журнала;
- поддерживаемый уровень детализации параллельности;
- возможные стратегии разрешения тупиковых ситуаций;
- поддержка усовершенствованных моделей управления транзакциями;
- параллельная обработка запросов.

Утилиты:

- измерение производительности;
- настройка производительности БД;
- инструменты загрузки/выгрузки данных;
- контроль активности пользователей;
- поддержка процедур администрирования БД.

Средства разработки:

- Инструменты, использующие языки четвертого и пятого поколений;
- CASE-инструменты;
- инструмент для работы с оконным интерфейсом;
- поддержка хранимых процедур, триггеров и правил.

Другие параметры:

- способность к модернизации;
- стабильность производителя СУБД;
- база пользователей;
- обучение и поддержка пользователей;
- взаимодействие с другими СУБД и прочими системами;
- поддержка работы в Internet;
- утилиты репликации;
- возможности распределенной работы;
- качество и полнота документации;
- требуемая операционная система;
- стоимость;
- оперативная справочная система;
- используемые стандарты;
- управление версиями;
- расширенная оптимизация запросов;
- масштабируемость;
- переносимость;
- требуемое аппаратное обеспечение;
- поддержка работы в сети;
- объектно-ориентированные свойства;
- поддержка двух- или трехуровневой архитектуры «клиент/сервер»;
- производительность;
- пропускная способность при обработке транзакций;
- максимальное количество одновременно работающих пользователей.

Если просто отмечать насколько хороши или плохи эти параметры в каждом случае, то сравнение разных СУБД может оказаться трудной задачей. Более полезный подход основан на введении весовых коэффициентов, определяющих

относительную важность для организации отдельных параметров.

Разработка приложений

Разработка приложений – это проектирование интерфейса пользователя и прикладных программ, предназначенных для работы с БД.

При этом проектирование интерфейса пользователя является одним из важнейших компонентов проектирования приложения в целом. Можно дать общие рекомендации по созданию макетов любых форм и отчетов. Такой макет должен включать:

- содержательное название;
- ясные и понятные инструкции;
- логически обоснованные группировки и последовательности полей;
- визуально привлекательный вид окна формы или поля отчета;
- легко узнаваемые названия полей;
- согласованную терминологию и сокращения;
- согласованное использование цветов;
- визуальное выделение пространства и границ полей ввода данных;
- удобные средства перемещения курсора;
- средства исправления отдельных ошибочных символов и целых полей;
- средства вывода сообщений об ошибках при вводе недопустимых значений;
- особое выделение необязательных для ввода полей;
- средства вывода пояснительных сообщений с описанием полей;
- средства вывода сообщений об окончании заполнения формы.

Создание прототипов. Это создание рабочей модели приложения баз данных.

Прототип создается для того, чтобы дать пользователям возможность опробовать его в работе и определить, какие из

функциональных средств системы отвечают своему назначению, а какие – нет. Т.е. это инструмент, позволяющий в значительной степени прояснить требования пользователя как для самих пользователей, так и для разработчиков.

Реализация. Физическая реализация БД и разработанных приложений.

Прикладные программы реализуются с помощью языков третьего или четвертого поколения. На этом же этапе реализуются также используемые приложением средства защиты базы данных и поддержки ее целостности.

Конвертирование и загрузка данных

Это перенос любых существующих данных в новую базу данных и модификация любых существующих приложений с целью организации совместной работы с новой базой данных.

Этот этап выполняется только в том случае, если новая БД заменяет старую.

Тестирование

Это процесс выполнения прикладных программ с целью поиска ошибок.

Прежде чем использовать новую систему на практике, ее следует тщательно протестировать. Этого можно добиться путем разработки продуманной стратегии тестирования с использованием реальных данных, которая должна быть построена таким образом, чтобы весь процесс тестирования выполнялся строго последовательно и методически правильно. Если тестирование проведено успешно, оно обязательно вскроет имеющиеся ошибки в прикладных программах и, возможно, в структурах БД.

Как и при проектировании БД, пользователи новой системы должны быть вовлечены в процесс ее тестирования.

Может использоваться несколько различных стратегий тестирования:

нисходящее тестирование;
восходящее тестирование;
тестирование потоков;
интенсивное тестирование.

Нисходящее тестирование начинается на уровне подсистем с модулями, которые представлены заглушками. Каждый модуль низкого уровня представлен заглушкой. По мере тестирования заглушки убираются и в конечном итоге все программные компоненты заменяются реальным кодом и снова тестируются.

Преимуществом этого подхода является то, что ошибки проектирования могут быть обнаружены еще на ранней стадии тестирования. Недостатком – необходимость создания многочисленных заглушек модулей. Кроме того, для то, чтобы получить выходные данные нужно организовывать их искусственным путем.

Восходящее тестирование выполняется в противоположном направлении. Оно начинается с тестирования модулей на самых низких уровнях иерархии системы. Преимущества и недостатки имеют обратный смысл.

Тестирование потоков – это стратегия тестирования систем, работающих в реальном масштабе времени, которые обычно состоят из большого количества взаимодействующих процессов. Эти системы довольно трудно тестируются, поскольку взаимодействие процессов системы зависит от времени. Стратегия тестирования направлена на слежение за отдельными процессами.

Некоторые системы создаются с целью работы в конкретных режимах максимальной и минимальной нагрузки. Стратегия *интенсивного тестирования* предназначена для проверки возможностей системы справляться с запланированной нагрузкой.

Эксплуатация и сопровождение

Это наблюдение за системой и поддержка ее нормального функционирования по окончании развертывания.

Этот этап включает в себя выполнение следующих действий:

- контроль производительности системы. Если производительность падает ниже приемлемого уровня, то

может потребоваться дополнительная настройка или реорганизация БД;

- сопровождение и модернизация приложений БД.

Администрирование данных и БД

АД и АБД отвечают за управление действиями, связанными с корпоративными данными и корпоративной БД соответственно. Рассмотрим цели и задачи, входящие в круг обязанностей АД и АБД в некоторой организации. В таблице ниже представлены этапы жизненного цикла приложений БД и указан вклад АД и АБД на каждом из них.

Администрирование данных и баз данных

Этап	Основная роль	Вспомогательная роль
Планирование разработки БД	АД	АБД
Определение требований к системе	АД	АБД
Сбор и анализ требований пользователей	АД	АБД
Концептуальное проектирование БД	АД	АБД
Выбор целевой СУБД	АБД	АД
Логическое проектирование БД	АД	АБД
Разработка приложений	АБД	АД
Физическое проетирование БД	АБД	АД
Создание прототипов	АБД	АД
Реализация	АБД	АД
Конвертирование и загрузка данных	АБД	АД
Тестирование	АБД	АД
Эксплуатация и сопровождение	АБД	АД

Администрирование данных – это управление информационными ресурсами, включая планирование БД, разработку и внедрение стандартов, определение ограничений и процедур, а также концептуальное и логическое проектирование БД.

АД отвечает за корпоративные информационные ресурсы и некомпьютиризированные данные. В разных организациях количество сотрудников, выполняющих функции АД, может отличаться и обычно определяется размерами организации. Должность АД обычно принадлежит отделу информационных систем. Основная обязанность АД состоит в обмене консультациями и советами со старшими менеджерами, а также в слежении за тем, чтобы применение технологий БД продолжало соответствовать корпоративным целям. Часто АД совмещает свои функции с АБД.

В настоящее время при обдумывании стратегии планирования информационной системы все больший акцент делается на важность АД. Организации все в большей и большей степени склонны уделять внимание значению данных как средству достижения более высокой конкурентноспособности. В результате возникает обязательное требование слияния стратегии построения информационных систем с бизнес - стратегиями организации.

Основные задачи администрирования данных:

- Выбор подходящих инструментов разработки
- Помощь в разработке корпоративных стратегий построения информационной системы, развития информационных технологий и бизнес -стратегий
- Предварительная оценка осуществимости и планирование процесса создания БД
- Разработка корпоративной модели данных
- Определение требований организации к используемым данным
- Определение стандартов сбора данных и выбора формата их представления
- Оценка объемов данных и вероятность их роста
- Определение способов и интенсивности использования данных

- Определение правил доступа к данным и мер безопасности, соответствующих правовым нормам и внутренним требованиям организации
- Концептуальное и логическое проектирование БД
- Взаимодействие с АБД и разработчиками приложений с целью обеспечения соответствия создаваемых приложений всем существующим требованиям
- Обучение пользователей – изучение существующих стандартов обработки данных и юридической ответственности за их некорректное применение
- Постоянная модернизация используемых информационных систем и технологий по мере развития бизнес -правил
- Обеспечение полноты всей требуемой документации
- Поддержка словаря данных организации
- Взаимодействие с конечными пользователями для определения новых требований и разрешение проблем, связанных с доступом к данным и недостаточной производительностью их обработки.

Администрирование базы данных – это управление физической реализацией приложений баз данных: физическое проектирование базы данных и ее реализация, организация поддержки целостности и защиты данных, наблюдение за текущим уровнем производительности системы, а также реорганизация базы данных по мере необходимости.

Деятельность АБД является в большей мере технической, чем деятельность АД, и предусматривает знание особенностей конкретных СУБД и операционных систем.

Задачи администрирования БД:

- Оценка и выбор целевой СУБД
- Физическое проектирование БД
- Реализация физического проекта БД в среде целевой СУБД
- Определение требований защиты и поддержки целостности данных

- Взаимодействие с разработчиками приложений БД
- Разработка стратегии тестирования
- Обучение пользователей
- Ответственность за сдачу в эксплуатацию готового приложения БД
- Контроль текущей производительности системы и соответствующая настройка БД
- Регулярное резервное копирование
- Разработка требуемых механизмов и процедур восстановления
- Обеспечение полноты используемой документации, включая материалы, разработанные внутри организации
- Поддержка актуальности используемого программного продукта и аппаратного обеспечения, включая заказ и установку пакетов обновления в случае необходимости.

Отличие АД от АБД заключается в самой природе, выполняемой ими работы. Работа АД является в большей степени управленческой, а работа АБД – технической.

8. ЯЗЫК МАНИПУЛИРОВАНИЯ ДАННЫМИ SQL

8.1. Определение данных

В этой главе довольно подробно рассматриваются предложения определения данных SQL. Удобно разделить эти предложения на два больших класса, которые весьма грубо можно охарактеризовать как логический и физический. Предложения «логического» класса должны иметь дело с объектами, которые на самом деле представляют интерес для пользователей, например, с базовыми таблицами и представлениями, а «физического» — с объектами, которые представляют интерес, главным образом, для системы, например, дисковые тома. Нет нужды говорить о том, что в действительности дело обстоит не настолько ясно, насколько предусматривается в этой простой классификации — некоторые «логические» предложения включают параметры, которые по своей природе в действительности являются «физическими», и наоборот. Кроме того, некоторые предложения не попадают строго в какую-либо одну из этих категорий. Тем не менее, такая классификация удобна как вспомогательное средство для понимания, и мы будем ее сейчас использовать. В данной главе рассматривается только «логическое» определение данных.

Ниже перечислены основные предложения логического определения данных:

CREATE TABLE (создать таблицу)	**CREATE VIEW (создать представление)**	**CREATE INDEX (создать индекс)**
ALTER TABLE (изменить таблицу)		
DROP TABLE (уничтожить таблицу)	**DROP VIEW (уничтожить представление)**	**DROP INDEX (уничтожить индекс)**

(*Примечание.* Имеется также предложение ALTER INDEX (изменить индекс), но оно полностью попадает в «физическую» категорию. Предложения ALTER VIEW (изменить представление) нет.)

Базовые таблицы

Базовая таблица — (важный) специальный случай более общего понятия «таблица». Поэтому давайте начнем с того, что несколько уточним это более общее понятие.

В реляционной системе *таблица* состоит из строки *заголовков столбцов* и нуля или более строк *значений данных* (число строк данных может быть каждый раз разным). Для заданной таблицы:

а) Строка заголовков столбцов специфицирует один или более столбцов, задавая наряду с прочим тип данных для каждого из них;

б) Каждая строка данных содержит в точности одно значение данных для каждого из столбцов, специфицированных в строке заголовков столбцов. Кроме того, все значения в заданном столбце имеют один и тот же тип, а именно: тип данных, специфицированный для этого столбца в строке заголовков столбцов.

Базовая таблица — это *автономная именованная таблица.* Под «автономностью» понимается то, что эта таблица существует *сама по себе* в отличие от представления, которое существует не *само по себе,* а является производным от одной или нескольких базовых таблиц. Представление служит просто альтернативным способом рассмотрения этих базовых таблиц. Под «именованной» понимается, что этой таблице с помощью соответствующего предложения CREATE явно задается некоторое имя в отличие от таблицы, которая строится как результат запроса и не имеет сама по себе какого-либо явного имени. Такая таблица существует в течение непродолжительного времени.

Предложение CREATE TABLE

Теперь мы в состоянии подробно обсудить предложение CREATE TABLE. Это предложение имеет следующий общий формат:

CREATE TABLE имя—базовой—таблицы
(определение — столбца [, определение—столбца] ...)
[другие—параметры];

Здесь «определение — столбца» в свою очередь имеет формат:

имя—столбца тип—данных [NOT NULL]

Факультативные «другие—параметры» имеют дело главным образом с проблемами физического хранения. *Примечание.* Прямые скобки в синтаксических определениях используются для того, чтобы указать, что конструкции, заключенные в эти скобки, являются необязательными (т. е. могут быть опущены). Многоточие указывает, что непосредственно предшествующая ему синтаксическая единица факультативно может повторяться один или более раз.

Ниже приведен пример предложения CREATE TABLE для таблицы S, теперь уже в полном виде:

```
CREATE TABLE S
    (НОМЕР_ПОСТАВЩИКА   CHAR (5) NOT NULL,
    ФАМИЛИЯ                  CHAR (20),
    СОСТОЯНИЕ                SMALLINT,
    ГОРОД                    CHAR (15));
```

Результат этого предложения состоит в том, что создается новая пустая базовая таблица, названная xyz.S, где xyz — имя, под которым известен системе пользователь, издающий предложение CREATE TABLE. В системный каталог при этом помещается статья, описывающая эту таблицу.

Пользователь xyz может обращаться к таблице по ее полному имени xyz.S или по сокращенному имени S. Другие пользователи должны обращаться к ней только по ее полному имени. Данная таблица состоит из четырех столбцов с именами xyz.S.НОМЕР_ПОСТАВЩИКА, xyz.S.ФАМИЛИЯ, xyz.S.СОСТОЯНИЕ и xyz.S.ГОРОД, имеющих указанные в определении типы данных. (Типы данных будут рассматриваться ниже). Пользователь xyz может обращаться к этим столбцам по их полным или по сокращенным именам: S.НОМЕР_ПОСТАВЩИКА, S.ФАМИЛИЯ, S.СОСТОЯНИЕ и S.ГОРОД. Другие пользователи должны применять только полные имена столбцов. Заметим, однако, что независимо от того, включается ли в имя часть «xyz», часть «S» может быть опущена, если это не приводит к двусмысленности, но ее включение никогда не является ошибкой.

Вообще относительно имен справедливы следующие правила. Имена пользователей, например xyz, являются уникальными во всей системе. Имена таблиц (неуточненные) уникальны для пользователя. Имена столбцов (неуточненные) уникальны для таблицы[1]. Под «таблицей» здесь понимаются как базовые таблицы, так и представления. Таким образом, представление не может иметь такое же имя, как и базовая таблица.

После того как таблица создана, в нее могут быть введены данные с помощью предложения INSERT (вставить) языка SQL.

Типы данных

В различных системах и реализациях систем, поддерживаемые типы данных и форматы могут отличаться.

---

[1] Кроме того, в качестве имен не могут использоваться зарезервированные и ключевые слова языка SQL. Первая литера любого имени должна быть буквой, а остальные литеры—буквами, цифрами или знаком подчеркивания. Использование # и $ специально оговаривается.

Например, система DB2 поддерживает следующие типы:

INTEGER — двоичное целое число, занимающее полное машинное слово, 31 бит со знаком

SMALLINT — двоичное целое число, занимающее полуслово, 15 бит со знаком

DECIMAL (p, q) — упакованное десятичное число, включающее p цифр и знак (0<p<16); предполагается q цифр справа от десятичной точки (q<p; если q=0, она может быть опущена)

FLOAT — число n с плавающей точкой, занимающее двойное слово и представленное шестнадцатеричной мантиссой f с точностью до 15 знаков (—1<f<+l) и двоичным целочисленным порядком e (—65<e<+64) таким образом, что n=f*(16**e); примерный диапазон значений n — от 5.4E—79 до 7.2E+75; см. также ниже пояснения для констант типа FLOAT

CHAR (n) — литерная строка фиксированной длины из n литер (0<n<255)

VARCHAR (n) — литерная строка переменной длины, не превышающей n литер (0<n; максимальное значение n зависит от ряда факторов, но в общем случае должно быть меньше, чем «размер страницы» — 4K либо 32K — пространства, содержащего данную таблицу).

Константы

Набор констант также различен для разных СУБД, например, в DB2 поддерживаются:

целочисленная — записывается как десятичное целое число со знаком или без знака, без десятичной точки; примеры: 4 -95 4 -364 0

десятичная — записывается как десятичное число со

	знаком или без знака, с десятичной точкой; примеры: 4.0 —95.7 +364.05 0.007
с плавающей точкой	— записывается как десятичная константа, за которой следует буква E с последующей целочисленной константой; примеры: 4E3 —95.7E46 +364E—5 0.7E1 (примечание: выражение xEy представляет значение x*(10**y))
строковая	— записывается либо как строка литер, заключенная в одиночные кавычки, либо как строка пар шестнадцатеричных цифр, представляющих рассматриваемые литеры в коде ASCII или EBCDIC, в зависимости от используемого компьютера

Неопределенное значение

Вернемся теперь к основной теме данной главы. Практически все системы поддерживают концепцию *неопределенного значения данных (NULL)*. Фактически любой столбец может содержать неопределенное значение, если в определении этого столбца в предложении CREATE TABLE явным образом не специфицировано NOT NULL (неопределенное значение не допускается). Неопределенное значение — это специальное значение, которое используется для того, чтобы представлять «неизвестное значение» или «неприменимое значение». Это не то же самое, что пробел или ноль. Например, запись поставки может содержать неопределенное значение поля КОЛИЧЕСТВО (известно, что поставка имела место, но неизвестен объем поставки). Запись поставщика может также содержать неопределенное значение в столбце СОСТОЯНИЕ (может быть, например, что СОСТОЯНИЕ не соотносится по некоторой причине поставщикам в Сан-Хосе).

Вернемся к предложению CREATE TABLE для базовой таблицы S. Мы специфицировали NOT NULL только для

столбца НОМЕР-ПОСТАВЩИКА. Результатом этой спецификации является гарантия того, что запись каждого поставщика в базовой таблице S всегда будет содержать какой-либо реальный (отличный от неопределенного значения) номер поставщика. Напротив, любое из значений ФАМИЛИЯ, СОСТОЯНИЕ и ГОРОД или все они могут быть неопределенными в той же самой записи.

Предложение ALTER TABLE

Точно так же, как в любое время можно с помощью предложения CREATE TABLE создать новую базовую таблицу, можно также в любое время изменить существующую базовую таблицу, добавляя к ней справа новый столбец. Для этого используется предложение ALTER TABLE:

ALTER TABLE имя—базовой—таблицы
ADD имя—столбца тип—данных;

Например,

ALTER TABLE S ADD СКИДКА SMALLINT;

Это предложение добавляет к таблице S столбец СКИДКА. Все существующие записи таблицы S расширяются с четырех значений полей данных до пяти, и во всех случаях новое пятое поле принимает неопределенное значение. Спецификация NOT NULL в предложении ALTER TABLE не допускается. Заметим, между прочим, что только что описанное расширение существующих записей, не означает, что в это время физически обновляются записи в базе данных. Изменяется лишь хранимое в каталоге описание таблицы. Отдельные записи физически не изменяются до тех пор, пока они в следующий раз не станут целевыми для предложения UPDATE языка SQL.

Предложение DROP TABLE

Существующую базовую таблицу можно в любое время уничтожить с помощью предложения:

DROP TABLE имя—базовой—таблицы;

Специфицированная базовая таблица удаляется из системы (точнее, из каталога удаляется описание этой таблицы). Автоматически удаляются также все индексы и представления, определенные над этой базовой таблицей.

Индексы

Подобно базовым таблицам индексы создаются и уничтожаются путем использования предложений определения данных языка SQL. Однако CREATE INDEX и DROP INDEX (а также ALTER INDEX и некоторые предложения/управления данными) вообще являются единственными предложениями в языке SQL, в которых имеются ссылки на индексы. Другие предложения, в частности предложения манипулирования данными, такие, как SELECT, намеренно не включают каких-либо таких ссылок. Решение о том, использовать или не использовать какой-либо индекс при обработке некоторого конкретного запроса в языке SQL, принимается не пользователем, а системой.

Предложение CREATE INDEX имеет следующий общий формат:

CREATE [UNIQUE] INDEX имя—индекса
ON имя—базовой—таблицы (имя—столбца [упорядочение]
[имя—столбца [упорядочение]] ...)
[другие—параметры];

Факультативные «другие—параметры» имеют отношение к проблемам физического хранения, как и в случае предложения CREATE TABLE. Каждая спецификация

«упорядочение» представляет собой либо ASC (возрастание), либо DESC (убывание). Если не специфицировано ни ASC, ни DESC, то по умолчанию предполагается ASC. Последовательность указания имен столбцов в предложении CREATE INDEX соответствует обычным образом упорядочению от старшего к младшему. Например, предложение

CREATE INDEX X ON B (P, Q DESC, R)

создает индекс с именем X над базовой таблицей B, в котором статьи упорядочиваются по возрастанию значений R в рамках убывающих значений Q в рамках возрастающих значений P. Столбцы P, Q и R не обязательно должны быть смежными. Не обязательно также, чтобы все они имели один и тот же тип данных. Наконец, не обязательно, чтобы все они были фиксированной длины.

После того как индекс создан, он автоматически поддерживается (программой управления хранимыми данными) с тем, чтобы отражать все обновления базовой таблицы, до тех пор, пока этот индекс не будет уничтожен.

Факультативный параметр UNIQUE (уникальный) в предложении CREATE INDEX специфицирует, что никаким двум записям в индексируемой базовой таблице не позволяется принимать одно и то же значение для индексируемого поля (или комбинации полей) в одно и то же время. В случае базы данных поставщиков и деталей, например, мы специфицировали бы, вероятно, следующие индексы с параметром UNIQUE:

CREATE UNIQUE INDEX XS ON S (НОМЕР_ПОСТАВЩИКА);
CREATE UNIQUE INDEX XP ON P (НОМЕР_ДЕТАЛИ);
CREATE UNIQUE INDEX XSP ON
SP (НОМЕР_ПОСТАВЩИКА, НОМЕР_ДЕТАЛИ);

Индексы, подобно базовым таблицам, могут создаваться и уничтожаться в любое время. В приведенном примере, однако, нам хотелось бы, вероятно, создать индексы XS, XP и XSP в то же время, когда создаются сами базовые таблицы S, P и SP. Если же эти базовые таблицы будут непустыми в, то время, когда издается предложение CREATE INDEX, ограничения уникальности могут уже оказаться нарушенными. Попытка создания индекса с параметром UNIQUE над таблицей, которая в настоящее время не удовлетворяет ограничению уникальности, будет завершаться неудачно.

Примечание. Два неопределенных значения считаются равными друг другу для целей индексирования с параметром UNIQUE, Смысл этого довольно загадочного замечания состоит в том, что неопределенные значения не всегда рассматриваются как эквивалентные друг другу во всех контекстах. Обсуждение этого вопроса содержится в следующей части пособия.

Над одной и той же базовой таблицей может быть построено любое число индексов. Например, другой индекс для таблицы S:

CREATE INDEX XSC ON S (ГОРОД);

В этом случае не было специфицировано UNIQUE, поскольку множество поставщиков может находиться в одном и том же городе. Предложение уничтожения индекса имеет вид:

DROP INDEX имя—индекса;

в результате индекс уничтожается, т. е. его описание удаляется из каталога. Если какой-либо существующий план прикладной задачи зависит от этого уничтоженного индекса, то этот план будет помечен как недействительный супервизором стадии исполнения. Когда этот план будет в дальнейшем вызываться для исполнения, супервизор стадии исполнения автоматически вызовет генератор планов прикладных задач с тем, чтобы сгенерировать заменяющий его план, который поддерживал бы

первоначальные предложения SQL, не используя исчезнувший теперь индекс. Этот процесс полностью скрыт от пользователя.

Упражнения

1. На рис. 8.1 приведены некоторые примеры значений данных для базы данных, содержащей информацию, касающуюся поставщиков (таблица S), деталей (таблица P) и проектируемых изделий (таблица J). Поставщики, детали и изделия уникально идентифицируются при этом соответственно номером поставщика, номером детали и номером изделия. Смысл записей таблицы SPJ состоит в том, что специфицированый поставщик поставляет специфицированную деталь для специфицированного проектируемого изделия в специфицированном количестве. Комбинация НОМЕР_ПОСТАВЩИКА, НОМЕР_ДЕТАЛИ и НОМЕР_ИЗДЕЛИЯ уникально идентифицирует такие записи. Напишите для этой базы данных соответствующее множество предложений CREATE TABLE.
2. Запишите множество предложений CREATE INDEX для базы данных из упражнения 1 таким образом, чтобы привести в действие требуемые ограничения уникальности.

На рисунке ниже приведены некоторые примеры значений данных для базы данных, содержащей информацию, касающуюся поставщиков (таблица S), деталей (таблица P) и проектируемых изделий (таблица J). Поставщики, детали и изделия уникально идентифицируются при этом соответственно номером поставщика, номером детали и номером изделия. Смысл записей таблицы SPJ состоит в том, что специфицированый поставщик поставляет специфицированную деталь для специфицированного проектируемого изделия в специфицированном количестве. Комбинация НОМЕР_ПОСТАВЩИКА, НОМЕР_ДЕТАЛИ и НОМЕР_ИЗДЕЛИЯ уникально идентифицирует такие записи. Напишите для этой базы данных соответствующее множество предложений CREATE TABLE. *Примечание.* Эта база данных будет использоваться в последующих частях.

S	НОМЕР_ ПОСТАВЩИКА	ФАМИЛИЯ	СОСТОЯНИЕ	ГОРОД
	S1	Смит	20	Лондон
	S2	Джонс	10	Париж
	S3	Блейк	30	Париж
	S4	Кларк	20	Лондон
	S5	Адамс	30	Атенс

P	НОМЕР_ ДЕТАЛИ	НАЗВАНИЕ	ЦВЕТ	ВЕС	ГОРОД
	P1	Гайка	Красный	12	Лондон
	P2	Болт	Зеленый	17	Париж
	P3	Винт	Голубой	17	Рим
	P4	Винт	Красный	14	Лондон
	P5	Кулачок	Голубой	12	Париж
	P6	Блюм	Красный	19	Лондон

J	НОМЕР_ИЗДЕЛИЯ	НАЗВАНИЕ	ГОРОД
	J1	Сортировщик	Париж
	J2	Перфоратор	Рим
	J3	Считыватель	Атенс
	J4	Консоль	Атенс
	J5	Сортировочно-подборочная машина	Лондон
	J6	Терминал	Осло
	J7	Лента	Лондон

SPJ	НОМЕР_ ПОСТАВ ЩИКА	НОМЕР_ ДЕТАЛИ	НОМЕР_ ИЗДЕЛИЯ	КОЛИЧЕСТВО
	S1	P1	J1	200
	S1	P1	J4	700
	S2	P3	J1	400
	S2	P3	J2	200
	S2	P3	J3	200
	S2	P3	J4	500
	S2	P3	J5	600
	S2	P3	J6	400
	S2	P3	J7	800
	S2	P5	J2	100
	S3	P3	J1	200
	S3	P4	J2	500
	S4	P6	J3	300
	S4	P6	J7	300
	S5	P2	J2	200
	S5	P2	J4	100
	S5	P5	J5	500
	S5	P5	J7	100
	S5	P6	J2	200
	S5	P1	J4	100
	S5	P3	J4	200
	S5	P4	J4	800
	S5	P5	J4	400
	S5	P6	J4	500

База данных поставщиков, деталей, изделий

8.2. Операции выборки данных

В языке SQL предусмотрено четыре предложения манипулирования данными: SELECT, UPDATE, DELETE и INSERT. В этой и следующей главе рассматривается предложение SELECT. В части 8.4 рассматриваются три других предложения. Предполагается также, если не оговорено противное, что все предложения языка вводятся интерактивным способом.

Примечание. Многие из примеров, особенно в следующей главе, являются весьма сложными. Дело, скорее, в том, что обычные операции настолько просты в SQL (а фактически, в большинстве реляционных языков), что примеры таких операций оказываются довольно неинтересными и не иллюстрируют полной мощности этого языка. Сначала, конечно, приводятся некоторые простые примеры затем более сложная, но чрезвычайно важная возможность, называемая *соединением.*

Примеры запросов

Начнем с простого примера — с запроса «Выдать номера и состояния для поставщиков, находящихся в Париже». Этот запрос может быть выражен в SQL следующим образом:

```
SELECT НОМЕР_ПОСТАВЩИКА, СОСТОЯНИЕ
FROM S
WHERE ГОРОД = 'Париж';
```

В качестве результата получим:

НОМЕР_ПОСТАВЩИКА	СОСТОЯНИЕ
S2	10
S3	30

Этот пример иллюстрирует самую общую форму предложения SELECT в языке SQL—

“SELECT (выбрать) специфицированные поля
FROM (из) специфицированной таблицы
WHERE (где) некоторое специфицированное условие является истинным”

Заметим, что результатом запроса является другая таблица — таблица, которая некоторым образом получается из заданных в базе данных таблиц. Иными словами, в реляционной системе типа ORACLE пользователь всегда действует в рамках простой табличной структуры, и это — весьма привлекательная особенность таких систем.

В данном случае было бы вполне возможно сформулировать запрос, используя *уточненные имена полей:*

SELECT S.НОМЕР_ПОСТАВЩИКА, S.СОСТОЯНИЕ
FROM S
WHERE S.ГОРОД = ’Париж’;

Использование уточненных имен никогда не рассматривается как ошибка, и иногда это существенно.

Для справочных целей ниже показана общая форма предложения SELECT, в которой, однако, опущена возможность UNION, обсуждаемая в следующей части:

SELECT [DISTINCT] элемент (ы)
FROM таблица (или таблицы)
[WHERE предикат]
[GROUP BY поле (или поля) [HAVING предикат]]
[ORDER BY поле (или поля)];

Перейдем теперь к иллюстрации основных особенностей этого предложения с помощью весьма продолжительной серии примеров.

Простая выборка

Выдать номера для всех поставляемых деталей:

```
SELECT НОМЕР_ДЕТАЛИ
FROM SP;
```

Имеем результат:

НОМЕР_ДЕТАЛИ
P1
P2
P3
P4
P5
P6
P1
P2
P2
P2
P4
P5

Обратим внимание на дубликаты номеров деталей в этом результате. Система не исключает дубликатов из результата предложения SELECT, если пользователь явно не потребует это сделать с помощью ключевого слова DISTINCT (различный, различные), как показано в следующем примере.

Выборка с исключением дубликатов

Выдать номера для всех поставляемых деталей, исключая избыточные дубликаты:

```
SELECT DISTINCT НОМЕР_ДЕТАЛИ
FROM SP;
```

В этом случае результат таков:

НОМЕР_ДЕТАЛИ
P1
P2
P3
P4
P5
P6

Выборка вычисляемых значений

Выдать номер и вес каждой детали в граммах для всех деталей, предполагая, что в таблице Р веса деталей заданы в фунтах (454 грамма):

```
SELECT НОМЕР_ДЕТАЛИ, ВЕС*454
FROM P;
```

Получаем результат:

НОМЕР_ДЕТАЛИ	
P1	5448
P2	7718
P3	7718
P4	6356
P5	5448
P6	8626

Фраза SELECT (и фраза WHERE) может включать арифметические выражения, а также простые имена полей. Можно, кроме того, осуществлять выборку просто констант. Например:

```
SELECT НОМЕР_ДЕТАЛИ, 'Вес в граммах = ', ВЕС*454
FROM P;
```

Получаем результат:

НОМЕР_ДЕТАЛИ		
P1	**’Вес в граммах =**	**5448**
P2	**’Вес в граммах =**	**7718**
P3	**’Вес в граммах =**	**7718**
P4	**’Вес в граммах =**	**6356**
P5	**’Вес в граммах =**	**5448**
P6	**’Вес в граммах =**	**8626**

Заметим, что в этом результате три столбца.

В связи с этим примером возникает следующий вопрос: что произойдет, если вес какой-либо детали имеет неопределенное значение (NULL)? Напомним, что NULL представляет неизвестное значение. Предположим, например, что вес детали P1 задан в базе данных как неопределенное значение вместо значения 12. Каково тогда значение выражения ВЕС*454 для этой детали? Ответ состоит в том, что оно также является неопределенным значением. В общем случае фактически *любое* арифметическое выражение считается имеющим неопределенное значение, если какой-либо из его операндов сам имеет неопределенное значение. Иными словами, если оказывается, что вес имеет неопределенное значение, то неопределенное значение имеют и все следующие выражения:

ВЕС+454
ВЕС — 454
ВЕС*454
ВЕС/454

Неопределенные значения показываются на терминале как тире или дефис.

Простая выборка «select *»

Выдать полные характеристики для всех поставщиков:

SELECT *
FROM S;

Результатом служит копия полной таблицы S.

Здесь звезда или звездочка служит кратким обозначением списка всех имен полей в таблице (таблицах), указанной(ых) во фразе FROM (из) в том порядке, в котором эти поля определяются в соответствующем(их) предложении(ях) CREATE TABLE. Таким образом, записанное выше предложение SELECT эквивалентно следующему:

```
SELECT НОМЕР_ПОСТАВЩИКА, ФАМИЛИЯ, СОСТОЯНИЕ, ГОРОД
FROM S;
```

Обозначение в виде звездочки удобно для интерактивных запросов, поскольку оно уменьшает число ударов по клавишам. Однако оно таит потенциальную опасность при использовании во встроенном SQL (т. е. в предложениях SQL в прикладной программе), поскольку смысл знака «*» может измениться, если для этой программы перегенерируется план прикладной задачи, а в данном промежутке времени к рассматриваемой таблице был добавлен другой столбец. В этом пособии «SELECT *» будет использоваться только в таких контекстах, где так делать безопасно (в основном только в интерактивных контекстах), и фактическим пользователям рекомендуется поступать подобным образом.

Отметим, наконец, что «*» может уточняться именем соответствующей таблицы. Допустима, например, следующая форма записи:

```
SELECT S.*
FROM S;
```

Ограниченная выборка

Выдать номера поставщиков, которые находятся в Париже и имеют состояние большее, чем 20:

```
SELECT НОМЕР_ПОСТАВЩИКА
FROM S
WHERE ГОРОД = 'Париж'
AND СОСТОЯНИЕ > 20;
```

Результат:

НОМЕР_ПОСТАВЩИКА
S3

Условие, или *предикат*, следующий за ключевым словом WHERE, может включать операторы сравнения =, != (неравно), >, > =, <, и <=, булевские операторы AND (и), OR (или) и NOT (нет), а скобки указывают требуемый порядок вычислений. В таком предикате числа сравниваются алгебраически — отрицательные числа считаются меньшими, чем положительные, независимо от их абсолютной величины. Строки литер сравниваются в соответствии с их представлением в коде ASCII. Если нужно сравнить две строки литер, имеющих разные длины, более короткая строка концептуально дополняется справа пробелами для того, чтобы обе строки имели одинаковую длину перед тем, как будет осуществляться их сравнение.

Выборка с упорядочением

Выдать номера и состояния поставщиков, находящихся в Париже, в порядке убывания их состояния:

```
SELECT НОМЕР_ПОСТАВЩИКА, СОСТОЯНИЕ
FROM S
WHERE ГОРОД = 'Париж'
ORDER BY СОСТОЯНИЕ DESC;
```

Результат:

НОМЕР_ПОСТАВЩИКА	**СОСТОЯНИЕ**
S3	**30**
S2	**10**

В общем случае не гарантируется, что результирующая таблица будет упорядочена каким-либо определенным образом. Здесь, однако, пользователь специфицировал, что результат перед тем, как он будет показан, должен быть организован в определенной последовательности. Упорядочение может быть специфицировано таким же образом, как в предложении CREATE INDEX:

имя—столбца [упорядочение] [, имя—столбца [упорядочение]] ...,

где «упорядочение», как и ранее, это ASC (возрастание) или DECS (убывание), и по умолчанию принимается ASC. Каждое «имя—столбца» должно идентифицировать некоторый столбец *результирующей таблицы.* Поэтому, например, следующее предложение недопустимо:

SELECT НОМЕР_ПОСТАВЩИКА
FROM S
ORDER BY ГОРОД;

Разрешается также идентифицировать столбцы во фразе ORDER BY (упорядочить по) «номерами—столбцов» вместо «имен—столбцов», где «номер—столбца» указывает порядковую позицию (слева направо) данного столбца в результирующей таблице запроса. Благодаря этому возможно упорядочение результата на основе «вычисляемых столбцов», которые не обладают именем. Например, упорядочить результат по возрастанию номера детали в рамках возрастания веса в граммах:

SELECT НОМЕР_ДЕТАЛИ, ВЕС*454
FROM P
ORDER BY 2, НОМЕР_ДЕТАЛИ; [или ORDER BY 2,1;]

Здесь «2» ссылается на второй столбец результирующей таблицы.

Получаем:

НОМЕР_ДЕТАЛИ	
P1	**5448**
P5	**5448**
P4	**6356**
P2	**7718**
P3	**7718**
P6	**8626**

Выборка с использованием between (между)

Выдать сведения о деталях, вес которых находится в диапазоне от 16 до 19 включительно:

```
SELECT НОМЕР_ДЕТАЛИ, НАЗВАНИЕ, ЦВЕТ, ВЕС, ГОРОД
FROM P
WHERE ВЕС BETWEEN 16 AND 19;
```

Имеем следующий результат:

НОМЕР_ ДЕТАЛИ	**НАЗВАНИЕ**	**ЦВЕТ**	**ВЕС**	**ГОРОД**
P2	**Болт**	З еленый	**17**	**Париж**
P3	**Винт**	**Голубой**	**17**	**Рим**
P6	**Блюм**	**Красный**	**19**	**Лондон**

Может быть также специфицировано NOT BETWEEN (не принадлежит диапазону между), например:

```
SELECT НОМЕР_ДЕТАЛИ, НАЗВАНИЕ, ЦВЕТ, ВЕС, ГОРОД
FROM P
WHERE ВЕС NOT BETWEEN 16 AND 19;
```

Получаем тогда:

НОМЕР_ДЕТАЛИ	НАЗВАНИЕ	ЦВЕТ	ВЕС	ГОРОД
P1	**Гайка**	**Красный**	**12**	**Лондон**
P4	**Винт**	**Красный**	**14**	**Лондон**
P5	**Кулачок**	**Голубой**	**12**	**Париж**

Выборка с использованием in (принадлежит)

Выдать детали, вес которых равен 12, 16 или 17:

```
SELECT НОМЕР_ДЕТАЛИ, НАЗВАНИЕ, ЦВЕТ, ВЕС, ГОРОД
FROM P
WHERE ВЕС IN (12, 16, 17);
```

Результат:

НОМЕР_ДЕТАЛИ	НАЗВАНИЕ	ЦВЕТ	ВЕС	ГОРОД
P1	**Гайка**	**Красный**	**12**	**Лондон**
P2	**Болт**	**Зеленый**	**17**	**Париж**
P3	**Винт**	**Голубой**	**17**	**Рим**
P5	**Кулачок**	**Голубой**	**12**	**Париж**

Предикат IN является в действительности просто краткой записью предиката, представляющего собой последовательность отдельных сравнений, соединенных операторами OR (или). Предыдущее предложение SELECT эквивалентно следующему:

```
SELECT НОМЕР_ДЕТАЛИ, НАЗВАНИЕ, ЦВЕТ, ВЕС, ГОРОД
FROM P
WHERE ВЕС = 12
OR ВЕС = 16
OR ВЕС = 17;
```

Имеется в распоряжении также предикат NOT IN (не принадлежит), например предложение;

SELECT НОМЕР_ДЕТАЛИ, НАЗВАНИЕ, ЦВЕТ, ВЕС, ГОРОД
FROM P
WHERE ВЕС NOT IN (12, 16, 17);

дает результат

НОМЕР_ ДЕТАЛИ	НАЗВАНИЕ	ЦВЕТ	ВЕС	ГОРОД
P4	**Винт**	**Красный**	**14**	**Лондон**
P6	**Блюм**	**Красный**	**19**	**Лондон**

Подобно предикату IN предикат NOT IN может рассматриваться только как сокращенная запись другого предиката, который не использует NOT IN. *Упражнение.* Запишите «развернутую форму» предложения из предшествующего примера.

Выборка с использованием предиката like (похоже на)

Выдать все детали, названия которых начинаются с буквы «С»:

SELECT НОМЕР_ДЕТАЛИ, НАЗВАНИЕ, ЦВЕТ, ВЕС, ГОРОД
FROM P
WHERE НАЗВАНИЕ LIKE 'С%';

Получаем результат:

НОМЕР_ ДЕТАЛИ	НАЗВАНИЕ	ЦВЕТ	ВЕ С	ГОРОД
P5	**Кулачок (Cam)**	**Голубой**	**12**	**Париж**
P6	**Блюм (Cog)**	**Красный**	**19**	**Лондон**

Обычно предикат LIKE имеет форму:

имя—столбца LIKE литерная—строковая—константа,
где «имя—столбца» должно обозначать столбец типа CHAR или VARCHAR. Этот предикат принимает для заданной записи значение истина, если значение в указанном столбце соответствует образцу, специфицируемому «литерной—строковой—константой». Литеры этой константы интерпретируются следующим образом:

- Литера «_» (разрыв или подчеркивание) обозначает *любую одиночную литеру.*
- Литера «%» (процент) обозначает *любую последовательность из* n *литер* (где n может быть нулем).
- Все другие литеры обозначают просто сами себя.

Следовательно, в приведенном примере предложение SELECT будет осуществлять выборку записей из таблицы P, для которых значение в столбце НАЗВАНИЕ начинается буквой «С» и содержит любую последовательность из нуля или более литер, следующую за этой буквой «С».

Ниже приведено еще несколько примеров, в которых используется LIKE:

АДРЕС LIKE '% Беркли %' будет принимать значение *истина,* если АДРЕС содержит где-либо внутри него строку 'Беркли'

НОМЕР_ПОСТАВЩИКА LIKE 'S__' будет принимать значение *истина,* если значение в столбце НОМЕР_ПОСТАВЩИКА состоит в точности из трех литер и первая из них литера «S»

НАЗВАНИЕ LIKE '% К___' будет принимать значение *истина,* если значение в столбце НАЗВАНИЕ состоит из четырех или более литер и трем последним из них предшествует литера «К»

ГОРОД NOT LIKE '% E %' — будет принимать значение *истина*, если значение ГОРОД не содержит литеры «Е»

Выборка, при которой вовлекается null (неопределенное значение)

Допустим, например, что значением в столбце СОСТОЯНИЕ для поставщика S5 является не 30, а неопределенное значение. Выдать номера поставщиков, у которых состояние больше, чем 25:

```
SELECT НОМЕР_ПОСТАВЩИКА
FROM S
WHERE СОСТОЯНИЕ > 25;
```

В результате получим:

НОМЕР_ПОСТАВЩИКА
S3

Здесь поставщик S5 не был назван в результате. Если неопределенное значение сравнивается с некоторым другим значением при вычислении предиката, то независимо от используемого оператора сравнения результатом сравнения никогда не является *истина,* даже если этот другой операнд также является неопределенным значением. Иными словами, если оказывается, что СОСТОЯНИЕ имеет неопределенное значение, то ни одно из следующих сравнений не будет принимать значение *истина*:

```
СОСТОЯНИЕ > 25
СОСТОЯНИЕ < = 25
СОСТОЯНИЕ = 25
СОСТОЯНИЕ !=25
СОСТОЯНИЕ = NULL (Это недопустимая синтаксическая конструкция. См. ниже)
СОСТОЯНИЕ != NULL (Это—тоже).
```

Поэтому если издать запрос:

```
SELECT НОМЕР_ПОСТАВЩИКА
FROM S
WHERE СОСТОЯНИЕ < = 25;
```

и сравнить его результат с результатом предыдущего запроса, то можно установить, что поставщик S5 не появляется ни в одном из них. Результат приведенного запроса:

НОМЕР_ПОСТАВЩИКА
S1
S2
S4

Для проверки наличия (или отсутствия) неопределенного значения предусмотрен специальный предикат вида:

имя — столбца IS [NOT] NULL

Например:

```
SELECT НОМЕР_ПОСТАВЩИКА
FROM S
WHERE СОСТОЯНИЕ IS NULL;
```

В результате имеем:

НОМЕР_ПОСТАВЩИКА
S5

Синтаксическая конструкция «СОСТОЯНИЕ = NULL» является некорректной, поскольку ничто — и даже само неопределенное значение — не считается равным неопределенному значению. (Несмотря на это, два неопределенных значения рассматриваются, однако, как дубликаты друг друга при исключении дубликатов.

Предложение SELECT DISTINCT (ВЫБРАТЬ РАЗЛИЧНЫЕ) даст в результате не более одного неопределенного значения. Аналогичным образом индекс со спецификацией UNIQUE (уникальный) будет допускать в индексируемом столбце не более одного неопределенного значения. Наконец, при упорядочении ORDER BY (УПОРЯДОЧИТЬ ПО) неопределенные значения интерпретируются, как будто бы они больше или равны всем значениям, не являющимся неопределенными).

Заметим, между прочим, что использование символа NULL во фразе SELECT не допускается. Например, следующая конструкция некорректна:

```
SELECT НОМЕР_ДЕТАЛИ, ’ВЕС = ’, NULL
FROM P
WHERE ВЕС IS NULL;
```

Запросы, использующие соединение

Способность «соединять» две или более таблицы в одну представляет собой одну из наиболее мощных возможностей реляционных систем. Фактически наличие операции соединения (join) — едва ли не самое главное, что отличает реляционные системы от систем других типов. Итак, что такое соединение? Говоря нестрого, это *запрос, в котором выборка данных осуществляется более чем из одной таблицы.* Ниже приводится простой пример.

Простое эквисоединение

Выдать все комбинации информации о таких поставщиках и деталях, которые размещены в одном и том же городе (иначе говоря, «соразмещены»—безобразный, но удобный термин):

```
SELECT S.*, P.*
FROM S, P
WHERE S.ГОРОД = P.ГОРОД;
```

Заметим, что здесь ссылки на поля во фразе WHERE должны уточняться именами содержащих их таблиц. В результате получим следующую ниже табл. 8.1. (Во избежание двусмысленности в этой таблице два столбца ГОРОД показаны явным образом как S.ГОРОД и P.ГОРОД.)

Таблица 8.1

Простое эквисоединение

НОМЕР_ ПОСТАВЩИКА	ФАМИЛИЯ	СОСТОЯНИЕ	S.ГОРОД
S1	Смит	20	Лондон
S1	Смит	20	Лондон
S1	Смит	20	Лондон
S2	Джонс	10	Париж
S2	Джонс	10	Париж
S3	Блейк	30	Париж
S3	Блейк	30	Париж
S4	Кларк	20	Лондон
S4	Кларк	20	Лондон
S4	Кларк	20	Лондон

НОМЕР_ ДЕТАЛИ	НАЗВАНИЕ	ЦВЕТ	ВЕС	P.ГОРОД
P1	Гайка	Красный	12	Лондон
P4	Винт	Красный	14	Лондон
P6	Блюм	Красный	19	Лондон
P2	Болт	Зеленый	17	Париж
P5	Кулачок	Голубой	12	Париж
P2	Болт	Зеленый	17	Париж
P5	Кулачок	Голубой	12	Париж
P1	Гайка	Красный	12	Лондон
P4	Винт	Красный	14	Лондон
P6	Блюм	Красный	19	Лондон

Пояснение. Из формулировки задачи на естественном языке ясно, что требуемые данные можно получить из двух таблиц — S и P. Поэтому в формулировке запроса на языке SQL мы, прежде всего, указываем эти две таблицы во фразе FROM, а

затем выражаем во фразе WHERE соединение между ними, т. е. тот факт, что значения ГОРОД должны быть равны. Для того чтобы понять, как это делается, представим себе две строки, по одной из каждой таблицы, например, строки, показанные ниже:

НОМЕР_ ПОСТАВЩИКА	ФАМИЛИЯ	СОСТОЯНИЕ	ГОРОД
S1	Смит	20	Лондон

НОМЕР_ ДЕТАЛИ	НАЗВАНИЕ	ЦВЕТ	ВЕС	ГОРОД
P1	Гайка	Красный	12	Лондон

Из этих двух строк можно видеть, что поставщик S1 и деталь P1 в действительности «соразмещены». Из таких двух строк будет сформирована строка результата:

НОМЕР_ ПОСТАВЩИКА	ФАМИЛИЯ	СОСТОЯНИЕ	ГОРОД
S1	Смит	20	Лондон

НОМЕР_ ДЕТАЛИ	НАЗВАНИЕ	ЦВЕТ	ВЕС	ГОРОД
P1	Гайка	Красный	12	Лондон

поскольку они удовлетворяют предикату во фразе WHERE (S.ГОРОД =P.ГОРОД). Это имеет место и для всех других пар строк, содержащих соответствующие значения ГОРОД. Обратите внимание на то, что поставщик S5, размещающийся в Атенсе, не попадает в результирующую таблицу, так как нет каких-либо деталей, хранимых в этом городе. Подобным же образом результат не содержит детали P3, хранимой в Риме, ввиду того, что нет поставщиков, размещенных в Риме.

Результат данного запроса называется *соединением* таблиц S и P по соответствию значений ГОРОД. Термин «соединение» используется также для обозначения операции

конструирования такого результата. Условие S.ГОРОД =P.ГОРОД называется *условием соединения* или *предикатом соединения.* В связи с приведенным примером нужно отметить ряд моментов. Одни из них имеют важное значение, другие не настолько существенны.

- Оба поля в предикате соединения должны быть либо числовыми, либо строками литер. Не обязательно, чтобы их типы данных были идентичны. Однако, по соображениям производительности, это было бы, вообще говоря, неплохо.
- Необязательно, чтобы поля в предикате соединения имели одинаковые имена, хотя очень часто это будет именно так.
- Нет необходимости в том, чтобы оператор сравнения в предикате соединения обязательно был равенством, хотя это будет очень часто. В дальнейшем будут приведены примеры такого рода. В случае оператора равенства соединение называют иногда *эквисоединением.*
- Фраза WHERE в SELECT-соединении может включать, помимо самого предиката соединения, другие условия.
- Можно, конечно, предусмотреть в SELECT выборку только специфицированных полей соединения, а не их всех.
- Выражение
-

SELECT S.*, P.*
FROM S, P

может быть еще более упрощено:

SELECT *
FROM S, P

С другой стороны, оно может быть записано и в расширенном виде:

SELECT НОМЕР_ПОСТАВЩИКА, ФАМИЛИЯ, СОСТОЯНИЕ, S.ГОРОД, НОМЕР_ДЕТАЛИ, НАЗВАНИЕ, ЦВЕТ, ВЕС, P.ГОРОД
FROM S, P

В такой формулировке для S.ГОРОД и P.ГОРОД во фразе SELECT следует указывать их уточненные имена, как показано в примере, поскольку неуточненное имя ГОРОД было бы двусмысленным.

- По определению, эквисоединение должно продуцировать результат, содержащий два идентичных столбца. Если исключить один из этих столбцов, то оставшееся называется *естественным соединением.* Для того чтобы построить естественное соединение таблиц S и P по городам в SQL, следовало бы записать:

SELECT НОМЕР_ПОСТАВЩИКА, ФАМИЛИЯ, СОСТОЯНИЕ, S.ГОРОД, НОМЕР_ДЕТАЛИ, НАЗВАНИЕ, ЦВЕТ, ВЕС
FROM S, P
WHERE S.ГОРОД = P.ГОРОД;

Естественное соединение является, вероятно, одной из наиболее полезных форм соединения — в такой степени, что мы часто используем неуточненный термин «соединение» специально для обозначения этого случая.

- Можно образовывать соединения также и трех, четырех, ... или любого числа таблиц.
- В табл. 8.2 рассматривается альтернативный (и полезный) способ, позволяющий представить себе, каким образом концептуально могут конструироваться соединения. Прежде всего, построим *декартово произведение* таблиц, перечисленных во фразе FROM. Декартово произведение

множества, состоящего из n таблиц, — это таблица, содержащая всевозможные строки r, такие, что r является конкатенацией какой-либо строки из первой таблицы, строки из второй таблицы, ... и строки из n-й таблицы.

Таблица 8.2

Альтернативный способ конструирования соединения

НОМЕР_ПОСТАВЩИКА	ФАМИЛИЯ	СОСТОЯНИЕ	S.ГОРОД	НОМЕР_ДЕТАЛИ	НАЗВАНИЕ	ЦВЕТ	ВЕС	P.ГОРОД
S1	Смит	20	Лондон	P1	Гайка	Красный	12	Лондон
S1	Смит	20	Лондон	P2	Болт	Зеленый	17	Париж
S1	Смит	20	Лондон	P3	Винт	Голубой	17	Рим
S1	Смит	20	Лондон	P4	Винт	Красный	14	Лондон
S1	Смит	20	Лондон	P5	Кулачок	Голубой	12	Париж
S1	Смит	20	Лондон	P6	Блюм	Красный	19	Лондон
S2	Джонс	10	Париж	P1	Гайка	Красный	12	Лондон
…	…	…	…	…	…	…	…	…
S5	Адамс	30	Атенс	P6	Блюм	Красный	19	Лондон

Например, табл. 8.2 (назовем ее CP) представляет собой декартовым произведением таблиц S и P (в указанном порядке). Полная таблица CP содержит 5x6=30 строк. Теперь исключим из этого декартова произведения все такие строки, которые не удовлетворяют предикату соединения. То, что останется, является требуемым соединением. В рассматриваемом случае мы исключаем из таблицы CP все те строки, в которых S.ГОРОД не равен P.ГОРОД. В результате получим в точности

приведенное выше соединение. Между прочим, вполне возможно, хотя, может быть, и несколько необычным образом, сформулировать в языке SQL запрос, результатом которого будет декартово произведение. Например:

```
SELECT S.*, P.*
FROM S, P;
```

Результат. Упомянутая выше таблица СР.

Соединение по условию «больше чем»

Выдать все комбинации информации о поставщиках и деталях, таких, что город местонахождения поставщика следует за городом, где хранится деталь, в алфавитном порядке:

```
SELECT S.*, P.*
FROM S, P
WHERE S.ГОРОД > P.ГОРОД;
```

Получим в результате следующую табл. 8.3.

Таблица 8.3

Соединение по условию «больше чем»

НОМЕР_ПОСТАВЩИКА	ФАМИЛИЯ	СОСТОЯНИЕ	S.ГОРОД
S2	Джонс	10	Париж
S2	Джонс	10	Париж
S2	Джонс	10	Париж
S3	Блейк	30	Париж
S3	Блейк	30	Париж
S3	Блейк	30	Париж

НОМЕР_ДЕТАЛИ	НАЗВАНИЕ	ЦВЕТ	ВЕС	P.ГОРОД
P1	Гайка	Красный	12	Лондон
P4	Винт	Красный	14	Лондон
P6	Блюм	Красный	19	Лондон
P1	Гайка	Красный	12	Лондон
P4	Винт	Красный	14	Лондон
P6	Блюм	Красный	19	Лондон

Соединение с дополнительным условием

Выдать все комбинации информации о поставщиках и информации о деталях, такие, что рассматриваемые поставщики и детали «соразмещены». Опустить при этом поставщиков с состоянием, равным 20:

```
SELECT S.*, P.*
FROM S, P
WHERE S.ГОРОД = P.ГОРОД
AND S.СОСТОЯНИЕ != 20;
```

Результат представлен в табл. 8.4.

Таблица 8.4

Соединение с дополнительным условием

НОМЕР_ ПОСТАВЩИКА	ФАМИЛИЯ	СОСТОЯНИЕ	S.ГОРОД
S2	Джонс	10	Париж
S2	Джонс	10	Париж
S3	Блейк	30	Париж
S3	Блейк	30	Париж

НОМЕР_ ДЕТАЛИ	НАЗВАНИЕ	ЦВЕТ	ВЕС	P.ГОРОД
P2	Болт	Зеленый	17	Париж
P5	Кулачок	Голубой	12	Париж
P2	Болт	Зеленый	17	Париж
P5	Кулачок	Голубой	12	Париж

Выборка специфицированных полей из соединения

Выдать все комбинации номеров поставщиков и номеров деталей, таких, что поставщик и деталь соразмещены:

```
SELECT S.НОМЕР_ПОСТАВЩИКА, P.НОМЕР_ДЕТАЛИ
FROM S, P
WHERE S.ГОРОД = P.ГОРОД;
```

Имеем результат:

НОМЕР_ПОСТАВЩИКА	НОМЕР_ДЕТАЛИ
S1	P1
S1	P4
S1	P6
S2	P2
S2	P5
S3	P2
S3	P5
S4	P1
S4	P4
S4	P6

Соединение трех таблиц

Выдать все пары названий городов, таких, что какой-либо поставщик, находящийся в первом из этих городов, поставляет некоторую деталь, хранимую во втором городе. Например, поставщик S1 поставляет деталь P1. Поставщик S1 находится в Лондоне, а деталь P1 хранится также в Лондоне. Поэтому пара городов «Лондон, Лондон» — это пара городов, которая содержится в результате.

```
SELECT DISTINCT S.ГОРОД, P.ГОРОД
FROM S, SP, P
WHERE S.НОМЕР_ПОСТАВЩИКА = SP.НОМЕР_ПОСТАВЩИКА
AND SP.НОМЕР_ДЕТАЛИ = P.НОМЕР_ДЕТАЛИ;
```

Получаем результат:

S.ГОРОД	P.ГОРОД
Лондон	Лондон
Лондон	Париж
Лондон	Рим
Париж	Лондон
Париж	Париж

В качестве упражнения следует установить, какие конкретно комбинации поставщик — деталь порождают каждую из строк результата в этом примере.

Соединение таблицы с ней самой

Выдать все пары номеров поставщиков, такие, что образующие их поставщики соразмещены.

```
SELECT      ПЕРВАЯ.НОМЕР_ПОСТАВЩИКА,
            ВТОРАЯ. НОМЕР_ПОСТАВЩИКА
FROM S ПЕРВАЯ, S ВТОРАЯ
WHERE ПЕРВАЯ.ГОРОД = ВТОРАЯ.ГОРОД;
```

Нетрудно видеть, что в этом запросе требуется соединение таблицы S с ней самой по соответствию городов. Поэтому таблица S дважды указывается во фразе FROM. Для того чтобы различать эти два ее вхождения, мы вводим в этой фразе два произвольных ее *псевдонима,* ПЕРВАЯ и ВТОРАЯ, и используем их как явные уточнители во фразах SELECT и WHERE.

Получаем результат:

НОМЕР_ПОСТАВЩИКА	НОМЕР_ПОСТАВЩИКА
S1	S1
S1	S4
S2	S2
S2	S3
S3	S2
S3	S3
S4	S1
S4	S4
S5	S5

Мы можем привести в порядок этот результат, расширив следующим образом фразу WHERE:

```
SELECT      ПЕРВАЯ.НОМЕР_ПОСТАВЩИКА,
            ВТОРАЯ.НОМЕР_ПОСТАВЩИКА
FROM S ПЕРВАЯ, S ВТОРАЯ
WHERE ПЕРВАЯ.ГОРОД = ВТОРАЯ.ГОРОД
AND   ПЕРВАЯ.НОМЕР_ПОСТАВЩИКА <
ВТОРАЯ.НОМЕР_ПОСТАВЩИКА
```

Условие ПЕРВАЯ.НОМЕР_ПОСТАВЩИКА < ВТОРАЯ.НОМЕР_ПОСТАВЩИКА дает двоякий эффект: а) оно исключает пары номеров поставщиков вида (x, x); б) оно гарантирует, что не будут появляться одновременно пары (x, y) и (y, x).

Имеем в результате:

НОМЕР_ПОСТАВЩИКА	НОМЕР_ПОСТАВЩИКА
S1	S4
S2	S3

Это первый пример, в котором мы видели, что использование синонимов необходимо. Однако введение таких синонимов никогда не будет ошибкой, даже если их использование не необходимо, и иногда они могут помочь в том, чтобы данное предложение стало более ясным.

Упражнения

Все последующие упражнения к данной части основываются на базе данных поставщиков-деталей-изделий. В каждом из них требуется записать предложение SELECT для указанного запроса. Для удобства ниже вновь приводится структура рассматриваемой базы данных:

S (НОМЕР_ПОСТАВЩИКА, ФАМИЛИЯ, СОСТОЯНИЕ, ГОРОД)

P (НОМЕР_ДЕТАЛИ, НАЗВАНИЕ, ЦВЕТ, ВЕС, ГОРОД)

J (НОМЕР_ИЗДЕЛИЯ, НАЗВАНИЕ, ГОРОД)

SPJ (НОМЕР_ПОСТАВЩИКА, НОМЕР_ДЕТАЛИ, НОМЕР_ИЗДЕЛИЯ, КОЛИЧЕСТВО)

Простые запросы

1. Выдать полный список деталей для всех изделий.
2. Выдать полный список деталей для всех изделий, изготавливаемых в Лондоне.
3. Выдать упорядоченный список номеров

поставщиков, поставляющих детали для изделия номер J1.

4. Выдать список всех поставок, в которых количество деталей находится в диапазоне от 300 до 750 включительно.

5. Выдать список всех комбинаций «цвет детали—город, где хранится деталь», исключая дубликаты пар (цвет—город).

6. Выдать список всех поставок, в которых количество не является неопределенным значением.

7. Выдать номера изделий и города, где они изготавливаются, такие, что второй буквой названия города является «О».

Соединения

8. Выдать все триплеты «номер поставщика, номер детали и номер изделия», такие, что образующие каждый из них поставщик, деталь и изделие являются соразмещенными.

9. Выдать все триплеты «номер поставщика, номер детали и номер изделия», такие, что образующие каждый из них поставщик, деталь и изделие не являются соразмещенными.

10. Выдать все триплеты «номер поставщика, номер детали и номер изделия», такие, что в каждом триплете указанные поставщик, деталь и изделие не являются попарно соразмещенными.

11. Выдать номера деталей, поставляемых каким-либо поставщиком из Лондона, для изделия, изготавливаемого также в Лондоне.

12. Выдать номера деталей, поставляемых каким-либо поставщиком из Лондона.

13. Выдать все пары названий городов, таких, что какой-либо поставщик из первого города поставляет детали для некоторого изделия, изготавливаемого во втором городе.

14. Выдать номера деталей, поставляемых для какого-либо изделия поставщиком, находящимся в том же городе, где изготавливается это изделие.

15. Выдать номера изделий, для которых детали поставляются, по крайней мере, одним поставщиком не из того

же самого города.

16. Выдать все пары номеров деталей, таких, что некоторый поставщик поставляет обе указанные детали.

8.3. Использование подзапросов и функции выборки данных

В этой части мы завершаем обсуждение предложения SELECT языка SQL. План этой части следующий:

— Сначала вводится понятие *подзапроса* (Subquery) или вложенного предложения SELECT. Представляет исторический интерес тот факт, что именно возможность вкладывать одно предложение SELECT внутрь другого первоначально послужила мотивировкой использования прилагательного «структуризованный» в названии языка «структуризованный язык запросов» (Structured Query Language—SQL). Однако более поздние дополнения к языку привели к тому, что сами по себе вложенные предложения SELECT стали значительно менее важными.
— Затем рассматривается *квантор существования* EXISTS (существует), который вместе с соединением расценивается, по мнению автора, как одна из наиболее важных и фундаментальных, хотя, может быть, и не самых легких для использования, возможностей полного языка SQL.
— Далее обсуждаются *стандартные функции* COUNT (число значений), SUM (сумма), AVG (среднее) и т. п. В нем описывается, в частности, использование в связи с этими функциями фраз GROUP BY (группировать по) и HAVING (имея).
— В завершение обсуждается оператор UNION (объединение).
— Для того чтобы попытаться связать воедино ряд идей, введенных в данной и предыдущей частях, в конце представлен пример весьма сложного предложения SELECT и принципиально показано, каким образом такое предложение могло бы обрабатываться системой.

Одна из причин большого размера данной главы заключается в большой избыточности языка SQL в том смысле, что он часто предоставляет несколько различных способов формулировки одного и того же запроса. Поскольку мы пытаемся дать достаточно исчерпывающее представление этого языка, материалы данной части по необходимости также в некоторой степени избыточны.

Необходимо сделать еще одно заключительное вводное замечание, которое может быть не совсем понятным пока не будет прочитана вся часть. Несмотря на то, что наша задача состоит в исчерпывающем обсуждении языка, в данную главу намеренно не включено какое-либо детальное описание вариантов ANY (любой) и ALL (все) операторов сравнения (> ANY, =ALL и т. д.), если необходимо такое детальное описание, можно обратиться к фирменному руководству по системе. Нет такого запроса, сформулированного с их использованием, который нельзя было бы в равной степени хорошо, а на самом деле лучше, сформулировать, используя конструкцию EXISTS (существует). Более того, они запутывают и порождают потенциальную опасность ошибок. Например, корректное предложение:

```
SELECT S.HOMEP_ПОСТАВЩИКА
FROM S
WHERE S.ГОРОД!= ANY (SELECT P.ГОРОД FROM P);
```

вовсе не осуществляет выборки номеров поставщиков, находящихся в городах, не совпадающих с любыми городами, где хранятся детали. Вместо этого оно производит выборку номеров поставщиков таких, что город, в котором размещен каждый из них, не совпадает с каким-нибудь городом, где хранятся детали. Эквивалентная формулировка с помощью EXISTS дает ясную корректную интерпретацию:

```
SELECT S.HOMEP_ПОСТАВЩИКА
FROM S
WHERE EXISTS (SELECT P.ГОРОД FROM P
                     WHERE P.ГОРОД! = S.ГОРОД);
```

(«выдать номера поставщиков таких, что существует некоторый город хранения деталей, который отличается от города, где находится данный поставщик»). Естественная интуитивная интерпретация! = ANY как «не совпадает с любыми» и некорректна и весьма обманчива. Подобная критика относится ко всем операторам, использующим ANY и ALL.

Подзапросы

В этом разделе обсуждаются *подзапросы* или *вложенные предложения* SELECT. Говоря нестрого, подзапрос представляет собой выражение SELECT-FROM-WHERE, которое вложено в другое такое предложение. Обычно подзапросы используются для представления множества значений, исследование которых должно осуществляться в каком-либо предикате IN, как иллюстрируется в следующем примере.

Простой подзапрос

Выдать фамилии поставщиков, которые поставляют деталь P2.

```
SELECT ФАМИЛИЯ
FROM S
WHERE НОМЕР_ПОСТАВЩИКА IN
        (SELECT НОМЕР_ПОСТАВЩИКА
        FROM SP
        WHERE НОМЕР_ДЕТАЛИ = ’P2’)
```

Результат:

ФАМИЛИЯ
Смит
Джонс
Блейк
Кларк

Пояснение. При обработке полного запроса система обрабатывает, прежде всего, вложенный подзапрос. Этот подзапрос возвращает множество номеров поставщиков,

которые поставляют деталь P2, а именно множество ('S1', 'S2', 'S3', 'S4'). Поэтому первоначальный запрос эквивалентен следующему простому запросу:

```
SELECT ФАМИЛИЯ
FROM S
WHERE НОМЕР_ПОСТАВЩИКА IN
              ('S1', 'S2', 'S3', 'S4');
```

и, следовательно, получаем приведенный ранее результат.

Неявное уточнение фамилии в этом примере требует дополнительного обсуждения. Заметим, в частности, что «НОМЕР_ПОСТАВЩИКА» слева от IN неявным образом уточняется именем таблицы S, в то время как «НОМЕР_ПОСТАВЩИКА» в подзапросе неявно уточняется именем таблицы SP. Справедливо следующее общее правило: предполагается, что неуточненное имя поля должно уточняться именем таблицы (или псевдонимом таблицы), указанным в той фразе FROM, которая является непосредственной частью того же самого запроса или подзапроса. В случае поля НОМЕР_ПОСТАВЩИКА слева от IN этой фразой является «FROM S», а в случае поля НОМЕР_ПОСТАВЩИКА в подзапросе—это фраза «FROM SP». Для большей ясности повторим первоначальный запрос с явно указанными предполагаемыми уточнениями:

```
SELECT S.ФАМИЛИЯ
FROM S
WHERE S.НОМЕР_ПОСТАВЩИКА IN
              (SELECT SP.НОМЕР_ПОСТАВЩИКА
              FROM SP
              WHERE SP.НОМЕР_ДЕТАЛИ = 'P2');
```

Неявные уточнения всегда можно отвергнуть путем задания явных уточнений.

Прежде чем перейти к нашему следующему примеру

подзапроса, необходимо отметить еще один важный момент. Первоначальная задача — «Выдать фамилии поставщиков, которые поставляют деталь P2» — может быть эквивалентным образом выражена как запрос с использованием *соединения:*

```
SELECT S.ФАМИЛИЯ
FROM S, SP
WHERE S.HOMEP_ПОСТАВЩИКА = SP.HOMEP_ПОСТАВЩИКА
AND SP.HOMEP_ДЕТАЛИ = 'P2';
```

Пояснение. Соединение S и SP по номерам поставщиков представляет собой таблицу из 12 строк (по одной строке для каждой строки SP), каждая из которых состоит из соответствующей строки SP, дополненной значениями ФАМИЛИЯ, СОСТОЯНИЕ и ГОРОД для поставщика, указываемого значением НОМЕР_ПОСТАВЩИКА в этой строке. Из этих 12 строк только четыре относятся к детали P2. Окончательный результат получается, таким образом, выделением значения ФАМИЛИЯ из этих четырех строк.

Обе формулировки первоначального запроса, одна из которых использует подзапрос, а другая — соединение, в равной степени корректны. Вопрос о том, какой из этих формулировок отдать предпочтение, — исключительно дело вкуса данного пользователя.

Подзапрос с несколькими уровнями вложенности

Выдать фамилии поставщиков, которые поставляют, по крайней мере, одну красную деталь.

```
SELECT ФАМИЛИЯ
FROM S
WHERE HOMEP_ПОСТАВЩИКА IN
        (SELECT HOMEP_ПОСТАВЩИКА
        FROM SP
        WHERE HOMEP_ДЕТАЛИ IN
                (SELECT HOMEP_ДЕТАЛИ
                  FROM P
                WHERE ЦВЕТ = 'Красный'));
```

Результат:

ФАМИЛИЯ
Смит
Джонс
Кларк

Пояснение. Результатом самого внутреннего подзапроса является множество ('P1', 'P4', 'P6'). Подзапрос следующего уровня в свою очередь дает в результате множество ('S1', 'S2', 'S4'). Последний, самый внешний SELECT, вычисляет приведенный выше окончательный результат. Вообще допускается любая глубина вложенности подзапросов.

Для того чтобы убедиться в Вашем понимании этого примера, попытайтесь выполнить следующие упражнения:

a. Перепишите данный запрос так, чтобы все уточнения имен были указаны явным образом.
b. Напишите эквивалентную формулировку этого же запроса с использованием соединения.

Коррелированный подзапрос

Выдать фамилии поставщиков, которые поставляют деталь P2. Этот пример уже рассматривался ранее. Однако для иллюстрации проблемы, рассматриваемой в данном разделе, приведем иное решение этой задачи.

```
SELECT ФАМИЛИЯ
FROM S
WHERE 'P2' IN
            (SELECT НОМЕР_ДЕТАЛИ
            FROM SP
            WHERE НОМЕР_ПОСТАВЩИКА =
                        S.НОМЕР_ПОСТАВЩИКА);
```

Пояснение. В последней строке приведенного запроса неуточненная ссылка на НОМЕР_ПОСТАВЩИКА уточняется неявным образом именем таблицы SP. Другая ссылка явно уточняется именем таблицы S. Этот пример отличается от

предыдущих тем, что внутренний подзапрос не может быть обработан прежде, чем будет обрабатываться внешний запрос, поскольку этот внутренний подзапрос зависит от переменной, а именно от S.НОМЕР_ПОСТАВЩИКА, значение которой *изменяется* по мере того, как система проверяет различные строки таблицы S. Следовательно, с концептуальной точки зрения обработка осуществляется следующим образом.

a. Система проверяет первую строку таблицы S. Предположим, что это строка поставщика «S1». Тогда переменная S.НОМЕР_ПОСТАВЩИКА в данный момент имеет значение 'S1', и система обрабатывает внутренний запрос

(SELECT НОМЕР_ДЕТАЛИ
FROM SP
WHERE НОМЕР_ДЕТАЛИ = 'S1'),

получая в результате множество ('P1', 'P2', 'P3', 'P4', 'P5', 'P6'). Теперь она может завершить обработку для S1. Выборка значения ФАМИЛИЯ для S1, а именно Смит, будет произведена тогда и только тогда, когда 'P2' принадлежит этому множеству, что, очевидно, справедливо.

b. Далее система будет повторять обработку такого рода для следующего поставщика и т.д. до тех пор, пока не будут рассмотрены все строки таблицы S.

Такой подзапрос, как в этом примере, называется *коррелированным.* Коррелированный подзапрос — это такой подзапрос, результат которого зависит от некоторой переменной. Эта переменная принимает свое значение в некотором внешнем запросе. Обработка такого подзапроса, следовательно, должна повторяться для каждого значения переменной в запросе, а не выполняться раз навсегда. Далее будет приведен другой пример коррелированного подзапроса.

Для того чтобы сделать более ясной связь коррелированных подзапросов с внешними запросами, некоторые пользователи любят вводить псевдонимы. Например:

```
SELECT SX.ФАМИЛИЯ
FROM S SX
WHERE ’P2’ IN
            (SELECT НОМЕР_ДЕТАЛИ
            FROM SP
            WHERE НОМЕР_ПОСТАВЩИКА =
                        SX.НОМЕР_ПОСТАВЩИКА);
```

В этом примере псевдонимом является имя SX, введенное во фразе FROM как альтернативное имя таблицы S и используемое далее в качестве явного уточнителя во фразе WHERE подзапроса, а также во фразе SELECT внешнего запроса. Действие приведенного выше полного предложения можно теперь описать более понятно и более точно следующим образом:

— SX — это переменная, областью определения которой является множество записей таблицы S, т. е. переменная, представляющая в любой заданный момент времени некоторую запись таблицы S.
— Поочередно для каждого возможного значения SX выполнить следующее:
  — вычислить подзапрос и получить множество номеров деталей, например, P;
  — добавить к результирующему множеству значение SX.ФАМИЛИЯ, если и только если P2 принадлежит множеству P.

В предыдущем варианте, этого запроса символ «S» в действительности выполнял две различные функции. Он обозначал, конечно, саму базовую таблицу, а также переменную, которая определена на множестве записей этой базовой таблицы. Как уже указывалось, многие считают более ясным использование двух различных символов для того, чтобы различать эти две различные функции.

Введение псевдонима никогда не является ошибкой, а иногда оно необходимо.

Случай использования одной и той же таблицы в подзапросе и внешнем запросе

Выдать номера поставщиков, которые поставляют, по крайней мере, одну деталь, поставляемую поставщиком S2.

```
SELECT DISTINCT НОМЕР_ПОСТАВЩИКА
FROM SP
WHERE НОМЕР_ДЕТАЛИ IN
        (SELECT НОМЕР_ДЕТАЛИ
        FROM SP
        WHERE НОМЕР_ПОСТАВЩИКА = 'S2');
```

Результат:

НОМЕР_ПОСТАВЩИКА
S1
S2
S3
S4

Отметим здесь, что ссылка на SP в подзапросе означает не то же самое, что ссылка на SP во внешнем запросе. В действительности, два имени SP обозначают *различные переменные.* Чтобы этот факт стал явным, можно использовать псевдонимы:

```
SELECT DISTINCT SPX.НОМЕР_ПОСТАВЩИКА
FROM SP SPX
WHERE SPX.НОМЕР_ДЕТАЛИ IN
        (SELECT SPY.НОМЕР_ДЕТАЛИ
        FROM SP SPY
        WHERE SPY.НОМЕР_ПОСТАВЩИКА = 'S2');
```

Эквивалентный запрос с использованием соединения имеет вид:

```
SELECT DISTINCT SPX.НОМЕР_ПОСТАВЩИКА
FROM SP SPX, SP SPY
WHERE SPX.НОМЕР_ДЕТАЛИ = SPY.НОМЕР_ДЕТАЛИ
AND SPY.НОМЕР_ПОСТАВЩИКА = 'S2';
```

Случай, когда в коррелированном и внешнем запросе используется одна и та же таблица

Выдать номера всех деталей, поставляемых более чем одним поставщиком:

```
SELECT DISTINCT SPX.НОМЕР_ДЕТАЛИ
FROM SP SPX
WHERE SPX.НОМЕР_ДЕТАЛИ IN
               (SELECT SPY.НОМЕР_ДЕТАЛИ
               FROM SP SPY
               WHERE SPY.НОМЕР_ПОСТАВЩИКА! =
                       SPX.НОМЕР_ПОСТАВЩИКА);
```

Результат:

НОМЕР_ДЕТАЛИ
P1
P2
P3
P5

Действие этого запроса можно пояснить следующим образом. «Поочередно для каждой строки таблицы SP, скажем SPX, выделить значение НОМЕР_ДЕТАЛИ, если и только если это значение входит в некоторую строку, скажем SPY, таблицы SP, значение столбца НОМЕР_ПОСТАВЩИКА в которой не является его значением в строке SPX». Заметим, что в этой формулировке *должен* быть использован, по крайней мере, один псевдоним — либо SPX, либо SPY, но не они оба, может быть заменен просто на SP.

Подзапрос с оператором сравнения, отличным от in

Выдать номера поставщиков, находящихся в том же городе, что и поставщик S1.

```
SELECT НОМЕР_ПОСТАВЩИКА
FROM S
WHERE ГОРОД =
        (SELECT ГОРОД
        FROM S
        WHERE НОМЕР_ПОСТАВЩИКА = 'S1');
```

Результат:

НОМЕР_ПОСТАВЩИКА
S1
S4

Иногда пользователь может знать, что заданный подзапрос должен возвратить в точности одно значение, как в рассматриваемом примере. В таком случае можно использовать вместо обычного IN более простой оператор сравнения (например, =, > и т. д.). Однако если подзапрос возвращает более одного значения и не используется оператор IN, будет возникать ошибка. Ошибка не возникнет, если подзапрос не возвратит вообще ни одного значения. При этом сравнение интерпретируется в точности так, как если бы подзапрос возвратил неопределенное значение. Иными словами, если x — переменная, то сравнение

x простой—оператор—сравнения (подзапрос),

где «подзапрос» возвращает пустое множество, имеет значение истинности не *истина* или *ложь,* а *неизвестно*.

Нужно отметить, что сравнение в предыдущем примере должно быть записано именно так, как показано — подзапрос должен следовать за оператором сравнения. Иначе говоря, следующая запись запроса некорректна:

```
SELECT НОМЕР_ПОСТАВЩИКА
FROM S
WHERE (SELECT ГОРОД
       FROM S
       WHERE НОМЕР_ПОСТАВЩИКА =
        ’S1’) = ГОРОД;
```

Более того, подзапрос не может включать фраз GROUP BY или HAVING, если он используется с простым оператором сравнения, например с =, > и т.д.

Квантор существования

Запрос, использующий exists

Выдать фамилии поставщиков, которые поставляют деталь Р2.

```
SELECT ФАМИЛИЯ
FROM S
WHERE EXISTS
       (SELECT *
       FROM SP
       WHERE НОМЕР_ПОСТАВЩИКА =
              S.НОМЕР_ПОСТАВЩИКА
       AND НОМЕР_ДЕТАЛИ = ’P2’);
```

Пояснение. EXISTS (существует) представляет здесь *квантор существования* — понятие, заимствованное из формальной логики. Пусть символ «х» обозначает некоторую произвольную переменную. Тогда в логике *предикат с навешенным квантором существования* EXISTS х (предикат—зависящий—от—х) принимает значение *истина* тогда и только тогда, когда «предикат—зависящий—от—х» имеет значение *истина* при каком-либо значении переменной х. Предположим, например, что переменная х обозначает любое целое число в диапазоне от 1 до 10. Тогда предикат

EXISTS x (x < 5)

принимает значение *истина.* Напротив, предикат

EXISTS x (x<0)

принимает значение *ложь.*

В языке SQL предикат с квантором существования представляется выражением вида

EXISTS (SELECT * FROM …).

Такое выражение считается истинным тогда и только тогда, когда результат вычисления подзапроса, представленного с помощью «SELECT * FROM …», является непустым множеством, иными словами, тогда и только тогда, когда существует какая-либо запись в таблице, указанной во фразе FROM подзапроса, которая удовлетворяет условию WHERE этого подзапроса. (На практике этот подзапрос всегда будет коррелированным множеством.)

Вернемся к приведенному выше примеру. Поочередно рассматриваем каждое значение столбца ФАМИЛИЯ и проверяем, является ли для него истинным условие существования. Предположим, что первое значение поля ФАМИЛИЯ — ’Смит’. Тогда соответствующее значение поля НОМЕР_ПОСТАВЩИКА — S1. Является ли пустым множество записей из SP, содержащих НОМЕР_ПОСТАВЩИКА, равный S1, и НОМЕР_ДЕТАЛИ, равный P2? Если ответ отрицателен, то существует запись в SP с НОМЕРОМ_ПОСТАВЩИКА, равным S1, и номером детали, равным P2, и, следовательно, ’Смит’ должно быть одним из результирующих значений. Аналогично поступаем для каждого из других значений столбца ФАМИЛИЯ.

Хотя этот первый пример только показывает иной способ формулировки запроса для задачи, с которой мы уже умеем справляться в языке SQL (используя либо соединение, либо оператор IN), EXISTS представляет собой одну из наиболее важных возможностей полного языка SQL.

Фактически любой запрос, который может быть выражен с использованием IN, может быть альтернативным образом сформулирован также с помощью EXISTS. Однако обратное высказывание несправедливо.

Запрос, использующий not exists

Выдать фамилии поставщиков, которые не поставляют деталь P2.

```
SELECT ФАМИЛИЯ
FROM S
WHERE NOT EXISTS
        (SELECT *
        FROM SP
        WHERE НОМЕР_ПОСТАВЩИКА =
                S.НОМЕР_ПОСТАВЩИКА
        AND НОМЕР_ДЕТАЛИ = 'P2');
```

Результат:

ФАМИЛИЯ
Адамс

Этот запрос можно перефразировать: «Выбрать фамилии поставщиков таких, что не существует поставки, связывающей их с деталью P2». Заметим, что легко преобразовать решение предыдущей задачи в решение данной.

Между прочим, заключенный в скобки подзапрос, входящий в выражение EXISTS, вовсе не обязательно должен использовать предложение SELECT вида «SELECT *». Можно использовать, например, предложение следующего вида: «SELECT имя-поля FROM …». Однако на практике оно почти всегда будет иметь вид «SELECT *», как уже было продемонстрировано в наших примерах.

Запрос, использующий not exists

Выдать фамилии поставщиков, которые поставляют все детали.

Имеется два квантора, обычно встречающихся в логике, EXISTS (существует) и FORALL (для всех). FORALL—это *квантор общности.* В логике предикат с навешенным квантором общности

FORALL x (предикат-зависящий-от-x)

принимает значение *истина* тогда и только тогда, когда «предикат-зависящий-от-x» принимает значение истина для всех значений переменной x. Например, если x снова обозначает любое целое число в диапазоне от 1 до 10, то предикат

FORALL x (x < 100)

принимает значение *истина* в то время, как предикат

FORALL x (x < 5)

принимает значение *ложь*.

В принципе, FORALL — это то, что нужно для формулировки рассматриваемого запроса. Нам хотелось бы сказать примерно следующее: «Выдать фамилии поставщиков таких, что ДЛЯ ВСЕХ (FORALL) деталей СУЩЕСТВУЕТ (EXISTS) запись в таблице SP, указывающая, что данный поставщик поставляет эту деталь». К сожалению, в языке SQL квантор FORALL непосредственно не поддерживается. Однако включающий FORALL предикат всегда может быть преобразован в эквивалентный предикат, содержащий вместо него квантор существования EXISTS, при помощи следующего тождества:

FORALL x (p) = NOT (EXISTS x (NOT (p))).

Здесь «p» — это любой предикат, который зависит от переменной x. Например, предположим еще раз, что x обозначает любое целое число в диапазоне от 1 до 10. Тогда предикат

FORALL x (x < 100)

(значение которого, конечно, *истина*) эквивалентен следующему предикату:

NOT (EXISTS x (NOT (x < 100)))

(«не существует такого х, для которого бы не имело места, что х меньше 100», т. е. «не существует х такого, что х>= 100»). Аналогичным образом, предикат

FORALL x (x < 5)

(который имеет значение *ложь*) эквивалентен предикату

NOT (EXISTS x (NOT (x < 5)))

(«не существует такого х, для которого было бы несправедливо, что х<5», т. е. «не существует такого х, что х> =5»).

В качестве другого примера предположим, что переменные х и у представляют собой действительные числа. Тогда предикат

FORALL x (EXISTS y (y> x)),

значение которого — *истина,* эквивалентен следующему:

NOT (EXISTS (NOT (EXISTS y (y > x))))

(«не существует такого действительного х, для которого не существует такого действительного у, что у больше х»).

Вернемся теперь к рассматриваемой задаче. Можно преобразовать выражение «поставщики такие, что ДЛЯ ВСЕХ деталей СУЩЕСТВУЕТ запись в таблице SP, указывающая, что данный поставщик поставляет эту деталь» в эквивалентное выражение «поставщики такие, что НЕ СУЩЕСТВУЕТ детали

такой, что НЕ СУЩЕСТВУЕТ записи в таблице SP, указывающей, что данный поставщик поставляет эту деталь». Следовательно, формулировка запроса в языке SQL такова:

```
SELECT ФАМИЛИЯ
FROM S
WHERE NOT EXISTS
        (SELECT *
        FROM P
        WHERE NOT EXISTS
                (SELECT *
                FROM SP
                WHERE
                НОМЕР_ПОСТАВЩИКА =
               S.НОМЕР_ПОСТАВЩИКА
                AND НОМЕР_ДЕТАЛИ =
               P.НОМЕР_ДЕТАЛИ));
```

Результат:

ФАМИЛИЯ
Смит

Данный запрос можно перефразировать: «Выдать фамилии поставщиков таких, что не существует детали, которую они бы не поставляли». Вообще говоря, наиболее легкий способ, позволяющий справляться со сложными запросами, подобными только что рассмотренному, состоит, вероятно, в том, чтобы сначала записывать их в «псевдоSQL» форме с использованием квантора FORALL, а затем более или менее механически трансформировать такую запись в реальный SQL, используя взамен NOT EXISTS.

Запрос, использующий not exists

Выдать номера поставщиков, которые поставляют, по крайней мере, все те детали, которые поставляет поставщик S2.

Один из способов справиться с этой сложной задачей состоит в том, чтобы разбить ее на множество более простых

запросов и заниматься ими последовательно. Так, можно сначала определить множество номеров деталей, которые поставляются поставщиком S2:

```
SELECT НОМЕР_ДЕТАЛИ
FROM SP
WHERE НОМЕР_ПОСТАВЩИКА = 'S2';
```

Результат:

НОМЕР_ДЕТАЛИ
P1
P2

Используя предложения CREATE TABLE и INSERT, которые будут обсуждаться в части 8.4, можно сохранить этот результат в некоторой таблице в базе данных, например, в таблице ВРЕМЕННАЯ. Далее можно перейти к определению множества номеров поставщиков, которые поставляют все детали, перечисленные в таблице ВРЕМЕННАЯ:

```
SELECT DISTINCT НОМЕР_ПОСТАВЩИКА
FROM SP SPX
WHERE NOT EXISTS
            (SELECT *
            FROM ВРЕМЕННАЯ
            WHERE NOT EXISTS
                        (SELECT *
                        FROM SP SPY
                              WHERE
                              SPY.НОМЕР_ПОСТАВЩИКА =
                                    SPX.НОМЕР_ПОСТ
                                    АВЩИКА
                              AND SPY.НОМЕР_ДЕТАЛИ =
                              ВРЕМЕННАЯ.НОМЕР_ДЕТАЛИ));
```

Результат:

НОМЕР_ПОСТАВЩИКА
S1
S2

(Заметим, однако, что этот запрос отличается от запроса выше необходимостью использовать, по крайней мере, один псевдоним, поскольку мы выделяем значения столбца НОМЕР_ПОСТАВЩИКА из таблицы SP, а не значения столбца ФАМИЛИЯ из таблицы S. По этой причине необходимо иметь возможность одновременно делать две различные ссылки на таблицу SP.)

Теперь таблица ВРЕМЕННАЯ может быть уничтожена. Идея о том, чтобы справляться со сложными запросами таким пошаговым образом для легкости понимания, часто оказывается полезной. Однако можно также выразить рассматриваемый полный запрос в виде единственного предложения SELECT, полностью исключая при этом необходимость в таблице ВРЕМЕННАЯ:

```
SELECT DISTINCT НОМЕР_ПОСТАВЩИКА
FROM SP SPX
WHERE NOT EXISTS
        (SELECT *
        FROM SP SPY
        WHERE НОМЕР_ПОСТАВЩИКА = ’S2’
        AND NOT EXISTS
            (SELECT *
            FROM SP SPZ
            WHERE
        SPZ.НОМЕР_ПОСТАВЩИКА =
             SPX.НОМЕР_ПОСТАВЩИКА
            AND SPZ.НОМЕР_ДЕТАЛИ =
                SPY.НОМЕР_ДЕТАЛИ));
```

Запрос, в котором используется импликация

Выдать номера поставщиков, поставляющих, по крайней мере, все те детали, которые поставляются поставщиком S2 (тот же самый запрос, что и в предыдущем примере).

В этом примере иллюстрируется еще одно очень полезное понятие — *логическая импликация.* Первоначальную задачу можно перефразировать следующим образом: «Выдать номера поставщиков, скажем, Sx, таких, что ДЛЯ ВСЕХ деталей Py, если поставщик S2 поставляет деталь Py, то поставщик Sx поставляет деталь Py».

Выражение

IF p THEN q (ЕСЛИ p TO q),

где p и q — предикаты, является *предикатом логической импликации.* Он определяется как эквивалент предиката:

NOT (p) OR q.

Иными словами, импликация «IF p THEN q» (читается также следующим образом: «из p СЛЕДУЕТ q») принимает значение *ложь* тогда и только тогда, когда q—*ложь,* а p—*истина,* как показывает табл. 8.5:

Таблица 8.5

Таблица истинности

p	**q**	**IF p THEN q**
T	T	T
T	F	F
F	T	T
F	F	T

Многие формулировки задач на обычном языке весьма естественным образом выражаются в терминах логической импликации. Несколько примеров можно найти в конце данной части среди предлагаемых упражнений. Язык SQL непосредственно не поддерживает импликацию. Но предыдущее определение показывает, каким образом любой содержащий импликацию предикат может быть трансформирован в другой предикат, который ее не содержит.

Пусть, например, p представляет собой предикат «Поставщик S2 поставляет деталь Py», а q — предикат «Поставщик Sx поставляет деталь Py». Тогда предикат

IF p THEN q

эквивалентен предикату

NOT (поставщик S2 поставляет деталь Py)
OR (поставщик Sx поставляет деталь Py);

или в языке SQL:

```
NOT EXISTS
    (SELECT *
    FROM SP SPY
    WHERE SPY.HOMEP_ПОСТАВЩИКА = ’S2’)
    OR EXISTS
        (SELECT *
        FROM SP SPZ
        WHERE SPZ.HOMEP_ПОСТАВЩИКА = Sx
        AND SPZ.HOMEP_ДЕТАЛИ =
    SPY.HOMEP_ДЕТАЛИ)
```

Следовательно, предикат

FORALL Py (IF p THEN q),

который эквивалентен предикату

NOT EXISTS Py (NOT (IF p THEN q)),

т. е, предикату

NOT EXISTS Py (NOT (NOT (p) OR q)),

может быть записан, таким образом, в виде:

NOT EXISTS Py (p AND NOT (q)),

или в языке SQL:

```
NOT EXISTS
    (SELECT *
    FROM SP SPY
```

```
WHERE SPY.НОМЕР_ПОСТАВЩИКА = 'S2'
AND NOT EXISTS
        (SELECT *
        FROM SP SPZ
        WHERE SPZ.НОМЕР_ПОСТАВЩИКА = Sx
        AND SPZ.НОМЕР_ДЕТАЛИ =
        SPY.НОМЕР_ДЕТАЛИ))
```

Поэтому полный запрос принимает вид:

```
SELECT DISTINCT НОМЕР_ПОСТАВЩИКА
FROM SP SPX
WHERE NOT EXISTS
        (SELECT *
        FROM SP SPY
        WHERE SPY.НОМЕР_ПОСТАВЩИКА = 'S2'
        AND NOT EXISTS
                (SELECT *
                FROM SP SPZ
                WHERE
                SPZ.НОМЕР_ПОСТАВЩИКА =
                  SPX.НОМЕР_ПОСТАВЩИКА
                AND SPZ.НОМЕР_ДЕТАЛИ =
                 SPY.НОМЕР_ДЕТАЛИ)):
```

Такой же вид имеет запрос выше. Таким образом, понятие импликации обеспечивает основу для систематического подхода к определенному классу (весьма сложных) запросов и их преобразованию в эквивалентную форму в языке SQL. Попрактиковаться в таком подходе позволяют упражнения в конце данной части.

Стандартные функции

Хотя и весьма мощное во многих отношениях, предложение SELECT в том виде, как оно было до сих пор описано, остается все еще неадекватным для многих

практических задач. Например, даже настолько простой запрос, как «Сколько имеется поставщиков?» нельзя выразить, используя только введенные до сих пор конструкции. Для того чтобы усилить его основные возможности по выборке данных, в SQL предусматривается ряд специальных *стандартных функций.* В настоящее время доступны функции COUNT (число значений), SUM (сумма), AVG (среднее), MAX (максимум) и MIN (минимум). Кроме специального случая «COUNT (*)» (см. ниже) каждая из этих функций оперирует совокупностью значений в одном столбце некоторой таблицы, возможно, *производной,* т. е. сконструированной некоторым образом из заданных базовых таблиц, и продуцирует в качестве ее результата единственное значение, определенное следующим образом:

COUNT — число значений в столбце

SUM — сумма значений по столбцу

AVG — среднее значение в столбце

MAX — самое большое значение в столбце

MIN — самое малое значение в столбце

Для функций SUM и AVG рассматриваемый столбец должен содержать числовые значения. В общем случае аргументу функции может факультативно предшествовать ключевое слово DISTINCT (различный), указывающее, что избыточные дублирующие значения должны быть исключены перед тем, как будет применяться функция. Однако для функций MAX и MIN ключевое слово DISTINCT не имеет отношения к делу и должно быть опущено. Для функции COUNT ключевое слово DISTINCT должно быть специфицировано. Специальная функция COUNT (*), для которой использование DISTINCT не допускается, предусмотрена для подсчета всех строк в таблице без исключения каких-либо дубликатов. Если DISTINCT специфицируется, то аргумент должен состоять только из имени столбца, например ВЕС. Если DISTINCT не специфицировано, аргумент может представлять собой арифметическое выражение, например ВЕС*454.

В столбце-аргументе всегда перед применением функции исключаются все неопределенные значения, независимо от того, специфицировано ли DISTINCT, за исключением случая COUNT (*), при котором неопределенные значения обрабатываются точно так же, как и значения, не являющиеся неопределенными. Если оказывается, что аргумент — пустое множество, функция COUNT принимает значение нуль. Все другие функции принимают в этом случае неопределенное значение.

Функция во фразе select

Выдать общее количество поставщиков.

SELECT COUNT (*)
FROM S;

Результат:
5

Функция во фразе select со спецификацией distinct

Выдать общее количество поставщиков, поставляющих в настоящее время детали:

SELECT COUNT (DISTINCT НОМЕР_ПОСТАВЩИКА)
FROM SP;

Результат:
4

Функция во фразе select с предикатом

Выдать количество поставок для детали P2.

SELECT COUNT (*)
FROM SP
WHERE НОМЕР_ДЕТАЛИ = ’P2’;

Результат:
4

Функция во фразе select с предикатом

Выдать общее количество поставляемых деталей P2.

```
SELECT SUM (КОЛИЧЕСТВО)
FROM SP
WHERE НОМЕР_ДЕТАЛИ = 'P2';
```

Результат:

1000

Функция в подзапросе

Выдать номера поставщиков со значением поля СОСТОЯНИЕ меньшим, чем текущее максимальное состояние в таблице S.

```
SELECT НОМЕР_ПОСТАВЩИКА
FROM S
WHERE СОСТОЯНИЕ <
            (SELECT MAX (СОСТОЯНИЕ)
            FROM S);
```

Результат:

НОМЕР_ПОСТАВЩИКА
S1
S2
S4

Функция в коррелированном подзапросе

Выдать номер поставщика, состояние и город для всех поставщиков, у которых состояние больше или равно среднему для их конкретного города.

```
SELECT НОМЕР_ПОСТАВЩИКА, СОСТОЯНИЕ, ГОРОД
FROM S SX
WHERE СОСТОЯНИЕ >=
            (SELECT AVG (СОСТОЯНИЕ)
            FROM S SY
            WHERE SY.ГОРОД = SX.ГОРОД);
```

Результат:

НОМЕР_ ПОСТАВЩИКА	СОСТОЯНИЕ	ГОРОД
S1	**20**	**Лондон**
S3	**30**	**Париж**
S4	**20**	**Лондон**
S5	**30**	**Атенс**

Включить в результат среднее состояние для каждого города невозможно.

Использование фразы group by

Выше показано, как можно вычислить общий объем поставок для некоторой конкретной детали. Предположим, что теперь требуется вычислить общий объем поставок для *каждой* детали, т. е. для каждой поставляемой детали выдать номер этой детали и общий объем поставок.

```
SELECT НОМЕР_ДЕТАЛИ, SUM (КОЛИЧЕСТВО)
FROM SP
GROUP BY НОМЕР_ДЕТАЛИ;
```

Результат:

НОМЕР_ДЕТАЛИ	
P1	**600**
P2	**1000**
P3	**400**
P4	**500**
P5	**500**
P6	**100**

Пояснение. С концептуальной точки зрения, оператор GROUP BY (группировать по) перекомпоновывает таблицу, представленную фразой FROM, в разделы или *группы* таким образом, чтобы в каждой группе все строки имели одно и то же значение поля, указанного во фразе GROUP BY. Это, конечно,

не означает, что таблица физически перекомпоновывается в базе данных. В рассматриваемом примере строки таблицы SP группируются таким образом, что в одной группе содержатся все строки для детали P1, в другой—все строки для детали P2 и т. д. Далее, к каждой группе перекомпонованной таблицы, а не к каждой строке исходной таблицы применяется фраза SELECT. Каждое выражение во фразе SELECT должно принимать *единственное значение для группы,* т. е. оно может быть либо самим полем, указанным во фразе GROUP BY, либо арифметическим выражением, включающим это поле, либо константой, либо такой функцией, как SUM, которая оперирует всеми значениями данного поля в группе и сводит эти значения к единственному значению.

Строки таблицы можно группировать по любой комбинации ее полей. Будет приведен пример, иллюстрирующий группирование более чем по одному полю. Заметим, что фраза GROUP BY не предполагает ORDER BY (упорядочить по). Чтобы гарантировать упорядочение результата этого примера по номерам деталей, следует специфицировать фразу ORDER BY НОМЕР_ДЕТАЛИ после фразы GROUP BY. Если поле, по значениям которого осуществляется группирование, содержит какие-либо неопределенные значения, то каждое из них порождает отдельную группу.

Использование фразы where с group by

Выдать для каждой поставляемой детали ее номер и общий объем поставок, за исключением поставок поставщика S1:

SELECT НОМЕР_ДЕТАЛИ, SUM (КОЛИЧЕСТВО)
FROM SP
WHERE НОМЕР_ПОСТАВЩИКА != ’S1’
GROUP BY НОМЕР_ДЕТАЛИ;

Результат:

НОМЕР_ДЕТАЛИ	
P1	**300**
P2	**800**
P4	**300**
P5	**400**

Строки, не удовлетворяющие фразе WHERE, исключаются до того, как будет осуществляться какое-либо группирование.

Использование having

Выдать номера деталей для всех деталей, поставляемых более чем одним поставщиком.

```
SELECT НОМЕР_ДЕТАЛИ
FROM SP
GROUP BY НОМЕР_ДЕТАЛИ
HAVING COUNT (*) > 1;
```

Результат:

НОМЕР_ДЕТАЛИ
P1
P2
P4
P5

Фраза HAVING играет такую же роль для групп, что и фраза WHERE для строк. (Конечно, если специфицирована фраза HAVING, то должна быть специфицирована и фраза GROUP BY.) Иными словами, HAVING используется для того, чтобы исключать группы, точно так же, как WHERE используется для исключения строк. Выражение во фразе HAVING должно принимать единственное значение для группы.

Было показано, что этот запрос может быть

сформулирован без GROUP BY (и без HAVING) с использованием коррелированного подзапроса. Однако этот пример в действительности основан на несколько ином восприятии логики, связанной с определением ответа на этот вопрос. Можно также сформулировать запрос, используя по существу *ту же* логику, что и в варианте GROUP BY/HAVING, но без явного использования фраз GROUP BY и HAVING вообще:

```
SELECT DISTINCT НОМЕР_ДЕТАЛИ
FROM SP SPX
WHERE 1 <
        (SELECT COUNT (*)
        FROM SP SPY
        WHERE SPY.НОМЕР_ДЕТАЛИ =
        SPX.НОМЕР_ДЕТАЛИ);
```

Следующий вариант, в котором вместо SPX используется таблица P, является, вероятно, более ясным:

```
SELECT НОМЕР_ДЕТАЛИ
FROM P WHERE 1 <
            (SELECT COUNT (НОМЕР_ПОСТАВЩИКА)
            FROM SP
            WHERE НОМЕР_ДЕТАЛИ =
            P.НОМЕР_ДЕТАЛИ);
```

Еще одна формулировка связана с использованием EXISTS:

```
SELECT НОМЕР_ДЕТАЛИ
FROM P
WHERE EXISTS
            (SELECT *
            FROM SP SPX
            WHERE SPX.НОМЕР_ДЕТАЛИ =
        P.НОМЕР_ДЕТАЛИ
```

```
AND EXISTS
        (SELECT *
        FROM SP SPY
        WHERE SPY.НОМЕР_ДЕТАЛИ =
       P.НОМЕР_ДЕТАЛИ
        AND SPY.НОМЕР_ПОСТАВЩИКА !=
       SPX.НОМЕР_ПОСТАВЩИКА);
```

Все эти альтернативные варианты являются в некотором отношении более предпочтительными по сравнению с вариантом GROUP BY/HAVING в связи с тем, что они, по крайней мере, логически более понятны и, в частности, не требуют этих дополнительных языковых конструкций. Из первоначальной формулировки задачи на естественном языке — «Выдать номера деталей для всех деталей, поставляемых более чем одним поставщиком» — без сомнения, не ясно, что группирование само по себе — это то, что необходимо для ответа на данный вопрос, и в нем, действительно, нет необходимости. Не является также непосредственно очевидным, что необходимо условие HAVING, а не условие WHERE. Вариант GROUP BY/HAVING более похож на процедурное предписание для решения задачи, чем просто на ясную логическую формулировку ее существа. С другой стороны, нельзя опровергнуть тот факт, что вариант GROUP BY/HAVING наиболее лаконичен. Далее, в свою очередь имеются некоторые задачи такого же общего характера, для которых GROUP BY и HAVING просто неадекватны, в силу чего следует использовать один из альтернативных подходов.

Наконец, конструкции GROUP BY свойственно серьезное ограничение — она работает только на одном уровне. Невозможно разбить каждую из этих групп на группы более низкого уровня и т.д., а затем применить некоторую стандартную функцию, например SUM или AVG на каждом уровне группирования.

Объединение

Объединением, двух множеств называется множество всех элементов, принадлежащих какому-либо одному или обоим исходным множествам. Поскольку отношение—это множество (множество строк), можно построить объединение двух отношений. Результатом будет множество, состоящее из всех строк, входящих в какое-либо одно или в оба первоначальных отношения. Если, однако, этот результат сам по себе должен быть другим отношением, а не просто разнородной смесью строк, то два, исходных отношения должны быть *совместимыми по объединению.* Нестрого говоря, строки в обоих отношениях должны быть одной и той же «формы». Что касается SQL, то две таблицы совместимы по объединению (и к ним может быть применен оператор UNION) тогда и только тогда, когда:

a. они, имеют одинаковое число столбцов, например, m;
b. для всех i (i=1, 2, ..., m) i-й столбец первой таблицы и i-й столбец второй таблицы имеют *в точности* одинаковый тип данных:
   - если тип данных—DECIMAL (p, q), то p должно быть одинаковым для обоих столбцов и q должно быть одинаковым для обоих столбцов;
   - если тип данных—CHAR (n), то должно быть одинаковым для обоих столбцов;
   - если тип данных—VARCHAR (n), то n должно быть одинаковым для обоих столбцов;
   - если NOT NULL специфицировано для какого-либо из этих столбцов, то такая же спецификация должна быть для другого столбца.

Запрос, требующий использования union

Выдать номера деталей, которые имеют вес более 16 фунтов либо поставляются поставщиком S2 (либо то и другое).

```
SELECT НОМЕР_ДЕТАЛИ
FROM P
```

```
WHERE ВЕС > 16
UNION
SELECT НОМЕР_ДЕТАЛИ
FROM SP
WHERE НОМЕР_ПОСТАВЩИКА = 'S2';
```

Результат:

P1
P2
P3
P6

Из этого простого примера следует несколько соображений:

— Избыточные дубликаты всегда исключаются из результата UNION. Поэтому, хотя в рассматриваемом примере деталь P2 выбирается обеими из двух составляющих предложений SELECT, в окончательном результате она появляется только один раз.
— Любое число предложений SELECT может быть соединено операторами UNION. Можно расширить данный пример с тем, чтобы включить номера красных деталей, дополнив приведенный выше запрос следующей конструкцией:

```
UNION
SELECT НОМЕР_ДЕТАЛИ
FROM P
WHERE ЦВЕТ = 'Красный'
```

перед заключительной точкой с запятой. Заметим, что такого же результата можно было достигнуть, добавляя к первому из первоначальных предложений SELECT фразу OR ЦВЕТ = 'Красный'.
— Любая фраза ORDER BY в запросе должна входить как часть только в последнее предложение SELECT и должна указывать столбцы, по которым осуществляется

упорядочение, путем указания их порядковых позиций, т. е. их номеров.

— В связи с оператором UNION часто оказывается полезной возможность включения констант во фразу SELECT. Например, можно указать, какому из двух условий WHERE удовлетворяет каждая из отдельных деталей:

```
SELECT НОМЕР_ДЕТАЛИ, ’ее вес> 16 фунтов’
FROM P
WHERE ВЕС > 16
UNION
SELECT НОМЕР_ДЕТАЛИ, ’деталь поставляется S2’
FROM SP
WHERE НОМЕР_ПОСТАВЩИКА = ’S2' ORDER BY 2, 1;
```

Результат:

P1	**деталь поставляется S2**
P2	**деталь поставляется S2**
P2	**ее вес> 16 фунтов**
P3	**ее вес> 16 фунтов**
P6	**ее вес> 16 фунтов**

Когда строковая константа выступает в качестве элемента, подлежащего выборке, считается, что она имеет тип VARCHAR и длину, равную числу литер в константе, и *допускаются неопределенные значения.*

— Может возникнуть желание узнать, поддерживаются ли в языке SQL какие-либо аналоги операторов INTERSECTION (пересечение) и DIFFERENCE (разность), поскольку объединение, пересечение и разность в теоретико-множественных рассмотрениях обычно выступают совместно. Пересечение двух множеств представляет собой множество всех элементов, принадлежащих обоим исходным множествам.

Разность двух множеств — это множество элементов, принадлежащих первому исходному множеству, но не

принадлежащих второму. В языке SQL эти два оператора непосредственно не поддерживаются, но каждый из них может быть смоделирован с помощью функции EXISTS. Пусть, например, A и B — таблицы, состоящие из единственного столбца, а именно, столбца номеров поставщиков. Пусть A представляет «поставщиков из Лондона», а B — «поставщиков, которые поставляют деталь P1».

Тогда

```
SELECT НОМЕР_ПОСТАВЩИКА
FROM A
WHERE EXISTS
        (SELECT НОМЕР_ПОСТАВЩИКА
        FROM B
        WHERE B.НОМЕР_ПОСТАВЩИКА =
  A.НОМЕР_ПОСТАВЩИКА);
```

представляет пересечение A и B, т.е. поставщиков из Лондона, которые поставляют деталь P1, а

```
SELECT НОМЕР_ПОСТАВЩИКА
FROM A
WHERE NOT EXISTS
        (SELECT НОМЕР_ПОСТАВЩИКА
        FROM B
        WHERE B.НОМЕР_ПОСТАВЩИКА =
              A.НОМЕР_ПОСТАВЩИКА);
```

представляет разность между A и B (в указанном порядке), т. е. поставщиков из Лондона, которые не поставляют деталь P1. *Упражнение.* Что представляет собой разность между B и A (именно в этом порядке)?

Заключение

Теперь мы рассмотрели все возможности предложения SELECT языка SQL. Чтобы завершить эту часть, приведем

весьма изощренный пример, который показывает, каким образом многие (но отнюдь не все) эти средства могут быть использованы вместе в едином запросе. Рассмотрим также концептуальный алгоритм обработки SQL — запросов общего вида.

Многоаспектный пример

Выдать номер детали, вес в граммах, цвет и максимальный объем поставки для всех красных и голубых деталей, таких, что общий объем их поставки больше, чем 350, исключая при этом из общего объема все такие поставки, для которых количество меньше или равно 200 деталей. Результат упорядочить по убыванию номеров деталей в рамках возрастающих значений этого максимального объема поставки.

```
SELECT P.НОМЕР_ДЕТАЛИ, ’вес в граммах = ’, P.ВЕС*454, P.ЦВЕТ’максимальный объем поставки =’,MAX (SP.КОЛИЧЕСТВО)
FROM P, SP
WHERE P.НОМЕР_ДЕТАЛИ = SP.НОМЕР_ДЕТАЛИ
AND P.ЦВЕТ IN (’Красный’, ’Голубой’)
AND SP.КОЛИЧЕСТВО > 200
GROUP BY P.НОМЕР_ДЕТАЛИ, P.ВЕС, P.ЦВЕТ
HAVING SUM (КОЛИЧЕСТВО) > 350
ORDER BY 6, P.НОМЕР_ДЕТАЛИ DESC;
```

Результат:

НОМЕР_ ДЕТАЛИ			**ЦВЕТ**		
P1	вес в граммах =	5448	Красный	максимальный объем поставки =	300
P5	вес в граммах =	5448	Голубой	максимальный объем поставки =	400
P3	вес в граммах =	7718	Голубой	максимальный объем поставки =	400

Пояснение. Фразы предложения SELECT применяются в таком порядке, в котором они записаны, за исключением самой фразы SELECT, которая применяется между фразами HAVING и ORDER BY, если они имеются. В данном примере, следовательно, можно представить себе, что результат строится следующим образом.

1. FROM. В результате обработки фразы FROM создается новая таблица, которая является декартовым произведением таблиц P и SP.
2. WHERE. Из результата шага 1 исключаются все строки, не удовлетворяющие фразе WHERE. В данном примере исключаются строки, не удовлетворяющие предикату: P.НОМЕР_ДЕТАЛИ = SP.НОМЕР_ДЕТАЛИ AND P.ЦВЕТ IN ('Красный', 'Голубой') AND SP.КОЛИЧЕСТВО > 200.
3. GROUP BY. Результат шага 2 группируется по значениям поля (полей), указанного во фразе GROUP BY. В нашем примере это поля P.НОМЕР_ДЕТАЛИ, P.ВЕС и P.ЦВЕТ.
4. HAVING. Группы, не удовлетворяющие условию
   SUM (КОЛИЧЕСТВО) > 350,
   исключаются из результата, полученного на шаге 3.
5. SELECT. Каждая группа, полученная на шаге 4, следующим образом генерирует единственную строку для результата. Во-первых, из группы выделяются номер детали, вес, цвет и максимальный объем поставки. Во-вторых, вес преобразуется в граммы. В-третьих, в соответствующие места полученной строки вставляются две строковые константы 'вес в граммах=' и 'максимальный объем поставки='.
6. ORDER BY. Результат шага 5 упорядочивается в соответствии со спецификацией фразы ORDER BY для получения окончательного результата.

Конечно, приведенный выше запрос весьма сложен, но представим себе, какую он выполняет работу. Обычная программа, в другом языке, которая выполняет ту же самую работу, вполне могла бы составить девять страниц по сравнению только с девятью строками, приведенными выше. При этом ра-

бота, необходимая для того, чтобы эта программа стала действующей, значительно больше, чем это необходимо для формулировки приведенного варианта запроса на языке SQL. Большинство запросов на практике будет, конечно, во всяком случае, значительно проще по сравнению с ним.

Упражнения

Как и в предыдущей части, все следующие упражнения основаны на базе данных поставщиков-деталей-изделий. В каждом из них требуется записать предложение SELECT для указанного запроса, за исключением упражнений 15—18 и 26. Для удобства повторим здесь структуру рассматриваемой базы данных:

S (НОМЕР_ПОСТАВЩИКА, ФАМИЛИЯ, СОСТОЯНИЕ, ГОРОД)
P (НОМЕР_ДЕТАЛИ, НАЗВАНИЕ, ЦВЕТ, ВЕС, ГОРОД)
J (НОМЕР_ИЗДЕЛИЯ, НАЗВАНИЕ, ГОРОД)
SPJ (НОМЕР_ПОСТАВЩИКА, НОМЕР_ДЕТАЛИ, НОМЕР_ИЗДЕЛИЯ, КОЛИЧЕСТВО)

В каждом разделе упражнения упорядочены приблизительно в порядке возрастания их сложности. Необходимо попытаться выполнить, по крайней мере, некоторые из легких упражнений в каждой группе. Упражнения 12—18 являются весьма трудными.

Подзапросы

1. Выдать названия изделий, для которых поставляются детали поставщиком S1.
2. Выдать цвета деталей, поставляемых поставщиком S1.
3. Выдать номера деталей, поставляемых для какого-либо изделия в Лондоне.
4. Выдать номера изделий, использующих, по крайней

мере, одну деталь, поставляемую поставщиком S1.

5. Выдать номера поставщиков, поставляющих, по крайней мере, одну деталь, поставляемую, по крайней мере, одним поставщиком, который поставляет, по крайней мере, одну красную деталь.

6. Выдать номера поставщиков, имеющих состояние меньшее, чем у поставщика S1.

7. Выдать номера поставщиков, поставляющих детали для какого-либо изделия с деталью P1 в количестве, большем, чем средний объем поставок детали P1 для этого изделия. *Примечание.* В этом упражнении нужно использовать стандартную функцию AVG.

Квантор EXISTS

8. Повторите упражнение 3.3 и используйте в Вашем решении EXISTS.

9. Повторите упражнение 3.4 и используйте в Вашем решении EXISTS.

10. Выдать номера изделий, для которых не поставляет какой-либо красной детали поставщик из Лондона.

11. Выдать номера изделий, для которых детали полностью поставляет поставщик S1.

12. Выдать номера деталей, поставляемых для всех изделий в Лондон.

13. Выдать номера поставщиков, поставляющих одну и ту же деталь для всех изделий.

14. Выдать номера изделий, для которых поставляются, по крайней мере, все детали, имеющиеся у поставщика S1.

Для следующих четырех упражнений (15—18) преобразуйте приведенное предложение SELECT языка SQL обратно в его эквивалент на естественном языке.

15. **SELECT DISTINCT НОМЕР_ИЗДЕЛИЯ**
FROM SPJ SPJX
WHERE NOT EXISTS
(SELECT *

```
                FROM SPJ SPJY
                WHERE     SPJY.НОМЕР_ИЗДЕЛИЯ     =
            SPJX.НОМЕР_ИЗДЕЛИЯ
                AND NOT EXISTS
                    (SELECT *
                    FROM SPJ SPJZ
                    WHERE SPJZ.НОМЕР_ДЕТАЛИ =
                         SPJY.НОМЕР_ДЕТАЛИ
                    AND
                SPJZ.НОМЕР_ПОСТАВЩИКА = 'S1'));
```

16. **SELECT DISTINCT НОМЕР_ИЗДЕЛИЯ**

```
FROM SPJ SPJX
WHERE NOT EXISTS
                (SELECT *
                FROM SPJ SPJY
                WHERE EXISTS
                    (SELECT *
                    FROM SPJ SPJA
                    WHERE
                    SPJA.НОМЕР_ПОСТАВЩИКА = 'S1'
                    AND SPJA.НОМЕР_ДЕТАЛИ =
                    SPJY.НОМЕР_ДЕТАЛИ)
                AND NOT EXISTS
                    (SELECT *
                    FROM SPJ SPJB
                    WHERE
                SPJB.НОМЕР_ПОСТАВЩИКА = 'S1'
                    AND SPJB.НОМЕР_ДЕТАЛИ =
                    SPJY.НОМЕР_ДЕТАЛИ
                    AND SPJB.НОМЕР_ИЗДЕЛИЯ =
                     SPJX.НОМЕР_ИЗДЕЛИЯ));
```

17. **SELECT DISTINCT НОМЕР_ИЗДЕЛИЯ**

```
FROM SPJ SPJX
WHERE NOT EXISTS
                (SELECT *
                FROM SPJ SPJY
```

```
                WHERE EXISTS
                        (SELECT *
                        FROM SPJ SPJA
                        WHERE
                    SPJA.НОМЕР_ДЕТАЛИ =
                         SPJY.НОМЕР_ДЕТАЛИ
                        AND
                    SPJA.НОМЕР_ИЗДЕЛИЯ =
                         SPJX.НОМЕР_ИЗДЕЛИЯ)
                AND NOT EXISTS
                    (SELECT *
                    FROM SPJ SPJB
                    WHERE
                    SPJB.НОМЕР_ПОСТАВЩИКА = 'S1'
                    AND SPJB.НОМЕР_ДЕТАЛИ =
                        SPJY.НОМЕР_ДЕТАЛИ
                    AND SPJB.НОМЕР_ИЗДЕЛИЯ =
                        SPJX.НОМЕР_ИЗДЕЛИЯ));
```

18. **SELECT DISTINCT НОМЕР_ИЗДЕЛИЯ**

```
FROM SPJ SPJX
WHERE NOT EXISTS
                (SELECT *
                FROM SPJ SPJY
                WHERE EXISTS
                        (SELECT *
                        FROM SPJ SPJA
                        WHERE
                    SPJA.НОМЕР_ПОСТАВЩИКА =
                        SPJY.НОМЕР_ПОСТАВЩИКА
                        AND
                    SPJA.НОМЕР_ДЕТАЛИ IN
                        (SELECT
                    НОМЕР_ДЕТАЛИ
                        FROM P
                        WHERE ЦВЕТ = 'Красный')
                AND NOT EXISTS
```

```
(SELECT *
FROM SPJ SPJB
WHERE SPJB.НОМЕР_ПОСТАВ-
ЩИКА =
      SPJY.НОМЕР_ПОСТАВЩИКА
AND SPJB.НОМЕР_ИЗДЕЛИЯ =
      SPJX.НОМЕР_ИЗДЕЛИЯ)));
```

Стандартные функции

19. Выдать общее число изделий, для которых поставляет детали поставщик S1.

20. Выдать общее количество деталей P1, поставляемых поставщиком S1.

21. Для каждой поставляемой для некоторого изделия детали выдать ее номер, номер изделия и соответствующее общее количество деталей.

22. Выдать номера изделий, для которых город является первым в алфавитном списке таких городов.

23. Выдать номера изделий, для которых средний объем поставки деталей P1 больше наибольшего объема поставки любой детали для изделия J1.

24. Выдать номера поставщиков, поставляющих деталь P1 для какого-либо изделия в количестве, большем среднего объема поставок детали P1 для этого изделия.

Объединение

25. Постройте упорядоченный список всех городов, в которых размещаются, по крайней мере, один поставщик, деталь или изделие.

26. Приведите результат следующего предложения SELECT:

```
SELECT P.ЦВЕТ
FROM P
UNION
SELECT P.ЦВЕТ
FROM P;
```

8.4. Операции обновления

В двух последних частях весьма подробно было рассмотрено предложение выборки данных SELECT языка SQL. Обратим теперь наше внимание на предложения обновления данных UPDATE (обновить), DELETE (удалить) и INSERT (вставить).

Как и предложение SELECT, три предложения обновления данных оперируют не только базовыми таблицами, но и представлениями. Однако *не все представления являются обновляемыми.* Если пользователь попытается выполнять операцию обновления над необновляемым представлением, система просто отвергнет эту операцию с соответствующим сообщением для пользователя. Предположим, следовательно, для целей данной части, что все таблицы, которые будут обновляться, являются базовыми таблицами.

В следующих трех разделах подробно обсуждаются три операции обновления. Синтаксис этих операций следует тому же общему образцу, который был уже показан для операции SELECT. Для удобства в начале соответствующих разделов приводится в общих чертах синтаксис обсуждаемых предложений языка SQL.

Предложение update

Предложение UPDATE имеет следующий общий формат:

```
UPDATE таблица
SET поле = выражение
        [, поле = выражение] …
[WHERE предикат];
```

Все записи в «таблице», которые удовлетворяют «предикату», обновляются в соответствии с присваиваниями «поле = выражение» во фразе SET (установить).

Обновление единственной записи

Изменить цвет детали P2 на желтый, увеличить ее вес на 5 и установить значение города «неизвестен» (NULL).

```
UPDATE P
SET ЦВЕТ = ’Желтый’, ВЕС = ВЕС + 5, ГОРОД =
NULL
WHERE НОМЕР_ДЕТАЛИ = ’P2’;
```

Для каждой записи, которая должна быть обновлена (т. е. для каждой записи, которая удовлетворяет предикату WHERE,. или для всех записей, если фраза WHERE опущена), ссылки во фразе SET на поля этой записи обозначают значения этих полей перед тем, как будет выполнено какое-либо присваивание в этой фразе SET.

Обновление множества записей

Удвоить состояние всех поставщиков, находящихся в Лондоне.

```
UPDATE S
SET СОСТОЯНИЕ = 2*СОСТОЯНИЕ
WHERE ГОРОД = ’Лондон’;
```

Обновление с подзапросом

Установить объем поставок равным нулю для всех поставщиков из Лондона.

```
UPDATE SP
SET КОЛИЧЕСТВО = 0
WHERE ’Лондон’ =
        (SELECT ГОРОД
        FROM S
        WHERE S.НОМЕР_ПОСТАВЩИКА =
              SP.НОМЕР_ПОСТАВЩИКА);
```

Обновление нескольких таблиц

Изменить номер поставщика S2 на S9.

UPDATE S
SET НОМЕР_ПОСТАВЩИКА = ’S9’
WHERE НОМЕР_ПОСТАВЩИКА = ’S2’;
UPDATE SP
SET НОМЕР_ПОСТАВЩИКА = ’S9’
WHERE НОМЕР_ПОСТАВЩИКА = ’S2’;

Невозможно обновить более одной таблицы в единственном запросе. Иными словами, в предложении UPDATE должна специфицироваться в *точности одна таблица.* Поэтому в данном примере мы сталкиваемся со следующей проблемой *целостности* (точнее, с проблемой *целостности по ссылкам): база* данных становится противоречивой после выполнения первого предложения UPDATE — она включает теперь некоторые поставки, для которых не имеется соответствующей записи о поставщике, и остается в таком состоянии до тех пор, пока не будет выполнено второе предложение UPDATE. Изменение порядка предложений UPDATE, конечно, не решает эту проблему. Поэтому важно обеспечить выполнение *обоих* эти предложений, а не только одного.

Предложение delete

Предложение DELETE имеет следующий общий формат:

DELETE
FROM таблица [WHERE предикат];

Удаляются все записи в «таблице», которые удовлетворяют «предикату».

Удаление единственной записи
Удалить поставщика S1.

```
DELETE
FROM S
WHERE НОМЕР_ПОСТАВЩИКА = 'S1';
```

И снова, если таблица SP в настоящее время содержит какие-либо поставки для поставщика S1, это удаление нарушит непротиворечивость базы данных.

Удаление множества записей
Удалить всех поставщиков из Лондона.

```
DELETE
FROM S
WHERE ГОРОД = 'Лондон';
```

Удаление множества записей
Удалить все поставки.

```
DELETE
FROM SP;
```

SP — все еще известная таблица, но она теперь пуста. Удалить все записи — это не уничтожить таблицу (операция DROP).

Удаление с подзапросом
Удалить все поставки для поставщиков из Лондона.

```
DELETE
FROM SP
WHERE 'Лондон' =
            (SELECT ГОРОД
             FROM S
             WHERE S.НОМЕР_ПОСТАВЩИКА =
                        SP.НОМЕР_ПОСТАВЩИКА);
```

Предложение insert

Предложение INSERT имеет следующий общий формат:

INSERT
INTO таблица [(поле [, поле] …)]
VALUES (константа [, константа] …);

или:

INSERT
INTO таблица [(поле [, поле] …)]
подзапрос;

В первом формате в «таблицу» вставляется строка, имеющая заданные значения для указанных полей, причем 1-я константа в списке констант соответствует 1-му полю в списке полей. Во втором формате вычисляется «подзапрос»; копия результата, представляющего собой, вообще говоря, множество строк, вставляется в «таблицу». При этом 1-й столбец этого результата соответствует 1-му полю в списке полей. В обоих случаях отсутствие списка полей эквивалентно спецификации списка всех полей в таблице.

Вставка единственной записи

Добавить в таблицу Р деталь Р7 (город 'Атенс', вес — 2, название и цвет в настоящее время неизвестны).

INSERT
INTO P (НОМЕР_ДЕТАЛИ, ГОРОД, ВЕС)
VALUES ('P7', 'Атенс', 2);

Создается новая запись для детали с заданным номером, городом и весом, с неопределенными значениями для названия и цвета. Эти два последних поля не должны быть, конечно, определены как NOT NULL в предложении CREATE TABLE для таблицы Р. Порядок слева — направо, в котором поля указаны в предложении INSERT, не обязательно должен совпадать с порядком слева — направо, в котором поля были

специфицированы в предложении CREATE (или ALTER).

Вставка единственной записи с опущенными именами полей

Добавить деталь P8 в таблицу P, при этом: название — ’Звездочка’, цвет — ’Розовый’, вес — 14, город — ’Ницца’.

INSERT
INTO P
VALUES (’P8’, ’Звездочка’, ’Розовый’, 14, ’Ницца’);

Отсутствие списка полей эквивалентно спецификации списка всех полей в таблице в порядке слева — направо, как они были определены в предложении CREATE (или ALTER). Как и «SELECT * », такая краткая нотация может быть удобной для интерактивного SQL. Она потенциально опасна, однако, во встроенном SQL, т. е. в предложениях SQL, используемых в прикладной программе, в связи с тем, что предполагаемый список полей может изменяться, если для программы заново осуществляется связывание, а определение таблицы было в этом промежутке времени изменено.

Вставка единственной записи

Вставить новую поставку с номером поставщика S20, номером детали P20 и количеством 1000.

INSERT
INTO SP (НОМЕР_ПОСТАВЩИКА, НОМЕР_ДЕТАЛИ, КОЛИЧЕСТВО)
VALUES (’S20’, ’P20’, 1000);

Подобно операциям UPDATE и DELETE операция INSERT при отсутствии соответствующего управления также может порождать проблему целостности по ссылкам.

Вставка множества записей

Для каждой поставляемой детали получить ее номер и общий объем поставок, сохранить результат в базе данных.

```
CREATE TABLE ВРЕМЕННАЯ
              (НОМЕР_ДЕТАЛИ CHAR (6),
              ОБЪЕМ_ПОСТАВКИ INTEGER);
INSERT
INTO ВРЕМЕННАЯ (НОМЕР_ДЕТАЛИ, ОБЪЕМ_ПОСТАВКИ)
       SELECT НОМЕР_ДЕТАЛИ, SUM (КОЛИЧЕСТВО)
       FROM SP
       GROUP BY НОМЕР_ДЕТАЛИ;
```

Здесь предложение SELECT выполняется точно так же, как обычно, но результат не возвращается пользователю, а копируется в таблицу ВРЕМЕННАЯ. Теперь с этой копией пользователь может делать все, что он пожелает — делать дальнейшие запросы, печатать и даже обновлять ее. Никакая из этих операций не будет оказывать какого-либо влияния на первоначальные данные. В конечном счете, таблицу ВРЕМЕННАЯ можно будет уничтожить, когда она больше не будет нужна:

```
DROP TABLE ВРЕМЕННАЯ;
```

Предыдущий пример очень хорошо показывает, почему свойство замкнутости реляционных систем, является таким важным. Приведенная полная процедура работает именно в связи с тем, что результатом предложения SELECT является другая таблица. Она *не* работала бы, если бы результат был чем-либо иным, кроме таблицы.

Между прочим, целевая таблица вовсе не обязательно должна быть первоначально пустой для вставки множества записей, хотя в приведенном примере это так. Если таблица не пуста, новые записи просто добавляются к тем, которые уже имеются.

Одно из важных применений INSERT … SELECT — построение так называемого *внешнего соединения.* Обычное (естественное) соединение двух таблиц не включает в результате строк какой-либо из двух таблиц, для которых нет соответствующих строк в другой таблице. Например, обычное соединение таблиц S и P по городам не включает какой-либо строки для поставщика S5 или для детали P3, поскольку в Атенсе не хранится никакая деталь и нет поставщиков, находящихся в Риме. Следовательно, в некотором смысле можно считать, что при обычном соединении *теряется информация* для таких несоответствующих строк. Однако иногда может потребоваться способность сохранять эту информацию. Рассмотрим следующий пример.

Использование insert ... select для построения внешнего соединения

Для каждого поставщика получить его номер, фамилию, состояние и город вместе с номерами всех поставляемых им деталей. Если данный поставщик не поставляет вообще никаких деталей, то выдать информацию для этого поставщика, оставляя в результате пробелы вместо номера детали.

```
CREATE TABLE ВНЕШ_СОЕДИНЕНИЕ
        (НОМЕР_ПОСТАВЩИКА CHAR (5),
        ФАМИЛИЯ CHAR (20),
        СОСТОЯНИЕ SMALLINT,
        ГОРОД CHAR (15),
        НОМЕР_ДЕТАЛИ CHAR (6);
INSERT
INTO ВНЕШ_СОЕДИНЕНИЕ
    SELECT S.*, SP.НОМЕР_ДЕТАЛИ
    FROM S, SP
    WHERE S.НОМЕР_ПОСТАВЩИКА =
    SP.НОМЕР_ПОСТАВЩИКА;
INSERT
INTO ВНЕШ_СОЕДИНЕНИЕ
```

```
SELECT S.*, 'bb'
FROM S
WHERE NOT EXISTS
            (SELECT *
             FROM SP
             WHERE
       SP.НОМЕР_ПОСТАВЩИКА=
        S. НОМЕР_ПОСТАВЩИКА);
```

Пояснение. Первые двенадцать строк приведенного результата соответствуют первому из двух INSERT … SELECT и представляют собой обычное естественное соединение таблиц S и SP по номерам поставщиков, за исключением того, что не включен столбец КОЛИЧЕСТВО. Последняя строка результата соответствует второму INSERT … SELECT и сохраняет информацию для поставщика S5, который не поставляет никаких деталей. Полный результат представляет собой *внешнее* соединение таблиц S и SP по номерам поставщиков, в котором опущен столбец КОЛИЧЕСТВО. В противоположность этому обычное соединение называется иногда *внутренним* соединением.

Заметим, что нужны два отдельных INSERT … SELECT, поскольку подзапрос не может содержать UNION.

Упражнения

Как обычно, все следующие упражнения основаны на базе данных поставщиков-деталей-изделий:

S (НОМЕР_ПОСТАВЩИКА, ФАМИЛИЯ, СОСТОЯНИЕ, ГОРОД)

P (НОМЕР_ДЕТАЛИ, НАЗВАНИЕ, ЦВЕТ, ВЕС, ГОРОД)

J (НОМЕР_ИЗДЕЛИЯ, НАЗВАНИЕ, ГОРОД)

SPJ (НОМЕР_ПОСТАВЩИКА, НОМЕР_ДЕТАЛИ, НОМЕР_ИЗДЕЛИЯ, КОЛИЧЕСТВО)

Запишите подходящее предложение INSERT, DELETE или UPDATE для каждой из следующих задач.

1. Измените цвет всех красных деталей на оранжевый.
2. Удалите все изделия, для которых нет поставок деталей.
3. Увеличьте размер поставки на 10 процентов для всех поставок тех поставщиков, которые поставляют какую-либо красную деталь.
4. Удалите все изделия из Рима и все соответствующие поставки.
5. Вставьте в таблицу S нового поставщика S10. Его фамилия и город — ’Уайт’ и ’Нью-Йорк’ соответственно, а состояние еще неизвестно.
6. Постройте таблицу, содержащую список номеров деталей, которые поставляются либо каким-нибудь поставщиком из Лондона, либо для какого-либо изделия в Лондоне.
7. Постройте таблицу, содержащую список номеров изделий, которые либо находятся в Лондоне, либо для них поставляются детали каким-нибудь поставщиком из Лондона.
8. Добавьте 10 к состоянию всех поставщиков, состояние которых в настоящее время меньше, чем состояние поставщика S4.
9. Постройте внешнее естественное соединение изделий и поставок по номерам изделий.
10. Постройте внешнее естественное соединение деталей и изделий по городам.
11. Постройте таблицу, содержащую полную информацию о поставщиках, деталях и изделиях, с указанием объема поставок для каждой поставки вместе с «сохраненной» информацией для каждого поставщика, детали и изделия, которые не входят в таблицу поставок.

ЗАКЛЮЧЕНИЕ

В данном учебном пособии представлен широкий тематический обзор материала, касающегося истории развития технологии баз данных, современных тенденций развития, а также современного состояния дел в данной области.

Первая глава данного пособия посвящена историческому обзору технологии баз данных, ее развитию, указываются сферы применения приложений в различное время.

Вторая глава касается вопросов первого этапа моделирования при разработке баз данных – методам описания предметной области. Также рассмотрены различные модели представления данных.

В третьей главе пособия подробно рассматриваются вопросы, касающиеся реляционной алгебры данной модели хранения данных.

Четвертая глава содержит обзор этапов проектирования баз данных, а также отдельно освещен один из важнейших моментов при проектировании реляционных баз данных – нормализация отношений.

Пятая и шестые главы посвящены механизмам поддержания защищенности и целостности информационных массивов.

Седьмая глава рассматривает различные аспекты эксплуатации баз данных.

Восьмая глава является подробным пособием по языку SQL – языку манипулирования данными в реляционных системах.

В каждой главе приведены примеры применения расмотренных приложений. В главах, касающихся непосредственно проектирования и реализации баз данных имеются задания и упражнения для самостоятельного решения вопросов разработки и приобретения практических навыков.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Глушаков, С. Базы данных [Текст] / С. Глушаков, Д. Ломотько. – Харьков: Фолио, М.: АСТ, 2000. – 504 с.

2. Гофман, В. Работа с базами данных в Delphi [Текст] / В. Гофман, А. Хомоненко. – СПб: БХВ-Петербург, 2003. – 656 с.

3. Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика [Текст] / Т. Коннолли, К. Бегг. – СПб: Питер, 2000. – 1120 с.

4. Шкарина, Л. Язык SQL: учебный курс [Текст] / Л. Шкарина. – СПб: Питер, 2001. – 592 с.

5. Бобровски, С. Oracle 7 и вычисления клиент/сервер [Текст]: пер. с англ. / С. Бобровски. – М.: Изд-во «Лори», 1995. – 652 с.

6. Oracle 7.3. Энциклопедия пользователя [Текст]: пер. с англ. / М. Ригардс и др. – Киев: Изд-во «Диа Софт», 1997. – 832 с.

7. Дейт, К. Руководство по реляционной СУБД DB2 [Текст]: пер. с англ. / К. Дейт. – М.: Финансы и статистика, 1988. – 320 с.

8. Хансен, Г. БД: разработка и управление [Текст]: пер с англ. / Г. Хансен, Д. Хансен. – М.: ЗАО «Издательство БИНОМ», 1999. – 704 с.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

ОСНОВНЫЕ ТЕРМИНЫ 5

1. ПОНЯТИЕ БАЗЫ ДАННЫХ. ФАЙЛОВЫЕ СИСТЕМЫ И БАЗЫ ДАННЫХ. КЛАССИФИКАЦИЯ ЗАДАЧ, РЕШАЕМЫХ С ИСПОЛЬЗОВАНИЕМ СУБД 6

2. МОДЕЛИ ДАННЫХ. ОТОБРАЖЕНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ. СУЩНОСТИ И СВЯЗИ. МЕТОДЫ АБСТРАГИРОВАНИЯ ДАННЫХ. ИЕРАРХИЧЕСКАЯ, СЕТЕВАЯ, РЕЛЯЦИОННАЯ МОДЕЛИ ДАННЫХ 18

2.1. Сущности и связи между ними 18
2.2. Модели данных 27

3. МАТЕМАТИЧЕСКИЕ ОСНОВЫ ПОСТРОЕНИЯ РЕЛЯЦИОННЫХ СУБД. РЕЛЯЦИОННАЯ АЛГЕБРА И БЕЗОПАСНЫЕ ВЫРАЖЕНИЯ. РЕЛЯЦИОННЫЕ ИСЧИСЛЕНИЯ, ПОСТРОЕННЫЕ НА ДОМЕНАХ И КОРТЕЖАХ 40

3.1. Реляционная модель 40
3.2. Описание учебной базы данных 48
3.3. Операции реляционной алгебры 58
3.4. Реляционное исчисление 75
3.5. Задания для самостоятельной работы 80

4. ЗАДАЧИ И ЭТАПЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ. ИСПОЛЬЗОВАНИЕ НОРМАЛЬНЫХ ФОРМ ПРИ ПРОЕКТИРОВАНИИ ПРИЛОЖЕНИЙ В РЕЛЯЦИОННЫХ СУБД. ЭТАПЫ НОРМАЛИЗАЦИИ ОТНОШЕНИЙ. МЕТОДОЛОГИИ ПРОЕКТИРОВАНИЯ 85

4.1. Этапы проектирования баз данных 85
4.2. Нормализация отношений 96

5. ЗАЩИТА БАЗЫ ДАННЫХ 112

6. СРЕДСТВА ПОДДЕРЖАНИЯ ЦЕЛОСТНОСТИ БАЗЫ ДАННЫХ 158
6.1. Основные понятия. Транзакции и их свойства 158
6.2. Понятие управление параллельностью 163
6.3. Методы управления параллельностью 173
6.4. Предупреждение взаимных блокировок 186
6.5. Восстановление данных 196
7. ЭКСПЛУАТАЦИЯ БАЗ ДАННЫХ 205
8. ЯЗЫК МАНИПУЛИРОВАНИЯ ДАННЫМИ SQL 224
8.1. Определение данных 224
8.2. Операции выборки данных 236
8.3. Использование подзапросов и функции выборки данных 262
8.4. Операции обновления 303
ЗАКЛЮЧЕНИЕ 313
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 314

Учебное издание

Плотников Денис Геннадьевич

БАЗЫ ДАННЫХ
И ИХ БЕЗОПАСНОСТЬ

В авторской редакции

Подписано к изданию 27.08.2015.

Объем данных 1,93 Мб.

ФГБОУ ВПО «Воронежский государственный
технический университет»
394026 Воронеж, Московский просп., 14