БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
Факультет радиофизики и электроники
Кафедра системного анализа

В. В. Скакун

Системы управления базами данных

Пособие
для студентов факультета
радиофизики и электроники

Минск
БГУ
2007

УДК 004.65(075.8)
ББК 32.973.26-018.2я73
С42

А в т о р - с о с т а в и т е л ь :
В. В. Скакун

Рекомендовано Ученым советом
факультета радиофизики и электроники
26 сентября 2006 г., протокол № 2

Р е ц е н з е н т ы :
кандидат экономических наук, профессор *А. И. Змитрович*;
кандидат технических наук, доцент *В. Н. Комличенко*

Системы управления базами данных: пособие v. С42 /
авт.-сост. В. В. Скакун. – Минск: БГУ, 2007. – 116 с.
ISBN

Рассматриваются понятия баз данных и систем их управления. Приводится описание и сравнительный анализ основных моделей данных, среди которых наиболее подробно излагается реляционная модель данных. Дан развернутый пример проектирования реляционной базы данных. Приводится достаточно полное описания языка SQL. Излагаются характеристики технологий доступа к данным и архитектур построения баз данных. По каждому разделу приведены контрольные вопросы и задания.

Предназначено для студентов факультета радиофизики и электроники БГУ, а также может быть полезной для студентов технических и экономических специальностей.

УДК 004.65(075.8)
ББК 32.973.26-018.2я73

ВВЕДЕНИЕ

Термин «базы данных» неразрывно связан с понятиями информации и данных. Поэтому логично было бы начать изложение материала из формулировки этих понятий.

Информация – это совокупность сведений, воспринимаемых из окружающей среды, передаваемых в окружающую среду либо сохраняемых внутри информационной системы. Под информационной системой будем понимать любую систему, осуществляющую сбор, накопление и обработку информации. Понятие информации предполагает обязательное наличие источника и приемника информации. Когда источником и приемником являются люди, то говорят, что они обмениваются сообщениями. Если источником информации является объект наблюдения, то наблюдатель получает информацию либо путем наблюдения, либо в процессе активного воздействия на объект наблюдения. Если источником и приемником информации являются технические устройства, то говорят, что они обмениваются сигналами. И если приемником информации является некоторая информационная система, то говорят, что она получает, выдает и преобразует данные.

Производным от информации являются данные. Данные – это запись в соответствующем коде наблюдения, факта, объекта и т. п., пригодная для интерпретации, передачи, обработки и получения информации. Следовательно, понятие информации предполагает существование материального носителя информации и форму, способ, которым она связана со своим носителем. Таким образом, *данные* – это информация, представленная в виде, позволяющем автоматизировать ее сбор, хранение и дальнейшую обработку человеком или информационным средством.

Хранящиеся в информационной системе данные должны быть доступны в том виде, в каком они нужны для конкретной деятельности человека. Сегодня мы можем встретить систему обработки данных традиционного типа, когда служащий вручную заполняет формы и помещает их в скоросшиватель, а рядом с ней современную систему с применением ЭВМ. Несмотря на поразительную несхожесть, обе эти системы обязаны предоставить достоверную информацию в определенное время, в определенном месте и с ограниченными затратами.

Информация получается из данных в процессе решения какой-либо задачи. Однако не вся информация может быть напрямую выведена из данных. Например, человеческие эмоции, отраженные в художественных произведениях, могут быть восприняты совершенно по-разному. Даже простая совокупность чисел в виде 25-05-99 может пониматься разным

способом: как номер телефона 25-05-99 либо как дата 25-05-99. Следовательно, для получения информации из данных необходимы определенные соглашения, называемые *интерпретацией данных*. Другая проблема состоит в том, что не существует определенного компактного кода для записи любой (в общем случае разнородной) информации.

Так как в дальнейшем мы будем говорить о технических информационных системах, то будем рассматривать только ту информацию, которая может быть переведена на язык, понимаемый компьютером. Другим доводом в пользу выбора компьютера как информационной системы является стоимость хранения единицы информации (в настоящее время стоимость хранения единицы информации на компьютерных носителях минимальна). Часто на язык, понимаемый компьютером, переводится только часть информации (набор данных) о какой-то конкретной предметной деятельности человека. Такой набор данных (например, телефонный справочник, список товаров, адреса клиентов некоторой фирмы) и называют базой данных (БД). В общем смысле термин «база данных» можно применить к любой совокупности связанной информации, объединенной по определенному признаку.

Приведем классическое определение БД, данное Дж. Мартином: «БД – это совокупность взаимосвязанных данных при такой их минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений в определенной предметной области человеческой деятельности». Под *предметной областью* мы будем понимать часть реального мира, представляющего интерес для исследования. Таким образом, БД является динамической информационной моделью некоторой предметной области деятельности человека.

Когда речь идет о БД, то под ней очень часто понимают не сам набор связанных данных, а совокупность, состоящую из набора данных и программы обслуживания, осуществляющую взаимодействие пользователя с набором данных, а также проведение операций с данными. К таким операциям можно отнести операции поддержки структуры данных, обеспечения целостности данных, поиск, сортировку, фильтрацию, добавление и удаление данных и т. д. Обычно минимальный набор таких операций осуществляется ядром (процессором) БД. В таком случае правомерно говорить о совокупности, состоящей из набора данных, ядра БД и программы, осуществляющей интерфейс с пользователем (рис. 1).

Полный набор операций с данными предоставляют системы управления базами данных (СУБД). Под СУБД понимают совокупность языковых и программных средств, обеспечивающих создание, поддержание и доступ к данным как со стороны пользователей, так и со стороны приложений. Кроме средств поддержки структуры данных и операций с данными СУБД также предоставляет развитый пользовательский интерфейс,

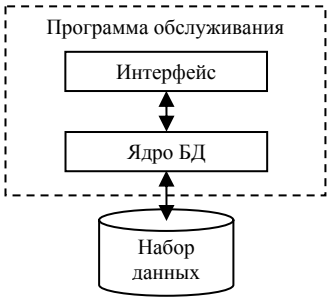


Рис. 1. Структура БД

средства программирования высокого уровня, средства администрирования, обеспечения секретности и безопасности информации.

Таким образом, под БД будем понимать набор взаимосвязанных данных, а под СУБД – программную среду, позволяющую управлять многими БД и имеющую средства их разработки. В то же время не всякий набор данных можно назвать термином БД. Рассмотрим требования к набору данных, чтобы его можно было бы интерпретировать как БД.

1. Связность и структурированность (логическая упорядоченность). Набор беспорядочных данных не может быть признан за БД (например, слова в некотором документе, хотя они объединены местом хранения). Логическая упорядоченность также требует хранения только однородных данных. Например, в поле «Фамилия» должна быть записана только фамилия, а не имея или отчество, и тем более, не название предприятия, где работает данный человек.

2. Контролируемая избыточность данных (Controlling Redundancy). Каждый объект должен быть описан только 1 раз. Данные должны быть избыточными настолько, насколько это требуется для нормального функционирования системы (некоторая избыточность требуется для повышения эффективности работы, ускорения поиска и восстановления информации после сбоев).

3. Целостность (Data Integrity). Подразумевает правильность данных (Data Validity) в любой момент времени.

4. Физическая и логическая независимость (Physical and Logical Independence). Физическая независимость данных предполагает независимость данных от их конкретного размещения на физических носителях, их типа, организации и способа доступа. Логическая независимость предполагает независимость приложений (представления, т. е. интерфейса данных) от изменения логической структуры данных. Например, добавление или удаление некоторых характеристик описываемых объектов и связей между ними не должно привести к необходимости переделки значительной части БД.

5. Безопасность и секретность. Под безопасностью данных понимают защиту данных от случайного или преднамеренного доступа к ним лиц, не имеющих на это право, от неавторизированной модификации данных или от их разрушения. Секретность определяется как право отдельных лиц

или организаций решать, как, когда и какое количество соответствующей информации может быть передано другим лицам или организациям.

Кроме классических БД, предназначенных в основном для хранения текстовой и числовой информации, в последнее время получили развитие новые направления применения технологии БД:

- БД мультимедиа (multimedia databases);
- географические информационные системы (geographic information systems – GIS), предназначенные для хранения карт, погодных данных и снимков со спутников;
- аналитическая обработка данных (Data warehouses, Data mining and on-line analytical processing – OLAP). Системы, использующиеся для выделения и анализа необходимой информации из очень больших БД для принятия решений;
- системы реального времени и активные БД (Real-time and active database technology) используются для контроля проектирования и производства;
- интернет БД включающие серверные БД и поисковые системы.

Работу с БД можно условно разбить на несколько этапов.

Первый этап заключается в определении структуры данных и имплементации ее с помощью некоторой СУБД. В реляционных БД информация сохраняется в связанных таблицах.

Второй этап заключается во вводе данных. Использование специальных форм упрощает ввод-вывод отдельных записей, ограничивая «рабочую зону» пользователя, сконцентрировав его внимание на требуемых данных.

Третий этап – поиск информации либо ее выборка. Осуществляется на основе критериев поиска. Совокупность критериев поиска, предназначенных для поиска и выделения информации, называется запросом.

Четвертый этап заключается в оформлении выбранных записей. Записи могут быть представлены в виде таблиц, форм, графиков, диаграмм и могут оформляться в виде отчетов.

Таким образом, можно выделить основные компоненты БД. Это таблицы (для реляционных БД), запросы, формы и отчеты. Сюда также необходимо добавить и программы, связывающие эти компоненты воедино в приложение.

1. ОСНОВНЫЕ ПОНЯТИЯ

Базы данных появились тогда, когда традиционная файловая система хранения перестала удовлетворять нарастающему потоку данных и требованиям по их обработке. В данной главе будут рассмотрены основные отличия способа хранения данных в БД от принятого в файловой системе, архитектура построения БД и основные свойства БД, выделяющих ее как технологию доступа, хранения и обработки данных.

1.1. ОТЛИЧИЕ ОТ ФАЙЛОВОЙ СИСТЕМЫ ХРАНЕНИЯ ДАННЫХ

Первое отличие состоит в том, что БД хранит данные в структурированном (упорядоченном) и специально организованном виде, исключающем совместное хранения разнородной информации и ее дублирование. К примеру, в текстовых файлах на порядок размещения данных не накладывается сколько-нибудь серьезных ограничений, и данные могут быть расположены произвольно. Некоторые данные могут неоднократно повторяться. В электронных таблицах данные по строкам и столбцам располагаются уже упорядоченно, но все еще достаточно произвольно. Человек сам решает в момент создания таблицы, как лучше и нагляднее разместить данные. И лишь в БД структура данных строго фиксирована и определяется стандартом используемой модели данных.

Второе отличие состоит в том, что БД хранит не только данные, но и описание их структуры (мета-данные). Мета-данные хранятся отдельно от самих данных в так называемом словаре (системном каталоге) данных. Таким образом, любая СУБД может работать с разными наборами данных, поскольку структура их хранения доступна при чтении данных. В файловой системе способ хранения данных – дело каждой программы, осуществляющей хранение и обработку данных. Структура данных встроена в программу доступа и не может быть прочитана другими программами.

В объектно-ориентированных и объектно-реляционных БД можно определить операции над данными как часть их определения. Операция (также называемая функцией) состоит из двух частей: интерфейса (или ее подписи – signature) включающего ее имя со списком аргументов и тела (или имплементации – method). Имплементация хранится отдельно и может быть изменена независимо от интерфейса. Таким образом, использование мета-данных проявляется:

- в независимости программ и данных (program-data independence);
- независимости программ и операций (program-operation independence).

Третьим серьезным отличием БД от файловой системы хранения является наличие расширенных средств поиска информации. Практически всегда в БД применяют индексированное хранение информации. Наряду с упорядоченностью данных индексированное хранение информации дает многократное повышение скорости поиска. Фактически появление БД было вызвано невозможностью дальнейшего повышения скорости поиска в файловых системах с ростом объема хранимой информации.

Соответственно БД снимают следующие ограничения, присущие файловой системе хранения данных:

- разделение и изоляция данных;
- дублирование данных;
- зависимость от данных;
- несовместимость файлов и программ доступа.

1.2. АРХИТЕКТУРА ПОСТРОЕНИЯ БАЗ ДАННЫХ

Так как любая упорядоченность накладывает серьезные ограничения на способ хранения и использования данных, то были предприняты действия, направленные на повышение гибкости доступа к данным. Результатом этих действий стала предложенная в 1978 г. трехуровневая архитектура построения баз данных (рис. 2). Данная схема была разработана как стандарт представления данных (ANSI/SPARC) и в настоящий момент ее поддерживает большинство коммерческих СУБД.

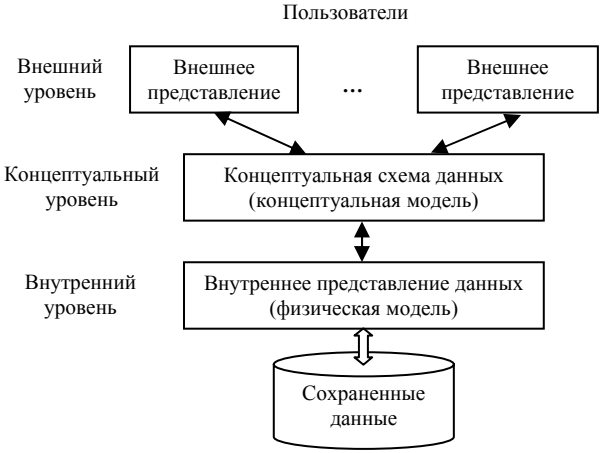


Рис. 2. Трехуровневая архитектура построения БД

Цель трехуровневой архитектуры заключается в отделении пользовательского представления БД от ее физического представления, т. е. обеспечении независимости от данных.

Первый уровень, внутренний (internal). Определяется физической моделью данных, которая описывает размещение данных на физических носителях и способы доступа к ним, структуру файлов, индексов и отдельных информационных единиц.

Второй уровень, концептуальный (conceptual). Определяется концептуальной моделью данных, которая описывает логическую структуру данных без указания деталей их физического хранения.

Третий уровень, внешний, или уровень представлений (интерфейса). Выводит необходимые данные в требуемом формате, скрывая остальную часть БД. Внешнее представление – это содержимое БД, каким его видит определенный пользователь. Одному пользователю нужны сведения о товарах и их размещение на складе и он может не иметь понятия, что в БД хранится еще информация о клиентах, поставщиках и т. д. Пользователь может также изменять свое представление, не оказывая влияние на другие представления. Внешний уровень предоставляет также свободу выбора языка общения с БД. Рядовой пользователь может использовать язык интерфейса, т. е. меню и другие запрограммированные действия. Опытный пользователь может воспользоваться языком запросов SQL. Системный программист может использовать расширение SQL и т. д.

Все эти уровни связаны между собой программами отображения одного уровня в другой путем трансляции запросов. Запрос от конечного пользователя на требуемые данные должен быть интерпретирован на концептуальном уровне и затем преобразован в конечный запрос на извлечение требуемых данных на физическом уровне. Затем эти данные должны быть преобразованы к виду, запрашиваемому пользователем.

1.3. СВОЙСТВА БАЗ ДАННЫХ

Независимость данных. Независимость данных должна обеспечиваться на логическом и физическом уровнях. *Логическая независимость* предполагает независимость приложений (внешнего представления) от изменения логической структуры данных. Схема, представленная на рис. 2, хорошо поясняет независимость данных. Видим, что независимость данных на логическом уровне обеспечивается тем, что внешнее представление данных развязано от способа организации данных. Поэтому изменения на первом и втором уровнях не повлияют на конечное представление данных, т. е. однажды разработанный интерфейс не придется переписывать заново.

Физическая независимость данных предполагает независимость данных от их конкретного размещения на физических носителях, их типа, организации и способа доступа. Независимость данных на этом уровне обеспечивается развязкой первого и второго уровней. При изменении физической модели не потребуется производить изменения на концептуальном и внешнем уровнях. В настоящее время этот уровень независимости обеспечивается СУБД и операционной системой, хотя на ранних порах развития БД представлял собой большую проблему. Для хранения данных применялись накопители на магнитных лентах, причем не существовало единых стандартов размещения данных на лентах и стандартов управления накопителями. Поэтому при смене типа носителя приходилось полностью переписывать все программы доступа к данным.

Контролирование избыточности данных. Избыточность данных в первую очередь связана с хранением повторяющейся информации. Наличие повторяющейся информации приводит не только к увеличению размера БД, но и приводит к возможности появления несогласованного состояния данных. Рассмотрим пример. Нам необходимо хранить информацию о покупках, сделанных в кредит. Тогда для каждой покупки отдельного покупателя придется повторно вводить его персональные данные, например ФИО и адрес. При покупке одинаковых товаров различными покупателями придется каждый раз дублировать описание товара. Такое дублирование информации приводит к следующим серьезным проблемам:

а) приходится тратить лишние усилия на ввод повторяющихся данных;

б) при вводе повторяющихся данных возрастает вероятность ошибки набора;

в) при изменении сведений об объекте необходимо корректировать все записи, содержащие эти сведения;

г) согласованность данных может быть нарушена либо при наличии отличающихся копий одинаковых данных, либо при обновлении не всех повторяющихся данных (аномалия модификации), либо при непреднамеренном удалении информации, которая напрямую не связана с удаляемыми данными (аномалия удаления); при добавлении неполных данных (аномалия добавления);

д) неоправданное увеличение размера данных приведет к снижению скорости выполнения запросов, поиска и т. п.

Основное требование контролируемой избыточности данных заключается в том, что каждая логическая единица данных должна быть сохранена лишь однажды. Данные должны быть избыточными настолько, насколько это требуется для нормальной работы системы, так как некоторая избыточность требуется для повышения эффективности работы,

ускорения поиска и восстановления информации после сбоев. Хранение согласованных данных лежит в основе следующего свойства БД.

Обеспечение целостности и правильности данных. Обеспечение целостности БД составляет необходимое условие ее успешного функционирования. *Целостность* есть свойство БД, означающее, что в ней содержится полная, непротиворечивая, согласованная и адекватно отражающая предметную область информация. Мы привыкли доверять данным, помещаемым в печатных изданиях. При подготовке книги к изданию размещаемые в ней данные проверяют несколько редакторов. Они же стараются сделать так, чтобы книга была написана грамотно и соответствовала нормам. При общении с компьютером соблюдается тот же принцип, т. е. компьютер сравнивается с известным издательством. Если что-то сделано на компьютере, то люди этому слепо верят, мотивируя тем, что компьютер не может ошибаться. Но это не совсем так. О достоверности данных придется забыть, если в компьютер будут помещены некорректные данные или они могут стать некорректными вследствие сбоев и ошибок.

Поддержание целостности включает проверку целостности и ее восстановление в случае обнаружения противоречий в БД. Целостное состояние БД описывается с помощью ограничителей целостности в виде условий, которым должны удовлетворять хранимые в базе данные. Ограничители целостности (constraints) бывают трех типов:

- ограничители значений. К ним относятся: задание типа и формата позволяющего ввод только определенных данных; задание диапазона значений; задание списка значений;
- ссылочная целостность. Обеспечивается контролем отношений между связанными данными и введением каскадного удаления и обновления связанных записей;
- целостность записи. Обеспечивается проверкой на уникальность некоторых данных и объявлением обязательных данных.

Другим важным механизмом поддержания целостности является введение транзакций. *Транзакцией* называется некоторая неделимая последовательность операций над данными БД, которая отслеживается от начала и до завершения. Если по каким-либо причинам (сбои и ошибки) транзакция остается незавершенной, то производится отмена всех операций, входящих в ее состав. Транзакции присущи следующие свойства:

- атомарность (выполняются все входящие в транзакцию операции или ни одна);
- согласованность (любая транзакция должна переводить БД из одного согласованного состояния в другое согласованное состояние);
- изолированность (транзакции выполняются независимо друг от друга);

- безопасность (даже аварийное завершение работы не приводит к потере данных).

Контроль транзакций особенно важен в многопользовательских БД, где транзакции могут быть запущены параллельно. Так как компьютер не может обрабатывать параллельно выполняемые процессы вследствие ограниченности его ресурсов (один центральный процессор), то обычно прибегают к разбиению выполняемых процессов на сравнительно небольшие части и к их поочередному выполнению. Если две или более транзакции читают или модифицируют разные данные, то это не приводит к возникновению каких-либо проблем. Другое дело, когда доступ осуществляется к одним и тем же данным. Тогда порядок выполнения частей транзакций может играть важную роль, так как от него будет зависеть конечное состояние данных.

В последнем случае говорят о сериализации транзакций, т. е. о составлении такого плана их выполнения (сериального плана), при котором суммарный эффект реализации транзакций эквивалентен эффекту их последовательного выполнения. При параллельном выполнении транзакций возможно возникновение конфликтов (блокировок). При обнаружении таких случаев обычно производится откат одной или нескольких транзакций.

Обеспечение безопасности и секретности. Практически всегда БД представляет собой важный ресурс, который должен быть надежно защищен. Потенциальными опасностями являются:

- похищение и фальсификация данных;
- утрата конфиденциальности;
- утрата целостности;
- потеря доступности;
- непредумышленное и умышленное повреждение данных.

В отношении опасностей могут быть предприняты самые разные контрмеры, начиная от компьютерных систем наблюдения, защиты и восстановления, и заканчивая правовыми и административными процедурами.

Обеспечение безопасности достигается защитой объектов БД и программного кода от модификации, запрещением редактирования по умолчанию, поддержкой блокировок, уровней изолированности транзакций и средств восстановления информации после сбоев (создание контрольных точек, ведение журнала, протоколирование и создание архивных копий).

Обеспечение секретности достигается шифрованием программ и данных, защитой паролем, авторизацией и аутентификацией пользователей, поддержкой различных уровней доступа к БД и отдельным ее элементам (таблицам, формам, отчетам и т.д).

Таким образом, можно выделить основные характеристики, определяющие БД как технологию хранения данных и доступа к ним:

6. Структурированное хранение данных при их минимальной избыточности.

7. Использование метаданных для хранения описания БД.

8. Независимость программ и данных (независимость программ и данных и независимость программ и операций).

9. Наличие встроенных средств обеспечения целостности, безопасности и секретности данных.

10. Поддержка разнообразных способов отображения (представления) и выборки данных.

11. Наличие расширенных средств поиска информации.

Контрольные вопросы и задания

1. Дайте определение БД.

2. Перечислите свойства БД.

3. Опишите пять основных компонентов БД и кратко поясните, для чего они предназначены.

4. Перечислите основные преимущества использования БД перед традиционной файловой системой хранения данных.

5. Нарисуйте архитектуру построения БД ANSI/SPARC. На ее основе поясните понятия логической и физической независимости данных.

6. Сформулируйте основные характеристики, определяющие БД как способ хранения и доступа к данным.

7. С помощью каких средств обеспечивается поддержка целостности данных в БД? Перечислите виды ограничителей целостности.

8. Сформулируйте основное требование контролируемой избыточности данных. Перечислите проблемы, появляющиеся при его нарушении.

2. МОДЕЛИРОВАНИЕ ДАННЫХ

Существует множество моделей, отражающих различные аспекты реального мира: физические, позволяющие понять физические свойства; математические, представляющие собой абстрактное описание мира с помощью математических знаков; экономические, отображающие тенденции экономики и позволяющие получить прогноз ее развития. Самой обобщенной моделью реального мира может служить естественный язык. Однако естественный язык пока малопригоден для компьютерной обработки преимущественно вследствие его многозначности, поэтому нам потребуется построить такую модель, которая бы наиболее адекватно отображала некоторую часть реального мира (предметную область) и была пригодна для компьютерной обработки.

2.1. ПОНЯТИЕ МОДЕЛИ

Модель данных – это средство абстракции, позволяющее отобразить информационное содержание данных. В самом простейшем виде модель данных может быть представлена простым перечнем той информации, которая должна храниться в информационной системе. Но так как данные, представляющие эту информацию, практически всегда структурированы и тесно взаимосвязаны, то необходимо представить каким-либо образом и их структуру. Иначе говоря, модель данных представляет собой совокупность правил, которые могут быть использованы для описания структуры данных. Моделей для отображения одной и той же информации можно придумать множество. Критерием выбора будет максимум количества информации, извлекаемой из данных. Чтобы понять процесс построения модели данных, необходимо знать ряд терминов, которые применяются при описании и представлении данных.

Объектом называется элемент информационной системы, сведения о котором мы сохраняем. Объекты могут быть реальными и абстрактными, а также могут представлять некоторую концепцию или событие. Например, клиент некой фирмы, отделы той же фирмы, записи купли-продажи и т. д. Всякий объект представляется посредством своих свойств и находится в некоторых отношениях с другими объектами. *Классом (типом) объектов* называют совокупность объектов, обладающих одинаковым набором свойств.

Выделенную по некоторым признакам совокупность объектов реального или абстрактного мира, или относящихся к какой-либо области знаний, или необходимых для достижения какой-то цели называют *предметной областью*.

Данные имеют три области своего представления. Первая область (см. рис. 3) – это *реальный мир*. Объекты являются понятиями реального мира, в котором они существуют и имеют некоторые свойства. Вторая – *область информации* (иначе область идей, которые существуют в голове программиста или разработчика), рассматривает атрибуты объектов. *Атрибут* – это информационное отражение свойств объекта. Каждый объект характеризуется рядом основных атрибутов. Например, человек может быть охарактеризован такими атрибутами, как ФИО, год рождения, номер паспорта, адрес и т. д. Третья область – это *область сохраненных данных*. Она может быть бумажным источником или представлять собой память компьютера, в которой используются строки символов или совокупность битов для кодирования элементов информации.

Так как объектов некоторого типа может быть множество, то области представления данных в свою очередь могут быть разбиты на две подобласти. Первая дает общую характеристику объекта как некоторого типа в виде совокупности имен атрибутов (полей). Для нашего примера, объект «человек» будет иметь атрибуты ФИО, год рождения, номер паспорта и т. д. Вторая подобласть состоит из совокупностей значений атрибутов каждого экземпляра объекта в отдельности. Например, Петров (Петров Петр Петрович, 1966, V-НО № 654723); Иванов (Иванов Иван Иванович, 1961, IV-ВЛ № 865278) и т. д. Исходя из этого, дадим определение экземпляра объекта: *экземпляром объекта* называется единичный набор принимаемых элементами данных значений.

Элементы данных (атрибуты) могут быть простыми и составными, однозначными и многозначными, сохраняемыми и вычисляемыми или быть пустыми. *Составные* складываются (или могут быть разделены) из нескольких простых, например адрес. Обычно разложение происходит в иерархическом порядке. Такие составные элементы данных могут использоваться, если ссылки на них возможны только как на целое. Если же возможны ссылки на часть элемента данных, то их лучше разложить на простые. *Многозначными* называют элементы данных, которые могут

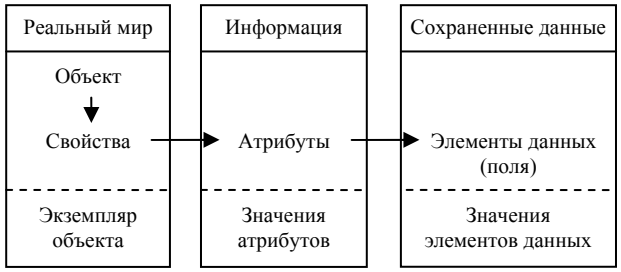


Рис. 3. Три области представления данных

принимать набор значений. Например, телефон и адрес клиента. *Вычисляемыми* (*производными*) называют элементы данных, которые могут быть вычислены на основе уже существующих. Например, возраст человека легко можно вычислить из текущей даты и дня рождения.

Специальное *пустое значение* (*Null*) вводится для элементов данных, которые не имеют значений или могут быть опущены. Различают следующие три случая: мы не знаем, какое значение имеет элемент данных; элемент данных не может иметь значения в данном конкретном случае (например, ученую степень имеет смысл заполнять только для научных работников и преподавателей); мы знаем, что элемент данных имеет значение, но оно для нас безразлично или было пропущено при вводе.

Каждый атрибут связан с набором значений, называемым доменом. Домен определяет все потенциальные значения, которые могут быть присвоены атрибуту. Таким образом, *домен* – это набор значений, который может быть присвоен атрибуту. Например, домен «Имена» содержит все возможные имена людей, а домен «Номера телефонов» – все возможные номера телефонов. Для доменов могут быть приняты соглашения по написанию (формат), к примеру, для имен вначале должна быть фамилия, затем имя и отчество. Домены нельзя путать с атрибутами. Атрибуты представляют собой различные интерпретации домена. Например, атрибуты «Имя клиента», «Имя поставщика» берут свои значения из одного и того же домена «Имена». В принципе, домен это не что иное, как тип данных. Его можно сравнить с таким типом данных в программировании, как перечисление *enum* в С. То есть мы можем задать тип Days и затем им пользоваться: enum Days {Пн., Вт., Ср., Чт., Пт., Сб., Вс.}.

Некоторые элементы данных обладают важным для построения информационной модели свойством. Если известно значение, какое принимает такой элемент данных некоторого объекта, то мы можем однозначно идентифицировать значения, принимаемые другими элементами данных того же объекта. Например, зная номер кузова автомобиля, мы вправе говорить о конкретном автомобиле, описывая его характеристики. Такой элемент данных, по которому можно однозначно идентифицировать остальные элементы данных, называется *ключевым элементом данных*. Ключевой элемент данных может состоять из нескольких элементов данных. Тогда его называют *составным* или *сцепленным*. В общем случае элементов данных, по которым можно однозначно идентифицировать остальные элементы данных, может быть несколько. Тогда один из них выбирают в качестве *первичного ключа*, а остальные будут называться *потенциальными* или *альтернативными ключами*.

Элемент данных является минимальным фрагментов данных. В большинстве случаев он сам по себе информации не несет и приобретает смысл только в связи с другими элементами данных. Совокупность значе-

ний связанных элементов данных называется *записью данных*. Некоторый объект может быть описан набором элементов данных, между которыми существует естественная связь. В дальнейшем такие элементы данных мы будем всегда рассматривать в совокупности и такой вид связи явно выделяться не будет.

Соответствующая модель объектов с составляющими их элементами данных и взаимосвязями называется *концептуальной моделью*. Поскольку такая модель позволяет наглядно отобразить структуру данных, то иначе ее называют *схемой данных*.

2.2. ТИПЫ СВЯЗЕЙ

В общем случае связи подразделяются по количеству связываемых объектов. Можно выделить унарные, бинарные, тернарные и более высоких порядков. Унарная связь (кроме рекурсивной) обычно смысла не имеет. Наибольший интерес представляют бинарные связи, так как связи более высоких порядков редко встречаются, труднореализуемы, нелегки для понимания и всегда могут быть разбиты на совокупность бинарных связей.

Первый тип связи – «один к одному»

(1:1 или 1 – 1) 

Эта связь означает, что одному экземпляру объекта A может соответствовать один и только один экземпляр связанного с ним объекта B. На практике такой вид связи встречается редко, так как почти всегда два связываемых объекта можно объединить в один. В качестве примера использования такой связи может послужить связь между персональной и служебной характеристикой работника некой фирмы. Поскольку доступ к служебной информации обычно ограничен, то защиту от несанкционированного доступа намного легче реализовать при раздельном хранении таких данных.

Второй тип связи – «один ко многим»

(1: M или 1 – ∞) 

Это означает, что одному экземпляру объекта A соответствует несколько (много) экземпляров объекта B. Данный вид связи является наиболее распространенным. Например: сведения о клиентах – записи о покупках; сведения о товаре – записи о наличии его на складе. Один клиент может сделать много покупок, а один и тот же товар может быть завезен многократно.

Третий тип связи – «многие к одному»

(M:1 или ∞ – 1) 

Это означает, что одному или нескольким экземплярам объекта A соответствует один экземпляр объекта B. Связи «один ко многим» и «многие к одному» являются обратимыми, поэтому на практике обычно говорят о связи «один ко многим».

Четвертый тип связи – «многие ко многим»

(M:N или ∞ – ∞) 

Это означает, что одному экземпляру объекта A соответствует несколько (много) экземпляров объекта B, и наоборот, каждому экземпляру объекта B может соответствовать несколько экземпляров объекта A. Например, сведения об авторах и написанных ими книгах или отношение между клиентами и товарами. Книга может быть написана несколькими авторами и, наоборот, один автор может написать много книг. Каждый товар может быть куплен несколькими клиентами и, наоборот, один клиент может купить несколько товаров.

Наряду с взаимосвязями между объектами можно выделить зависимости между атрибутами объектов. Зависимости бывают *функциональные*, *транзитивные* и *многозначные*. Понятие функциональной зависимости является базовым, так как на его основе формулируются определения всех остальных видов зависимостей. Атрибут B функционально зависит от атрибута A, если каждому значению атрибута A соответствует одно и только одно значение атрибута B. То есть если нам известно значение атрибута A (его иначе называют детерминантом), то мы однозначно можем определить значение атрибута B. Однако данному значению атрибута B может соответствовать несколько различных значений атрибута A. Математически функциональная зависимость B от A обозначается записью $A \rightarrow B$. Необходимо отметить, что A и B могут быть составными, т. е. состоять из двух и более атрибутов. Если же между атрибутами A и B существует функциональная зависимость вида $A \rightarrow B$ и $B \rightarrow A$ (т. е. между A и B имеется взаимнооднозначное соответствие), то говорят о функциональной взаимозависимости (обозначается как $A \leftrightarrow B$ или $B \leftrightarrow A$). Наличие функциональной взаимозависимости между атрибутами A и B (например, хранение в БД наряду с идентификационным номером клиента и номера его паспорта) приводит к образованию связи «один к одному» между этими атрибутами.

Функциональная зависимость может быть полной либо частичной. Частичной зависимостью (частичной функциональной зависимостью) называется зависимость атрибута B от части составного атрибута A. Полная функциональная зависимость определяется зависимостью атрибута B от всего составного атрибута A.

Атрибут C зависит от атрибута A транзитивно (существует транзитивная зависимость), если для атрибутов A, B, C выполняются условия

$A \rightarrow B$ и $B \rightarrow C$, но обратная зависимость отсутствует. Например ФИО $\rightarrow$ Должность $\rightarrow$ Оклад.

Между атрибутами может иметь место и многозначная зависимость. Атрибут B многозначно зависит от атрибута A, если каждому значению A соответствует множество значений B. Многозначные зависимости могут образовывать связи «один ко многим», «многие к одному» или «многие ко многим», обозначаемые соответственно $A \Leftarrow B$, $A \Rightarrow B$ и $A \Leftrightarrow B$. Например, имя клиента и его идентификационный номер существуют совместно. Клиентов с одинаковыми именами может быть много, но все они имеют различные идентификационные номера (один ко многим). Или информация о клиентах и обслуживаемых их продавцах хранится совместно. Тогда несколько клиентов с одинаковыми именами могут быть обслужены разными продавцами и наоборот, несколько продавцов с одинаковыми именами могут получить заказы от нескольких клиентов (многие ко многим). Атрибуты могут быть и независимыми. Два или более атрибута будут взаимно независимыми, если ни один из этих атрибутов не является функционально зависимым от других.

Проектирование концептуальной модели можно провести на основе анализа существующих зависимостей между атрибутами. Понятно, что все атрибуты, характеризующиеся полной функциональной зависимостью от ключевого атрибута, конечно, будут атрибутами объекта одного типа. Те атрибуты, которые характеризуются транзитивной зависимостью от ключевого атрибута, необходимо выделить и сформировать отдельные объекты. Многозначные зависимости также требуют дополнительного разбиения по объектам. Практически процесс формирования объектов на основе анализа зависимостей необходимо продолжать до тех пор, пока все неключевые атрибуты будут характеризоваться только полной функциональной зависимостью от соответствующих ключевых атрибутов. На первом же шаге проектирования можно значительно сократить множество анализируемых атрибутов, исключив тривиальные зависимости, т. е. те зависимости, которые не могут не выполняться. Атрибуты, связанные тривиальной зависимостью, необходимо всегда рассматривать вместе. В качестве примера схематично представим концептуальную модель расписания вылета самолетов некоторого аэропорта (рис. 4).

Существует большое количество видов концептуальных моделей, позволяющих отобразить семантику данных. Критериями выбора модели могут служить следующие характеристики:

- выразительность. Модель должна содержать достаточно средств для выражения типов, атрибутов, связей и ограничений;
- формализованность;
- простота и легкость восприятия. Модель должна восприниматься конечными пользователями;

- минимальность. Модель должна обладать минимальным (но достаточным) набором не перекрывающихся по смыслу концепций;
- представительность. Модель должна обладать выразительной диаграммной техникой представления.

Рис. 4. Концептуальная модель расписания вылета самолетов

Всеми этими характеристиками обладает модель сущность-связь, специально разработанная для концептуального моделирования реляционных БД. Большим достоинством ее также является возможность автоматизации процесса проектирования.

2.3. МОДЕЛЬ СУЩНОСТЬ–СВЯЗЬ

Модель сущность-связь (entity-relationship model или ER model) представляет собой высокоуровневую концептуальную модель данных, созданную Питером Ченом (Chen P.) в 1976 г. в целях упрощения задачи проектирования БД.

Основным объектом в ER модели является сущность, или объект (entity), которая представляет собой реальную вещь и может быть реально существующим (физическое существование) или абстрактным (концептуальное существование) объектом. Каждая сущность имеет набор атрибутов, представляющих ее свойства. Сущность имеет имя – тип, который представляет совокупность сущностей с одинаковым набором атрибутов. Таким образом, некоторая сущность определяется ее типом (именем) и набором атрибутов. В свою очередь, некоторый экземпляр сущности будет характеризоваться набором значений атрибутов. Различаются сильные и слабые типы сущностей. *Слабый тип сущности* определяется как тип, существование которого зависит от какого-то другого типа сущности, а *сильный тип сущности* – как тип, существование которого не зависит от всех других типов сущностей. Слабые сущности могут быть дочерними, подчиненными и зависимыми, а сильные – родительскими, владельцами или доминантными.

Между типами сущностей могут существовать связи. В общем случае связи могут быть унарными (рекурсивная), бинарными, тернарными и быть более высокого порядка (степень связи). Как и для сущностей, следует различать связи и типы связей. В свою очередь связи могут также иметь атрибуты. На связи накладываются ограничения по степени участия и по показателю кардинальности. Показатель кардинальности соответствует типу связи в ее обычной формулировке, т. е. 1:1, 1:M, M:1, M:N. Если ER модель составляется для некоторого предприятия (смотри пример на рис. 5), то показатели кардинальности прежде всего будут определяются производственными правилами (бизнес-правилами), установленными на данном предприятии.

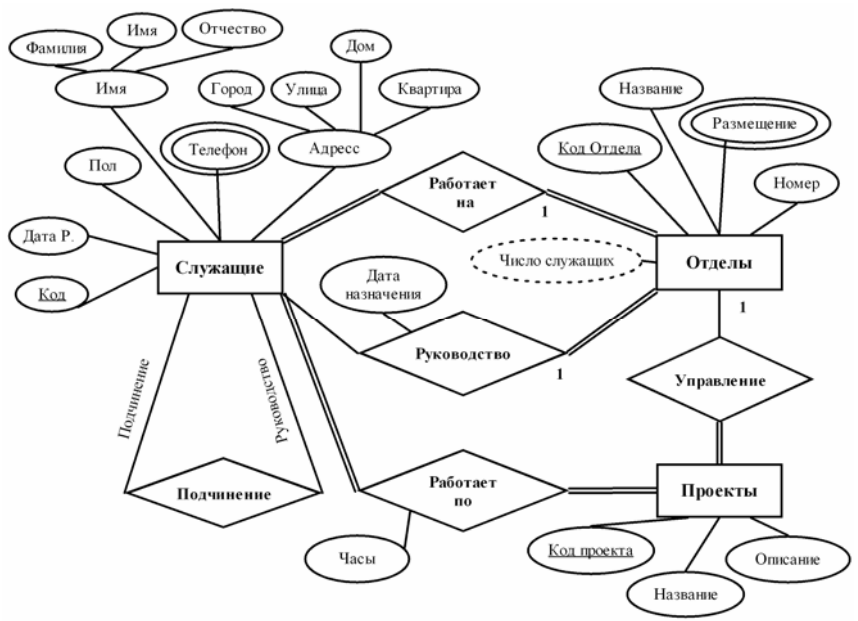


Рис. 5. Пример ER модели

Термин «бизнес-правило» получил широкое распространение, поскольку при моделировании любой предметной области важнейшей задачей является выделение и учет всех без исключения отношений между объектами (если, конечно, этого позволяет сама модель). Связям могут также присваиваться ролевые имена для однозначного определения назначения каждой связи. Если назначение каждой связи определено недвусмысленно, то ролевые имена не указываются. Степень участия может быть полной и частичной. Полной она будет в том случае, если для

существования объекта, участвующего в связи, требуется существование другого объекта (называется иначе зависимостью существования). Например, если правила предприятия требуют, чтобы любой служащий работал в некотором отделе, то экземпляр сущности «Служащий» будет существовать, только если он участвует в связи «Работает на» с сущностью «Отдел». В то же время только некоторые служащие могут руководить отделами. Значит, участие сущности «Служащий» в связи «Руководство» будет только частичным.

Допустимо использование и альтернативного варианта обозначения структурных ограничений (см. рис. 6), накладываемых на связь в виде

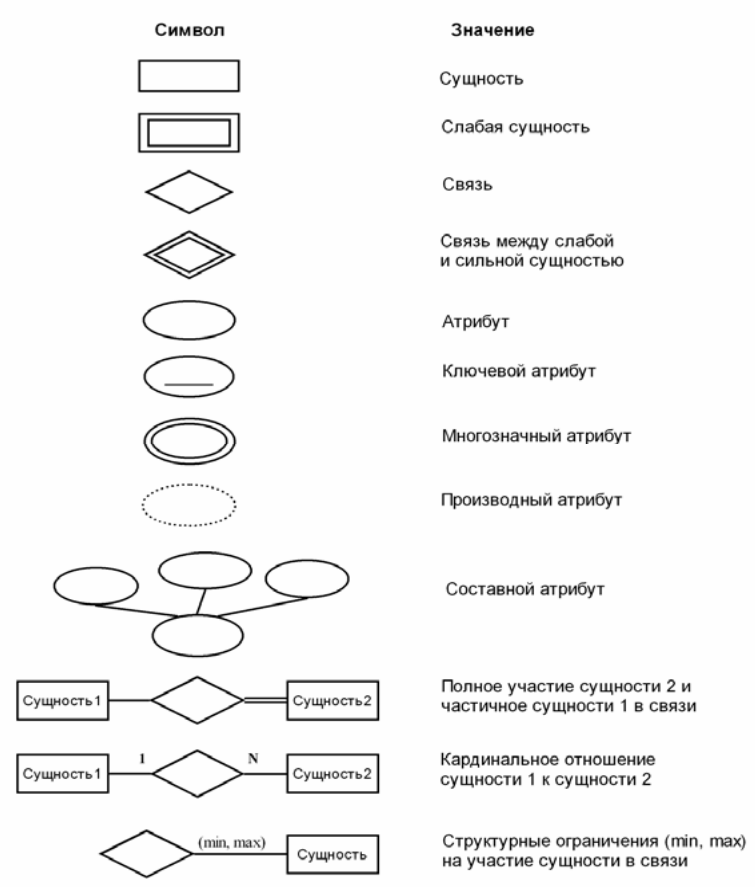


Рис. 5 Система обозначений, принятая в ER модели

двух чисел (min, max), означающих минимальное и максимальное количество экземпляров сущности ($0 \le min \le max$ и $max \ge 1$), участвующей в связи. Такой вид обозначения позволяет отобразить больше информации о степени участия: min = 0 означает частичное участие, min > 0 – полное, а max = N указывает на отсутствие ограничений со стороны максимума. Например, надпись (0, 1) над связью «Руководство» означает, что любой служащий не обязательно должен руководить некоторым отделом, да и руководить одновременно двумя отделами он тоже не может.

Достоинством этой модели является наличие удобной и выразительной диаграммной техники построения модели (см. рис. 6). На диаграмме каждый тип объекта показан в виде отдельного прямоугольника с его именем внутри, причем слабые типы объектов изображаются в двойной рамке. Атрибуты показаны в виде эллипсов с названием атрибута, соединенных сплошной линией с соответствующим объектом или типом связи. Эллипс обведен штриховой линией, если атрибут производный, и двойной линией, если атрибут многозначный. Если атрибут составной, то составляющие его атрибуты показаны в виде других эллипсов, соединенных с эллипсом составного атрибута с помощью дополнительных линий. Ключевые атрибуты, как правило, подчеркиваются, а множества значений не показываются вовсе.

Каждый тип связи показан в виде ромба с названием связи внутри. Ромб окружен двойной линией, если связь задана между слабым типом объекта и типом объекта, от существования которого зависит слабый тип объекта. Участники каждой связи присоединены к соответствующему типу связи сплошными линиями. Каждая такая линия содержит надпись «1», «M» или «N» для обозначения показателя кардинальности. В большинстве случаев указывается только показатель «1». Тогда отсутствие показателя кардинальности с обеих сторон типа связи означает наличие связи «многие ко многим» между типами сущностей. Двойная линия обозначает полное участие. Типы связей, аналогично типам объектов, могут иметь атрибуты.

К недостаткам этой модели можно отнести возможность появления «ловушек» при недостаточном определении связей между объектами. К таким ловушкам можно отнести ловушки разветвления и разрыва.

Ловушка разветвления имеет место в том случае, когда модель отображает связи между типами сущностей, но путь между отдельными сущностями этих типов указан не однозначно. Например, ловушка разветвления может возникать, когда две и более связей «один ко многим» разветвляются из одной сущности, как показано на рис. 7. В данном примере схема данных, представленная слева, не позволяет однозначно ответить на вопрос, к каким группам относятся студенты. Устранить

такую ловушку можно с помощью реорганизации связей между типами сущностей – так, как показано на схеме справа.

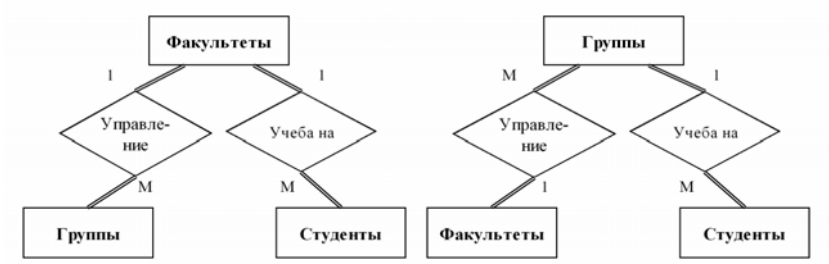


Рис. 7. Пример возникновения ловушки разветвления

Ловушка разрыва появляется в том случае, когда в модели предполагается наличие связи между типами сущностей, но не существует пути между отдельными сущностями этих типов. Ловушка разрыва может возникать между тремя и более сущностями, связанными между собой отношениями «один ко многим» при наличии связи с частичным участием. Пример представлен на рис. 8.

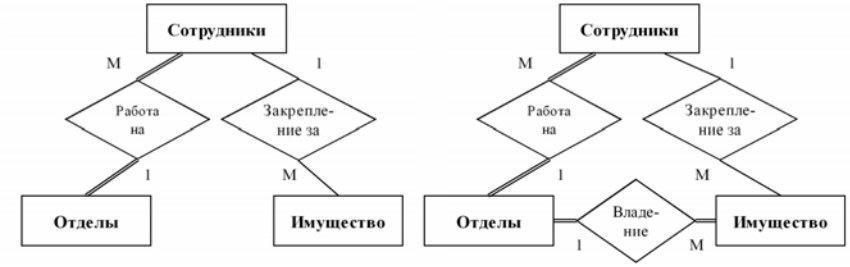


Рис. 8. Пример возникновения ловушки разрыва

Из схемы, представленной слева, следует, что в отделе работает несколько сотрудников и за ними закреплено определенное имущество. Но не за каждым сотрудником должно быть закреплено какое-то имущество и не все имущество обязательно должно быть закреплено за сотрудниками в данный момент времени. Такая схема не позволяет ответить на вопрос, какое имущество (закрепленное или не закрепленное) числится за отделом. Такая ловушка устраняется введением дополнительной связи между типами сущностей, как показано на схеме справа.

Контрольные вопросы и задания

1. Дайте определение модели данных.
2. Отобразите схематично три области представления данных и дайте определения базовым понятиям моделирования данных: объекту, предметной области, атрибуту данных, домену и записи данных.
3. Перечислите виды атрибутов данных.
4. Что такое ключевой элемент данных?
5. Дайте определение концептуальной модели данных.
6. Перечислите типы связей и дайте их определение.
7. Перечислите виды зависимостей между атрибутами объекта.
8. Приведите пример частичной и полной функциональной зависимости.
9. Приведите пример транзитивной зависимости.
10. Дайте определение ER модели.
11. Назовите основные достоинства и недостатки ER модели.
12. Что такое слабый тип сущности?
13. Какие ограничения накладываются на связи между типами сущностей?
14. Разработайте ER модель расписания вылета самолетов, представленной на рис. 4.
15. Что такое ловушки разрыва и разветвления и когда они возникают? Приведите пример для каждой из ловушек.

3. МОДЕЛИ ДАННЫХ

Хранимые в базе данные имеют определенную логическую структуру, т. е. описываются некоторой моделью представления данных (в дальнейшем просто моделью данных). К числу классических относятся иерархическая, сетевая и реляционная. В последние годы появились и стали внедряться на практике постреляционная, многомерная и объектно-ориентированная модели. Разрабатываются также всевозможные системы, основанные на моделях данных, расширяющих известные модели, например объектно-реляционная.

3.1. ИЕРАРХИЧЕСКАЯ МОДЕЛЬ

Исторически первой была разработана и воплощена иерархическая модель данных. В 1960 г. была разработана БД для большой ЭВМ IBM System 360. Первой коммерческой СУБД стала СУБД IMS фирмы IBM. Среди отечественных можно назвать СУБД ОКА.

Иерархическая модель данных представляется связным графом типа дерева, вершины (типы) которого расположены на разных иерархических уровнях (рис. 9). При этом одна вершина, расположенная на самом верху дерева, называется корнем и не подчиняется ни одной вершине, а все остальные связаны с одной и только одной вершиной, расположенной на более высоком уровне. Элементы, расположенные в конце ветви, т. е. не имеющие порожденных, называются листьями. Потомков одного и того же типа называют близнецами. Элементы могут быть простыми, а также представляют собой записи.

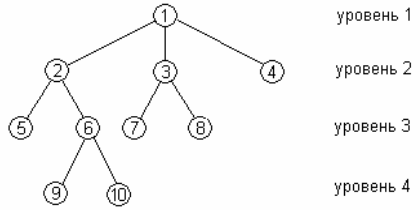


Рис. 6. Иерархическая модель данных

Иерархическая БД представляет собой упорядоченную совокупность экземпляров типа «дерево», содержащих в свою очередь совокупность экземпляров типа «запись». Обход всех элементов иерархической БД обычно производится сверху вниз и слева направо. Иерархической модели присущи связи «один к одному» и «один ко многим». Связь «многие ко многим» не может быть реализована, так как вершина не может иметь более одного родителя. Представим расписание занятий на факультете с помощью иерархической модели (рис. 10). Видим, что иерархической модели данных присуще дублирование информации. В нашем примере для каждой группы будут дублироваться записи расписания. При измене-

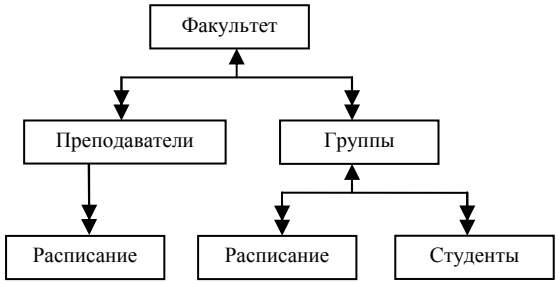


Рис. 7. Иерархическая модель расписания занятий

нии расписания общих курсов придется менять записи расписания для всех групп. К тому же отсутствует связь расписания с преподавателями и преподавателей со студентами.

К достоинствам иерархической модели данных относятся эффективность использования памяти ЭВМ и неплохие показатели времени выполнения основных операций с данными. К недостаткам можно отнести:

- невозможность установления связи M:M;
- сильную логическую зависимость данных. Добавление или удаление вершины невозможно без разрушения всей структуры данных;
- избыточность информации;
- затрудненный поиск «снизу вверх». Это приводит к простому перебору всех значений в БД.

3.2. СЕТЕВАЯ МОДЕЛЬ ДАННЫХ

Впервые сетевые СУБД появились в 1970 г. с появлением миникомпьютеров DEC. Это IDMS (Computer Association), VAX_DBMS (Digital), db_VistaIII и наша СЕТЬ.

Если в отношении между данными порожденный элемент имеет более одного исходного элемента, то такое отношение уже нельзя описать как древовидное. Его описывают в виде сетевой структуры. Сетевая модель данных позволяет отображать взаимосвязи элементов данных в виде произвольного графа, обобщая тем самым иерархическую модель данных (рис. 11). В сетевой модели данные представлены в виде записей и связей. Запись может иметь множество как подчиненных ей записей, так и записей, которым она подчинена. Такая сеть позволяет реализовывать отношение M:M на всех связях. С некоторой избыточностью сетевые модели всегда можно разложить на несколько древовидных, вводя дополнительные вершины. Рассмотрев обратный переход к сетевой модели, легко

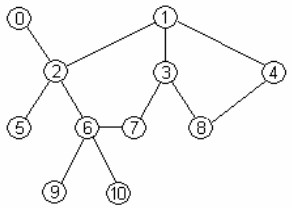


Рис. 9. Сетевая модель данных

заметить, что дублирование информации сильно уменьшается. К другим достоинствам этой модели данных можно отнести большую гибкость системы. В ней можно использовать не только предопределенные связи, но и добавлять новые (наряду с узлами) в процессе работы. Разрешены также симметричные запросы (сверху вниз и снизу вверх), выполняющиеся по схожим алгоритмам. Сетевые модели получили достаточно широкое распространение. Результатом явился стандарт CODASYL. Основными языками программирования для создания сетевых, а также иерархических БД можно считать PL/1 и COBOL.

К недостаткам сетевой модели данных можно отнести:

- недостаточную скорость выполнения поиска;
- некоторую избыточность информации (для каждой записи дублируются ссылки на подчиненные и родительские элементы;
- громоздкое и труднообозримое представление данных;
- ослабление контроля целостности вследствие допустимости установления произвольных связей между записями.

На рис. 12 представлена сетевая модель расписания занятий. Здесь связь типа M:M между группами и преподавателями реализована в виде записей пересечения «расписание», указывающих день, время и номер аудитории. Между студентами и преподавателями также можно установить связь типа M:M с помощью записей пересечения «задание», например индивидуальные задания, курсовые работы и т. п.

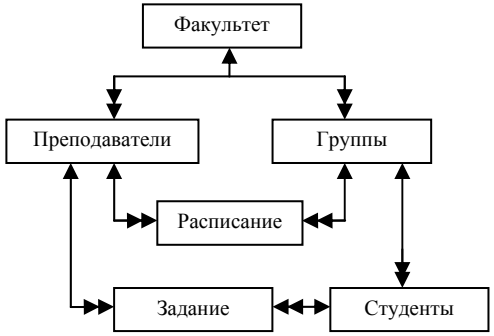


Рис. 8. Сетевая модель расписания занятий

3.3. РЕЛЯЦИОННАЯ МОДЕЛЬ

3.3.1. Основные понятия

В реляционной модели данных информация хранится в одной или нескольких связанных таблицах. Отдельная таблица обычно представляет совокупность (группу) либо реальных объектов, либо некоторых абстрактных концепций, либо событий одного типа. Каждая запись в таблице идентифицирует один объект группы. Таблица состоит из строк и столбцов, называемых соответственно записями и полями. Таблицы обладают следующими свойствами.

1. Каждый ячейка таблицы представляет собой один элемент данных, т. е. совокупность значений в одном столбце одной строки недопустима.

2. Все столбцы в таблице однородные. Это означает, что элементы столбца имеют одинаковую природу. Столбцам присвоены имена.

3. В таблице нет двух одинаковых строк.

4. Порядок размещения строк и столбцов в таблице может быть произвольным. В операциях с такой таблицей ее строки и столбцы могут просматриваться в любом порядке безотносительно к их информационному содержанию и смыслу.

Таблицы, обладающие такими свойствами, являются точным прообразом математического двухмерного множества – отношения (relation). Но эти два понятия не эквивалентны. Отношение – это абстрактный математический объект, а таблица – конкретное изображение этого абстрактного объекта. Различие проявляется в их свойствах. В отношении строки и столбцы не могут быть упорядочены, а в таблице строки упорядочены сверху вниз, столбцы – слева направо. В таблице могут повторяться строки, а в отношении нет. В реляционной модели каждая строка таблицы уникальна. Это обеспечивается применением ключей, которые содержат одно или несколько полей таблицы. Ключи хранятся в упорядоченном виде, обеспечивающем прямой доступ к записям таблицы во время поиска. Связь между таблицами осуществляется посредством значений одного или нескольких совпадающих полей (преимущественно ключевых).

Приведем ряд терминов, применяющихся в реляционной модели.

• ***Отношением*** (relation) называется двухмерное множество – таблица, удовлетворяющая вышеперечисленным требованиям.

• ***Атрибут*** – это свойство, характеризующее объект. В структуре таблицы каждый атрибут именуется и ему соответствует заголовок некоторого столбца таблицы. Количество атрибутов называется ***степенью отношения.***

• ***Кортежем*** (tuple) называется строка таблицы. В общем случае кортежи представляют собой набор пар атрибут, значение. Каждое значение должно быть атомарным, т. е. оно не может быть многозначным или

составным. Следовательно, многозначные и составные атрибуты в реляционной модели не поддерживаются. Количество кортежей называется ***кардинальным числом.***

• ***Домен*** представляет собой множество всех возможных значений определенного атрибута отношения.

• ***Первичным ключом*** называется атрибут отношения, однозначно идентифицирующий каждый из его кортежей. Ключ может быть составным (сложным), т. е. состоять из нескольких атрибутов.

• ***Потенциальный ключ*** – это подмножество атрибутов отношения, обладающее следующими свойствами:

– свойством уникальности. Нет одинаковых кортежей с теми же значениями потенциальных ключей;

– свойством неизбыточности. Никакое из подмножеств потенциального ключа не обладает свойством уникальности.

Каждое отношение обязательно имеет комбинацию атрибутов, которая может служить ключом. Его существование гарантируется тем, что отношение – это математическое множество, которое не может содержать одинаковых кортежей, т. е. по крайней мере вся совокупность атрибутов обладает свойством однозначной идентификации кортежей отношения. Возможны случаи, когда отношение имеет несколько комбинаций атрибутов, каждая из которых однозначно определяет кортежи отношения. Эти комбинации атрибутов являются потенциальными или возможными ключами отношения. Один потенциальный ключ выбирается в качестве первичного, остальные будут называться вторичными (альтернативными). Потенциальные ключи имеют очень большое значение в реляционной теории. Они служат для адресации кортежей. Указав значение потенциального ключа, мы гарантированно получим не более одного кортежа. Для отношений, участвующих в связи «многие ко одному» с другим отношением, существуют еще внешние ключи, использующиеся для установления связи.

• ***Внешний ключ*** – это атрибут подчиненного отношения, который используется для установления связи с базовым отношением. Он содержит значения, всегда совпадающие со значениями первичного ключа базового отношения.

Исходя их вышеприведенных понятий, математически отношение можно описать следующим образом. Пусть даны n множеств $D1, D2, D3, ..., Dn$, тогда отношение R есть множество упорядоченных кортежей $d1$, $d2$, $d3, ..., dn$, где $dk \in Dk$, dk – атрибут, а Dk – домен отношения R.

В середине 1970-х годов инженер IBM Е.Ф. Коддом (Codd E.F.) предложил модель данных, основанную на математических операциях исчисления отношений и реляционной алгебре. Основной структурной едини-

цей этой модели являлось отношение (relation). Поэтому такая модель данных получила название реляционной. Кодд также разработал язык манипулирования данными, представленными в виде отношений. Он предложил два эквивалентных между собой по своим выразительным возможностям варианта языка манипулирования данными.

1. Реляционная алгебра. Это процедурный язык, так как отношение, являющееся результатом запроса к реляционной БД, вычисляется при выполнении последовательности реляционных операторов, применяемых к отношениям. Операторы состоят из операндов, в роли которых выступают отношения, и реляционных операций. Результатом реляционной операции является отношение. Операции можно разделить на две группы. Первую группу составляют операции над множествами, к которым относятся операции объединения, пересечения, разности, деления и декартова произведения. Вторую группу составляют специальные операции над отношениями: проекция, выборка и соединение.

2. Реляционное исчисление. Это непроцедурный язык описательного или декларативного характера, содержащий лишь информацию о желаемом результате. Процесс получения этого результата скрыт от пользователя. К языкам такого типа относятся SQL и QBE. Первый основан на реляционном исчислении кортежей, второй – на реляционном исчислении доменов.

С помощью этих языков можно извлекать подмножество столбцов и строк таблицы, создавая таблицы меньшей размерности, а также объединять связанные данные из нескольких таблиц, создавая при этом таблицы большей размерности. Результат каждой (реляционной) операции над отношениями также является отношением. Это реляционное свойство получило название ***свойства замкнутости***. Следовательно, различные пользователи могут выделять в реляционной БД различные наборы данных и связей между ними. Этот способ представления данных наиболее естественен и обозрим для конечного пользователя. Реляционная модель данных очень гибкая, поскольку любое представление данных с некоторой избыточностью можно свести к двухмерным таблицам.

3.3.2. Связи в реляционной БД

Отношение между объектами определяет тип связи между таблицами. Поддерживаются связи четырех типов: «один к одному», «один ко многим», «многие к одному» и «многие ко многим». Рассмотрим подробнее типы связей в применении к реляционной модели данных.

«Один к одному». Связь «один к одному» означает, кто каждой записи из первой таблицы соответствует одна и только одна запись из другой таблицы. Рассмотрим таблицы, содержащие персональные и служебные сведения о работниках некоторой фирмы (ключевые поля выделены).

Персональные сведения

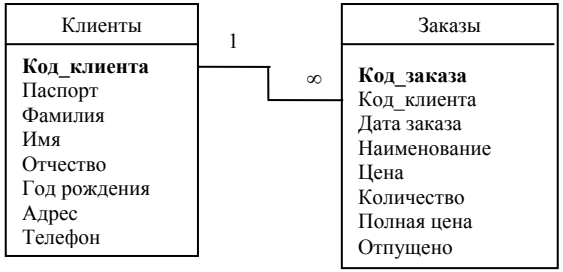
Код_пс	Паспорт	Фамилия	Имя
1	АВ 2358955	Сидоров	Сергей
2	DB 2456886	Петров	Петр
3	МА 8654212	Иванов	Иван

Служебные сведения

Код_сс	Место работы	Должность	Разряд
1	Бухгалтерия	Бухгалтер	3
2	Цех № 2	Токарь	7
3	Лаборатория	Начальник отдела	

Между таблицами «Персональные сведения» и «Служебные сведения» существует связь «один к одному», поскольку для одного человека, работающего в определенной фирме, может существовать только одна запись о его служебном положении. Табельные номера «Код_пс» и «Код_сс» были добавлены для однозначной идентификации записей. Эти же поля и приняты в качестве первичных ключей. Связь между этими таблицами поддерживается при помощи совпадающих значений полей «Код_пс» и «Код_сс». Легко убедиться, что между двумя ключевыми полями может существовать только связь «один к одному», поскольку любые дублирования одного и того же табельного номера исключены с обеих сторон.

«Один ко многим» и «многие к одному». Связь «один ко многим» означает, кто каждой записи из первой таблицы может соответствовать одна либо много записей из другой таблицы. Связи «один ко многим» и «многие к одному» являются обратимыми, поэтому подробнее остановимся на связи «один ко многим». Рассмотрим таблицы, содержащие сведения о клиентах некоторой фирмы и сделанных ими заказах. Предполагается, что один и тот же клиент может сделать несколько заказов. Таким образом, между этими таблицами существует связь «один ко многим».



Клиенты

Код_клиента	Паспорт	Фамилия	Имя
1	АВ 2358955	Сидоров	Сергей
2	СВ 2456886	Петров	Петр
3	МА 8654212	Иванов	Иван

Заказы

Код_заказа	Код_клиента	Дата заказа	Наименование
1	1	05.03.2007	Пылесос «Вихрь 150»
2	2	15.04.2007	Холодильник «Минск 131»
3	2	21.04.2007	Телевизор «Горизонт ТЦ 655»
4	1	02.10.2007	Стол компьютерный
5	3	12.12.2007	Кресло «Президент»

Для установления связи необходимо в таблицу «Заказы» ввести поле «Код_клиента», которое будет являться внешним ключом для таблицы «Заказы». Связь между таблицами будет осуществляться на основании значений полей «Клиенты.Код_клиента» и «Заказы.Код_клиента». Причем подчеркнем, что связь устанавливается на основе значений совпадающих полей, а не их наименований. Таким образом, если связь устанавливается между ключевым полем одной таблицы и неключевым полем второй таблицы, то это будет связь типа «один ко многим».

«Многие ко многим». Связь «многие ко многим» возникает между двумя таблицами в тех случаях, когда каждой записи из первой таблицы может соответствовать одна либо много записей из второй таблицы и наоборот, одной записи из второй таблицы может соответствовать одна либо много записей из первой таблицы.

Типичным примером является связь между клиентами некой фирмы и покупаемыми ими товарами. Каждый товар может быть куплен несколькими клиентами и наоборот, один клиент может купить несколько товаров. Такая связь может быть реализована с помощью дополнительной таблицы, содержащей ключевые поля обеих таблиц, участвующих в связи (являющимися для нее внешними ключами).

Клиенты

Код_клиента	Паспорт	Фамилия	Имя
1	АВ 2358955	Сидоров	Сергей
2	СВ 2456886	Петров	Петр
3	МА 8654212	Иванов	Иван

Товары

Код_товара	Наименование	Цена
1236	Пылесос «Вихрь 150»	88 999
5543	Холодильник «Минск 131»	310 056
445	Телевизор «Горизонт ТЦ 655»	260 324
3245	Электрочайник BOSCH	50 346

Заказы

Код_заказа	Код_клиента	Код_товара	Дата заказа
1	1	1236	05.03.2007
2	2	5543	15.04.2007
3	2	445	21.04.2007
4	1	3245	02.10.2007
5	3	1236	12.12.2007
6	3	445	30.12.2007

Связь «многие ко многим» также автоматически возникает между таблицами, связанными посредством неключевых полей. Пусть первая таблица содержит информацию о том, на каких станках могут работать рабочие некоторой бригады. Вторая таблица содержит сведения о том, кто из бригады ремонтников какие станки обслуживает.

Работа

Работает	Станок
Иванов А. В.	Станок 1
Иванов А. В.	Станок 2
Петров Н. Г.	Станок 1
Петров Н. Г.	Станок 3
Сидоров В. К.	Станок 2

Обслуживание

Обслуживает	Станок
Голубев Б. С.	Станок 1
Голубев Б. С.	Станок 2
Зыков А. Ф.	Станок 2
Зыков А. Ф.	Станок 3

Первой и третьей записям таблицы «Работа» соответствует первая запись таблицы «Обслуживание». Четвертой записи таблицы «Работа» соответствуют вторая и четвертая записи таблицы «Обслуживание». Такой вид связи «многие ко многим» характеризуется как слабый вид связи или даже как отсутствие связи, поскольку никакого контроля за целостностью данных в этом случае не осуществляется.

Информация, размещенная в связанных таблицах, может быть легко объединена с помощью естественного соединения (по значениям внешних ключей). Для нашего примера будем иметь следующее.

Заказы

Фамилия	Имя	Наименование	Цена	Дата заказа
Сидоров	Сергей	Пылесос «Вихрь 150»	88 999	05.03.2007
Петров	Петр	Холодильник «Минск 131»	310 056	15.04.2007
Петров	Петр	Телевизор «Горизонт ТЦ 655»	260 324	21.04.2007
Сидоров	Сергей	Электрочайник «BOSCH»	50 346	02.10.2007
Иванов	Иван	Пылесос «Вихрь 150»	88 999	12.12.2007
Иванов	Иван	Телевизор «Горизонт ТЦ 655»	260 324	30.12.2007

Кроме вышеупомянутых связей, в реляционной модели возможны также рекурсивные связи. Предположим, мы должны хранить информацию о сотрудниках некоторой компании с указанием отношений подчиненности между отдельными сотрудниками. Поскольку один экземпляр объекта «Сотрудники» ссылается на другой экземпляр того же объекта, который в свою очередь может ссылаться на третий экземпляр, то связь будет унарной и рекурсивной. Данная связь легко реализуется путем введения внешнего ключа, ссылающегося на первичный ключ той же таблицы.

3.3.3. Целостность в реляционной БД

Рассмотрим подробнее ограничители целостности в применении к реляционной модели данных. Понятие ссылочной целостности неразрывно связано с понятием внешних ключей и формулируется для реляционной БД следующим образом: реляционная БД не должна содержать несогласованных значений внешних ключей, т. е. таких значений, для которых не существует отвечающих им значений соответствующих потенциальных ключей базовых таблиц. Иначе говоря, внешние ключи могут содержать только те значения, которые принимают соответствующие им ключевые (в общем случае потенциальные) поля базовых таблиц. Если в БД уже есть клиент с номером 10 и товар с номером 25, то только тогда можно внести запись о покупке клиентом 10 товара 25 в таблицу «Заказы». С другой стороны, нельзя удалить клиента с номером 25, или даже изменить его номер, если в БД хранится хотя бы одна запись о сделанном им заказе. В противном случае согласованность внешних ключей тоже нарушается,

так как запись в таблице «Заказы» будет содержать значение, указывающее на несуществующего клиента. Заметим, что внешний ключ также может содержать значения первичного ключа своего же отношения для обеспечения рекурсивной связи. Ограничители ссылочной целостности могут быть схематично отображены с помощью линий со стрелками, направленных от внешних ключей подчиненных отношений к потенциальным ключам базовых отношений.

Есть два правила внешних ключей, обеспечивающих согласованное состояние БД. Первое заключается в ограничении обновления и удаления. При работе с базовой таблицей это правило запрещает удаление записей, которые содержат связанные записи в подчиненных таблицах, и изменение значений их первичных ключей. При работе с подчиненной таблицей это правило запрещает установку внешнего ключа в значение, которое не существует для соответствующего потенциального ключа базовой таблицы. Второе из них заключается в каскадном обновлении внешних ключей и каскадном удалении записей подчиненных таблиц при операциях с базовой таблицей. При изменении значения базового ключа, использующегося в связи, значения внешних ключей подчиненных таблиц должны быть изменены соответственно (каскадное обновление). При удалении записи из базовой таблицы все связанные с ней записи из подчиненных таблиц должны быть также удалены (каскадное удаление).

Кроме ссылочной целостности можно сформулировать еще три ограничителя целостности для реляционной БД:

- ограничители домена (domain constraints), запрещающие ввод значений атрибутов, не принадлежащих домену. К ним можно отнести также все ограничители значений (тип, формат, задание списка и диапазона значений);
- ограничители ключей (key constraints), обеспечивающие уникальность значений потенциальных ключей (обеспечивается применением уникальных индексов);
- ограничители записи (entity constraints), запрещающие *Null* значения для первичных ключей, так как в противном случае будет нарушено требование идентифицируемости записи.

3.4. НОРМАЛИЗАЦИЯ ДАННЫХ В РЕЛЯЦИОННОЙ МОДЕЛИ

Один и тот же набор данных в реляционной модели можно представить различными способами. Процесс нормализации данных позволяет решить вопрос о наиболее эффективной их структуре, обладающей минимальной избыточностью. Это позволяет решить следующие задачи:

а) исключить ненужное повторение данных,

б) обеспечить быстрый доступ к данным,

в) обеспечить целостность данных.

Обычно говорят только об уменьшении избыточности данных и нормализацией называют процесс уменьшения избыточности данных в реляционной БД.

Теория нормализации оперирует с пятью нормальными формами и одной промежуточной, уточняющей третью. На практике обычно руководствуются только первыми тремя нормальными формами. Процесс проектирования БД с использованием нормальных форм является итерационным и заключается в последовательном применении правил нормальных форм от низшей к высшей. Каждая следующая нормальная форма уменьшает избыточность данных и сохраняет свойства всех предыдущих форм. Перевод отношения в следующую нормальную форму осуществляется методом «декомпозиции без потерь», т. е. разбиением исходной таблицы на несколько связанных. Такая декомпозиция должна обеспечить равенство результатов выборок из исходного отношения и выборок, основанных на совокупности полученных отношений, т. е. говоря математическим языком, должно соблюдаться следующее правило: $\bigcup_{i=1}^{m} R_i = R$, где $R = \{A_1, A_2, \ldots, A_n\}$ – исходное отношение, а $D = \{R_1, R_2, \ldots, R_m\}$ – декомпозиция, т. е. множество нормализованных отношений. При декомпозиции следует не только опасаться потери информации, но и избегать появления так называемых подложных записей (например, такие записи могут появляться при объединении таблиц посредством неключевых полей). И наконец, при установлении зависимостей между таблицами следует придерживаться связей, существующих между реальными объектами.

Рассмотрим ненормализованную таблицу «Продажи», которая будет содержать сведения о покупателях, сведения о проданных товарах и оформленных заказах:

Продажи

№	Наименование	№	Наименование
1	Фамилия	9	Дата_заказа
2	Имя	10	Наим_товара
3	Отчество	11	Категория
4	Паспорт	12	Размещение
5	Телефон	13	Цена
6	Адрес	14	Количество
7	Кредит	15	Полная_цена
8	Примечание	16	Прим_заказа

Таблицу «Продажи» можно рассматривать как однотабличную БД. Основная проблема состоит в том, что в ней содержится значительное

количество повторяющейся информации. Например, сведения о каждом покупателе повторяются для каждого сделанного им заказа. Наличие повторяющейся информации ведет к возможной потере согласованности данных, т. е. к появлению аномалий обновления, неоправданному увеличению размера БД, и, как следствие, к снижению скорости поиска и выполнения запросов. Аномалии обновления можно условно разбить на три вида:

а) аномалии добавления. Возникают в случаях, когда информацию в таблицу нельзя поместить до тех пор, пока она не полная. Например, мы хотим добавить информацию о пришедшем на склад новом товаре, у которого пока еще нет покупателей. Нам придется оставить незаполненными все поля, касающиеся клиентов и оформления заказа. Но если некоторые поля были объявлены обязательными, то либо в них придется ввести что-нибудь, либо вообще отказаться от записи. Кроме того, после первой же покупки этого товара необходимость в такой записи вообще отпадет;

б) аномалии удаления. Состоят в том, что при удалении некоторой информации может пропасть и другая информация, не связанная напрямую с удаляемой. Например, при удалении последней записи о продаже некоторого товара сведения о нем будут утеряны, хотя это никак не будет означать, что данного товара больше нет на складе;

в) аномалии модификации. Проявляются в том, что изменение значения некоторого поля в одной записи может привести к просмотру всей таблицы и соответствующему изменению других ее записей. Например, такая аномалия возникнет при обновлении не всех записей о покупках какого-либо клиента при смене его адреса. Причем простая автозамена здесь поможет не во всех случаях, так как адрес может быть записан разным образом.

3.4.1. Первая нормальная форма

Первая нормальная форма требует соответствия исходной таблицы требованиям, предъявляемым к отношениям, т. е.:

1) таблица не должна иметь повторяющихся записей;

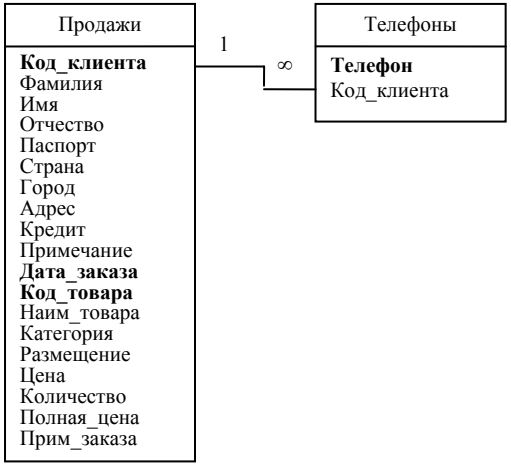
2) все атрибуты должны быть простыми (скалярными).

Первое требование легко решается введением ключевого поля, однозначно определяющего остальные поля таблицы. Разумеется, можно использовать одно из полей, имеющихся в таблице. К примеру, номер паспорта. Но паспорт легко сменить, сменив фамилию. В таком случае рекомендуется ввести так называемый табельный номер, основным назначением которого будет идентификация некоторой записи. Если этот номер не имеет практического смысла, то его можно скрыть от пользователей. Ввод табельного номера рекомендуется если:

- ключевой атрибут не является уникальным (например, фамилия);
- естественный атрибут может меняться со временем;
- данный атрибут участвует во многих связях. Так как связь осуществляется посредством ключевых полей, то во всех подчиненных таблицах значение ключевого поля придется повторить. При применении не числовых и достаточно громоздких полей скорость выполнения операций доступа и поиска будет существенно падать. Лучшим решением является применение целочисленных значений.

Второе требование постулирует, чтобы в каждой ячейке было представлено только одно значение, а не массив или перечисление. Составные поля должны быть разложены на простые, а многозначные – вынесены в отдельные таблицы или разложены по отдельным строкам (с дублированием значений всех остальных полей). В нашем случае поле «Адрес» должно быть разложено на поля «Страна», «Город» и «Адрес», чтобы сохранить возможность поиска и сортировки по данным полям. Поле «Телефон» является многозначным и, следовательно, должно быть выделено в отдельную таблицу.

После ввода составного ключа «Код_клиента», «Код_товара» и «Дата_заказа» таблица «Продажи» будет находиться в первой нормальной форме. Получим:



3.4.2. Вторая нормальная форма

Таблица находится во второй нормальной форме, если:

1) она удовлетворяет условиям первой нормальной формы;

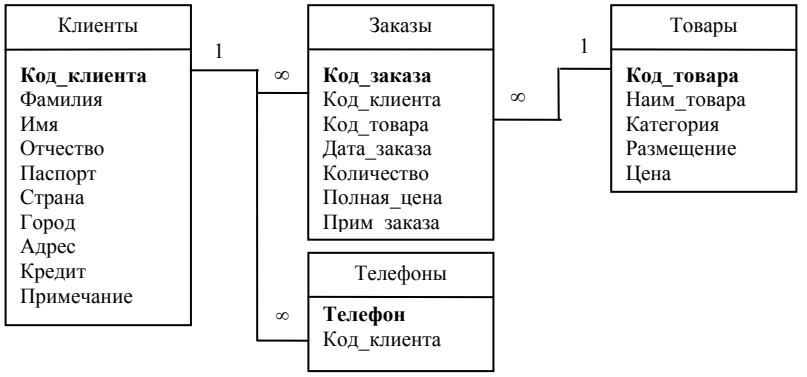
2) любое поле, не входящее в ключ, должно однозначно идентифицироваться значением первичного ключа. Если первичный ключ является

составным, то остальные поля должны зависеть от его полного выражения, а не от части (неключевые атрибуты должны характеризоваться полной функциональной зависимостью от первичного ключа).

Легко заметить, что атрибуты «Фамилия», «Имя», «Отчество», «Паспорт», «Страна», «Город», «Адрес», «Предприятие», «Руководитель», «Кредит» и «Примечание» зависят только от части составного ключа – поля «Код_клиента», а атрибуты «Наим_товара», «Категория» и «Цена» зависят только от поля «Код_товара». Следовательно, они должны быть вынесены в две отдельные таблицы.

Иначе говоря, второе требование также постулирует устранение повторяющейся информации для группы полей. Такие повторяющиеся группы полей обычно соответствуют некоторым реальным объектам, информацию о которых мы сохраняем. Следовательно, таблица должна содержать данные, относящиеся только к одному объекту. Поскольку здесь явно выделяются объекты «Покупатели», «Товары» и «Заказы», нам необходимы три таблицы. Записи первой таблицы будут содержать сведения о покупателях, другой – информацию о товарах, третьей – информацию о каждом из заказов. Вначале разобьем таблицу «Продажи» на две отдельные таблицы («Клиенты» и «Заказы») и определим поле «Код_клиента» в качестве совпадающего поля для связывания таблиц.

Аналогично выделим из таблицы «Заказы» таблицу «Товары», которая будет содержать информацию о товарах каждого типа. Для связывания таблиц «Заказы» и «Товары» будем использовать поле «Код_товара». Между таблицами «Клиенты» – «Заказы» и «Товары» – «Заказы» будут отношения один ко многим. В таблице «Заказы» в качестве ключевого поля можно выбрать поля «Код_клиента», «Код_товара» и «Дата_заказа», а также ввести табельный номер «Код_заказа». Получим:



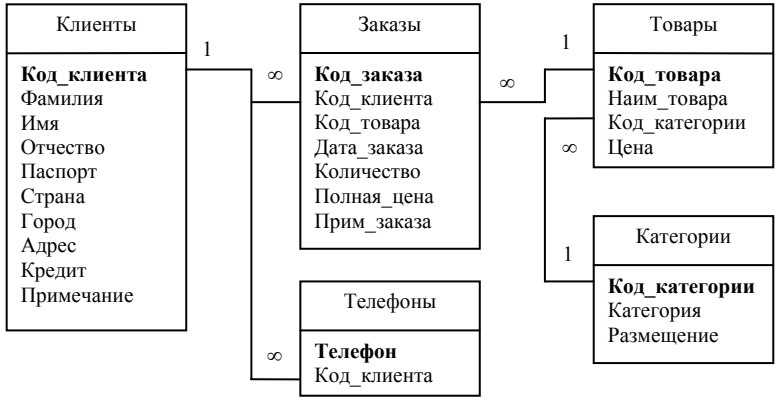
3.4.3. Третья нормальная форма

Таблица находится в третьей нормальной форме, если:

1) она удовлетворяет условиям второй нормальной формы;

2) ни одно из полей таблицы, не входящих в ключ, не должно идентифицироваться с помощью другого поля, не входящего в ключ (иначе, отношение не должно иметь неключевых атрибутов, которые бы находились в транзитивной функциональной зависимости от первичного ключа).

Сведение таблицы к третьей нормальной форме предполагает ее разделение с целью помещения в отдельную таблицу (или несколько таблиц) столбцов, которые не зависят напрямую от выражения первичного ключа. В результате такого разбиения каждое из полей, не входящих в первичный ключ, должно оказаться независимым от какого-либо другого неключевого поля. Обратимся к таблице «Товары». Поле «Размещение» этой таблицы содержит описание места размещения товара на складе, которое однозначно определяется значением поля «Категория». Поскольку поле «Размещение», не входящее в индекс, однозначно определяется другим неключевым полем «Категория», таблица «Товары» не является таблицей в третьей нормальной форме. Для приведения этой таблицы к третьей нормальной форме создадим новую таблицу «Категории». Получим:



Третья нормальная форма также постулирует отсутствие полей, которые могут быть вычислены на основе других. Если полная цена может быть вычислена на основе полей «Цена» и «Количество», то его лучше исключить из таблицы и вычислять динамически. Но в нашем случае это поле содержит отпускную цену, которая может включать скидки и налоги и может не зависеть напрямую от цены товара.

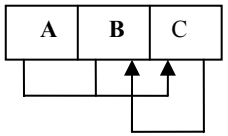
На этом процесс нормализации обычно заканчивается. Если же необходима более детальная проработка данных, то дополнительно рассматривают требования нормальных форм более высокого порядка.

3.4.4. Нормальная форма Бойса – Кодда (Boyce - Codd)

Третья нормальная форма постулирует отсутствие зависимости одних неключевых атрибутов от других неключевых атрибутов, забывая о том, что в ряде случаев может наблюдаться зависимость части ключевого составного атрибута от некоторого неключевого. Если в таблице имеется такая зависимость, то необходимо перейти к усиленной третьей нормальной форме, или форме Бойса – Кодда (BCNF). Таблица находится в форме Бойса – Кодда, если:

1) удовлетворяет требованиям третьей нормальной формы;

2) в ней отсутствуют зависимости атрибутов составного ключа от не ключевых атрибутов.



Этот вид зависимости отображен схематично на схеме. Неключевой атрибут C функционально зависит от составного ключа A + B. Атрибут B в свою очередь функционально зависит от атрибута C. Такая взаимозависимость требует дальнейшей декомпозиции таблицы.

3.4.5. Нормальные формы более высоких порядков

Четвертая нормальная форма устраняет нетривиальные многозначные зависимости. Если таблица содержит два и более никак не связанных между собой многозначных атрибута, то по требованиям первой нормальной формы нам потребуется сделать отдельную запись для каждого значения каждого многозначного атрибута. Общее число записей будет равно числу сочетаний значений, принимаемых многозначными атрибутами, и может быть очень большим. Легко увидеть, что часть информации повторяется. Следовательно, для уменьшения избыточности информации и устранения возможности появления несогласованного состояния данных необходимо прибегнуть к дальнейшей декомпозиции отношения.

Пятая нормальная форма основывается на концепции устранения зависимостей объединения. Может случиться ситуация, когда нельзя разбить таблицу на две таким образом, чтобы получить после объединения исходный результат (некоторые записи могут быть потеряны или, наоборот, могут появиться подложные записи). Однако при этом существует ситуация, что при выполнении всех правил нормализации такую таблицу можно разбить на три и более таблиц. На практике этот вид зависимости очень трудно обнаружить и, следовательно, устранить.

3.4.6. Правила нормализации

Резюмируя вышесказанное, сформулируем несколько правил, которых следует придерживаться при нормализации таблиц.

1. Разрабатывайте схему данных таким образом, чтобы можно было бы легко объяснить ее, т. е. не комбинируйте атрибуты независимых объектов и не создавайте сложные связи.

2. Разрабатывайте схему данных таким образом, чтобы исключить возможность появления аномалий обновления.

3. Разрабатывайте схему данных таким образом, чтобы в связях участвовали только первичные (можно допустить потенциальные) и внешние ключи. Это позволит избежать появления подложных записей.

Требование, чтобы все таблицы были нормализованы, не является обязательным. В нашем случае вполне можно было бы обойтись без таблицы «Телефоны». В заключение сформулируем определение реляционной базы данных как *совокупность нормализованных отношений*.

3.4.7. Выводы

Резюмируя вышесказанное, выделим основные свойства реляционной модели, выделяющие ее из остальных моделей данных:

а) основной единицей обработки данных является отношение. Данные хранятся в связанных отношениях, любая выборка из БД представляет собой отношение, пусть она даже будет единственным числом;

б) данные представляются одним и только одним способом, а именно явными скалярными значениями. Даже связи задаются явными значениями, а не указателями;

в) нет указателей на отдельные строки таблицы. Строка идентифицируется по явному значению ключевого поля.

Ключи играют огромную роль в реляционных БД и используются для достижения следующих целей:

1) исключения дублирования кортежей и, как следствие, адресации кортежей;

2) организации связывания отношений.

Достоинства реляционной модели заключаются в следующем.

1. Простота. Использование таблиц наглядно и легко постигаемо.

2. Гибкость. Один и тот же набор данных можно представить различными способами.

ID	Имя			Адрес					Телефон	
	Фамилия	Имя	Отчество	Страна	Город	Доп. адрес			Номер	Тип
						Улица	Д.	Кв.		
1	Сидоров	Петр	Петрович	Беларусь	Минск	Ландера	6	67	2786544	Дом.
						Коржа	33	23	2367678	Раб.
									6257533	Моб.
2	Иванов	Иван	Иванович	Беларусь	Гродно	Ленина	23	12	244466	Дом.
				Литва	Вильнюс	Лесная	43	65	567543	Дом.

3. Точность. Действия с данными основаны на точных математических операциях.

4. Связность. Реляционное представление дает ясную картину связей.

5. Логическая независимость данных. Можно добавлять новые таблицы, поля и связи, не разрушая старых. Можно удалять таблицы, связи и поля, не участвующие в связи и не являющиеся ключевыми.

Среди недостатков можно выделить следующие.

1. Однородная структура данных, и как следствие, невозможность хранения объектов со сложной структурой в одном кортеже отношения. Для хранения таких объектов приходится выделять несколько связанных отношений, что приводит к потере реальности восприятия сохраненных объектов.

2. Отсутствие стандартных средств идентификации записей и, как следствие, возможность повторного добавления одного и того же объекта.

3. Запрещение многозначных и составных полей.

4. Сложность описания рекурсивных, иерархических и сетевых связей;

5. Фиксированный и ограниченный набор операций (нет возможности определения новых операций).

Примерами реляционных СУБД для ПЭВМ являются следующие: Paradox (Borland) и Access (Microsoft), DB2 (IBM), MS SQL Server (Microsoft), Ingres (ASK Computer Systems) и Oracle (Oracle). Первым коммерческим продуктом стал Oracle (Oracle) в 1978 году.

3.5. ПОСТРЕЛЯЦИОННАЯ МОДЕЛЬ

Классическая реляционная модель предполагает неделимость данных, хранящихся в полях записей таблиц. Существует ряд случаев, когда это ограничение мешает эффективной реализации приложений.

Постреляционная модель данных представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся в ячейках таблицы. Поскольку атомарность данных постулируется первой нормальной формой, то исходя из этого определения ее называют также Non-First Normal Form (NFNF, или NF^2) реляционной моделью. Постреляционная модель данных допускает многозначные и составные поля. Набор значений многозначных полей считается самостоятельной таблицей, встроенной в основную таблицу. Вследствие этого ее еще часто называют вложенной (Nested) реляционной моделью. Все атрибуты в постреляционной модели рассматриваются как составные, имеющие иерархическую структуру, причем на количество ветвей не накладывается каких-либо ограничений. Это позволяет на одном уровне иерархии хранить целые массивы данных, как показано ниже в таблице.

По сравнению с реляционной в постреляционной модели данные хранятся более эффективно, а при обработке не требуется выполнять операцию соединения данных из двух и более таблиц. Помимо обеспечения вложенности полей, постреляционная модель поддерживает ассоциированные многозначные поля. Совокупность ассоциированных полей называется *ассоциацией*. При этом в некоторой строке первое значение одного столбца ассоциации соответствует первым значениям всех остальных столбцов ассоциации. Аналогичным образом связаны все вторые значения столбцов и т. д. На длину и количество полей в записях таблицы не накладывается требование постоянства. Это означает, что структура данных и таблиц имеет большую гибкость. Но поскольку постреляционная модель допускает хранение в таблицах ненормализованных данных, возникает проблема обеспечить целостность и непротиворечивость этих данных. Такая проблема решается с помощью процедур, автоматически выполняемых до или после обращения к данным.

Достоинством постреляционной модели является возможность представления совокупности связанных реляционных таблиц одной постреляционной таблицей. Это обеспечивает высокую наглядность представления информации и повышение эффективности ее обработки. Недостатком постреляционной модели является сложность решения проблемы обеспечения целостности и непротиворечивости хранимых данных. Эта модель пока не реализована в полном объеме в коммерческих СУБД. Примером использования некоторых концепций модели может служить Oracle 8. В этой СУБД реализована поддержка составных и многозначных атрибутов, а также вложенных таблиц (Nested Tables). Для хранения составных атрибутов в Oracle 8 используется тип данных OBJECT. Например, составной атрибут «Телефон» описывается так:

CREATE TYPE Tel_Type AS OBJECT(Tel_Number CHAR(10), Tel_Descr CHAR(5));

Для хранения многозначных атрибутов в Oracle 8 используется тип данных VARRAY, представляющий собой массив некоторых объектов переменной длины, имеющий два свойства: 1) Count – число элементов в массиве и 2) Limit – максимальное число элементов в массиве (задается при определении этого типа). Для формирования массива объектов вначале требуется определить тип объектов, которые будут в нем храниться, например:

CREATE TYPE Tel_Type AS OBJECT(Tel_Number CHAR(15), Tel_Descr CHAR(5));

Затем необходимо определить тип массива, указав его максимальную размерность: CREATE TYPE Tel_List_Type AS VARRAY(5) OF Tel_Type;

Этот же набор данных можно определить и как вложенную таблицу:

CREATE TYPE Tel_List_Type AS TABLE OF Tel_Type;

Оба объекта, VARRAY и TABLE, имеют как похожие свойства, так и существенные различия. Во-первых, тип VARRAY имеет верхний предел числа элементов, а TABLE их не имеет. Во-вторых, во вложенных таблицах можно обращаться к отдельным их элементам и дополнительно определять индексы, что недопустимо в VARRAY.

3.6. МНОГОМЕРНАЯ МОДЕЛЬ

Многомерный подход к представлению данных в базе появился практически одновременно с реляционным, но реально работающих многомерных СУБД (МСУБД) до недавнего времени было очень мало. С середины 1990-х с развитием систем OLAP (On Line Analytical Processing – оперативная аналитическая обработка данных), Data Warehousing (хранилища данных) и Data Mining (добыча данных), применяющихся для сложного анализа данных, принятия решений, менеджмента и получения новых знаний, интерес к ним стал приобретать массовый характер. Основное назначение систем OLAP заключается в динамическом синтезе, анализе и консолидации больших объемов многомерных данных, а это, в свою очередь, связано с необходимостью извлекать большое количество записей из больших БД и производить их оперативную обработку (например, быстро вычислять итоговые значения). Если дать самую краткую характеристику системе Data Warehousing, то ее можно определить как предметно (проблемно) ориентированную, интегрированную, постоянную в структуре, но зависящую от времени совокупность данных, предназначенных для принятия некоторых решений. Главное отличие от стандартных БД состоит в том, что оно представляется собранием данных из многих БД, объединенных вместе для решения какой-либо частной задачи, либо является построенной по определенным правилам выдержкой из очень большой БД. Существенным отличием можно считать также отсутствие операций обновления данных.

Data Mining также является системой, хотя и основанной на данных, но весьма далекой от целей их хранения. Ее главная цель исследовать данные с целью получить новые знания, т. е. сложный анализ данных, таких, например, как поиск цепочек ДНК и т. д. Data Mining преследует следующие цели:

- предсказание (например, прогноз погоды);
- идентификация (например, по образцу, …);
- классификация;
- оптимизация (создание более оптимально действующих систем на основе анализа уже существующих).

Многомерные СУБД являются узкоспециализированными СУБД, предназначенными для интерактивной аналитической обработки инфор-

мации. Многомерность модели данных в них означает не многомерность визуализации цифровых данных, а многомерное логическое представление структуры информации при описании и в операциях манипулирования данными. При этом часто создается только многомерное логическое представление данных, а не их хранение. То есть сами данные могут храниться в больших БД (например, реляционных), а в требуемый момент времени извлекаться оттуда. Обычно в этом случае доступ производится с помощью многомерных индексов. Поскольку многомерные модели предназначены в основном для аналитической обработки данных, а не для создания удобных хранилищ информации, основной задачей в них является операции консолидации, т. е. расчет всех промежуточных и основных итоговых значений, причем по всем размерностям. Это позволяет в много раз уменьшить объем представляемой информации и значительно ускорить доступ к данным и их последующую обработку. Такое предварительное обобщение является особенно ценным при работе с иерархически связанной информацией, например при работе с данными, зависящими от времени.

По сравнению с реляционной многомерная модель данных обладает более высокой скоростью доступа к данным, наглядностью и информативностью. Для сравнения ниже приведены реляционное (табл. 1) и многомерное (табл. 2) представления одних и тех же данных о продаже товаров

Реляционное представление данных *Таблица 1*

Клиент	Товар	Цена	Количество	Дата
Иванов	Масло	1000,00	1	02.02.2007
Петров	Молоко	400,00	2	02.02.2007
Иванов	Сметана	750,00	2	06.02.2007
Иванов	Масло	1000,00	1	22.02.2007
Петров	Молоко	400,00	3	03.03.2007
Сидоров	Масло	1000,00	2	02.03.2007
Иванов	Сметана	1500,00	2	12.03.2007
Иванов	Молоко	500,00	1	05.04.2007
Сидоров	Сметана	800,00	3	14.04.2007
Петров	Масло	1200,00	2	28.04.2007

Многомерное представление данных *Таблица 2*

Товар	Дата		
	Июнь 2007	Июль 2007	Август 2007
Масло	2000	2000	2400
Молоко	800	1200	500
Сметана	1500	3000	2400

Если размерностей всего две (см. табл. 2), то такое представление называется перекрестной или сводной таблицей. Объем продаж вычисляется на основе группированного представления продаж по товарам, клиентам и месяцам. На рис. 13 показан пример трехмерной модели данных, где введена еще одна размерность *Клиент*. Если число размерностей велико, то информация не обязательно должна представляется визуально в виде многомерных объектов (n-мерных гиперкубов), тем более, что это не представляется возможным. Пользователь привык иметь дело с двумерными таблицами, диаграммами, графиками. Многомерная модель как раз предоставляет расширенный набор средств для выборки данных из многомерного хранилища данных, выполненных с разной степенью детализации.

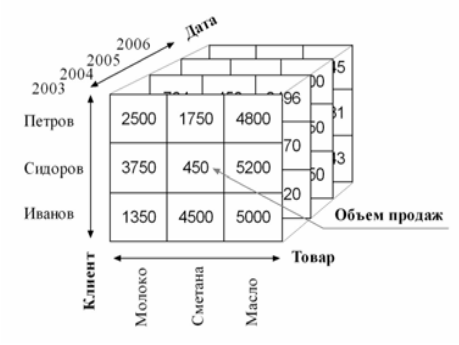


Рис. 10. Пример трехмерной модели

Основными понятиями многомерных моделей являются измерение и ячейка, а операциями – консолидация, срез, вращение, агрегация и детализация:

Измерение (Dimension) – это множество однотипных данных, образующих одну из граней гиперкуба. В многомерной модели данных измерения играют роль индексов, которые служат для идентификации конкретных значений в ячейках гиперкуба.

Ячейка (Cell), или *показатель* – это поле, значение которого однозначно определяется фиксированным набором измерений. В зависимости от того, как формируются значения некоторой ячейки, она может быть переменной (значения изменяются и могут быть загружены из внешнего источника данных или сформированы программно) либо формулой (значения, подобно формульным ячейкам электронных таблиц, вычисляются по заранее заданным формулам).

Консолидация включает в себя простые обобщающие операции типа расчета итоговых значений, матричную арифметику, а также расчет сложных выражений.

Срез (Slice) представляет собой подмножество гиперкуба, полученное в результате фиксации одного или нескольких измерений. Формирование среза выполняется для ограничения выборки данных, поскольку все значения гиперкуба очень трудно представить визуально.

Вращение (Rotate) заключается в изменении порядка измерений при визуальном представлении данных.

Агрегация (Drill Up) и *детализация* (Drill Down) означают соответственно переход к более общему и к более детальному представлению информации пользователю из гиперкуба. Например, в БД хранится информация о дате продажи товара. Тогда с помощью операции детализации можно уточнить момент продажи, а с помощью операции агрегации представить суммарную информацию по месяцам, кварталам и годам.

Основным достоинством многомерной модели данных является удобство и эффективность аналитической обработки больших объемов данных, связанных со временем. При организации обработки аналогичных данных на основе реляционной модели происходит нелинейный рост трудоемкости операций в зависимости от размерности БД и существенное увеличение затрат оперативной памяти на индексацию. Недостатком многомерной модели данных является ее громоздкость для простейших задач обычной оперативной обработки информации.

Примерами систем, поддерживающих многомерные модели данных, являются Essbase (Arbor Software), Media Multi-matrix (Speedware), Oracle Express Server (Oracle), Cache (Inter Systems). В последней СУБД, основанной на многомерной модели данных, реализованы три способа доступа к данным: прямой (на уровне узлов многомерных массивов), объектный и реляционный.

3.7. ОБЪЕКТНО–ОРИЕНТИРОВАННАЯ МОДЕЛЬ

Основное назначение объектно-ориентированных БД (ООБД) состоит в длительном хранении структур данных и объектов программ. Эта задача появилась очень давно, так как для достаточно большой программы фактически всегда актуальной задачей являлось сохранение и восстановление промежуточных результатов работы. Обычно для этой цели использовались файлы и потоковый ввод-вывод. Но в этом случае главный недостаток файловой системы хранения данных, заключающийся в невозможности использования сохраненных данных другими программами, являлся серьезным препятствием при разработке больших программных комплексов, состоящих из многих взаимодействующих приложений. Для каждой программы приходилось писать специальные конверторы объектов в формат, пригодный для совместного хранения и общего использования. И наоборот, чтение данных подразумевало написание конвертора сохраненных в файле объектов обратно в переменные оперативной памяти компьютера. С появлением БД для хранения общих данных стали использовать технологию БД. Но это не решило проблемы, так как приходилось конвертировать объекты в формат БД, и наоборот. По разным оценкам на это уходило до 30 % времени выполнения программы.

Успех объектно-ориентированного стиля программирования (ООП), наращивание объема программ, необходимость создания очень сложных программных комплексов, состоящих из многих компонентов, привело к созданию стандартов совместного хранения и использования объектов, что в свою очередь привело к созданию объектно-ориентированной модели данных (ООМД). С другой стороны, оказалось очень удобным использовать классы как определенные пользователем инкапсулированные типы объектов произвольной внутренней сложности для хранения данных. При этом подходе появилась возможность представления и реализации сложных моделей данных, так как не было необходимости учитывать многочисленные ограничения, накладываемые моделью данных, например требование атомарности атрибутов в реляционной модели. Несомненным достоинством ООМД является также способность к хранению объектов произвольной сложности, отражающих сложность реальных объектов. Для хранения подобного объекта в реляционной БД потребовалось бы множество записей в нескольких таблицах, что привело бы к потере соответствия между реальным объектом и сохраненным.

Объектно-ориентированные БД преимущественно используются вместе с объектно-ориентированными программами и служат для длительного хранения их объектов. Такие программы специально разрабатываются для совместной работы с ООБД. В обычных объектно-ориентированных программах объекты существуют в оперативной памяти компьютера в течение времени выполнения программы и соответственно называются временными. ООБД расширяют время существования объектов так, чтобы они были доступными длительное время и после завершения работы программы. Другими словами, ООБД позволяет сохранять области оперативной памяти, представляющие собой копии объектов, во внешней памяти. С другой стороны, ООБД обеспечивает возможность восстановления сохраненных объектов в оперативной памяти и обеспечение доступа к ним другим программам. Такая возможность длительного хранения объектов требует включения механизмов поиска, индексации, обеспечения целостности и безопасности информации, принятых в БД.

Структура ООМД графически представима в виде дерева, узлами которого являются объекты, а соединительные линии предоставляют связи объектов. Говорить о строгой иерархической структуре нельзя, так как объекты могут наследовать свойства нескольких базовых объектов. Несмотря на внешнюю схожесть логической структуры, ООМД коренным образом отличается от иерархической модели. Основное отличие состоит в методах обработки данных.

Для моделирования объектно-ориентированных систем разработан язык моделирования UML (Universal Modeling Language), который можно определить как промышленный объектно-ориентированный стандарт моделирования. С помощью построителя диаграмм классов этого языка можно визуально разрабатывать (и представлять) объектно-ориентированные приложения, в том числе БД.

3.7.1. Понятие класса

Понятие класса является центральным понятием ООП. Класс может быть определен как определенный пользователем инкапсулированный тип объектов произвольной внутренней сложности. Классы играют роль некоторого шаблона для определения набора подобных объектов. Каждый объект обладает своими собственными характеристиками, однако объекты, имеющие одинаковые свойства и ведущие себя сходным образом, могут быть сгруппированы вместе с образованием класса. Атрибуты и связанные с ними методы (операции над атрибутами) определяются один раз для всего класса, а не для каждого объекта в отдельности. Объекты некоторого класса называются его экземплярами.

3.7.2. Инкапсуляция данных

Понятие инкапсуляции означает, что объект содержит как структуру данных, так и набор операций, с помощью которым этой структурой можно манипулировать. Инкапсуляция ограничивает область видимости данных, принадлежащих объекту и, следовательно, доступ к ним. В противоположность реляционным БД, где операции над данными (вставка, удаление, модификация) являются общими и могут быть применены к любым объектам (таблицам) БД, в ООБД инкапсулированные данные одного объекта недоступны для методов другого объекта. Доступ к таким данным предоставляется только операциям этого объекта. В идеале для каждого атрибута объекта должна быть определена операция доступа (get). Но требования полной инкапсуляции приводят к невозможности выполнения параллельных транзакций и быстрого поиска на основе индексов. Инкапсуляция также приводит к тому, что обеспечение целостности становится задачей каждого объекта в отдельности. Лишь ссылочная целостность может контролироваться системой снаружи. Поэтому требования инкапсуляции объектов в ООБД выполняются только на уровне пользователей, а не системы в целом.

Инкапсулированными являются как данные, так и операции. Только часть операций доступна снаружи и предоставляет интерфейс объекта. Операции в ООБД состоят из двух частей: подписи (интерфейса), содержащей имя операции и список аргументов, и тела (метода), содержащего собственно имплементацию операции. Такой подход реализует требование независимости программ и операций, поскольку позволяет модифицировать операции объектов, не затрагивая внешние программы, имеющие доступ к этим операциям.

3.7.3. Наследование

Некоторые объекты могут иметь подобные, но не идентичные атрибуты и методы. Наследование позволяет определять один класс на основе другого класса, если имеет смысл совместно использовать подобные атрибуты и методы. Класс, от которого производится наследование, называется базовым классом или суперклассом. Производный класс называется подклассом. Подкласс наследует все основные свойства суперкласса и дополнительно определяет свои собственные. Процесс образования суперкласса называется обобщением, а процесс образования подкласса – специализацией. Такая способность к порождению классов от уже существующих делает возможным поступательное наращивание сложности программы и определяет легкость модификации. Наследование существенно сокращает избыточность данных, так как общие свойства могут быть легко перенесены в суперклассы. В свою очередь подклассы могут переопределять свойства суперклассов. Возможность переопределения является важной характеристикой наследования, поскольку позволяет легко управлять отдельными классами с минимальным влиянием на остальную часть системы. Переопределение позволяет повторно использовать имя операции в нескольких классах, что дает возможность определять одно и то же имя для одной и той же операции независимо от ее типа. Конкретный тип операции определяется из контекста при выполнении программы. Такое переопределение имени операции называется перегрузкой.

3.7.4. Полиморфизм

Переопределение (перегрузка) является частным случаем полиморфизма. Полиморфизм означает способность одного и того же программного кода работать с разнотипными данными. Иначе говоря, для разных классов можно определить функции с одинаковыми именами, а вызов конкретной функции будет определяться типом данных либо из контекста. Полиморфизм позволяет обеспечить динамическое (позднее) связывание объектов, когда тип объекта становится известным не в процессе написания программы, а во время ее выполнения.

3.7.5. Идентификация объектов в ООБД

Для идентификации объектов в объектно-ориентированных моделях данных применяется автоматически генерируемый уникальный идентификатор (OID – Object Identifier), который назначается каждому объекту в

момент его создания. Объекты в ООБД могут и не иметь OID, если они не представляют собой некоторой сущности и могут использоваться разными объектами (например числа и другие простые типы данных). Кроме OID для идентификации объектов может также применяться традиционный подход с использованием ключевых атрибутов. OID обладает следующими свойствами:

- генерируется СУБД;
- уникально обозначает реальный объект и не зависит от значений его атрибутов;
- инвариантен. Его нельзя изменить во время всего жизненного цикла программы. После создания объекта его OID не может быть использован повторно ни для какого другого объекта, даже после его удаления.

OID также применяется для связывания объектов и обеспечения ссылочной целостности. Бинарные связи чаще всего реализуются на основе инверсных ссылок, т. е. включения OID объектов друг в друга. Инвариантность OID упрощает контроль целостности, исключая каскадное обновление связанных объектов.

Применение OID дает неоспоримые преимущества над реляционной системой, где идентификация объекта производится по значению ключа, часто надуманного, и который к тому же может меняться. Например, в любой момент времени можно удалить некоторый объект и значение его ключа использовать для другого объекта. Также можно отметить следующие достоинства применения OID:

- эффективность и быстрота. OID обычно представляет собой указатель на фактический адрес объекта в памяти и поэтому доступ к нему занимает очень мало времени;
- независимость от данных. Объект может неоднократно менять свое содержание, но при этом он останется одним и тем же объектом;
- невозможность изменения пользователем.

Но и в ООМД все еще остается возможность создания двух идентичных копий некоторого объекта, различающихся только OID, и поэтому применение ключевых атрибутов для дополнительной идентификации объектов оправдано.

3.7.6. Обеспечение доступности и перманентности объектов

Доступность объектов в ООБД обеспечивается на основе присвоения уникальных имен (ключей) объектам, поскольку OID объектов скрыты для пользователей. Прямое присвоение имени каждому объекту в отдельности не производится, так как БД может состоять из сотен и тысяч объектов, и наличие большого числа сложных имен может только усложнить доступ. Но так как обычно объектно-ориентированная система

построена на основе иерархии контейнеров, где одни объекты, по крайней мере, концептуально содержатся в других (обычно содержатся не сами объекты, а их глобальные уникальные идентификаторы), то имена назначают только базовым объектам либо даже их коллекциям. Тогда по ссылкам (OID) можно легко добраться до необходимого объекта. Такой механизм доступа сильно отличается от применяемого в реляционных системах, где все объекты предполагаются постоянными и доступными через значения их потенциальных ключей.

Перманентность объектов, т. е. возможность длительного хранения объектов во внешней памяти, может быть реализована тремя различными способами: созданием контрольных точек, сериализацией и явной подкачкой страниц.

В первом способе на диск копируется все адресное пространство программы (или его некоторая часть). В тех случаях, когда копируется все адресное пространство, программу можно вновь запустить с некоторой контрольной точки. Этот подход имеет два серьезных недостатка. Во-первых, контрольная точка может быть использована только той программой, которая ее создала; во-вторых, на диск копируется множество совершенно ненужной информации.

Во втором способе на диск копируется отдельная иерархия контейнеров объектов. Вначале производится обход связанного графа объектов, доступных из выбранного объекта, затем вся эта структура пишется на диск. Но такой подход не позволяет восстанавливать связанные структуры данных, которые совместно используют одну и ту же подчиненную структуру, при их раздельной записи, так как подчиненная структура после восстановления уже не будет совместной. К тому же сериализация производится сразу большими блоками, что приводит к неэффективному расходу памяти компьютера при сохранении небольших изменений.

Третий подход связан с явным обменом областей оперативной памяти и внешнего устройства. Этот процесс напрямую связан с преобразованием указателей объектов, существующих в оперативной памяти, и указателей на те же объекты в адресном пространстве внешней памяти. Существуют две реализации этого подхода: в первой объект считается перманентным (т. е. сохраняемым) тогда, когда он достижим для перманентного корневого объекта, а во второй он будет перманентным только в том случае, если он явно объявлен перманентным в прикладной программе. В обоих случаях перманентными будут только те объекты, в которых еще на этапе программирования заложена эта возможность. Недостатками такого подхода являются необходимость сборки «мусора» после выполнения программы, так как объект будет перманентным до тех пор, пока не будет явно удален приложением, и необходимость постоянной обработки двух систем указателей. Последний метод можно считать наиболее перспек-

тивным, если напрямую включить схемы обеспечения перманентности в базовый язык программирования.

3.7.7. Стандарт ODMG

Совместное использование внешними программами объектов, хранящихся в ООБД, может быть осуществлено только при наличии единого стандарта создания, хранения и использования объектов. Фактически даже отсутствие языка запросов типа SQL, стандартного языка БД, может являться серьезной проблемой при использовании ООБД. Растущий интерес к ООБД предопределил появление такого стандарта. Группа ODMG (Object Database Management Group), включающая ряд ведущих производителей программного обеспечения, разработала стандартную объектно-ориентированную модель данных, которая может быть использована в ООБД. Вначале был создан стандарт ODMG-93, или ODMG-1. Затем он перерос в ODMG-2.

ODMG-2 состоит из следующих частей:

- объектной модели;
- языка определения объектов (ODL – object definition language);
- объектного языка запросов (OQL – object query language);
- расширения объектно-ориентированных языков программирования (ООЯП), таких как C++, Java, Smalltalk и др.

Объектная модель и язык определения объектов стандартизируют ключевые слова, типы данных, их конструкторы и т. д., наряду с механизмами идентификации, связывания, именования, обеспечения доступности и перманентности объектов (смотри систему принятых обозначений на рис. 14). К примеру, этот стандарт рекомендует применение только ин-

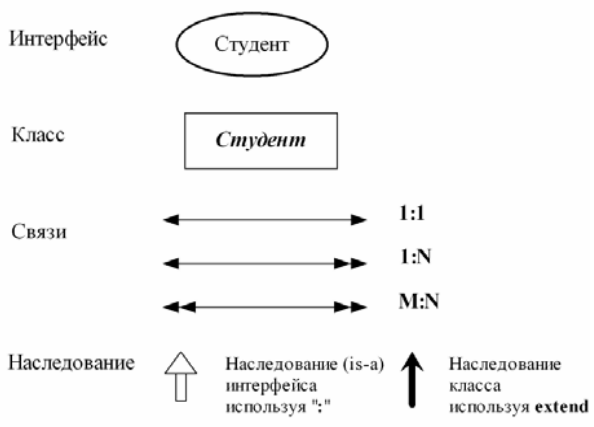


Рис. 11. Система принятых обозначений

версных связей на основе OID и вводит такие ключевые слова, как attribute, key, relationship, extent и т. д. Объектный язык запросов представляет собой расширение существующих ООЯП. Синтаксис его похож на SQL и дополнен концепциями наследования, инкапсуляции и полиморфизма. Например, представим определение двух классов, лежащих в основе структуры данных ООБД некоторого университета:

```
class Person (extent persons, key ssn) {
  attribute struct Pname {string fname, string mname, string (name} name;
  attribute string ssn;
  attribute date birthdate;
  attribute enum Gender{M, F) sex;
  attribute struct Address {short no, string street, short aptno, string city, string
state, short zip}  address;
  short age();  };

class Student extends Person(extent students) {
  attribute string class;
  attribute Department minors_in;
  relationship Department majors_in inverse Department::has_majors;
  relationship set<Grade> completed_sections inverse Grade::student;
  void change_major(in string dname) raises(dname_not_valid);
  float gpa();
  void register(in short secno) raises(section_not_valid);
  void assign_grade(in short secno; in GradeValue grade)
raises(section_not_valid, grade_not_valid);  };
```

и саму структуру (см. рис. 15)

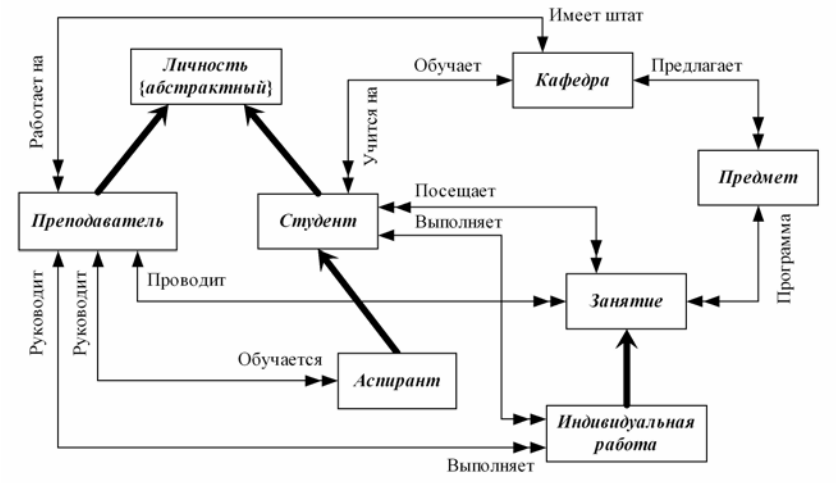


Рис. 15. Структура данных ООБД университета

3.7.8. Выводы

Объектно-ориентированные БД имеют следующие преимущества.

1. Однозначная идентификация объектов.

2. Расширенные возможности моделирования. Позволяют более точно моделировать реальный мир.

3. Расширяемость.

4. Устранение проблем несоответствия. Программы могут читать объекты напрямую, без промежуточной конвертации.

5. Поддержка эволюции схемы.

6. Применимость для разработки сложных специализированных БД.

Также можно сформулировать и недостатки, присущие ООБД.

1. Отсутствие универсальной модели данных.

2. Недостаточный опыт эксплуатации и отсутствие стандартов.

3. Сложность.

4. Отсутствие поддержки стандартных видов представлений.

5. Недостаточность средств обеспечения целостности и безопасности информации.

Концептуальное различие объектно-ориентированной и реляционной модели заключается в следующем.

1. В типах данных. Простые в реляционной БД и определенные пользователем классы в ООБД.

2. В идентификации объектов. OID в ООБД и потенциальные ключи в реляционной БД.

3. В установлении связей. Инверсные включения OID в ООБД и первичные и внешние ключи в реляционной БД.

4. В возможности наследования в ООБД.

5. В применении и разработке операций. В ООБД операции принадлежат объектам и разрабатываются вместе с ними. В реляционной БД операции являются общедоступными и разрабатываются после определения структуры данных.

Примером ООБД можно считать СУБД OpenODB (Hewlett Packard), O2 (Ardent Software) и Object Store (Object Design).

3.8. ОБЪЕКТНО–РЕЛЯЦИОННАЯ МОДЕЛЬ

Объектно-реляционные системы получили свое развитие на основе концептуального совпадения назначения и свойств двух фундаментальных понятий в реляционных и объектно-ориентированных базах данных: доменов и классов. Если определить домен немного шире, чем в реляционной модели, а именно как определенный пользователем инкапсулированный тип данных с произвольной внутренней сложностью, то увидим, что домен и класс – это одно и то же. Следовательно, в объектно-

реляционной БД (ОРБД) домены и соответственно реляционные атрибуты могут содержать абсолютно все: массивы, списки, объекты вместе с методами обработки своих данных. Таким образом, реляционная модель данных, где в качестве доменов могут выступать классы, в принципе могла бы решать все задачи, выполняемые в ООБД и невыполняемые в реляционной БД.

Для обеспечения функциональности таких систем со стороны объектно-ориентированного подхода требуется поддержка определенных пользователем типов данных произвольной сложности вместе с методами обработки своих данных, а со стороны реляционного подхода требуется, чтобы отношения содержали сами объекты (по крайней мере концептуально), а не ссылки или указатели на них.

Примером ОРБД можно считать DB2, Informix и продукты Oracle 8.x.

Контрольные вопросы и задания

1. Назовите основные модели данных.
2. Укажите достоинства и недостатки иерархической модели данных.
3. Укажите достоинства и недостатки сетевой модели данных.
4. Какой вид связи невозможно реализовать в иерархической модели данных?
5. Назовите условия, при соблюдении которых таблицу (реляционная модель данных) можно считать отношением.
6. Дайте определения каждому из следующих понятий реляционной модели данных: отношению, домену, кортежу, степени отношения, кардинальному числу, первичному и внешнему ключу.
7. Укажите различия между потенциальными ключами и первичным ключом отношения.
8. Что может служить идентификатором записи в реляционной модели данных: а) значение первичного ключа; б) значение потенциального ключа; в) значение внешнего ключа; г) порядковый номер записи.
9. Какие из нижеперечисленных терминов означают одно и то же понятие в реляционной модели данных: домен, кортеж, отношение, строка, столбец, поле, запись, атрибут?
10. Покажите на простых примерах виды связей между отношениями в реляционной модели данных.
11. Перечислите средства обеспечения ссылочной целостности в реляционной модели данных.
12. Что дает каскадное обновление записей?
13. Что дает каскадное удаление записей?
14. Перечислите отличительные свойства реляционной модели данных.

15. Сформулируйте назначение ключей в реляционной модели данных.

16. Какие задачи решает теория нормализации данных?

17. Опишите проблемы, связанные с наличием повторяющихся данных.

18. Сформулируйте правила нормализации.

19. Дайте определение а) первой нормальной формы; б) второй нормальной формы; в) третьей нормальной формы.

20. Зачем необходимо разбивать информацию, хранящуюся в одной таблице, на несколько связанных таблиц в реляционной модели данных.

21. Следуя правилам нормализации, разработайте реляционную модель данных расписания занятий, представленную на рис. 12.

22. Следуя правилам нормализации, разработайте реляционную модель данных предприятия, представленную на рис. 6.

23. Перечислите основные достоинства и недостатки реляционной модели данных.

24. Чем отличается постреляционная модель данных от реляционной?

25. Сформулируйте назначение и основные принципы построения объектно-ориентированных БД.

26. Перечислите основные достоинства и недостатки многомерной модели данных.

27. На базе относительного сходства какого понятия в реляционной и объектно-ориентированной модели данных получили развитие объектно-реляционные БД.

28. Сформулируйте принципиальные отличия реляционной, объектно-ориентированной и объектно-реляционной модели данных.

29. Чем отличается расширенная ER модель от обычной ER модели?

30. Дайте определение многомерной модели данных. В чем состоит основное отличие многомерной модели от других моделей.

31. Перечислите основные достоинства и недостатки многомерной модели данных.

Ответы на некоторые вопросы

Схема данных для ER-модели, представленной на рис. 6, может быть представлена следующим образом

Рис. 16 Схема данных ER модели, представленной на рис. 6

4. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

Все тонкости построения информационной модели некоторой предметной области деятельности человека преследуют одну цель – получить хорошую БД. Поясним термин – хорошая БД и сформулируем требования, которым она должна удовлетворять.

1. Должна удовлетворять информационным потребностям пользователей (организаций) и по структуре и содержанию соответствовать решаемым задачам.

2. Должна обеспечивать получение требуемых данных за приемлемое время, т. е. отвечать требованиям производительности.

3. Должна легко расширяться при реорганизации предметной области.

4. Должна легко изменяться при изменении программной и аппаратной среды.

5. Корректные данные, загруженные в БД, должны оставаться корректными (данные должны проверяться на корректность при их вводе).

4.1. ЭТАПЫ ПРОЕКТИРОВАНИЯ

Основные этапы проектирования можно отобразить с помощью схемы, представленной на рис. 17. Рассмотрим каждый из этапов подробнее:

Первый этап. Планирование разработки БД. На этом этапе выделятся наиболее эффективный способ реализации этапов жизненного цикла системы.

Второй этап. Определение требований к системе. Производится определение диапазона действий и границ приложения БД, а также сбор и анализ требований пользователей.

Третий этап. Проектирование концептуальной модели БД. Процесс создания БД начинается с определения концептуальной модели, представляющей объекты и их взаимосвязи без указания способов их физического хранения. Усилия на этом этапе должны быть направлены на структуризацию данных и выявление взаимосвязей между ними. Этот процесс можно разбить еще на несколько под этапов:

а) *Уточнение задачи*. Перед началом работы над конкретным приложением разработчик должен явно представлять то, что он будет разрабатывать. В случаях, когда разрабатывается небольшая персональная БД, такие представления могут быть достаточно полными. В других случаях, когда разрабатывается большая БД под заказ, таких представлений или может быть очень мало, или они будут поверхностными. Сразу начинать разработку с определения таблиц, полей и связей между ними будет явно рановато. Такой подход в большинстве случаев приведет к полной переделке приложения. Поэтому следует затратить некоторое время на составление списка всех основных задач, которые должны решаться этим приложением, включая и те, которые могут возникнуть в будущем.

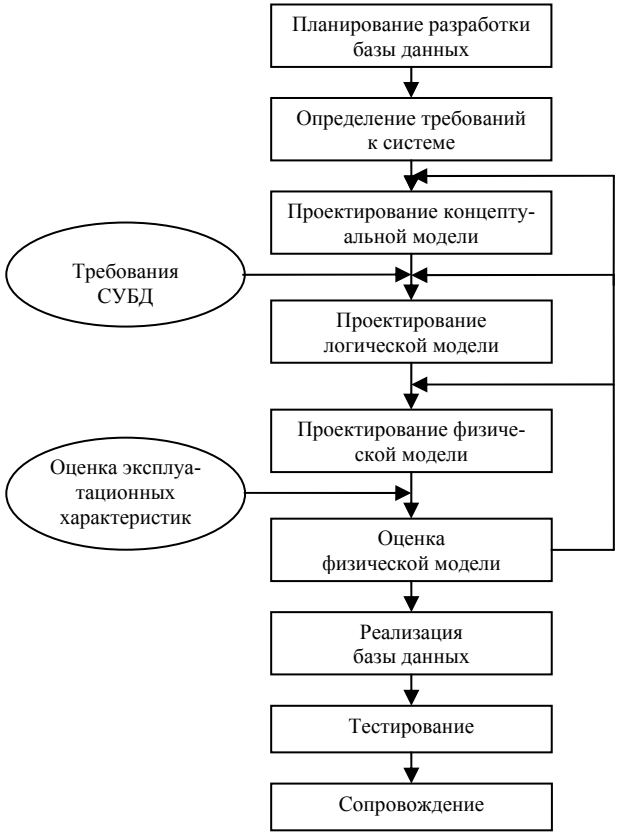


Рис. 17. Схема проектирования БД

б) *Уточнение последовательности выполнения задач*. Чтобы приложение работало логично и удобно, лучше всего объединить основные задачи в группы и затем упорядочить задачи каждой группы так, чтобы они располагались в порядке их выполнения. Группировка и графическое представление последовательности их выполнения поможет определить естественный порядок выполнения задач.

в) *Анализ данных*. После определения списка задач необходимо для каждой задачи составить полный перечень данных, требуемых для ее решения. После анализа данных можно приступать к разработке концептуальной модели, т. е. к выделению объектов, атрибутов и связей.

Четвертый этап. Построение логической модели. Этап начинается с выбора модели данных. При выборе модели важную роль играет ее простота, наглядность и сравнение естественной структуры данных с моделью, ее представляющей. Но часто этот выбор определяется успехом (или наличием) той или иной СУБД. То есть разработчик выбирает СУБД, а не модель данных. Таким образом, на этом этапе концептуальная модель транслируется в модель данных, совместимую с выбранной СУБД. Возможно, что отображенные в концептуальной модели взаимосвязи между объектами либо некоторые атрибуты объектов окажутся впоследствии нереализуемыми средствами выбранной СУБД. Это потребует изменение концептуальной модели. Версия концептуальной модели, которая может быть обеспечена конкретной СУБД, называется ***логической моделью***. Иногда процесс определения концептуальной и логической моделей называется определением структуры данных.

Пятый этап. Построение физической модели. Физическая модель определяет размещение данных, методы доступа и технику индексирования. На этапе физического проектирования мы привязываемся к конкретной СУБД и расписываем схему данных более детально, с указанием типов, размеров полей и ограничений. Кроме разработки таблиц и индексов, производится также определение основных запросов.

При построении физической модели приходится решать две взаимно противоположные по своей сути задачи. Первой из них является минимизация места хранения данных, а второй – достижение максимальной производительности, целостности и безопасности данных. Например, для обеспечения высокой скорости поиска необходимо создать индексы, причем их число будет определяться всеми возможными комбинациями полей, участвующими в поиске; для восстановления данных требуется ведения журнала всех изменений и создание резервных копий БД; для эффективной работы транзакций требуется резервирование места на диске под временные объекты и т. д., что приводит к увеличению (иногда значительному) размера БД.

Шестой этап. Оценка физической модели. На этом этапе проводится оценка эксплуатационных характеристик. Здесь можно проверить эффективность выполнения запросов, скорость поиска, правильность и удобство выполнения операций с БД, целостность данных и эффективность расхода ресурсов компьютера. При неудовлетворительных эксплуатационных характеристиках возможен возврат к пересмотру физической и логической моделей данных, выбору СУБД и типа компьютера.

Седьмой этап. Реализация БД. При удовлетворительных эксплуатационных характеристиках можно перейти к созданию макета приложения, т. е. набору основных таблиц, запросов, форм и отчетов. Этот предварительный макет можно продемонстрировать перед заказчиком и получить его одобрение перед детальной реализацией приложения.

Восьмой этап. Тестирование и оптимизация. Обязательным этапом является тестирование и оптимизация разработанного приложения.

Девятый этап, заключительный. Сопровождение и эксплуатация. Так как выявить и устранить все ошибки на этапе тестирования не получается, то этап сопровождения является обычным для БД.

Существуют два основных подхода к проектированию схемы данных: нисходящий и восходящий. При восходящем подходе работа начинается с нижнего уровня – уровня определения атрибутов, которые на основе анализа существующих между ними связей группируются в объекты, и связи между ними. Процесс нормализации таблиц для реляционной модели данных является типичным примером этого подхода. Этот способ хорошо подходит для проектирования относительно небольших БД. При увеличении числа атрибутов до нескольких сотен и даже тысяч более приемлемой стратегией проектирования является нисходящий подход. Начинается он с определения нескольких высокоуровневых сущностей и связей между ними. Затем эти объекты детализируются до необходимого уровня. Примером такого подхода проектирования является использование модели сущность–связь. На практике эти подходы обычно комбинируются. В этом случае можно говорить о смешанном подходе проектирования.

4.2. СРЕДСТВА АВТОМАТИЗИРОВАННОЙ РАЗРАБОТКИ ПРИЛОЖЕНИЙ

Проектирование БД можно проводить с помощью автоматизированных систем разработки приложений, так называемых CASE (Computer Aided Software Engineering) систем. Автоматизированные системы разработки приложений представляют собой программные средства, поддерживающие процессы создания и сопровождения информационных систем, такие как анализ и формулировка требований, проектирование приложений, генерация кода, тестирование, управление конфигурацией и проектом. Основная цель CASE систем состоит в том, чтобы отделить процесс проектирования программного обеспечения от его кодирования и последующих этапов разработки (тестирование, документирование и т. д.), а также автоматизировать весь процесс создания программных систем.

Разработка БД с помощью CASE систем на этапе концептуального проектирования обычно проводится с помощью ER модели, которую необходимо рассматривать как способ концептуального проектирования данных, а не как самостоятельную модель данных. Результатом проектирования практически всегда (или до недавнего времени) являлась реляци-

онная модель данных. В последнее время с появлением объектно-ориентированной модели данных конечным результатом проектирования все чаще стали объектно-ориентированная и объектно-реляционная модели данных.

Для автоматизации проектирования объектно-ориентированных баз данных CASE системы предоставляют специальный язык UML (Unified Modeling Language), который можно определить как промышленный объектно-ориентированный стандарт моделирования. Его составляющими можно назвать языки OMT (Object Modeling Technique) и OOSE (Object-Oriented Software Engineering). Большую роль в создании этого языка сыграл консорциум OMG (Object Management Group), включающий ряд ведущих производителей программного обеспечения.

CASE системы различаются по ориентации, функциональной полноте и по типу используемой модели. По ориентации можно выделить CASE системы, предназначенные для анализа предметной области, для проектирования БД и для полной разработки приложений. К числу последних можно отнести MS Visual Studio. По функциональной полноте они разделяются на системы, предназначенные для решения отдельных задач проектирования и на интегрированные системы, поддерживающие весь цикл разработки. По типу используемой модели они разделяются на структурные, объектно-ориентированные и комбинированные.

Примерами CASE систем могут служить ERWin (Logic Works), Rational Rose (Rational Software) (OODBMS), S-Designer (SPD), DataBase Designer, Developer/Designer 2000 (Oracle Corp.).

Контрольные вопросы и задания

1. Сформулируйте требования, которым должна удовлетворять БД.
2. На что должны быть направлены усилия разработчиков при построении концептуальной модели.
3. Отобразите схематично основные этапы проектирования БД и дайте их характеристику.
4. Чем отличается логическая модель данных от концептуальной?
5. Какие две противоположные по сути задачи приходится решать при построении физической модели?
6. Что такое CASE системы?
7. Дайте классификацию CASE систем.

5. СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Под СУБД понимают совокупность языковых и программных средств, обеспечивающих создание, поддержание (редактирование) и доступ к данным как со стороны пользователей, так и со стороны приложений. СУБД предоставляет:

- средства поддержки логической модели данных;
- развитый пользовательский интерфейс;
- средства программирования высокого уровня, с помощью которых можно создавать свои собственные приложения – БД;
- средства администрирования, обеспечения целостности, безопасности и секретности информации.

5.1. АРХИТЕКТУРА ПОСТРОЕНИЯ СУБД

Основные операции (функции) СУБД заключаются в хранении, управлении, обработки и представлении данных. Их можно условно разбить на следующие классы:

а) операции определения структуры данных;

б) операции изменения данных: добавление (вставка) данных; удаление данных; модификация (редактирование) данных. Все эти операции зачастую заменяются одним термином – обновление данных;

в) операции поиска и доступа к информации: выборка (получение данных из БД и обеспечение пользователю доступности выбранных данных); поиск (предусматривает установку курсора на записи, удовлетворяющей определенному критерию); сортировка; фильтрация;

г) операции поддержки целостности и восстановления данных;

д) операции контроля за доступом;

е) операции поддержки обмена данными;

ж) операции администрирования.

Сюда также можно отнести так называемые низкоуровневые операции: управление данными во внешней памяти; управление буферами оперативной памяти; управление параллельностью и транзакциями; ведение журнала изменений в БД (имеется не во всех СУБД).

Для работы с хранящейся в БД информацией СУБД предоставляет программам и пользователям следующие два вида языков:

- язык описания данных (DDL – Data Definition Language) – высокоуровневый непроцедурный язык декларативного типа, предназначенный для описания логической структуры данных. Может дополнительно подразделяться на SDL (Storage Definition Language) и VDL (View Definition Language);

• язык манипулирования данными (DML – Data Manipulation Language) – совокупность инструкций, обеспечивающих выполнение основных операций по работе с данными: ввод, удаление, модификацию и выборку данных. Обычно инструкции DML указывают, какие данные необходимо извлечь, а не как они должны быть извлечены, и поэтому его также называют декларативным языком.

СУБД представляет собой сложную систему, состоящую из многих компонентов. Структурно схему построения СУБД можно представить следующей схемой (рис. 18).

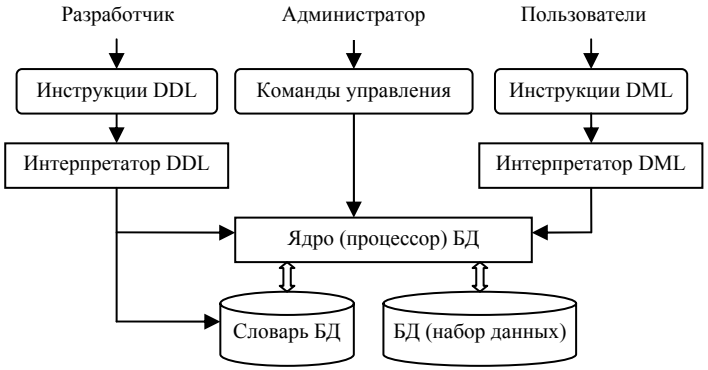


Рис. 18. Структурная схема СУБД

База данных и словарь данных (системный каталог, хранящий метаданные) хранятся на внешних носителях. Операции доступа к данным на низком уровне (запись/чтение) контролируются операционной системой. Если требуется высокая производительность, то такие операции могут контролироваться самой СУБД. На более высоком уровне операции доступа контролируются ядром БД, которая использует для этого информацию, содержащуюся в словаре данных. Практически всегда для каждой операции доступа в оперативной памяти (ОП) выделяется область, называемая буфером. Необходимость буферизации обусловлена тем, что объем ОП всегда меньше объема внешней памяти. Такой подход повышает производительность системы и позволяет предоставить доступ к одним и тем же данным разным процессам (в конечном случае пользователям). Ядро БД также ответственно за поддержку безопасности и целостности БД.

Интерпретатор DDL обрабатывает инструкции определения/изменения структуры данных и сохраняет результаты в системном каталоге. Запросы от пользователей интерпретируются интерпретатором DML и затем обрабатываются ядром БД. Так как чаще всего запросы формируются через высокоразвитый интерфейс и могут представлять собой нечто достаточно абстрактное, то они проходят предварительную предкомпиляцию в инструкции DML. Операции управления и администрирования выполняются непосредственно ядром БД и промежуточной обработки не требуют.

Признаки, по которым можно различать СУБД.

1. По типу поддерживаемой модели: реляционная, объектная, объектно-реляционная, иерархическая, сетевая, другие (многомерная, постреляционная).

2. По количеству поддерживаемых пользователей: однопользовательские и многопользовательские.

3. По количеству компьютеров, на которых располагается СУБД: централизованные (один компьютер) и распределенные (Distributed DBMS – DDBMS).

4. По назначению: общецелевые и специальные (обслуживание банков, продажа и резервирования билетов и т. д. Часть из них образует категорию OLTP – on-line transaction processing systems).

5.2. ИНДЕКСЫ

Индексы предназначены для ускорения доступа к данным и повышения скорости поиска. Основным их назначением является устранение последовательного или пошагового сканирования записей при поиске нужных данных. Помимо этого индексы также позволяют проверять значения поля на уникальность и сортировку записей в таблице. *Индекс* – это внутренняя таблица, имеющая два столбца: упорядоченные значения выражения, содержащего все поля, включенные в индекс, и местоположение каждой записи с данным значением индексного выражения. Поля, на основе которых создается индекс, обычно могут быть упорядочены как по возрастанию своих значений, так и по убыванию, хотя некоторые СУБД поддерживают только возрастающую сортировку. При выполнении сортировки изменение физического положения записей не происходит. Меняется только визуальное представление очередности следования записей. Таким образом, порядок следования записей в БД легко менять, назначая соответствующий индекс таблице.

В первом поле индекса можно хранить значения индексных полей таблицы либо свертку поля (так называемый хеш-код). Преимущество хранения хеш-кода вместо соответствующих значений состоит в том, что длина свертки независима от длины исходных значений и имеет достаточно малую величину (например, 4 байта), что существенно снижает время поиска. Недостатком хеширования является необходимость выполнения операции свертки, что требует определенных затрат времени, и

возможность возникновения коллизий (свертка различных значений может дать одинаковый хеш-код).

Для организации ссылки на запись таблицы часто используются абсолютные или относительные адреса дисковой памяти компьютера. Так как таблицы хранятся в виде совокупности блоков данных фиксированного размера (например, целого числа кластеров), то преимущественно в качестве адресов записей используются адреса начала этих блоков.

Таким образом, индекс обычно сохраняет каждое значение индексированного поля вместе со списком указателей на все блоки данных физического носителя, которые содержат записи с данным значением индексируемого поля. Значения всегда хранятся в упорядоченном виде, что позволяет применить быстрые алгоритмы поиска, например бинарный поиск. Число обращений к индексу будет $\log_2 n$, где n – число записей в индексе. Индексный файл имеет намного меньший размер, чем файл данных, тем самым еще более повышая эффективность поиска.

Индексы различают на первичные и вторичные в зависимости от того, определяет ли поле, на котором основан индекс, физический порядок записей в таблице – или нет. Первичный (primary) индекс создается на основе первичного ключа таблицы, т. е. поля, которое определяет физический порядок следования записей в таблице и является уникальным. Так как таблица может содержать не более одного поля, определяющего физический порядок записей, то для нее может быть определен только один первичный индекс. Дополнительно для таблицы можно определить несколько вторичных (secondary) индексов, основанных на неупорядоченных физически полях.

5.2.1. Первичные индексы

Первичный индекс состоит из двух полей. Первое поле имеет тот же тип данных, что и поле, на котором основан индекс, а второе содержит указатели на блоки данных физического носителя. Так как в одном блоке данных обычно помещается несколько записей, то первое поле индекса будет содержать не все значения индексированного поля, а только первые для блока данных. Соответственно, в первом поле записывается значение первичного ключа для первой записи в блоке, а во втором – указатель на этот блок данных (рис. 19).

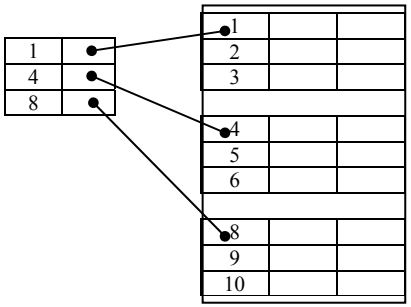


Рис. 12. Структура первичного индекса

Число записей в первичном индексе будет определяться числом блоков данных, отведенных для хранения таблицы. Поиск некоторой записи будет вначале проводиться по первому полю индекса, затем по упорядоченным значениям индексируемого поля в блоке данных. Скорость поиска данных с использованием первичного индекса будет максимально высокой, так как число записей в индексе намного меньше, чем в таблице и для индекса и для блоков данных можно использовать быстрый алгоритм поиска вследствие упорядоченности значений.

5.2.2. Вторичные индексы

Структура вторичного индекса от первичного отличается тем, что второе поле индекса содержит указатели на блоки данных с неупорядоченными записями и тем, что для индексируемых полей с повторяющимися значениями приходится хранить список указателей на все блоки данных физического носителя, которые содержат записи со значениями индексируемого поля. Помимо указателей на начала блоков данных применяются также указатели на записи с данным значением индексируемого поля (рис. 20).

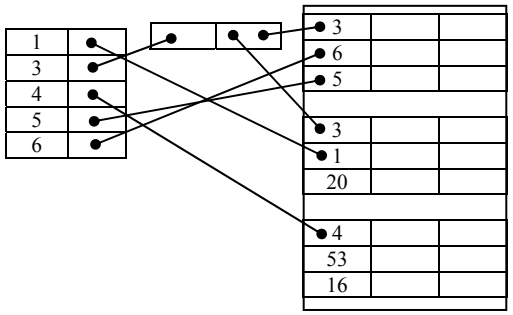


Рис. 13. Структура вторичного индекса

Иначе говоря, первое поле вторичного индекса содержит упорядоченные значения индексируемого поля, а второе – указатели на блоки данных физического носителя, которые содержат записи со значениями индексируемого поля.

5.2.3. Многоуровневые индексы

По способу своей организации индексы разделяются на одноуровневые и многоуровневые. Разбиение некоторого индекса на несколько уровней производится для дальнейшего увеличения скорости поиска записей при увеличении числа записей в индексируемой таблице. Число записей в самом индексе и, следовательно, размер индексного файла, может быть настолько большим, что операция открытия индекса и поиск в нем может занимать значительное время. Бинарный поиск требует приблизительно $\log_2 b_i$ обращений к индексу, ссылающемуся на b_i блоков данных. Если разбить такой индекс на блоки записей одинаковой длины и

включить в дополнительный уровень только значения первых записей полученных блоков и соответствующие указатели на эти блоки, то общее число обращений к индексу значительно сократится. Количество обращений к индексу уже будет $\log_{fs} b_i$, где *fs* это число записей в блоке индекса первого уровня. В такой схеме второй уровень будет являться первичным индексом для первого уровня, как представлено на рис. 21.

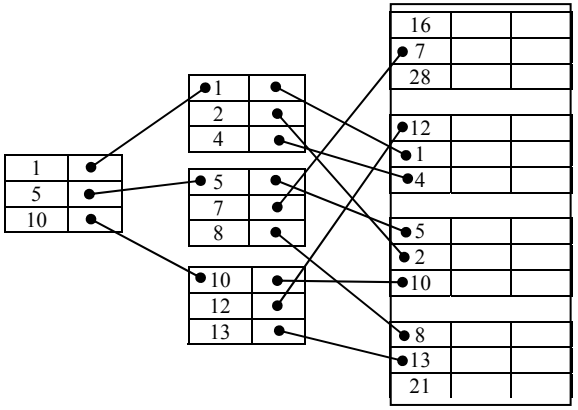


Рис. 14. Структура многоуровневого индекса

5.2.4. Составные индексы

Если в таблице поиск часто ведется по нескольким полям одновременно, то для его ускорения необходимо определить составной индекс. Составной индекс по своей структуре от обычных индексов значительно не отличается. Отличие состоит в том, что первое поле индекса содержит свертку значений всех полей, входящих в индекс. К примеру, нам требуется часто искать некоторое предприятие по стране регистрации, городу и собственно по его имени. Тогда при создании индекса вначале производится сортировка по всем полям, включенным в индекс, в порядке их следования (т. е. будет произведена сортировка по стране регистрации, затем внутри каждой страны по городу, и, наконец, внутри каждого города по названию предприятия). Затем на основе отсортированных значений происходит создание первого поля индекса и заполнение второго поля указателями на блоки данных физического носителя, содержащими записи со значениями индексированных полей. Очевидно, что совокупность вторичных индексов, созданных для каждого из полей, входящих в составной индекс, не сможет заменить составного индекса, поскольку результат вложенной сортировки не равен результату последовательной сортировки.

Ускорение поиска достигается возможностью применения быстрых алгоритмов поиска для всех полей составного индекса (вследствие их упорядоченности). В большинстве систем существует одно ограничение на использование составного индекса, а именно только последнее условие (т. е. сравнение с последним полем индекса) может быть неравенством. Например, Страна = 'Беларусь' And Город = 'Минск' And Предприятие <'С'.

Контрольные вопросы и задания

1. Дайте определение СУБД.
2. Перечислите основные операции СУБД.
3. Перечислите три основные операции СУБД, отвечающие за поиск данных.
4. Зарисуйте структурную схему СУБД и поясните назначение каждого блока.
5. Дайте классификацию СУБД.
6. Перечислите признаки, по которым можно различать СУБД.
7. Сформулируйте назначение индексов в БД.
8. Представьте схематично структуру первичного индекса.
9. Представьте схематично структуру вторичного индекса.
10. Для чего используются многоуровневые индексы?
11. Поясните, почему скорость поиска с использованием первичного индекса является наибольшей.
12. Чем отличается первичный индекс от вторичного?
13. Чем отличается составной индекс от простого?

6. ЯЗЫК ЗАПРОСОВ SQL

Структурированный язык запросов SQL (Structured Query Language) предназначен для создания запросов в реляционных БД. Он позволяет выполнять операции над таблицами (создание, удаление, изменение структуры) и над данными таблиц (выборка, изменение, добавление и удаление), а также производить некоторые сопутствующие операции. SQL основан на реляционном языке исчисления кортежей и является непроцедурным языком, т. е. не содержит операторов управления, организации подпрограмм, ввода-вывода и т. п. В связи с этим SQL автономно не используется, обычно он погружен в среду встроенного языка программирования СУБД. В различных СУБД состав операторов SQL может несколько отличаться. В данной главе при изложении синтаксиса будем придерживаться стандарта, принятого в СУБД Access.

Основные операторы языка SQL можно условно разделить на два подъязыка: язык определения данных DDL и язык манипулирования данными DML. Операторы DDL и DML представлены в табл. 4.

Таблица 4

Вид	Название	Назначение
DDL	CREATE TABLE	создание таблицы
	DROP TABLE	удаление таблицы
	ALTER TABLE	изменение структуры таблицы
	CREATE INDEX	создание индекса
	DROP INDEX	удаление индекса
	CREATE VIEW	создание представления
	DROP VIEW	удаление представления
	CREATE DOMAIN	создание домена
	ALTER DOMAIN	изменение домена
	DROP DOMAIN	удаление домена
DML	SELECT	выборка записей
	UPDATE	изменение записей
	INSERT	вставка новых записей
	DELETE	удаление записей
	TRANSFORM	создание перекрестного запроса
	UNION	создание запроса на объединение

Запросы DDL предназначены для определения структуры данных, т. е. позволяют создавать домены, таблицы, индексы, представления, а также позволяют обеспечить целостность данных посредством задания ограничителей целостности.

Запросы DML различаются на запросы выборки, запросы действий и на специальные запросы. Результатом запроса выборки является некоторая таблица. Выходной набор может быть редактируемым, т. е. изменения, сделанные в выходном наборе, будут произведены и в таблицах, на которых основан запрос, если соблюдены определенные (различные для каждой СУБД) требования. Специальные запросы, к которым относятся запрос на объединение, запрос на создание перекрестной таблицы и некоторые другие, также возвращают таблицу как результат своего действия. Запросы действий, напротив, оказывают лишь некоторое действие над данными. Результат этого действия можно просмотреть, открыв соответствующие таблицы.

Кроме запросов DDL и DML можно выделить еще две группы запрсов: запросы управления доступом к данным (операторы GRANT и REVOKE) и запросы управления транзакциями (операторы START TRANSACTION, COMMIT, ROLLBACK).

Запросы SQL представляют собой инструкции, состоящие из фраз (предложений). Инструкции называются по имени оператора, определяющего суть инструкции. Такой оператор практически всегда следует первым. Фразы, в свою очередь, также называются по имени ключевого слова, с которого они начинаются.

6.1. ЗАПРОСЫ МАНИПУЛИРОВАНИЯ ДАННЫМИ

6.1.1. Запросы выборки

Основу запросов DML составляет инструкция SELECT.

SELECT [ALL|DISTINCT|DISTINCTROW|TOP] <список полей либо выражений>
FROM <список таблиц> [IN <имя базы данных>]
[<Таблица1> INNER|RIGHT|LEFT JOIN <Таблица2>
ON <Таблица1>.<Поле1> = <Таблица2>.<Поле2>]
[WHERE <условие отбора>]
[GROUP BY <список полей>] [HAVING <условие отбора>]
[ORDER BY <спецификация> [,<спецификация>] ...]

Инструкция SELECT позволяет производить выборку и вычисления над данными из одной или нескольких таблиц. Результатом выполнения инструкции является ответная таблица. Простейшая форма инструкции SELECT включает фразы SELECT и FROM. Фраза SELECT определяет поля, подлежащие выводу в выходной набор, а фраза FROM определяет имена таблиц, включенных в запрос. Имена полей и таблиц отделяются запятыми, а предложение запроса заканчивается точкой с запятой. Если в запросе участвует несколько таблиц, то для исключения двусмысленности имена полей следует записывать в полной форме: <таблица>.<поле> (например, Клиенты.Адрес). Для повышения эффективности вначале

указываются меньшие по размеру таблицы. Если имена полей и таблиц включают пробелы, то их необходимо заключать в квадратные скобки. Список полей следует задавать в той последовательности, в которой они должны быть представлены в выходном наборе. Например:

SELECT Фамилия, Имя, Отчество, [год рождения] FROM Клиенты;

Для выбора всех полей применяется шаблон *:

SELECT * FROM Клиенты;

Список полей или выражений может содержать имена полей, участвующих в запросе, выражения над столбцами, а также строковые константы, заключенные в кавычки. В выражениях (вычисляемых полях) могут принимать участие имена полей, знаки арифметических операций (+, – ,* ,/ ,^), множество встроенных функций, константы, круглые скобки и следующие операции:

\ – возвращает целое от деления двух арифметических выражений (заранее округленных);

MOD – возвращает остаток от деления двух арифметических выражений (заранее округленных);

& – операция конкатенации;

IIF(условие, выражение1, выражение2) – Если условие истинно, то возвращается результат выражения1, в противном случае – выражения2 (но в любом случае вычисляется как выражение1, так и выражение2). Например:

SELECT [Фамилия] & " " & [Имя] & " " & [Отчество] AS Полное_имя, "возраст", DateDiff("yyyy", [Дата Рождения], Date()) AS Возраст FROM Клиенты;

Именам полей и таблиц можно назначать альтернативные имена (псевдонимы). Псевдонимы записываются через ключевое слово AS (<имя таблицы> или <имя поля> AS <псевдоним>). В предыдущем примере псевдонимы использовались для задания имен вычисляемым полям. В противном случае имя вычисляемого поля имело бы вид: Выражение1 и т. д. В большинстве случаев псевдонимы используются для сокращения набора длинных имен. Особенно это эффективно для замены длинных имен таблиц, поскольку в многотабличных запросах приходится включать имена таблиц в описание каждого поля. Псевдонимы также используются при создании рекурсивных запросов, когда приходится соединять записи из одной и той же таблицы. Тогда для различия копий таблиц им приходится назначать псевдонимы.

Если в запрос включены не все поля некоторой таблицы, то выходной набор может содержать одинаковые строки. Предикаты ALL, DISTINCT, DISTINCTROW (записываются сразу после оператора SELECT) служат для управления выводом повторяющихся строк. По умолчанию принимается предикат ALL, т. е. в ответную таблицу включаются все строки, в

том числе и повторяющиеся. Предикат DISTINCT исключает записи, которые содержат повторяющиеся значения в выбранных полях. В выходной набор включаются только уникальные значения каждого из полей, находящихся в списке фразы SELECT. Если фраза SELECT содержит более одного поля, то для включения записи в результат выполнения запроса необходимо, чтобы совокупность значений во всех этих полях была уникальной. Предикат DISTINCTROW исключает полностью повторяющиеся записи и применяется для многотабличных запросов. Он игнорируется, если запрос содержит только одну таблицу или все поля всех таблиц. В Access может дополнительно применяться предикат TOP, возвращающий определенное число записей, находящихся в начале или в конце диапазона, описанного с помощью фразы ORDER BY.

В одном запросе SQL позволяет обращаться к данным нескольких баз данных различных форматов. Такие запросы получили название гетерогенных. В разных СУБД синтаксис записи гетерогенных запросов несколько отличается. В ACCESS для полного описания имени и типа таблицы применяется ключевое слово IN: имя таблицы IN "путь к файлу" "тип таблицы". Например:

SELECT * FROM Клиенты IN "c:\dBase\Clients.dbf" "dBASE".

В системах, ориентированных на доступ к данным через BDE (Borland Database Engine), для указания пути к таблице необходимо указать имя псевдонима базы (BDE Alias), заключенного в двоеточие с обеих сторон. Например: SELECT * FROM :BCDEMOS:Clients.

Фраза WHERE не обязательна. Она задает условия, которым должны удовлетворять записи в результирующей таблице. Выражение <условие отбора> является логическим. Его элементами могут быть имена полей, операции сравнения <, <=, >, >=, =, <>, арифметические операции, логические операторы (NOT, AND, OR, XOR), скобки, функции IN, BETWEEN, LIKE, IS (NOT) NULL и множество встроенных функций. Строки заключаются в кавычки, а константы типа Дата/Время – в символы #.

Функция IN проверяет на равенство любому значению из списка: [поле1] IN ("Минск", "Москва", "Киев");

Функция BETWEEN задает диапазон значений. Границы диапазона разделяются оператором And: [поле2] BETWEEN 50 And 100 (эквивалентно выражению [поле2] <=100 AND [поле2] >=50).

Функция LIKE проверяет на соответствие заданному шаблону символов. В качестве символов шаблона используются:

* – любое число произвольных символов;

? – один произвольный символ;

– одна произвольная цифра;

[] – диапазон допустимых символов. К примеру, [А-Я], [3-9]. Если же необходимо исключить символы, то перед ними ставится !: [!А-Я]. Например, LIKE "А*" – любая строка, начинающаяся с А, LIKE "220###" задает условие для почтового индекса, начинающегося с 220, LIKE "####АА[А-Х]" – отбирает номера автомобилей серий ААА, … , ААХ.

Запрос может быть основан на нескольких таблицах. Простое включение полей из нескольких таблиц дает простой перебор всех их возможных значений. Для двух таблиц общее число записей будет равно произведению числа записей в первой и второй таблицах. Но так как реляционная база данных практически всегда состоит из таблиц, связанных между собой посредством совпадающих значений полей, участвующих в связи, то для правильного объединения данных необходимо включать в запрос явное определение соответствующих связей. Связь можно задать с помощью двух способов: с помощью оператора INNER|RIGHT|LEFT JOIN и с помощью дополнительного условия выборки во фразе WHERE. Причем в SQL объединение данных можно произвести даже по неравенству, т. е. поддерживаются операции сравнения =, <>, <, <= ,> ,>=. Оператор INNER|RIGHT|LEFT JOIN является не обязательной частью инструкции SELECT и оформляется как часть фразы FROM:

SELECT Товары.[Наименование товара], Заказы.Дата, Заказы.[Полная цена] FROM Товары INNER JOIN Заказы ON Товары.Код_товара = Заказы.Код_товара;

Этот же запрос можно записать следующим образом (второй способ задания связи): SELECT Товары.[Наименование товара], Заказы.Дата, Заказы.[Полная цена] FROM Товары, Заказы WHERE Товары.Код_товара = Заказы.Код_товара;

Для большинства многотабличных запросов набор записей формируется на основе совпадающих значений полей, участвующих в связи, т. е. с помощью естественного, иначе внутреннего объединения (эквисоединение). Внутреннее соединение задается с помощью оператора INNER JOIN. Для двух таблиц, связанных отношением «один ко многим», из главной таблицы будут отобраны только те записи, которые имеют связанные записи в подчиненной таблице. Допустим, мы имеем две таблицы, «Клиенты» и «Заказы». С помощью внутреннего объединения получим сведения по клиентам, имеющим хотя бы один заказ. Если заказов у некоторых клиентов несколько, то сведения по ним повторятся столько раз, сколько было сделано заказов. Но предположим, что мы хотим получить информацию по всем клиентам и посмотреть, кто заказывал что-либо, а кто нет. Ответ на такой вопрос позволяют дать запросы с внешним объединением (LEFT|RIGHT [OUTER] JOIN). Тогда в запрос будут включены все записи из таблицы «Клиенты» и те записи из таблицы «Заказы», которые имеют связанные записи в главной таблице. Выбор LEFT либо RIGHT зависит от

того, с какой стороны от слова JOIN находится та таблица, из которой необходимо отобрать все записи:

SELECT Фамилия, Имя, [Дата заказа], Цена FROM Клиенты LEFT JOIN Заказы ON Клиенты.код_клиента = Заказы.код_клиента;

Этот же запрос можно записать и с правым объединением: Клиенты RIGHT JOIN Заказы. При задании ссылочной целостности такой запрос не имеет большого смысла, так как результат ничем не будет отличаться от внутреннего соединения, поскольку записей в подчиненной таблице, не связанных с записями в главной таблице, просто не может быть. Кроме соединения записей из нескольких таблиц, можно также провести рекурсивное соединение записей из одной и той же таблицы (используется, когда в таблице есть записи, которые ссылаются на другие записи этой же таблицы). Предположим, мы имеем таблицу о сотрудниках, где кроме полей *Фамилия, Имя*, *Должность* и т. д. есть поле *Подчиняется*, в котором записывается тот код сотрудника (первичный ключ данной таблицы), которому данный сотрудник подчиняется:

SELECT Сотр.Должность, Сотр.Фамилия, Сотр.Имя, Сотр1.Должность AS Подчиняется FROM Сотрудники As Сотр LEFT JOIN Сотрудники AS Сотр1 ON Сотр.Подчиняется = Сотр1.КодСотрудника; (в этом запросе целесообразно выполнить внешнее левое объединение для отображения и тех сотрудников, которые не подчиняются никому другому).

6.1.2. Запросы с группировкой

Иногда интерес представляет не каждая строка таблицы в отдельности, а итоговые значения по группам данных. Например, может понадобиться общая сумма продаж для клиентов, проживающих в определенном районе, или интересно знать средний объем продаж по месяцам, чтобы выяснить тенденции сбыта. Получить ответы на такие вопросы можно с помощью итоговых запросов (запросов с группировкой). Фраза GROUP BY позволяет выделять группы в результирующем множестве записей. Группой являются записи с совпадающими значениями в полях, перечисленных во фразе GROUP BY. Оператор перегруппирует таблицу таким образом, чтобы в каждой группе все строки имели одно и то же значение поля. Фраза SELECT затем применяется уже к группам перекомпонованной таблицы. Каждое выражение во фразе SELECT должно принимать единственное значение для группы, т. е. оно может быть либо самим полем, либо арифметическим выражением, включающем это поле, либо константой, либо агрегатной функцией, возвращающей единственное значение для группы. В запросах с группировкой необходимо тщательно следить за включением полей во фразу SELECT, так как в противном случае можно не получить желаемого результата. Если во фразу SELECT будет помещено хотя бы одно поле, которое не является единственным

для группы, например ключевое поле подчиненной таблицы, то создание групп будет прервано, так как в результате каждая строка запроса будет уникальна.

Группирование записей само по себе ничего не дает. Обычно производятся вычисления для групп. Для этой цели имеется ряд групповых (иначе агрегатных) функций, производящих следующие действия над значениями заданного поля (аргумента функции) для каждой группы:

SUM – вычисляет сумму;

AVG – вычисляет среднее арифметическое;

STDEV – вычисляет стандартное отклонение;

VAR – вычисляет дисперсию;

COUNT – вычисляет число записей, для которых значение заданного поля отлично от NULL. Для подсчета всех записей необходимо использовать операцию *. Например: Count(*);

MIN – возвращает минимальное значение;

MAX – возвращает максимальное значение;

FIRST – возвращает первое значение;

LAST – возвращает последнее значение.

Групповые функции могут также применяться сами по себе, без выполнения группировки. Тогда группой будут считаться все отобранные фразой WHERE записи. Например: SELECT Count(*) FROM Заказы; – произведет подсчет всех записей в таблице «Заказы»,

SELECT Sum([Отпускная цена] + [Транс издержки]) As Полная_цена FROM Заказы WHERE Город = "Минск"; – подсчитает полную сумму цен по всем заказам, сделанным из Минска,

SELECT Клиенты.Фамилия, Sum(Заказы.Цена) AS Стоимость FROM Клиенты INNER JOIN Заказы ON Клиенты.КодКлиента = Заказы.КодКлиента WHERE Клиенты.Город ="Минск" GROUP BY Клиенты.Фамилия; – подсчитает полную сумму цен по всем заказам для каждого клиента, проживающего в Минске.

Фраза HAVING используется для дополнительной селекции записей во время определения групп. Она выполняет те же функции, что и WHERE, но уже в рамках выходного набора. Фраза HAVING устанавливает, какие записи, сгруппированные посредством GROUP BY, должны отображаться и участвовать в групповых операциях. Правила записи <условия отбора> аналогичны правилам формирования <условия отбора> во фразе WHERE:

SELECT код_товара FROM Заказы GROUP BY код_товара HAVING Count(*) > 1; – отбирает коды товаров, заказываемых более чем одним покупателем.

Фраза ORDER BY замыкает инструкцию SELECT и задает порядок сортировки результирующего множества. Каждая спецификация сортировки представляет собой пару вида: <имя поля> [ASC/DESC]. Необязательный модификатор ASC задает сортировку по возрастанию, DESC – убыванию. Большинство СУБД требуют, чтобы поле, участвующее в сортировке, ьыло перечислено во фразе SELECT:

SELECT Товар, Цена FROM Товары ORDER BY Цена DESC;

6.1.3. Параметрические запросы

До сих пор условие отбора мы записывали явно во фразе WHERE. Но во многих случаях оно становится известным только во время выполнения программы. SQL позволяет вводить условия отбора в виде параметров запроса. В этом случае такие запросы называются параметрическими или динамическими. Для увеличения эффективности выполнения таких запросов в некоторых СУБД можно вызвать операцию PREPARE, которая подготовит запрос к запуску, т. е. зарезервирует необходимую память и проведет его оптимизацию. Тогда сразу после задания параметра запрос будет готов к запуску. Для освобождения зарезервированной памяти в таком случае применяется операция UNPREPARE.

Для задания параметра в Access некоторый текст, являющийся именем параметра, необходимо заключить в квадратные скобки. Этот же текст будет выводится в виде приглашения в окне задания значения параметра. Имя параметра должно отличаться от названий полей таблиц, включенных в запрос. Дополнительно можно явно определить типы параметров с помощью инструкции PARAMETERS в виде [имя параметра] тип данных, [имя параметра] тип данных, и т. д., что позволяет контролировать значения параметров на соответствие типу еще при их вводе. В таком случае инструкция PARAMETERS располагается перед описанием запроса:

PARAMETERS [Введите год:] INT;

SELECT Клиенты.Фамилия FROM Клиенты INNER JOIN Заказы ON Клиенты.КодКлиента = Заказы.КодКлиента WHERE Year(Заказы.Дата) = [Введите год:];

6.1.4. Вложенные запросы

Инструкции SELECT могут многократно вкладываться друг в друга. Вложенная инструкция SELECT записывается как часть фразы WHERE и служит для отбора записей основного запроса. SQL выполняет вложенный подзапрос и затем сравнивает каждую строку основного запроса с результатом вложенного. Вложенные запросы записываются внутри скобок. Например: SELECT Фамилия, Имя FROM Клиенты WHERE Кредит < (SELECT AVG(Кредит) FROM Клиенты);

Если подчиненный запрос возвращает набор записей, то вместо операторов сравнения можно использовать функции ALL, SOME, ANY,

EXISTS, IN, получившие по функции EXISTS название кванторов. Квантор существования EXISTS обычно записывается следующим образом: EXISTS(SELECT * FROM …). Перед ним можно поставить NOT для инверсии результата. Выражение EXISTS считается истинным тогда и только тогда, когда результат вычисления подзапроса является непустым множеством. В запросах данного типа необходимо явно определять связь между таблицами в виде условия отбора фразы WHERE вложенного запроса. В качестве примера выберем фамилии покупателей, которым был продан компьютер «Pentium II 350»:

SELECT Фамилия FROM Клиенты WHERE EXISTS(SELECT * FROM Заказы WHERE Заказы.код_клиента = Клиенты.код_клиента And [Наименование товара] = "Pentium II 350");

Кванторы SOME и ANY (синонимы) используются для отбора в главном запросе записей, которые удовлетворяют сравнению с записями, отобранными подчиненным запросом. Например:

SELECT * FROM Товары WHERE Цена > ANY(SELECT Цена FROM Заказано WHERE Скидка >= 0.25); Этот запрос отбирает все товары, цена которых больше, чем цена любого товара (т. е. самого недорогого), проданного со скидкой 25 %.

Квантор ALL используется для отбора в главном запросе записей, которые удовлетворяют сравнению со всеми записями, отобранными подчиненным запросом:

SELECT Марка FROM Товары WHERE Цена > ALL(SELECT Цена FROM Заказано WHERE Скидка >= 0.25); Этот запрос отбирает все товары, цена которых больше, чем цена всех товаров (т. е. самого дорогого), проданных со скидкой 25 %.

Квантор IN используются для отбора в главном запросе только тех записей, которые содержат значения, совпадающие с одним из отобранных подчиненным запросом:

SELECT Марка FROM Товары WHERE Цена IN(SELECT Цена FROM Заказано WHERE Скидка >= 0.25); Этот запрос отбирает все товары, цена которых совпадает с ценой товаров, проданных со скидкой 25%.

6.2. ЗАПРОСЫ ДЕЙСТВИЙ

С помощью запросов действий пользователь может изменять или переносить данные в таблицах, обновлять, добавлять или удалять группы записей, а также создавать новые таблицы. Некоторые возможности редактирования существуют и в запросах выборки, т. е. изменения, сделанные в запросе, автоматически переносятся на базовые таблицы запроса. Но редактируемыми могут быть только некоторые запросы. Это простые однотабличные запросы выборки без применения предикатов, группировки и агрегатных функций.

Существуют четыре запроса действий: на добавление записей, на удаление записей, на обновление записей и на создание таблицы.

6.2.1. Запрос на обновление

С помощью запроса на обновление можно изменить группу записей, отобранных по заданному критерию. Инструкция запроса на обновление имеет формат вида:

UPDATE <имя таблицы> SET <имя поля> = {<выражение> , NULL} [, SET <имя поля > = {<выражение> , NULL } ...]

[WHERE <условие отбора>]

Новые значения полей в записях могут быть пустыми (NULL), либо вычисляться в соответствии с арифметическим выражением. Правила записи арифметических и логических выражений аналогичны соответствующим правилам для вычисляемых полей. Например:

UPDATE Товары
SET Наименование = "Pentium III 800", Цена = Цена + 250
WHERE Наименование = "Pentium II 350";

6.2.2. Запрос на добавление

С помощью запроса на добавление записи одной таблицы (все или отобранные запросом) можно добавить в конец другой таблицы либо просто добавить в таблицу всего одну запись, состоящую из литералов. Соответственно запрос на добавление имеет форматы двух видов:

INSERT INTO <имя таблицы> [(<список полей>)] VALUES (<список значений>) и

INSERT INTO <имя таблицы> [(<список полей>)] <инструкция SELECT>

В первом формате инструкция INSERT выполняет ввода одной записи, для которой значения полей задаются литералами или выражениями. Порядок перечисления полей должен соответствовать порядку значений, перечисленных в списке значений фразы VALUES. При явном перечислении можно опускать задание некоторых полей. Если список полей опущен, то в списке значений должны быть перечислены все значения в порядке следования полей таблицы. Во втором формате инструкция INSERT предназначена для добавления записей, отобранных из другой таблицы с помощью инструкции SELECT (таблицы должны быть разными). Здесь также необходимо обеспечить соответствие полей (типов и размеров полей или, по крайней мере, возможность полноценной конвертации данных), перечисленных как после оператора INSERT INTO, так и после SELECT. Например:

INSERT INTO Товары(Наименование, Цена) VALUES ("Pentium II 233", 450);

INSERT INTO Клиенты(Фамилия, Имя, Отчество) SELECT Фамилия, Имя, Отчество FROM Поставщики WHERE Город = "Минск";

6.2.3. Запрос на удаление

С помощью запроса на удаление можно сразу удалить группу записей, удовлетворяющих определенному критерию. Этот запрос особенно эффективен при удалении большого числа записей. С помощью запроса на удаление можно явно удалить записи только из одной таблицы. Но если было определено каскадное удаление, то будут также удалены связанные записи из подчиненных таблиц. Запрос на удаление имеет формат вида:

DELETE FROM <имя таблицы> [WHERE <условие отбора>]

Если необязательная фраза WHERE опущена, т. е. условие отбора удаляемых записей отсутствует, удалению подлежат все записи таблицы. Например:

DELETE FROM Товары WHERE Наименование LIKE "Celeron*";

6.2.4. Запрос на создание новой таблицы

С помощью этого запроса можно создать новую таблицу на основе уже существующей и перенести в нее все или удовлетворяющие некоторому критерию записи. Новая таблица будет иметь ту же структуру. Обычно этот запрос применяется для создания архивных копий таблиц. Запрос на создание новой таблицы имеет формат вида:

SELECT <список полей> INTO <имя новой таблицы>

FROM <имя таблицы> [WHERE <условие отбора>].

В новую таблицу будут перенесены поля, перечисленные в списке полей инструкции SELECT, для всех записей, удовлетворяющих условию отбора. Например:

SELECT Клиенты.Фамилия, Клиенты.Имя, Клиенты.Отчество INTO Совершеннолетние FROM Клиенты WHERE Клиенты.[год рождения]>#1984/01/04#;

6.3. СПЕЦИАЛЬНЫЕ ЗАПРОСЫ

6.3.1. Перекрестные запросы

Инструкция TRANSFORM служит для создания перекрестных запросов. Они позволяют создавать перекрестные (сводные) таблицы, в которой можно было бы просмотреть зависимость некоторых данных от двух параметров, один из которых откладывается по строкам, а второй – по

столбцам. В общем случае инструкция TRANSFORM записывается следующим образом:

TRANSFORM <выражение с групповой функцией (вычисляет значения ячеек)>

SELECT <инструкция с применением GROUP BY (отбирает записи и задает имена строк)>

PIVOT <выражение, задающее имена столбцов>.

К примеру, нам надо определить объем выручки от продажи каждого товара по месяцам. Тогда каждая строка полученной таблицы (см. табл. 3) будет соответствовать определенному товару, в качестве столбцов будут названия месяцев, а в ячейках таблицы будут представлены соответствующие объемы продаж.

Перекрестная таблица *Таблица 3*

Товар	Дата			
	Янв 2007	Февр 2007	Март 2007	Апр 2007
Молоко	4363	674	8555	12
Мясо	34636	8654	22	436
Птица	3363	54375	25252	131
Рыба	33656	34633	2554	0

TRANSFORM Sum(Заказы.Цена*Заказы.Количество) AS СуммаПоТовару SELECT Товары.Товар FROM Товары INNER JOIN Заказы ON Товары.КодТовара = Заказы.КодТовара WHERE Заказы.[Дата заказа] BETWEEN #2002/1/1# And #2002/30/4# GROUP BY Товары.Товар PIVOT Format([Дата заказа],"mmm yyyy");

В выражении PIVOT также можно применить функцию DatePart("m",[ДатаРазмещения],1,0), возвращающую номер месяца.

6.3.2. Запрос на объединение

Оператор UNION позволяет объединить выходные наборы нескольких инструкций SELECT в одну результирующую таблицу (так называемое вертикальное объединение). Этот запрос является аналогом операции объединения отношений $R = A \cup B$ в реляционной алгебре. При объединении записи возвращаемые второй и последующими инструкциями SELECT, будут дописываться в конец первой, причем из результата выборки всегда будут исключаться повторяющиеся записи. В выходных наборах всех инструкций SELECT должно быть одинаковое количество полей с одинаковыми характеристиками (последнее требование в некоторых СУБД не является обязательным). В крайнем случае некоторую инструкцию SELECT можно дополнить строковыми константами или вычисляемыми полями. В качестве заголовков столбцов для выходного набора будут приниматься имена полей первой инструкции. Для выпол-

нения сортировки результирующего набора данных объединение может дополняться фразой ORDER BY, которая записывается в конце последней инструкции SELECT:

SELECT Фамилия & " " & Имя & " " & Отчество As Название, Город FROM Клиенты

UNION SELECT Поставщик, Город FROM Поставщики

ORDER BY Название, Город;

Кроме UNION в стандарте SQL2 включены еще две операции реляционной алгебры: исключения – EXCEPT (аналог операции разности $R = A - B$. Отношение R включает записи, присутствующие в A и отсутствующие в B) и пересечения – INTERSECT (аналог операции пересечения $R = A \cap B$. Отношение R включает записи, присутствующие как в A, так и в B). Как и для объединения, операнды (таблицы) этих операций должны содержать соответствующий по типам набор полей.

6.4. ЗАПРОСЫ ОПРЕДЕЛЕНИЯ ДАННЫХ

6.4.1. Создание представлений

На основе запроса выборки можно построить представление. В SQL представление является виртуальной таблицей построенной на основе данных одной или нескольких таблиц, т. е. запрос выборки может быть многотабличным. По одним и тем же таблицам можно построить несколько представлений. Поведение представлений отличается от поведения запросов. Представление всегда содержит только «свежие» данные. Любые изменения в таблицах немедленно отражаются и в представлении. Забота об обновлении данных лежит на СУБД. Таким образом, представление дает возможность работы с выделенными данными как с некоторой локальной таблицей. Как и запросы, представления не всегда могут быть редактируемыми. Как правило, редактируемыми могут быть только представления, основанные на одной таблице, без использования предикатов, группировки и агрегатных функций. В некоторых СУБД, например в Access, не делается различия между запросом и таблицей. Это позволяет основывать запросы на других запросах, что внешне напоминает работу с представлениями.

Само представление описывается путем указания идентификатора представления и запроса, который должен быть выполнен для его получения. Инструкция создания представления имеет следующий формат:

CREATE VIEW <имя представления>

[(<имя поля> [,<имя поля>]...)] AS <инструкция SELECT>

Если имена столбцов в представлении не указываются, то будут использоваться имена столбцов из запроса, описываемого соответствующим

оператором SELECT. Инструкция удаления представления имеет формат вида: DROP VIEW <имя представления>.

Использование представлений для однопользовательской БД имеет целью лишь упрощение структуры запросов. Однако для многопользовательской сетевой СУБД представления играют ключевую роль в определении доступа к данным и защите информации. Использование представлений дает следующие преимущества.

1. Независимость от данных. С помощью представлений можно создать согласованную, неизменную картину структуры БД, которая будет оставаться стабильной даже в случае изменения (незначительного) исходных таблиц.

2. Актуальность. Представление содержит только «свежие» данные.

3. Повышение защищенности данных. Каждому пользователю может быть предоставлен ограниченный набор представлений, дающих доступ только к определенной информации.

4. Снижение сложности. Использование представлений позволяет упростить структуру запросов.

5. Возможность индивидуальной настройки. Каждый пользователь может работать только с теми данными, которые ему действительно нужны, и к тому же в определенной форме.

К недостаткам можно отнести снижение производительности, наличие структурных ограничений и ограниченные возможности обновления.

6.4.2. Создание доменов

Инструкция создания домена имеет формат вида:

CREATE DOMAIN <имя домена> [AS] тип данных [DEFAULT <значение по умолчанию>] [CHECK (<условия целостности>)]

SQL (стандарт SQL2) поддерживает пять скалярных типов данных: символьный (строковый), битовый, численный, даты/время и интервал. Символьный тип данных позволяет определить строки фиксированной (CHAR(n) или CHARACTER(n)) и переменной длины (VARCHAR(n) или CHAR VARYING(n) или CHARACTER VARYING(n)), где n означает максимальное количество символов. Спецификация VARYING позволяет сохранять в базе только вводимые символы, иначе введенная строка дополняется пробелами до n символов. Численный тип данных позволяет задавать целые числа разного размера (SMALLINT, INT или INTEGER) и вещественные числа разной точности (FLOAT [precision], REAL, DOUBLE PRECISION, NUMERIC(precision, scale), DECIMAL(precision, scale) или DEC (precision, scale)), где precision определяет полное число десятичных цифр, а scale – число цифр после десятичной точки. Битовые типы данных позволяют хранить последовательности двоичных цифр фиксированной (BIT(n)) и переменной длинны (BIT VARYING(n)). Для

хранения данных типа даты/время введены три типа данных: DATE, TIME и TIMESTAMP. Типы DATE и TIME позволяют хранить соответственно дату и время по отдельности, а TIMESTAMP – вместе. Тип данных INTERVAL определен для хранения интервалов времени. При определении доменов типы данных задаются после оператора ключевого слова AS, например: Name AS VARCHAR(20).

Условие проверки CHECK позволяет задать ограничители целостности на значения, которые может принимать домен. Например:

CREATE DOMAIN sex_type AS CHAR CHECK (VALUE IN ('M', 'F'));

Значения в операторе IN могут также выбираться и из некоторой таблицы, например IN (SELECT s_type FROM STypes).

Изменить определение домена можно с помощью инструкции ALTER DOMAIN. Удалить созданный домен можно с помощью инструкции DROP DOMAIN, имеющего следующий формат записи:

DROP DOMAIN <имя домена> [RESTRICT|CASCADE]

Опция CASCADE позволяет после удаления домена изменить все типы полей, основанных на этом домене, на соответствующий тип данных и произвести необходимую их конвертацию, насколько это будет возможно.

6.4.3. Создание таблиц

Инструкция создания таблицы имеет формат вида:

CREATE TABLE <имя таблицы> (<имя поля> <тип данных> [NOT NULL] [,<имя поля> <тип данных> [NOT NULL]] ...)

Обязательными операндами инструкции являются имя создаваемой таблицы и имя хотя бы одного поля с указанием типа данных. При создании таблицы для отдельных полей могут указываться некоторые дополнительные правила контроля вводимых в них значений. Конструкция NOT NULL требует, чтобы в этом столбце должно быть определено значение. Например: CREATE TABLE Товары(Код CHAR(5) NOT NULL, Тип CHAR(8), Наименование VARCHAR(20) NOT NULL, Цена DECIMAL(8,2));

Инструкция изменения структуры таблицы имеет формат вида:

ALTER TABLE <имя таблицы> ({ADD, MODIFY, DROP} <имя поля> [<тип данных>] [NOT NULL] [,{ADD, MODIFY, DROP} <имя поля> [<тип данных>] [NOT NULL]]...)

Изменение структуры таблицы может состоять в добавлении (ADD), изменении (MODIFY) или удалении (DROP) одного или нескольких столбцов. Правила записи инструкции ALTER TABLE такие же, как и инструкции CREATE TABLE, разве что при удалении столбца указывать тип данных не требуется. Для примера добавим одно поле:

ALTER TABLE Товары(ADD Категория VARCHAR(20));

Инструкция удаления таблицы имеет формат вида:

DROP TABLE <имя таблицы>.

Инструкции создания и изменения таблицы имеют и более сложный вид записи, позволяющий не только задать ограничители целостности значений, но и определить ссылочную целостность связанных таблиц. Ограничители целостности можно также задать отдельно с помощью оператора ASSERTION. Расширенный вариант инструкции создания таблицы имеет формат вида:

CREATE TABLE <имя таблицы>

{<имя поля> <тип данных> [NOT NULL] [UNIQUE] [DEFAULT <значение по умолчанию>] [CHECK (<условия целостности>)] [, …]}

[PRIMARY KEY (<список полей>),]

{[UNIQUE (<список полей>),] [, …]}

{[FOREIGN KEY (<список внешних полей>)] REFERENCES <имя базовой таблицы> [<список ключевых полей базовой таблицы>] MATCH {PARTIAL | FULL} [ON UPDATE <действие>] [ON DELETE < действие >] [, …]} {[CHECK (<условия целостности>)] [, …]}.

Фраза PRIMARY KEY определяет первичный ключ таблицы. Фраза UNIQUE позволяет определить альтернативные (потенциальные) ключи. Фразы FOREIGN KEY и REFERENCES используются для задания связей между таблицами. Можно дополнительно определить каскадное удаление и каскадное обновление. Для выполнения ссылочной целостности SQL определяет четыре вида действий: CASCADE, SET NULL, SET DEFAULT, NO ACTION (используется по умолчанию). Фраза CHECK служит для задания дополнительных условий для значений полей таблицы. Вышеперечисленные ограничители целостности могут дополнительно предваряться фразой CONSTRAINT <имя ограничителя целостности>, что позволит впоследствии отменить это ограничение в операторе ALTER TABLE. Например,

CREATE TABLE Заказы (КодЗаказа INTEGER PRIMARY KEY, КодКлиента INTEGER, ДатаЗаказа DATE, ПримечанияЗаказа VARCHAR(255), CONSTRAINT ВнКлЗаказыКодКлиента FOREIGN KEY (КодКлиента) REFERENCES Клиенты ON UPDATE CASCADE ON DELETE CASCADE);

Расширенный вариант инструкции изменения таблицы имеет формат вида:

ALTER TABLE <имя таблицы>

[ADD <имя поля> <тип данных> [NOT NULL] [UNIQUE] [DEFAULT <значение по умолчанию>] [CHECK (<условия целостности>)]]

[DROP <имя поля> [RESTRICT | CASCADE]]

[ADD CONSTRAINT <имя ограничителя> PRIMARY KEY | UNIQUE | FOREIGN KEY | CHECK]

[DROP CONSTRAINT <имя ограничителя> [RESTRICT | CASCADE]]

[ALTER SET DEFAULT <значение>]
[ALTER DROP DEFAULT]

6.4.4. Создание индексов

Инструкция создания индекса имеет формат вида:

CREATE [UNIQUE] INDEX <имя индекса> ON <имя таблицы> (<имя поля> [ASC | DESC] [,<имя поля> [ASC | DESC]...) [WITH {PRIMARY | DISALLOW NULL | IGNORE NULL}]

Инструкция позволяет создать индекс для одного или нескольких столбцов заданной таблицы. Для одной таблицы можно создать несколько индексов. Индексы можно создавать только для таблиц БД, но не для представлений. Задав необязательную опцию UNIQUE, можно обеспечить уникальность значений во всех указанных в операторе полях. По существу, создание индекса с указанием признака UNIQUE означает определение потенциального ключа в созданной ранее таблице. С помощью зарезервированного слова PRIMARY можно объявить индексированные поля первичным ключом. В этом случае опцию UNIQUE можно опустить, поскольку уникальность все равно будет обеспечена. При создании индекса можно задать порядок сортировки значений в столбцах – в порядке возрастания ASC (по умолчанию) или в порядке убывания DESC. Для разных полей можно задавать различный порядок сортировки. Например:

CREATE INDEX AscName ON Клиенты(Фамилия, Имя, Отчество);

С помощью ключевого слова WITH можно запретить значения NULL в индексированных полях новых записей (DISALLOW NULL) или запретить включение в индекс записей, имеющих значения NULL в индексированных полях (IGNORE NULL).

Инструкция удаления индекса имеет формат вида:

DROP INDEX <имя индекса>.

6.5. ИСПОЛЬЗОВАНИЕ ТРАНЗАКЦИЙ

Транзакцией называется логическая единица работы, состоящая из одной или более инструкций SQL, которые с точки зрения воздействия на БД рассматриваются и обрабатывается системой как единое неделимое действие. Результаты действия всех инструкций, входящих в транзакцию, либо полностью принимаются, либо полностью отвергаются.

Обычно в SQL транзакция автоматически запускается любым оператором манипулирования данных либо вызовом оператора BEGIN TRANSACTION. Завершение транзакции может быть выполнено одним из следующих действий:

а) вызовом оператора COMMIT, означающим успешное завершение транзакции. После него все изменения в таблицах БД гарантированно фиксируются во внешней памяти;

б) вызовом оператора ROLLBACK, означающим откат транзакции, в результате чего выполняется откат всех изменений в БД, внесенных при выполнении этой транзакции;

в) выходом из процедуры, содержащей несколько инструкций SQL.

В последнем случае, если при выполнении всех входящих в процедуру инструкций SQL не произойдет ни одной ошибки, то будет автоматически вызван оператор COMMIT, в противном случае – ROLLBACK. С помощью оператора SET TRANSACTION можно установить тип транзакции (READ ONLY или READ WRITE) и уровень изоляции данных, необходимый для того, чтобы гарантировать отсутствие конфликтных ситуаций при параллельном выполнении нескольких транзакций. Точнее, требуется, чтобы обновления, выполняемые данной транзакцией *T1*, не были доступны для любой другой транзакции *T2* до тех и только до тех пор, пока не будет завершено выполнение транзакции *T1*. Можно задать следующие уровни изоляции данных: READ UNCOMITED (незавершенное считывание), READ COMMITED (завершенное считывание), REPEATABLE READ (повторяемое считывание), SERIALIZABLE. Полная безопасность данных гарантируется только при использовании уровня SERIALIZABLE (способность к упорядочению), который предусматривает составление сериального плана выполнения параллельных транзакций.

6.6. УПРАВЛЕНИЕ ДОСТУПОМ К ДАННЫМ

Каждая СУБД должна предоставлять механизм, гарантирующий, что доступ к БД смогут получить только те пользователи, которые имеют на это соответствующее разрешение. Язык SQL включает два оператора GRANT и REVOKE, предназначенные для защиты данных от неавторизированного доступа. Применяемый механизм защиты основан на использовании идентификаторов пользователей, предоставляемых им прав владения и привилегий. Каждому пользователю БД назначается собственный идентификатор (обычно определяется именем пользователя и паролем), который применяется для определения того, на какие объекты БД может ссылаться данный пользователь, а также какие операции с этими объектами он имеет право выполнять. Таким образом, каждый оператор SQL выполняется от имени некоторого пользователя исходя из назначенных ему прав доступа (привилегий).

Каждый объект в SQL имеет своего владельца (часто пользователя, который его создал). Только владелец имеет полный доступ к объекту.

Для предоставления прав доступа другим пользователям к этому объекту применяется оператор GRANT, имеющий следующий формат:

GRANT {<список привилегий> | ALL PRIVILEGES} ON <имя объекта> TO {<список идентификаторов пользователей> | PUBLIC} [WITH GRANT OPTIONS]

В SQL определен следующий набор привилегий, позволяющий:

SELECT – просматривать данные;

INSERT [<список полей>] – добавлять данные;

UPDATE [<список полей >] – обновлять данные;

DELETE – удалять данные;

REFERENCE [<список полей >] – ссылаться на указанные поля при определении ссылочной целостности;

USAGE – использовать домены и ограничители целостности.

После списка идентификаторов пользователей можно указать опцию WITH GRANT OPTIONS, которая позволит этим пользователям в свою очередь назначать привилегии другим пользователям. Таким образом, владелец может четко контролировать, кто получил права доступа к объекту и какие права доступа ему предоставлены.

Для отмены указанных привилегий применяется оператор REVOKE:

REVOKE [GRANT OPTION FOR] {<список привилегий> | ALL PRIVILEGES} ON <имя объекта> FROM {<список идентификаторов пользователей> | PUBLIC} [RESTRICT | CASCADE]

Опция GRANT OPTION FOR позволяет для всех заданных привилегий отменять возможность их передачи другим пользователям.

Контрольные вопросы и задания

1. К какому виду реляционных языков относится SQL: к реляционной алгебре или реляционному исчислению?
2. На какие группы можно разбить основные операторы SQL?
3. Перечислите операторы языка описания запросов выборки в SQL.
4. Какие виды вычислений можно производить в SQL?
5. Исправьте ошибку(и) в запросе: SELECT Фамилия, Имя, Отчество FROM Клиенты INNER JOIN Заказы AS Клиенты = Заказы SORT BY Фамилия WHERE Город = Минск;
6. Что сделает следующий запрос действия (каскадное удаление не установлено): DELETE * FROM Клиенты INNER JOIN Заказы ON Клиенты.КодКлиента = Заказы.КодКлиента;

а) удалит из таблицы «Клиенты» информацию о тех клиентах, которые имеют заказы;

б) удалит из таблицы «Клиенты» информацию о тех клиентах, которые имеют хотя бы один заказ, и сделанные ими заказы из таблицы «Заказы»;

в) завершится выдачей сообщения о том, что данный запрос не может быть выполнен.

7. Чем отличается внутреннее (естественное) соединение от внешнего. Напишите два запроса: один с внутренним объединением, один с внешним, между таблицами «Клиенты» и «Заказы»?
8. Что позволяют запросы с группировкой?
9. Какой из запросов действия имеет две формы записи?
10. Напишите запрос выборки фамилии, имени и отчества всех клиентов, которые покупали товары в текущем году на языке SQL. Записи должны быть упорядочены по возрастанию по полю «Фамилия». Повторяющиеся записи должны быть исключены.
11. Что возвратит следующий запрос: SELECT КодКлиента, SUM(Цена*Количество) FROM Клиенты INNER JOIN Заказы ON Клиенты.КодКлиента = Заказы.КодКлиента WHERE EXISTS(SELECT * FROM Товары WHERE НаименованиеТовара = "Молоко" And Товары.КодТовара = Заказы.КодТовара) GROUP BY КодКлиента;
12. Что возвратит следующий запрос (связи между таблицами заданы): SELECT * FROM Клиенты, Заказы;

а) информацию о клиентах и сделанных ими заказах с учетом связи, определенной в окне схемы данных,

б) информацию о клиентах, затем информацию о товарах,

в) любые возможные сочетания записей из таблицы «Клиенты» и записей из таблицы «Заказы».

13. Перечислите основные операторы языка определения данных в SQL.
14. Какие типы данных поддерживаются в SQL?
15. Напишите запрос создания таблицы «Клиенты».
16. Перечислите достоинства и недостатки представлений в SQL.
17. Какие объекты базы данных позволяют создавать операторы языка определения данных в SQL?
18. Как организуется защита данных от неавторизированного доступа в SQL.

7. АРХИТЕКТУРА БАЗ ДАННЫХ

В данной главе мы кратко рассмотрим архитектуру доступа к данным ведущего производителя программного обеспечения Microsoft, а также основные аспекты распределенного хранения данных и доступа к ним в многопользовательском режиме. Интерес к распределенному совместному доступу к данным возник не случайно. Сегодня любая крупная организация содержит большой штат работников и географически распределена. Доступ же к общим корпоративным данным может потребоваться каждому в любой момент времени. Простое дублирование информации на каждом персональном компьютере не решает проблему, поскольку данные могут динамично меняться. К тому же хранение больших объемов данных на каждом компьютере очень ресурсоемко. Решение подобных проблем требует серьезного развития технологий совместного доступа к данным и систем управления и обработки данных.

7.1. ТЕХНОЛОГИИ ДОСТУПА К ДАННЫМ

Стандартные технологии доступа к данным. В первых версиях операционной системы Windows пользователи могли совместно использовать данные в разных приложениях, копируя и вставляя их с помощью буфера обмена (clipboard). Затем был предложен протокол обмена данными Dynamic Data Exchange (DDE) для более динамичного режима обмена данными. Однако он функционировал медленно и ненадежно, и на смену ему был разработан значительно более эффективный протокол связывания и внедрения объектов Object Linking and Embedding (OLE).

OLE – это технология, которая позволяет создавать составные приложения, включающие в себя объекты, созданные с помощью других приложений. Объекты могут быть встроены в основное приложение или просто быть связаны с ним. Приложение, включающее в себя другие объекты, называется контейнером OLE, а приложение, поставляющее свои объекты для встраивания либо связывания – сервером OLE. Объектами OLE-приложения могут быть текстовые документы, диаграммы, электронные таблицы, графические изображения т. д. После вставки или внедрения объект отображается внутри клиентского приложения и хранится вместе с ним. Причем для редактирования связанных данных пользователю достаточно дважды щелкнуть мышью на встроенном объекте, в результате чего будет запущено приложение, в котором этот объект был создан. При связывании объектов в контейнере хранится лишь ссылка на объект-источник. После обновления исходного файла объекта обновляется и его представление в составном приложении. Помимо встраивания и связывания объектов, эта технология позволяет вызывать функции одного

приложения из другого. Другими словами, OLE является объектно-ориентированной технологией разработки повторно используемых программных компонентов.

В целях дополнительной интеграции объектов концепция OLE была значительно расширена, позволив создавать самостоятельные функциональные компоненты, предоставляющие свои функции (сервисы) другим объектам. В такой архитектуре создание и сопровождение одних объектов может производиться совершенно независимо от других объектов. Взаимодействие объектов определятся посредством специально организованных интерфейсов. Компонентная модель объектов Component Object Model (COM) является объектно-ориентированной моделью, состоящей из спецификации, определяющей интерфейс между объектами внутри системы, и конкретной реализации в виде динамически связываемой библиотеки Dynamic Link Library (DLL). Технология COM предоставляет стандартный метод поиска и инициализации объектов, а также организации связи между приложением и объектом. Одним из основных достоинств технологии COM является то, что она предоставляет унифицированный (двоичный) стандарт взаимодействия, не зависящий от языка программирования, использовавшегося при создании приложения и объекта. Идеология COM была реализована в 1993 г. в спецификации OLE 2.0.

COM позволяет создавать централизованные приложения. Для создания распределенных корпоративных систем была предложена архитектура Distributed Component Object Model (DCOM). DCOM расширяет архитектуру COM до распределенной компонентной среды, в которой компоненты одинаково выглядят для клиентов на локальном и удаленном компьютерах. DCOM реализует это, заменяя сообщение между процессами клиента и компонента соответствующим сетевым протоколом.

Технология Open Database Connectivity (ODBC). Технология открытого доступа к данным ODBC была разработана фирмой Microsoft для обеспечения возможности взаимосвязи между различными SQL-совместимыми БД, причем в этой технологии SQL используется как стандартный механизм доступа к данным. Необходимость создания ODBC появилась вследствие того, что каждая фирма-разработчик СУБД использовала свой диалект SQL, что делало невозможным обмен данными между двумя БД различных форматов. Поэтому вначале был разработан общий стандарт на SQL, получивший название CLI (Call Level Interface). В его основу были положены уже существующие стандарты X/Open и ISO. Затем каждой фирме-разработчику СУБД было предписано разработать драйвер перевода своего диалекта SQL в CLI, и наоборот.

Таким образом, основное назначение ODBC состоит в абстрагировании приложения от особенностей ядра используемой БД. Технология ODBC предусматривает создание дополнительного уровня между прило-

жением и используемой СУБД. Предоставленный интерфейс обеспечивает высокую степень взаимодействия, позволяя одному приложению обращаться к разным базам данных с помощью одного и того же кода. Это позволяет создавать распределенные (преимущественно клиент-серверные) гетерогенные приложения без учета особенностей конкретных СУБД. В качестве сервера может выступать любой сервер БД, имеющий драйвер ODBC или даже обычная БД, если требуется совместная обработка данных, написанных в разных форматах. ODBC находится как бы посередине между приложениями и используется как средство коммуникации между клиентской и серверной частями. Службы ODBC обеспечивают соединение с БД, получение от приложения запросов на выборку информации и перевод их на язык ядра адресуемой базы данных для доступа к хранимой в ней информации. Одна из главных целей создания ODBC – скрыть сложность соединения с сервером и по мере возможности автоматизировать выполнение многочисленных процедур, связанных с получением данных. ODBC требует от разработчика указания только имени источника данных (DSN – data source name), при этом функции, драйверы, адреса серверов, сети и шлюзы скрыты от пользователя.

Достоинством технологии ODBC является простота разработки приложений, обусловленная высоким уровнем абстрактности интерфейса доступа к данным практически любых существующих типов СУБД. Основной недостаток технологии ODBC связан с необходимостью трансляции запросов, что снижает скорость доступа к данным.

Технология Object Linking and Embedding Database. Реляционные БД – не единственный источник данных. Данные могут быть представлены в любом виде и формате. Например, в качестве данных могут выступать объектно-ориентированные БД, электронные таблицы, документы в RTF, XML формате, почтовые системы и т. д. Соответственно возникла потребность либо создать единый формат хранения данных, что дорого и неэффективно, либо нарастить имеющиеся технологии интерфейсами доступа к любым типам данных. Технология OLE DB реализует это требование, являясь более универсальной, нежели стандартные технологии OLE и COM.

В технологии OLE DB используется механизм провайдеров, под которыми понимают поставщиков данных, находящихся в надстройке над физическим форматом данных. Провайдер OLE DB представляет собой компонент COM, позволяющий принимать вызовы OLE DB API и выполнять все необходимое для обработки запроса к источнику данных. Кроме поставщиков данных имеются также сервис-провайдеры, реализующие самые различные сервисные функции. Технология OLE DB может использовать ODBC для доступа к реляционным БД. В этом случае применяется OLE DB-провайдер для доступа к ODBC данным. Таким образом,

технология OLE DB не заменяет технологию ODBC, она позволяет организовывать доступ к источникам данных через различные интерфейсы и в том числе через ODBC.

Технологии Data Access Objects и ActiveX Data Objects. Хотя ODBC и OLE DB считаются хорошими интерфейсами передачи данных, но как программный интерфейс они имеют много ограничений, поскольку являются низкоуровневыми. Для снятия этих ограничений были предложены технологии Data Access Objects (DAO) и ActiveX Data Objects (ADO). Данные технологии представляют собой высокоуровневые объектные модели (библиотеки функций) и создают еще один уровень абстракции между приложением и функциями ODBC и OLE DB. Технология DAO предназначена преимущественно для создания БД с помощью СУБД Access, так как кроме замены совокупности низкоуровневых функций ODBC несколькими высокоуровневыми осуществляет также прямой доступ к функциям ядра Microsoft Jet базы данных Access.

Технология ADO предоставляет иерархическую модель объектов для доступа к различным OLE DB–провайдерам данных и характеризуется еще более высоким уровнем абстракции. Объектная модель ADO включает объекты, обеспечивающие соединение с провайдером данных, создание запросов SQL к данным, создание набора записей на основе запроса и т. д. Особенностью технологии ADO является возможность ее использования в приложениях Интранет/Интернет для доступа к различным источникам данных. В целом, ADO можно охарактеризовать как наиболее современную технологию разработки приложений для работы с распределенными БД по архитектуре клиент-сервер.

В общем случае, архитектуру доступа к данным, предоставляемую фирмой Microsoft можно представать с помощью следующей схемы (см. рис. 22).

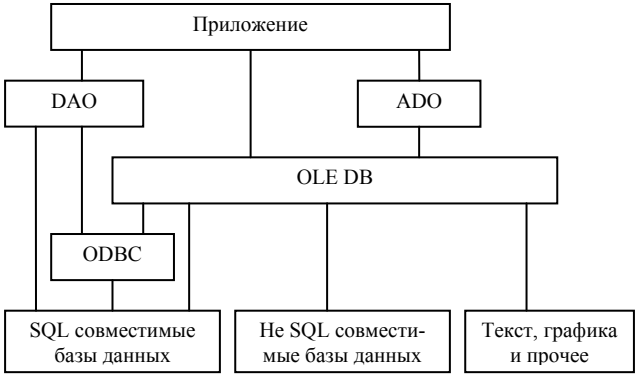


Рис. 22. Архитектура доступа к данным Microsoft

7.2. АРХИТЕКТУРА КЛИЕНТ–СЕРВЕР

Если речь идет о некоторой БД как самостоятельной функциональной единице, то под ней понимают совокупность набора данных и программы обслуживания. Программа обслуживания, реализующая функции управления, обработки и представления данных, может быть как некоторой коммерческой СУБД, так и самостоятельной программой, где ядро СУБД, обеспечивающее управление данными, будет представлено лишь несколькими DLL. При размещении БД в сети возможны различные варианты распределения данных и функций БД по узлам сети. Данные могут храниться на одном выделенном компьютере и быть распределены по всем узлам сети. Функции обработки и представления данных также могут быть самым различным способом распределены по узлам сети. В зависимости от числа узлов сети, между которыми выполняется распределение функций СУБД, можно выделить двухзвенные модели, трехзвенные и т. д.

Исторически первыми появились распределенные БД с применением файл-серверной архитектуры. В таких БД по запросам пользователей файлы БД передаются на персональные компьютеры (ПК), где и производится их обработка. В таком случае БД может располагаться на файл-сервере, в качестве которого может использоваться наиболее мощный компьютер. Функция файл-сервера заключается в основном в хранении БД и обеспечении доступа к ним пользователей, работающих на различных компьютерах. Эти функции обеспечиваются, как правило, той же СУБД, которая работает и на компьютерах пользователя. Для каждого клиента во время работы создается локальная копия данных, с которой он работает. При этом решаются задачи, связанные с возможным одновременным доступом нескольких пользователей к одним и тем же данным. Забота о целостности данных при такой организации работы целиком возлагается на программы клиентов. Если они недостаточно продуманы, в базу данных можно легко занести ошибки, которые могут отразиться на всех пользователях. Если используемая СУБД не имеет достаточных средств для обеспечения многопользовательского доступа к данным, или неправильно сконфигурирована, то нарушение целостности может произойти и при попытке одновременного изменения одних и тех же данных. Часто такие действия приводят к полному повреждению внутренней структуры БД.

При небольших объемах данных архитектура файл-сервер вполне соответствует современным требованиям, но с увеличением числа компьютеров в сети или ростом БД начинают возникать проблемы, связанные с резким падением производительности системы. Это связано с увеличением объема данных, передаваемых по сети, так как вся обработка производится на компьютере пользователя. Если пользователю потребуется несколько строк из таблицы объемом в сотни тысяч записей, то сначала

вся таблица с файл-сервера будет передана на его компьютер, а затем СУБД отберет нужные записи. Даже реализация постепенной постраничной подкачки данных не решает проблему. Намного более удобной для совместной обработки данных является архитектура клиент-сервер.

Архитектура клиент-сервер разделяет приложение на две части, используя лучшие качества с обеих сторон. Клиентская часть (front-end) находится на компьютерах пользователей и обеспечивает легкий в использовании интерактивный интерфейс. Сервер (back-end) находится на выделенном компьютере и обеспечивает управление данными, разделение

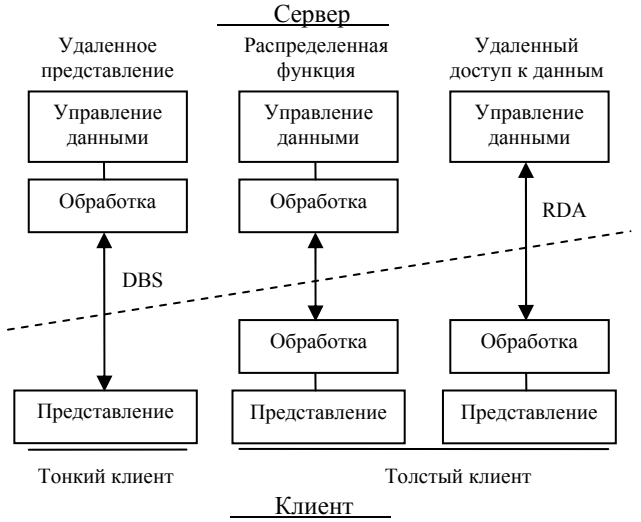


Рис. 15. Модели архитектуры клиент-сервер

информации, администрирование, обеспечение целостности, безопасности и секретности.

В общем случае сервером определенного ресурса в компьютерной сети называется компьютер (программа), управляющий этим ресурсом, клиентом – компьютер (программа), использующий этот ресурс. В качестве ресурса компьютерной сети могут выступать, к примеру, базы данных, службы печати, почтовые службы. Тип сервера определяется видом ресурса, которым он управляет. Например, если управляемым ресурсом является БД, то соответствующий сервер называется сервером БД. Архитектура клиент-сервер предполагает централизованное хранение данных с двухзвенным распределением функций СУБД. В этой архитектуре данные обычно хранятся на выделенном компьютере под управлением специальной программы сервера, а доступ к данным и их представление организу-

ются через клиентские программы. Достоинством организации информационной системы по архитектуре клиент-сервер является удачное сочетание централизованного хранения, обслуживания и коллективного доступа к общей корпоративной информации с индивидуальной работой пользователей над персональной информацией. В такой системе легко реализовать многопользовательский доступ к данным, поскольку появляется возможность предоставлять одни и те же данные нескольким клиентам одновременно, решая при этом проблемы совместного доступа.

В архитектуре клиент-сервер клиент устанавливает соединение с сервером и формирует запрос к серверу БД. Выполнение запроса происходит на сервере. Затем результат запроса посылается клиенту для использования и просмотра. Так как обычно результатом запроса является небольшая часть хранимой в БД информации, то нагрузка на сеть сильно уменьшается. На сервере происходит также обработка транзакций и правил целостности (бизнес-логики). Так как SQL предоставляет стандартный интерфейс для СУБД различных типов, то он может использоваться как средство коммуникации между сервером и клиентом. В таком случае сервер часто называется сервером запросов (SQL сервером).

Если вся обработка данных происходит на стороне сервера, то клиент выполняет только функции интерфейса с пользователем. В этом случае клиентское приложение называют «тонким» клиентом. Если часть обработки данных производится на стороне клиента, то говорят о «толстом» клиенте. По разделению функций между клиентом и сервером можно выделить следующие типы архитектур (рис. 23):

1) удаленное представление,
2) распределенная функция,
3) удаленный доступ к данным.

В модели удаленного доступа к данным (Remote Data Access – RDA) программы, реализующие функции представления информации и функции их обработки, совмещены и выполняются на компьютере-клиенте. Функции сервера фактически ограничиваются управлением данных и обработкой запросов со стороны клиентов. Основное достоинство RDA-модели состоит в наличии большого числа готовых СУБД и других инструментальных средств, обеспечивающих быстрое создание программ клиентской части. Недостатками RDA-модели являются, во-первых, довольно высокая загрузка системы передачи данных вследствие того, что вся логика обработки сосредоточена на компьютере-клиенте, а обрабатываемые данные расположены на удаленном узле. Во-вторых, RDA-системы неудобны с точки зрения разработки, модификации и сопровождения. Основная причина состоит в том, что в получаемых приложениях прикладные функции и функции представления тесно взаимосвязаны. Поэтому даже при незначительном изменении функций системы требуется переделка всей прикладной ее части. Если функции сервера заключаются лишь в хранении данных, то эта модель ничем не отличается от архитектуры файл-сервер.

Модель удаленного представления, иначе модель сервера БД (Database Server – DBS) отличается от предыдущей модели тем, что функции компьютера-клиента ограничиваются функциями представления, в то время как прикладные функции (обязательно включающие обеспечение целостности, безопасности и секретности), реализуются на стороне сервера. Логика работы приложения реализуются в виде хранимых процедур и триггеров. Процедуры хранятся в словаре БД и разделяются несколькими клиентами. В некоторых СУБД на сервере можно хранить и сами запросы, называемые в таком случае хранимыми командами. Хранимые команды выполняются значительно быстрее, так как не требуется каждый раз производить их синтаксический разбор и оптимизацию кода. Достоинствами модели DBS являются возможность централизованного администрирования приложений и обеспечения целостности, а также эффективное использование вычислительных и коммуникационных ресурсов. К недостаткам модели следует отнести ограниченность действий, которые можно выполнять с помощью хранимых процедур и триггеров и сравнительно низкая эффективность использования вычислительных ресурсов обоих компьютеров.

В модели распределенной функции логика обработки данных распределена по двум узлам. Такую модель могут иметь БД, в которых общая часть прикладных функций реализована на компьютере-сервере, а специфические функции обработки информации выполняются на компьютере-клиенте. Функции общего характера могут включать в себя стандартное обеспечение целостности данных, например, в виде хранимых процедур и триггеров, а оставшиеся прикладные функции реализуют специальную прикладную обработку.

Кроме перечисленных способов, можно еще выделить распределенное представление, где все функции БД, включая представление информации сосредоточены на сервере, а клиентская часть системы практически вырождена, и распределенную БД, где по функциональной нагрузке клиент фактически становится равен серверу. В распределенном представлении основной функцией клиентской части является просто отображение информации на экране монитора и связь с основным компьютером через локальную сеть. Модель распределенной БД, наоборот, предполагает использование мощного компьютера-клиента, причем данные могут храниться как на компьютере-сервере, так и на компьютере-клиенте.

Если функцию хранения данных вынести на отдельный компьютер сети, то получим трехъярусный (трехзвенный) вариант представления данных. На нижнем уровне на компьютерах пользователя расположены

приложения клиентов, обеспечивающих пользовательский интерфейс. На втором уровне расположен сервер приложений, обеспечивающий управление данными и реализующий несколько прикладных функций, каждая из которых оформлена как служба предоставления услуг всем требующим этого программам. Серверов приложений может быть несколько, причем каждый из них предоставляет свой вид сервиса. Любая программа, запрашивающая услугу у сервера приложений, является для него клиентом. На третьем уровне расположен удаленный сервер баз данных. Поскольку центральным звеном является сервер приложений, такую модель называют моделью сервера приложений, или AS-моделью (Application Server). Достоинством AS-модели является разгрузка сервера БД, а к недостаткам можно отнести увеличение нагрузки на сеть.

7.3. РАСПРЕДЕЛЕННЫЕ БАЗЫ ДАННЫХ

Распределенные базы данных (РДВ) можно рассматривать как подвид распределенных вычислительных систем, занимающихся обработкой данных. Распределенная вычислительная система состоит из совокупности элементов (не обязательно однородных), соединенных между собой с помощью коммуникационной сети и взаимодействующих при решении некоторой общей задачи. Можно выделить два преимущества такой системы: 1) увеличение мощности системы при решении общей задачи; 2) возможность автономной работы отдельных элементов системы. Таким образом, мы можем определить распределенную БД (Distributed Database – DDB), как совокупность логически взаимосвязанных баз данных, распределенных в компьютерной сети, и распределенную СУБД (Distributed Database Management System – DDBMS) как совокупность программ предназначенных для управления распределенной БД.

Существуют два основных способа организации РБД с распределенным хранением данных: фрагментация и репликация (тиражирование). Фрагментация бывает горизонтальной и вертикальной. При *горизонтальной фрагментации* разбиение происходит за счет помещения в отдельные таблицы с одинаковой структурой не перекрывающихся групп строк. При *вертикальной фрагментации* разбиение происходит по столбцам: одни столбцы формируют одну таблицу, другие – другую. При этом для сохранения идентификации целой записи в отдельных фрагментах приходится в каждый фрагмент добавлять первичный ключ таблицы. Информация о местоположении каждого фрагмента обычно хранится в так называемом глобальном словаре данных, который в свою очередь также может быть распределенным. При раздельном хранении фрагментов данных СУБД должна обеспечивать такой механизм работы, чтобы для программ и

пользователей распределенная система воспринималась как единая централизованная БД.

Если БД или хотя бы один фрагмент данных может располагаться более чем на одном компьютере, то говорят о репликации (или иначе тиражировании) данных. Соответственно репликация бывает полной и частичной. При *полной репликации* на всех компьютерах размещаются синхронизируемые копии одной и той БД. Безопасность и степень доступности данных в такой системе самые высокие. Система остается работоспособной пока хотя бы один компьютер системы находится в рабочем режиме. Скорость выборки локальных данных также максимальна (если пренебречь эффектом увеличения скорости вследствие уменьшения размера БД при ее разбиении). Недостатком такой системы можно считать трудность синхронизации реплик при обновлении данных и то, что между обновлениями копии БД могут отличаться друг от друга. Противоположностью полной репликации является отсутствие репликации, где каждый фрагмент данных имеет только одну копию. Между этими крайними случаями находится широкий спектр вариантов *частичной репликации*. Степень репликации зависит от требуемой доступности данных и от обеспечения необходимой производительности операций обновления данных. Например, если требуется максимальная степень доступности данных и нет необходимости в частом их обновлении, то полная репликация является наилучшим решением. К фрагментации данных прибегают в том случае, если доступ к некоторому фрагменту данных требуется преимущественно для пользователей одного или нескольких компьютеров. Такой подход позволяет обеспечить максимальную скорость работы с данными для пользователей этих компьютеров. В общем случае поиск оптимального решения размещения данных может представлять сложную задачу оптимизации.

Первым фактором, по которому можно различать РБД, является степень однородности. Если все пользователи РБД используют одно и то же программное обеспечение (СУБД) для доступа к данным, и если данные, расположенные на всех компьютерах сети, контролируются все той же СУБД, то такую РБД называют однородной. В противном случае – неоднородной или гетерогенной.

Второй фактор – это степень автономности. С одной стороны, мы можем иметь распределенную СУБД с полным отсутствием локальной автономности, которая имеет единую концептуальную схему данных, единый центр обработки запросов и транзакций, где части единой БД просто распределены по разным компьютерам. С другой стороны, мы можем иметь распределенную СУБД, которая хоть и имеет некоторую общую схему данных, но составлена из полностью автономных СУБД. Такая СУБД называется федеративной СУБД (Federated Database System –

FDBMS). Федеративная СУБД может быть даже составлена из СУБД, поддерживающих различные модели данных, типы, ограничения и языки манипулирования данными. Такой вариант распределенной СУБД более надежен и перспективен, но связан со значительными сложностями реализации. Проблемы в таких системах могут возникать не только вследствие отличия поддерживаемых моделей данных, но и вследствие отличия имен таблиц и полей, и наоборот, когда поля с одинаковыми именами могут обозначать разные данные. Федеративная СУБД поддерживает глобальную схему, на основании которой пользователи могут строить распределенные запросы и обновлять данные. Она работает только с общей схемой данных, поскольку все локальные СУБД имеют свой собственные схемы данных и собственными силами обеспечивают доступ к данным всех их пользователей. Глобальная схема создается посредством слияния локальных схем данных. Программное обеспечение федеративной СУБД предварительно транслирует глобальные запросы в запросы к локальным БД. Затем результаты всех локальных запросов объединяются и предоставляются пользователю.

Основываясь на трехуровневой архитектуре БД, архитектура распределенную СУБД может быть представлена следующим образом (рис. 24).

Рис. 24. Архитектура РСУБД

Глобальная концептуальная схема представляет собой логическое описание всех БД, предоставляя ее так, как будто она не является распределенной. Схемы фрагментации, репликации и распределения определяют размещение данных по локальным компьютерам. Для различных типов распределенных СУБД эти схемы могут иметь разную значимость. Для федеративной СУБД схемы фрагментации, репликации и распределения могут быть опущены вообще, а локальные схемы отображения могут быть представлены локальными внешними схемами.

Распределенные БД имеют преимущества перед традиционными централизованными БД, но не лишены и некоторых недостатков. РБД обладает следующими преимуществами.

1. Разделяемость и локальная автономность. Географическая распределенность организации может быть отображена в распределении ее данных. В результате пользователи отдельных частей БД получают локальный контроль над данными и могут устанавливать локальные ограничения и права доступа.

2. Управление распределенными данными на разных уровнях «прозрачности». В идеальном варианте реальное расположение данных должно быть полностью скрыто от пользователя. Он должен работать с распределенной БД как с системой, расположенной в одном месте.

3. Увеличение стабильности и надежности системы. С выходом из строя отдельных частей распределенная СУБД будет продолжать функционировать.

4. Увеличение производительности. В распределенных БД фрагменты данных можно разместить там, где они наиболее нужны. Следовательно, падает нагрузка на сеть при пересылке данных. Размер фрагмента данных на локальном компьютере будет много меньшим, что также приведет к увеличению скорости работы с БД.

5. Увеличение гибкости за счет модульности системы.

К недостаткам таких систем можно отнести:

1. Повышение сложности, влекущее увеличение стоимости и срока разработки РБД.

2. Усложнение контроля за целостностью данных.

3. Усложнение контроля за безопасностью и секретностью данных, связанное с обеспечением защиты не только БД самих по себе, но и сетевых соединений.

7.4. МНОГОПОЛЬЗОВАТЕЛЬСКИЕ БАЗЫ ДАННЫХ

Важнейшей целью создания БД является организация параллельного доступа многих пользователей к общим данным, используемым ими совместно. Обеспечить параллельный доступ несложно, если все пользо-

ватели будут только читать данные, помещенные в базу. В этом случае работа каждого из них не оказывает никакого влияния на работу остальных пользователей. Но если два или больше 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 потере согласованности данных.

Существуют два подхода решения проблем совместного доступа к общим данным: установка блокировок и управление параллельностью выполнения транзакций. В первом случае на время выполнения какой-либо операции в БД доступ к используемому объекту со стороны других пользователей временно запрещается или ограничивается. Во втором случае при одновременном доступе к одним и тем же данным операции манипулирования данными, заключенные в рамки транзакции, чередуются, и таким образом достигается их параллельное выполнение. Однако несмотря на то что каждая из транзакций может сама по себе выполнена вполне корректно, подобное чередование операций способно приводить к неверным результатам, из-за чего целостность и согласованность базы данных будет нарушена. В большинстве случаев эти два подхода применяются совместно.

Различают оптимистический и пессимистический подходы в управлении параллельностью выполнения транзакций. Пессимистический подход обеспечивает более высокий уровень безопасности, поскольку откладывается выполнение любых транзакций, потенциально способных войти в конфликт с другими транзакциями. Оптимистический подход разрешает асинхронное выполнение транзакций в предположении, что вероятность конфликта невысока. Проверка на наличие конфликта откладывается до завершения транзакции и ее фиксации в БД.

7.4.1. Блокировки

Для организации многопользовательского доступа в СУБД применяется механизм блокировок. Суть блокировки состоит в том, что на время выполнения какой-либо операции в БД доступ к используемому объекту со стороны других пользователей временно запрещается или ограничивается. Например, при копировании таблицы она блокируется от изменения, хотя и разрешено просматривать ее содержимое. Блокировки устанавливаются СУБД автоматически, но возможно и их явное определение.

Рассмотрим четыре вида блокировок, перечисленные в порядке убывания строгости ограничений на возможные действия.

1. Полная блокировка. Означает полное запрещение любых операций над объектами БД. Обычно применяется при изменении структуры объектов.

2. Блокировка от записи. Накладывается в случаях, когда можно читать данные, но не изменять их. Изменение структуры данных также запрещается.

3. Предохраняющая блокировка от записи. Предохраняет объект от наложения на него со стороны других операций полной блокировки либо блокировки от записи. Этот вид блокировки позволяет тому, кто раньше «захватил» объект, успешно завершить модификацию объекта. Такая блокировка обычно применяется при совместном изменении данных несколькими пользователями.

4. Предохраняющая полная блокировка. Предохраняет объект от наложения на него со стороны других операций полной блокировки. Обеспечивает максимальный уровень совместного использования объектов. Используется, например, для обеспечения одновременного просмотра несколькими пользователями одной таблицы. Тогда никому не будет позволено изменить структуру общей таблицы.

При одновременном выполнении различных операций над одним и тем же объектом производится попытка их совмещения. Совмещение возможно тогда, когда совместимыми сказываются блокировки, накладываемые конкурирующими операциями. В отношении перечисленных выше блокировок действуют следующие правила совмещения:

- полная блокировка несовместима с другими блокировками, т. е. при наличии полной блокировки над объектом нельзя производить другие операции, накладывающие со своей стороны любую блокировку;
- блокировка от записи совместима с аналогичной блокировкой и предохраняющей полной блокировкой;
- предохраняющая блокировка от записи совместима с обоими видами предохраняющих блокировок;
- предохраняющая полная блокировка совместима со всеми видами блокировок, кроме полной.

7.4.2. Транзакции

Понятие транзакции имеет непосредственную связь с понятием целостности БД. В системах с развитыми средствами контроля целостности каждая транзакция начинается при целостном состоянии БД и должна оставить это состояние целостным после своего завершения. Несоблюдение этого условия приводит к тому, что вместо фиксации результатов транзакции происходит ее откат, и БД остается в таком состоянии, в котором находилась к моменту начала транзакции, т. е. в целостном состоянии. Часто БД могут обладать такими ограничениями целостности, которые просто невозможно не нарушить, выполняя только одну операцию изменения данных. Поэтому для поддержания подобных ограничений целостности допускается их нарушение внутри транзакции с тем

условием, чтобы к моменту завершения транзакции условия целостности были соблюдены.

Различаются два вида ограничений целостности: немедленно проверяемые и откладываемые. К немедленно проверяемым ограничениям целостности относятся такие ограничения, проверку которых бессмысленно или даже невозможно откладывать. Немедленно проверяемые ограничения целостности соответствуют уровню отдельных операторов языкового уровня СУБД. При их нарушениях не производится откат транзакции, а лишь отвергается соответствующий оператор. Откладываемые ограничения целостности – это ограничения на базу данных, а не на какие-либо отдельные операции. По умолчанию такие ограничения проверяются при конце транзакции, и их нарушение вызывает автоматическую замену оператора COMMIT на оператор ROLLBACK. Однако в некоторых системах поддерживается специальный оператор насильственной проверки ограничений целостности внутри транзакции. Если после выполнения такого оператора обнаруживается, что условия целостности не выполнены, пользователь может сам выполнить оператор ROLLBACK или постараться устранить причины нецелостного состояния БД внутри транзакции.

7.4.3. Уровни изолированности транзакций

В связи со свойством сохранения целостности БД транзакции являются подходящими единицами изолированности пользователей. Действительно, если с каждым сеансом работы с БД ассоциируется транзакция, то каждый пользователь начинает работу с согласованным состоянием БД, т. е. с таким состоянием, в котором БД могла бы находиться, даже если бы пользователь работал с ней в одиночку. Следовательно требуется, чтобы обновления, выполняемые некоторой транзакцией *T1*, не были доступны для любой другой транзакции *T2* до тех и только до тех пор, пока не будет завершено выполнение транзакции *T1*. Завершение выполнения транзакции открывает доступ ко всем обновлениям, выполненным данной транзакцией.

Существует несколько уровней изолированности транзакций. По приоритету их можно расположить в следующем порядке: READ UNCOMMITTED (чтение незафиксированных данных) < READ COMMITTED (чтение зафиксированных данных) < REPEATABLE READ (гарантия повторяемости считывания) < SERIALIZABLE (способность к упорядочению). Далеко не каждая СУБД поддерживает все уровни изолированности транзакций. Последний уровень имеется лишь у наиболее развитых.

Если все транзакции выполняются на уровне способности к упорядочению (принятом по умолчанию), то чередующееся выполнение любого

множества параллельных транзакций может быть упорядочено. Однако если любая транзакция выполняется на более низком уровне изоляции, то существует множество различных способов нарушения способности к упорядочению. Существуют четыре особых случая нарушения способности к упорядочению.

1. *Потерянные изменения*. Результаты успешно завершенной операции обновления одной транзакции могут быть перекрыты результатами выполнения другой транзакции. Допустим, транзакция *T1* изменяет объект базы данных *A*. До завершения транзакции *T1* транзакция *T2* также изменяет объект *A*, перекрывая таким образом результат предыдущей транзакции. Чтобы избежать такой ситуации, явно противоречащей требованию изолированности пользователей, требуется до завершения транзакции *T1* запретить любой другой транзакции изменять объект *A*. Отсутствие потерянных изменений является минимальным требованием к многопользовательской СУБД, поскольку в этом случае обеспечивается требование целостности БД при завершении любой транзакции.

2. *Неаккуратное считывание или чтение «грязных данных»*. Допустим, что транзакция *T1* выполняет вставку новой строки, затем транзакция *T2* извлекает эту строку, после чего выполнение транзакции *T1* отменяется. В результате транзакция *T2* обнаружит, что данной строки больше не существует в том смысле, что она никогда не существовала (поскольку транзакция *T1* действительно не была выполнена). Можно рассмотреть еще один сценарий совместного выполнения транзакций *T1* и *T2*. Транзакция *T1* изменяет объект базы данных *A*, а транзакция *T2* читает объект *A*. Поскольку операция изменения еще не завершена, транзакция *T2* видит несогласованные «грязные» данные, поскольку в следующий момент времени транзакция *T1* может быть отвергнута вследствие нарушения немедленно проверяемого ограничения целостности. Чтобы избежать ситуации чтения «грязных» данных, до завершения транзакции *T1*, изменившей объект *A*, никакая другая транзакция не должна иметь возможности читать объект *A*.

3. *Неповторяемое считывание*. Допустим, транзакция *T1* читает объект *A*. До завершения транзакции *T1* транзакция *T2* изменяет объект *A* и успешно завершается. Транзакция *T1* повторно читает объект *A* и видит его измененное состояние. Чтобы избежать неповторяющихся чтений, до завершения транзакции *T1* никакая другая транзакция не должна иметь возможности изменять объект *A*. Отсутствие неповторяющихся чтений часто является максимальным требованием изолированности транзакций, хотя это еще не гарантирует реальной изолированности пользователей.

4. *Наличие фиктивных элементов*. Допустим, что транзакция *T1* извлекает множество всех записей, которые удовлетворяют некоторому условию (например, записи всех поставщиков из Москвы). Затем транзак-

ция *T2* вставляет новую строку, которая удовлетворяет тому же условию и успешно завершается. Если транзакция *T1* вновь повторит ту же операцию извлечения, что и раньше, то ею будет обнаружена ранее отсутствовавшая – «фиктивная» строка. Данная ситуация также противоречит идее изолированности пользователей и может возникнуть даже на третьем уровне изолированности транзакций. Для ее устранения требуется более высокий уровень синхронизации транзакций.

Возможности возникновения этих нарушений при различных уровнях изоляции транзакций приведены в табл. 5:

Уровни изоляции транзакций *Таблица 5*

Уровень изоляции	Потерянные изменения	Неаккуратное считывание	Неповторяемое считывание	Фиктивные элементы
READ UNCOMMITTED	Нет	Да	Да	Да
READ COMMITTED	Нет	Нет	Да	Да
REPEATABLE READ	Нет	Нет	Нет	Да
SERIALIZABLE	Нет	Нет	Нет	Нет

Для обеспечения реальной изолированности транзакций в СУБД применяется метод сериализации транзакций. *Сериализация транзакций* – это определение последовательности выполнения транзакций, когда результат совместного выполнения транзакций эквивалентен результату последовательного выполнения этих же транзакций. Система, в которой поддерживается сериализация транзакций, обеспечивает реальную изолированность пользователей. Основная проблема состоит в выборе такого метода сериализации, который не слишком ограничивал бы их параллельность. Тривиальным решением является действительно последовательное выполнение транзакций. Но существуют ситуации, в которых можно выполнять операторы разных транзакций в любом порядке с сохранением сериальности. Примерами могут служить только читающие транзакции, а также транзакции, не конфликтующие по доступу к объектам БД.

7.4.4. Методы сериализации транзакций

Существуют два базовых подхода к сериализации транзакций: подход, основанный на синхронизационных захватах (блокировках) объектов БД, и подход, основанный на использовании временных меток. Суть обоих подходов состоит в обнаружении конфликтов транзакций и их устранении. Практические методы сериализации транзакций основываются на учете этих конфликтов. Для каждого из подходов имеются две разновидности – пессимистическая и оптимистическая. При применении пессимистических методов (ориентированных на ситуации, когда конфликты возникают часто) конфликты распознаются и разрешаются немедленно при их возникновении (реальном или подразумеваемом). Оптимистические методы основываются на том, что результаты всех операций модификации БД сохраняются в рабочей памяти транзакций. Реальная модификация БД производится только на стадии фиксации транзакции. Тогда же проверяется, не возникают ли конфликты с другими транзакциями.

Наиболее распространенным является подход, основанный на соблюдении двухфазного протокола синхронизационных блокировок объектов БД. В общих чертах протокол состоит в том, что перед выполнением любой операции в транзакции *T* над объектом БД *r* от имени транзакции *T* запрашивается синхронизационный захват объекта *r* в режиме, зависящем от вида операции (захват по чтению или захват по записи).

Одним из наиболее чувствительных недостатков метода сериализации транзакций на основе синхронизационных захватов является возможность возникновения тупиков (deadlocks) между транзакциями. Приведем пример возникновения тупика между транзакциями *T1* и *T2*: транзакции *T*1 и *T2* установили монопольные захваты объектов *r1* и *r2*; после этого *T1* требуется совместный захват *r2*, а *T2* – совместный захват *r1*. Ни одна из транзакций не может продолжаться, следовательно, монопольные захваты не будут сняты, а совместные – не будут удовлетворены. Поскольку тупики возможны и никакого естественного выхода из данной ситуации не существует, то эти ситуации необходимо обнаруживать и искусственно устранять. Разрушение тупика начинается с выбора в цикле транзакций так называемой транзакции-жертвы, т. е. транзакции, которой решено пожертвовать, чтобы обеспечить возможность продолжения работы других транзакций. Критерием выбора обычно является стоимость транзакции. Стоимость транзакции определяется на основе многофакторной оценки, в которую с разными весами входят время выполнения, число накопленных захватов, их приоритет и т. д. После выбора транзакции-жертвы выполняется откат этой транзакции, который может носить полный или частичный характер. При этом освобождаются блокировки объектов и, следовательно, может быть продолжено выполнение других транзакций. Естественно, такое насильственное устранение тупиковых ситуаций является нарушением принципа изолированности пользователей, которого невозможно избежать.

Альтернативный метод сериализации транзакций, хорошо работающий в условиях редких конфликтов транзакций, основан на использовании временных меток. Основная идея метода (у которого существует множество разновидностей) состоит в следующем: если транзакция *T1* началась раньше транзакции *T2*, то система обеспечивает такой режим выполнения, как если бы *T1* была целиком выполнена до начала *T2*. Для этого каждой транзакции *T* предписывается временная метка *t*, соответствующая времени начала *T*. При выполнении операции над объектом *r*

транзакция *T* помечает его своей временной меткой и типом операции (чтение или изменение). Перед выполнением операции над объектом *r* транзакция *T1* проверяет, не закончилась ли транзакция *T*, пометившая этот объект. Если *T* закончилась, *T1* помечает объект *r* и выполняет свою операцию. Если транзакция *T* не завершилась, то *T1* проверяет конфликтность операций. Если операции неконфликтны, при объекте *r* остается или проставляется временная метка с меньшим значением, и транзакция *T1* выполняет свою операцию. Если операции *T1* и *T* конфликтуют, то при $t(T) > t(T1)$ (т. е. транзакция *T* является более «молодой», чем *T1*), производится откат *T*, и *T1* продолжает работу. Если же $t(T) < t(T1)$, то *T1* получает новую временную метку и начинается заново.

К недостаткам метода временных меток относятся потенциально более частые откаты транзакций, чем в случае использования синхронизационных захватов. Это связано с тем, что конфликтность транзакций определяется более грубо. Кроме того, в распределенных системах не очень просто вырабатывать глобальные временные метки. Но все эти недостатки окупаются тем, что не нужно распознавать и устранять тупики, реализация чего в распределенных системах стоит очень дорого.

Контрольные вопросы и задания

1. Дайте характеристику технологий доступа к данным Microsoft.
2. Что такое распределенная БД?
3. Что подлежит распределению в распределенной СУБД?
4. В чем состоит отличие архитектуры файл-сервер от архитектуры клиент-сервер?
5. Чем отличается удаленное представление от удаленного доступа к данным в архитектуре клиент-сервер?
6. Назовите преимущества и недостатки распределенных БД.
7. Назовите основные способы организации распределенной БД с распределенным хранением данных.
8. Что такое федеративная распределенная СУБД?
9. Перечислите способы решения проблем совместного доступа к общим данным в многопользовательских БД.
10. Перечислите виды блокировок и правила их совмещения.
11. Какой из уровней изоляции транзакций обеспечивает максимальную изолированность пользователей: READ UNCOMMITTED, REPEATABLE READ, READ COMMITTED, SERIALIZABLE?
12. Какие вы знаете случаи нарушения способности к упорядочению транзакций?
13. На каком уровне изоляции транзакций может происходить чтение «грязных данных»?

ЛИТЕРАТУРА

Elmasri, R. Fundamentals of Database systems /R. Elmasri, S. B. Navathe. – Adisson-Wesley, 2000. – 1038 с.

Горев, А. Эффективная работа с СУБД /Горев А. – СПб.: Питер, 1998. – 704 с.

Дейт, Дж. Введение в системы баз данных. 8-ое издание. Пер. с англ. /Дж. Дейт. – Москва: Вильямс, 2005. – 1328 с.

Змитрович, А.И. Базы данных. Учебное пособие для вузов /А. И. Змитрович – Минск.: Университетское, 1991. – 271 с.

Конолли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика /Т. Конолли. – Москва, СПб, Киев: Вильямс, 2000. – 1120 с.

Мартин, Дж. Организация баз данных в вычислительных системах /Дж. Мартин. – Москва: Мир, 1980. – 664 с.

Михеева, В. Microsoft Access 2003 /В. Михеева, И. Харитонова. – СПб, БХВ, 2004. – 1072 с.

Хомоненко, А. Д. Базы данных. Учебное пособие /Хомоненко А. Д. – СПб.: Корона, 2002. – 672 с.

СОДЕРЖАНИЕ