Министерство науки и высшего образования РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Технология разработки и защиты баз данных:
сборник учебно-методических материалов

для специальности 09.02.07 Информационные системы и программирование

Благовещенск

2020

Печатается по решению
редакционно-издательского совета

Составитель: Самохвалова С.Г.

Технология разработки и защиты баз данных: сборник учебно-методических материалов для специальность 09.02.07 Информационные системы и программирование – Благовещенск: Амурский гос. ун-т, 2020. - 64 с.

КРАТКОЕ ИЗЛОЖЕНИЕ ЛЕКЦИОННОГО МАТЕРИАЛА

Сегодня трудно себе представить сколько-нибудь значимую информационную систему, которая не имела бы в качестве основы или важной составляющей базу данных. Концепции и технологии баз данных складывались постепенно и всегда были тесно связаны с развитием систем автоматизированной обработки информации. Создание баз данных после появления реляционного подхода превратилось из искусства в науку, но, как показала практика последних лет, все же окончательно его не исключившая. Тем не менее, сейчас это вполне сложившаяся дисциплина (хотя являющаяся скорее инженерной, чем чисто научной), основанная на достаточно формализованных подходах и включающая широкий спектр приемов и методов создания баз данных.

Базам свойственна «перманентность» данных. Соответственно назначение систем управления базами данных — обеспечение в течение длительного времени их сохранности, а также возможностей выборки и актуализации. Данные существуют всегда, пока есть потребность в их использовании, хотя характер использования, как и пути извлечения практической пользы могут быть самыми разными: от оперативной актуализации значений до уничтожения данных, от их использования для совершенствования сложных систем управления до формирования «чемоданов компромата».

Базы данных в стремительно, а в какой-то степени и сумбурно развивающихся информационных технологиях — это сравнительно консервативное направление, где СУБД и сами базы представляют собой «долговременные сооружения». Элементная база ЭВМ и парадигмы программирования меняются быстрее, чем хранимые данные теряют актуальность. В таких условиях, в отличие от прикладных программистов, создатели баз данных (от разработчиков СУБД до администраторов БД) должны постоянно помнить о проблеме «наследственности» — о том, как интегрировать в создаваемую систему наследуемые данные, находящиеся под управлением устаревшей СУБД, и о том, как построить систему, чтобы вновь создаваемые данные могли быть, в свою очередь, наследованы следующим поколением систем и разработчиков.

Достаточно консервативны и концепции баз данных. Эта консервативность — следствие не только свойства «долговечности», но и того факта, что базы вторичны по отношению к описываемым ими реальным процессам и объектам, достаточно стабильным и типичным. Кроме того, модели данных строились в значительной степени «по аналогии» с организационными и технологическими структурами — иерархическими, сетевыми, матричными.

Широкое использование баз данных различными категориями пользователей привело, с одной стороны, к созданию интерфейсов, требующих минимум времени на освоение средств управления системой, а с другой — к построению мощных, гибких СУБД, имеющих в том числе развитые средства защиты данных от случайного или преднамеренного разрушения. Появились и средства автоматизации разработки, позволяющие создать базу данных любому пользователю, даже не владеющему основами теории БД.

Но, как было отмечено ранее, база данных — это важная, но не основная (функционально), а обеспечивающая (информационная) составляющая некоторой, обычно достаточно крупной человеко-машинной системы. И здесь интересно отметить принципиальное отличие в развитии способностей взаимодействующих субъектов (человек — машина). Разделение информации на табличную (числовую), текстовую и графическую отражает последовательность, в которой эти виды информации «осваивались» компьютерами. Первые языки программирования были рассчитаны исключительно на обработку числовой информации (Fortran, Algol). Первыми появляются и табличные базы данных, также преимущественно рассчитанные на обработку числовых таблиц (файлов). Затем осваиваются текстовые файлы и текстовые БД (автоматизированные информационно-поисковые системы с библиографическими и полнотекстовыми базами). Наконец, с существенным повышением быстродействия и емкости памяти компьютеров на сцену выходят графические и мультимедийные базы.

Создание практически полезной «серьезной» базы данных в равной степени зависит как от «фундаментальности» знаний разработчика в области концепций и технологий СУБД, так и от

степени понимания им сегодняшних и будущих прикладных задач пользователя, не только от адекватности применения тех или иных типовых или оригинальных решений, но и от качества представления (описания) этих решений, с той или иной степенью успешности позволяющих использовать, сопровождать и развивать систему после разработчика.

Кроме того, возможности накапливать и оперативно обрабатывать большие объемы информации, характеризующие деятельность предприятий за достаточно длительные периоды и в различных аспектах, дали новый импульс к развитию аналитических систем. Такого рода системы поддержки принятия решений обычно используются для оценки и выбора альтернативных решений, прогнозирования, идентификации объектов и состояний и т. д. Однако, поскольку для получения необходимых данных в этих случаях нужно использовать сложные SQL-запросы или специализированные процедуры, и при этом обрабатывать большие объемы записей, то уже это может приводить к сознательному отказу от классических нормализованных схем, так как чем выше степень нормализации, тем больше число операций соединения отношений и, соответственно, больше времени необходимо для получения конечного результата.

Понятие базы и банка данных

Развитие вычислительной техники и появление емких внешних запоминающих устройств прямого доступа предопределило интенсивное развитие автоматических и автоматизированных систем разного назначения и масштаба, в первую очередь заметное в области бизнес-приложений Такие системы работают с большими объемами информации, которая обычно имеет достаточно сложную структуру, требует оперативности в обработке, часто обновляется и в то же время требует длительного хранения Примерами таких систем являются автоматизированные системы управления предприятием, банковские системы, системы резервирования и продажи билетов и т д (рис 1 1) Другими направлениями, стимулировавшими развитие, стали, с одной стороны, системы управления физическими экспериментами, обеспечивающими сверхоперативную обработку в реальном масштабе времени огромных потоков данных от датчиков, а с другой — автоматизированные библиотечные информационно-поисковые системы

Рис 1 1 Схема автоматизированной информационной системы

Это привело к появлению новой информационной технологии интегрированного хранения и обработки данных — концепции баз данных, в основе которой лежит механизм предоставления обрабатывающей программе из всех хранимых данных только тех, которые ей необходимы, и в форме, требуемой именно этой программе. При этом сама форма (структура данных и форматы полей, входящих в эту структуру) описывается на логическом, т. е. «видимом» из программы, уровне. Более того, поскольку различные программы могут по-разному «видеть» (а следовательно,

и использовать) одни и те же данные, то система должна сделать «прозрачными» для программы все данные, кроме тех, которые для нее являются «своими».

Банк данных (БнД) — это система специально организованных данных, программных, языковых, организационных и технических средств, предназначенных для централизованного накопления и коллективного многоцелевого использования данных.

Под базой данных (БД) обычно понимается именованная совокупность данных, отображающая состояние объектов и их отношений в рассматриваемой предметной области.

Характерной чертой баз данных является постоянство: данные постоянно накапливаются используются; состав и структура данных, необходимых для решения тех или иных прикладных задач, обычно постоянны и стабильны во времени; отдельные или даже все элементы данных могут меняться — но и это есть проявление постоянства — постоянная актуальность.

Система управления базами данных (СУБД) — это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Иногда в составе банка данных выделяют архивы. Основанием для этого является особый режим использования данных, когда только часть данных находится под оперативным управлением СУБД. Все остальные данные (собственно архивы) обычно располагаются на носителях, оперативно не управляемых СУБД. Одни и те же данные в разные моменты времени могут входить как в базы данных, так и в архивы Банки данных могут не иметь архивов, но если они есть, то в состав банка данных может входить и система управления архивами.

Проблемы совместного использования данных и периферийных устройств компьютеров и рабочих станций быстро породили модель вычислений, основанную на концепции файлового сервера — сеть создает основу для коллективной обработки, сохраняя простоту работы с персональным компьютером, позволяет совместно использовать данные и периферию.

В этом смысле главной отличительной чертой баз данных является использование централизованной системы управления данными, причем как на уровне файлов, так и на уровне элементов данных. Централизованное хранение совместно используемых данных приводит не только к сокращению затрат на создание и поддержание данных в актуальном состоянии, но и к сокращению избыточности информации, упрощению процедур поддержания непротиворечивости и целостности данных

Эффективное управление внешней памятью является основной функцией СУБД. Эти обычно специализированные средства настолько важны с точки зрения эффективности, что при их отсутствии система просто не сможет выполнять некоторые задачи уже потому, что их выполнение будет занимать слишком много времени. При этом ни одна из таких специализированных функций, как построение индексов, буферизация данных, организация доступа и оптимизация запросов, не является видимой для пользователя и обеспечивает независимость между логическим и физическим уровнями системы прикладной программист не должен писать программы индексирования, распределять память на диске и т. д.

Развитие теории и практики создания информационных систем, основанных на концепции баз данных, создание унифицированных методов и средств организации и поиска данных позволяют хранить и обрабатывать информацию о все более сложных объектах и их взаимосвязях, обеспечивая многоаспектные информационные потребности различных пользователей

Основные требования, предъявляемые к банкам данных, можно сформулировать следующим образом

Многократное использование данных: пользователи должны иметь возможность использовать данные различным образом

Простота пользователи должны иметь возможность легко узнать и понять, какие данные имеются в их распоряжении

Легкость использования' пользователи должны иметь возможность осуществлять (процедурно) простой доступ к данным, при этом все сложности доступа к данным должны быть скрыты в самой системе управления базами данных

Гибкость использования: обращение к данным или их поиск должны осуществляться с помощью различных методов доступа.

Быстрая обработка запросов на данные: запросы на данные, в том числе незапланированные, должны обрабатываться с помощью высокоуровневого языка запросов, а не только прикладными программами, написанными с целью обработки конкретных запросов (разработка таких программ в каждом конкретном случае связана с большими затратами времени).

Пользователь должен иметь возможность кратко выразить нетривиальные запросы (в нескольких словах или несколькими нажатиями клавиш мыши). Это означает, что средство формулирования должно быть достаточно «декларативным», т. е. упор должен быть сделан на «что», а не на «как».

Кроме того, средство обработки запросов не должно зависеть от приложения, т. е. оно должно работать с любой возможной базой данных.

Язык взаимодействия конечных пользователей с системой должен обеспечивать конечным пользователям возможность получения данных без использования прикладных программ.

База данных — это основа для будущего наращивания прикладных программ: базы данных должны обеспечивать возможность быстрой и дешевой разработки новых приложений. Сохранение затрат умственного труда: существующие программы и логические структуры данных (на создание которых обычно затрачивается много человеко-лет) не должны переделываться при внесении изменений в базу данных.

Наличие интерфейса прикладного программирования: прикладные программы должны иметь возможность просто и эффективно выполнять запросы на данные; программы должны быть изолированы от расположения файлов и способов адресации данных.

Распределенная обработка данных: система должна функционировать в условиях вычислительных сетей и обеспечивать эффективный доступ пользователей к любым данным распределенной БД, размещенным в любой точке сети.

Адаптивность и расширяемость: база данных должна быть настраиваемой, причем настройка не должна вызывать перезаписи прикладных программ. Кроме того, поставляемый с СУБД набор предопределенных типов данных должен быть расширяемым — в системе должны иметься средства для определения новых типов и не должно быть различий в использовании системных и определенных пользователем типов.

Контроль за целостностью данных: система должна осуществлять контроль ошибок в данных и выполнять проверку взаимного логического соответствия данных.

Восстановление данных после сбоев: автоматическое восстановление без потери данных транзакции. В случае аппаратных или программных сбоев система должна возвращаться к некоторому согласованному состоянию данных.

Вспомогательные средства должны позволять разработчику или администратору базы данных предсказать и оптимизировать производительность системы.

Автоматическая реорганизация и перемещение: система должна обеспечивать возможность перемещения данных или автоматическую реорганизацию физической структуры.

Пользователи баз данных

В информационных системах, создаваемых на основе СУБД, способы организации данных и методы доступа к ним перестали играть решающую роль, поскольку оказались скрытыми внутри СУБД Массовый, так называемый конечный пользователь, как правило, имеет дело только с внешним интерфейсом, поддерживаемым СУБД.

Эти преимущества, как уже понятно, не могут быть реализованы путем механического объединения данных в БД. Предполагается, что в системе обязательно существует специальное должностное лицо (группа лиц) — администратор базы данных (А БД), который несет ответственность за проектирование и общее управление базой данных. АБД определяет информационное содержание БД. С этой целью он идентифицирует объекты БД и моделирует базу, используя язык описания данных. Получаемая модель служит в дальнейшем справочным документом для администраторов приложений и пользователей. Администратор решает также все вопросы, связанные с

размещением БД в памяти, выбором стратегии и ограничений доступа к данным В функции АБД входят также организация загрузки, ведения и восстановления БД и многие другие действия, которые не могут быть полностью формализованы и автоматизированы.

Администратор приложений (или, если таковой специально не выделяется — администратор БД) определяет для приложений подмодели данных. Тем самым разные приложения обеспечиваются собственным «взглядом», но не на всю БД, а только на требуемую для конкретного приложения («видимую») ее часть. Вся остальная часть БД для данного приложения будет «прозрачна»

Прикладные программисты имеют, как правило, в своем распоряжении один или несколько языков программирования, с помощью которых генерируются прикладные программы.

Типология баз данных

Классификация баз и банков данных может быть произведена по разным признакам (относящимся к разным компонентам и сторонам функционирования банков данных (БнД), среди которых выделяют, например, следующие.

По форме представляемой информации можно выделить фактографические, документальные, мультимедийные, в той или иной степени соответствующие цифровой, символьной и другим (нецифровой и несимвольной) формам представления информации в вычислительной среде. К последним можно отнести картографические, видео-, аудио-, графические и другие БД.

По типу хранимой (не мультимедийной) информации можно выделить фактографические, документальные, лексикографические БД. Лексикографические базы — это классификаторы, кодификаторы, словари основ слов, тезаурусы, рубрикаторы и т. д , которые обычно используются в качестве справочных совместно с документальными или фактографическими БД

Документальные базы подразделяются по уровню представления информации на полнотекстовые (так называемые «первичные» документы) и библиографическо-реферативные («вторичные» документы, отражающие на адресном и содержательном уровнях первичный документ).

По типу используемой модели данных выделяют три классических класса БД: иерархические, сетевые, реляционные. Развитие технологий обработки данных привело к появлению постреляционных, объ-ектноориентированных, многомерных БД, которые в той или иной степени соответствуют трем упомянутым классическим моделям.

По топологии хранения данных различают локальные и распределенные БД.

По типологии доступа и характеру использования хранимой информации БД могут быть разделены на специализированные и интегрированные.

По функциональному назначению (характеру решаемых с помощью БД задач и, соответственно, характеру использования данных) можно выделить операционные и справочно-информацион-ные. К последним можно отнести ретроспективные БД (электронные каталоги библиотек, БД статистической информации и т. д.), которые используются для информационной поддержки основной деятельности и не предполагают внесения изменений в уже существующие записи, например, по результатам этой деятельности. Операционные БД предназначены для управления различными технологическими процессами. В этом случае данные не только извлекаются из БД, но и изменяются (добавляются) в том числе в результате этого использования.

По сфере возможного применения можно различать универсальные и специализированные (или проблемно-ориентированные) системы.

По степени доступности можно выделить общедоступные и БД с ограниченным доступом пользователей. В последнем случае говорят об управляемом доступе, индивидуально определяющем не только набор доступных данных, но и характер операций, которые доступны пользователю.

Следует отметить, что представленная классификация не является полной и исчерпывающей. Она в большей степени отражает исторически сложившееся состояние дел в сфере деятельности, связанной с разработкой и применением баз данных.

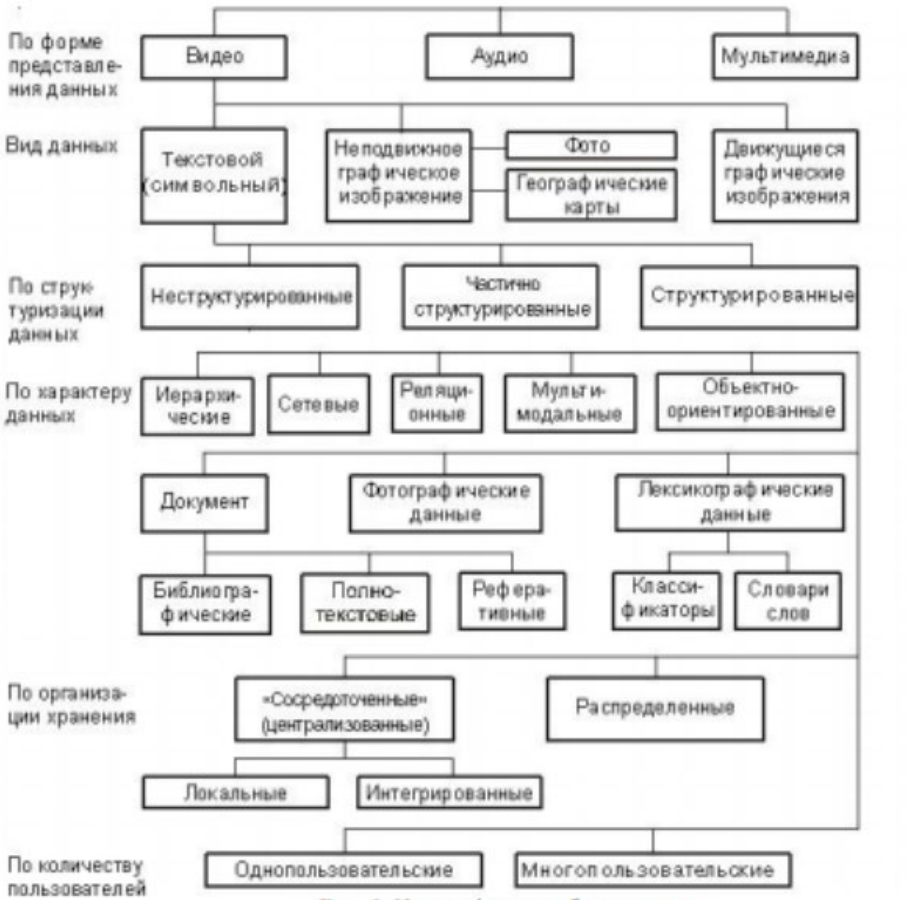


Рис. 1. Классификация баз данных

Типология моделей

Основные отличия любых методов представления информации заключаются в том, каким способом фиксируется семантика предметной области Однако следует особо отметить, что для всех уровней и для любого метода представления предметной области (нам важен контекст создания и использования машинных баз данных) в основе отображения (т е собственно формирования представления) лежит кодирование понятий и отношений между понятиями Многоуровневая система моделей представления информации иллюстрируется рис 1 7 .

Ключевым этапом при разработке любой информационной системы является проведение системного анализа формализация предметной области и представление системы как совокупности компонент Системный анализ позволяет, с одной стороны, лучше понять «что надо делать» и «кому надо делать» (аналитику, разработчику, руководителю, пользователю), а с другой — отслеживать во времени изменения рассматриваемой модели и обновлять проект Декомпозиция как основа системного анализа может быть функциональной (построение иерархий функций) или объектной

Рис 1 7 Система моделей представления информации

Однако в большинстве систем, если говорить, например, о базах данных, типы данных являются более статичным элементом, чем способы их обработки Поэтому получили интенсивное развитие такие методы системного анализа, как диаграммы потоков данных (Data Flow Diagram)

Развитие реляционных баз данных в свою очередь стимутировало развитие методик построения моделей данных, и в частности, ER-диаграмм (Entity Relationship Diagram)

Однако и Функциональная декомпозиция и диаграммы потоков данных дают только некоторый срез исследуемой предметной области, но не позволяют получить представление системы в целом

Различаются и методы отображения, используемые на этапе построения даталогических моделей, отражающих способ идентификации элементов и связей, но, что особенно важно — в контексте их будущего представления в одномерном пространстве памяти вычислительной машины .Модели подразделяются на фактографические — ориентированные на представление хорошо структурированной информации, и документальные — представляющие наиболее распространенный способ отражения слабоструктурированной информации. Если в первом случае говорят о реляционной, иерархической или сетевой моделях данных, то во втором — о семантических сетях и документальных моделях.

При проектировании информационных систем свойства объектов (их характеристики) называются атрибутами. Именно значения атрибутов позволяют выделить как в предметной области различные объекты (типы объектов), так и среди объектов одного типа — их различные экземпляры. Представление атрибутов удобнее всего моделируется теоретико-множественными отношениями. Отношение наглядно представляется как таблица, где каждая строка — кортеж отношения, а каждый столбец (домен) представляет множество значений атрибута. Список имен атрибутов отношения образует схему отношения, а совокупность схем отношений, используемых для представления БД, в свою очередь образует схему базы данных

Представление схем БД в виде схем отношений упрощает процедуру проектирования БД

Этим объясняется создание систем, в которых проектирование БД ведется в терминах реля-

ционной модели данных, а работа с БД поддерживается СУБД одного из упомянутых ранее типов

Основное отличие методов представления информации заключается в том, каким способом фиксируется семантика предметной области. Первые, фактографические БД, задают четкую схему соответствия, в рамках которой и отображается предметная область. Подобное построение по сути своей является довольно статичным, требует априорного знания типов отношений. В нем достаточно сложно вводить информацию о новых типах отношений между объектами, но с другой стороны, зафиксированная схема базы данных позволяет довольно эффективно организовать поиск информации

Во втором случае предметная среда отображается (по крайней мере, на уровне модели) в виде однородной сети, любые изменения которой, как по вводу новых классов объектов, так и новых типов отношений, не связаны с какими-либо структурными преобразованиями сети. В силу большого количества типов отношений манипулирование подобной «элементарной» информацией достаточно затруднено, поэтому для данного случая характерно введение большого количества более общих понятии (и соответствующих им отношений), что упрощает работу с сетью.

Модель данных должна, так или иначе, дать основу для описания данных и манипулирования ими, а также дать средства анализа и синтеза структур данных. Любая модель, построенная более или менее аккуратно с точки зрения математики, сама создает объекты для исследования и начинает жить как бы параллельно с практикой.

Реляционная модель данных в качестве основы отображения непосредственно использует понятие отношения. Она ближе всего находится к так называемой концептуальной модели предметной среды и часто лежит в основе последней . В отличие от теоретико-графовых моделей в реляционной модели связи между отношениями реализуются неявным образом, для чего используются ключи отношений. Например, отношения иерархического типа реализуется механизмом первичных / внешних ключей, когда в подчиненном отношении должен присутствовать набор атрибутов, связывающих это отношение с основным. Такой набор атрибутов в основном отношении будет называться первичным ключом, а в подчиненном — вторичным

Прогресс в области разработки языков программирования, связанный в первую очередь с типизацией данных и появлением объектно-ориентированных языков, позволил подойти к анализу сложных систем с точки зрения иерархических представлений — с помощью классов объектов со свойствами инкапсуляции, наследования и полиморфизма, схемы которых отображают не только данные и их взаимосвязи, но и методы обработки данных

В этом смысле объектно-ориентированный подход является гибридным методом и позволяет получить более естественную формализацию системы в целом. В итоге это позволяет снизить существующий барьер между аналитиками и разработчиками (проектировщиками и программистами), повысить надежность системы и упростить сопровождение, в частности, интеграцию с другими системами Модель будет структурно объектно-ориентированной, если она поддерживает сложные объекты, модель будет поведенчески объектно-ориентированной, если она обеспечивает процедурную расширяемость, для того чтобы модель была полностью объектно-ориентированной, она должна обладать обоими свойствами

В заключение отметим, что представленная здесь типология моделей не претендует на полноту, и она не является классификацией в точном смысле этого слова. Она скорее иллюстрирует эклектичносгь преобладающих в разное время взглядов, методов и решений, используемых при проектировании и реализации баз данных

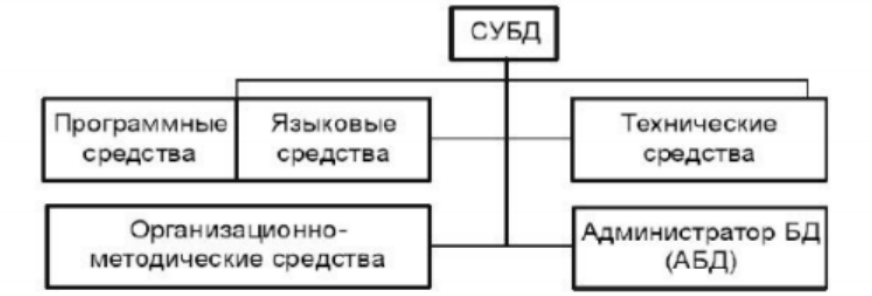
Компоненты банка данных

Определение банка данных предполагает, что с функционально-организационной точки зрения банк данных является сложной человеко-машинной системой, включающей в себя все подсистемы, необходимые для надежного, эффективного и продолжительного во времени функционирования.

В структуре банка данных выделяют следующие компоненты (подсистемы):

- информационная база;
- лингвистические средства;

• программные средства;
• технические средства;
• организационно-административные подсистемы и нормативно-методическое обеспечение.



Информационная база

Данные, отражающие состояние определенной предметной области и используемые информационной системой, принято называть информационной базой. Информационная база состоит из двух компонент: 1) коллекции записей собственно данных; 2) описания этих данных — метаданных.

Данные отделены от описаний, но в то же время данные не могут использоваться без обращения к соответствующим описаниям.

Уже из определения базы данных и приведенных ранее основных требований следует, что данные могут использоваться (т. е. представляться) по-разному. С одной стороны, разные прикладные задачи требуют разных наборов данных, в совокупности обеспечивающих функциональную полноту информации, а с другой — они должны быть различны для различных категорий субъектов (разработчиков или пользователей). Также должны быть различными и способы описания самих данных, их природы, формы хранения, условий взаимной непротиворечивости.

В литературе по базам данных упоминаются три уровня представления данных — концептуальный, внутренний и внешний (рис. 1.2).

Эти уровни представлений введены исходя из различного рас смотрения БД. Например, прикладному программисту требуются не все данные БД, а только некоторая их часть, используемая в его программе. Внешний уровень представления обеспечивает именно эту форму обмена данными.

Внутренний уровень — глобальное представление БД, определяет необходимые условия для организации хранения данных на внешних запоминающих устройствах Описание БД на концептуальном уровне представляет собой обобщенный взгляд на данные с позиций предметной области (разработчика приложений, пользователя или внешней информационной системы).

Внешний уровень представления данных не затрагивает физической организации (размещения) данных во внешней памяти, поэтому его называют иногда логическим уровнем.

Соответственно внутренний уровень называют физическим уровнем.

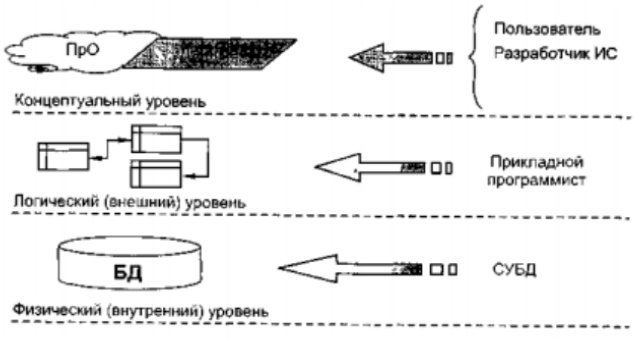


Рис 1.2 Уровни представления данных

Лингвистические средства

Многоуровневое представление БД предполагает соответствующие описания данных на каждом уровне и согласование одних и тех же данных на разных уровнях. С этой целью в состав СУБД включаются специальные языки для описания представлений внутреннего и внешнего уровней. Кроме того, СУБД должна включать в себя язык манипулирования данными (ЯМД).

Желательно также наличие тех или иных дополнительных сервисных средств, например средств генерации отчетов.

Работа с базами данных предполагает несколько этапов:

• описание БД;

• описание частей БД, необходимых для конкретных приложений (задач, групп задач);

• программирование задач или описание запросов в соответствии с правилами конкретного языка и использованием языковых конструкций для обращения к БД;

• загрузка БД и т. д.

Для выражения обобщенного взгляда на данные применяют язык описания данных (ЯОД) внутреннего уровня, включаемый в состав СУБД3.

Описание представляет собой модель данных и их отношений, т. е. структур, из которых образуется БД.

ЯОД позволяет определять схемы базы данных, характеристики хранимых и виртуальных данных и параметры организации их хранения в памяти и может включать в себя средства поддержки целостности базы данных, ограничения доступа, секретности.

ЯМД обычно включает в себя средства запросов к базе данных и поддержания базы данных (добавление, удаление, обновление данных, создание и уничтожение БД, изменение определений БД, обеспечение запросов к справочнику БД).

Исторически первым типом структур данных, который был включен в языки программирования, была иерархическая структура Некоторые ранние СУБД также предполагали использование в качестве основной модели иерархические структуры типа дерева. Основанием для такого выбора было удобство представления (моделирования) естественных иерархических структур данных, существующих, например, в организациях .

В ряде предметных областей структура данных имеет более сложный вид, в котором поддерживаются связи типа «многие к одному», и которые могут быть представлены ориентированным графом Такие структуры называют сетевыми. Для управления БД сетевой структуры международной ассоциацией Кодасил была предложена обобщенная архитектура системы с ЯОД схемы (модели БД) и подсхемы (модели части БД для конкретного приложения), а также ЯМД для оперирования данными БД в прикладных программах.

В настоящее время разработаны десятки языков, основанных на реляционной исчислении, различие которых обусловлено особенностями математических теорий, положенных в основу их построения. Среди этих языков можно выделить базирующиеся на С-исчислении, предложенном Коддом, и исчислении, предложенном Пиротти. С-исчисление базируется на классическом прикладном исчислении предикатов, исчисление представляет собой разновидность прикладного многотипного исчисления предикатов. Существенное различие между этими исчислениями, а следовательно и языками заключается в том, что в С-исчислении в качестве области изменения значений предметной переменной используется множество выборок (кортежей) отношения, а в /"-исчислении каждому типу переменных или констант соответствует определенный домен базы данных.

Функциональные характеристики языков отражают возможности описания данных, средств представления запроса, обновления, поддержки целостности и секретности, включения в языки программирования, управления форматом ответов, средств запроса к словарю данных БД и т. д.

Качественные характеристики языков запросов могут определяться такими свойствами, как полнота, селективная мощность, простота изучения и использования, степень процедурности и модульности, унифицированность, производительность и эффективность

Модульность построения языка характеризует возможность существования нескольких

уровней языка и зависит от специфических свойств математической теории, лежащей в его основе.

Минимальный уровень языка, обычно легко понимаемый пользователем, бывает достаточным для формулирования большинства запросов, и лишь формулировка сложных запросов может потребовать использования всех выразительных средств языка, о существовании которых пользователи начального уровня могут и не знать. Языки, не обладающие модульностью, требуют от пользователя знания почти всего объема средств языка, что усложняет процесс их изучения.

Наиболее распространенным языком для работы с базами данных является SQL (Structured Query Language), в своих последних реализациях предоставляющий не только средства для спецификации и обработки запросов на выборку данных, но также и функции по созданию, обновлению, управлению доступом и т. д.

По существу SQL уже соединяет в себе язык описания данных и язык манипулирования данными. Он не является полноценным языком программирования, и в случае его использования для организации доступа к БД из прикладных программ, SQL-выражения встраиваются в конструкции базового языка.

Являясь внутренним языком баз данных, SQL естественно отражает особенности конкретной СУБД. Сегодня это единственный стандартизованный язык фактографических баз данных, достаточно мощный и в то же время простой для понимания и использования. Сочетание этих факторов вместе с поддержкой ведущих производителей, таких как IBM и Microsoft, привели не только к широкому его распространению, но и совершенствованию. Сегодня, благодаря независимости от конкретных СУБД и межплатформенной переносимости, SQL стал языком распределенных баз данных и языком шлюзов, позволяющим совместно использовать СУБД разного типа

Программные средства

Обработка данных и управление этой обработкой в вычислительной среде, а также взаимодействие с операционной системой и прикладными программами осуществляется комплексом программных средств, взаимосвязь которых иллюстрируется рис. 1.3 В составе комплекса обычно выделяют следующие компоненты:

- ядро, обеспечивающее управление данными во внешней и оперативной памяти, а также протоколирование изменений;
- процессор языка базы данных, обеспечивающий обработку (трансляцию или компиляцию) и оптимизацию запросов на выборку и изменение данных;
- подсистему (библиотеку) поддержки программных вызовов, которая обслуживает прикладные программы управления данными, взаимодействующие с СУБД через средства пользовательского интерфейса;
- сервисные программы (системные и внешние утилиты), обеспечивающие настройку СУБД, восстановление после сбоев и ряд дополнительных возможностей обслуживания.

Большинство СУБД работают в среде операционной системы и тесно с ней связаны.

Многопользовательские приложения, обработка распределенных запросов, защита данных требуют эффективно использовать ресурсы, управление которыми обычно является функцией ОС.

Использование многопроцессорных систем и мультипоточных технологий обработки данных позволяет эффективно обслуживать параллельно выполняемые запросы, но требует координации использования ресурсов между ОС и СУБД Соответственно, управление доступом и обеспечение зашиты также обычно интегрируются с соответствующими средствами операционной системы.

Именно централизованное управление данными обеспечивает:

- сокращение избыточности в хранимых данных;
- совместное использование хранимых данных;
- стандартизацию представления данных, упрощающую эксплуатацию БД;
- разграничение доступа к данным;
- целостность данных, обеспечиваемую процедурами, предотвращающими включение в БД неверных данных, и ее восстановление после отказов системы.

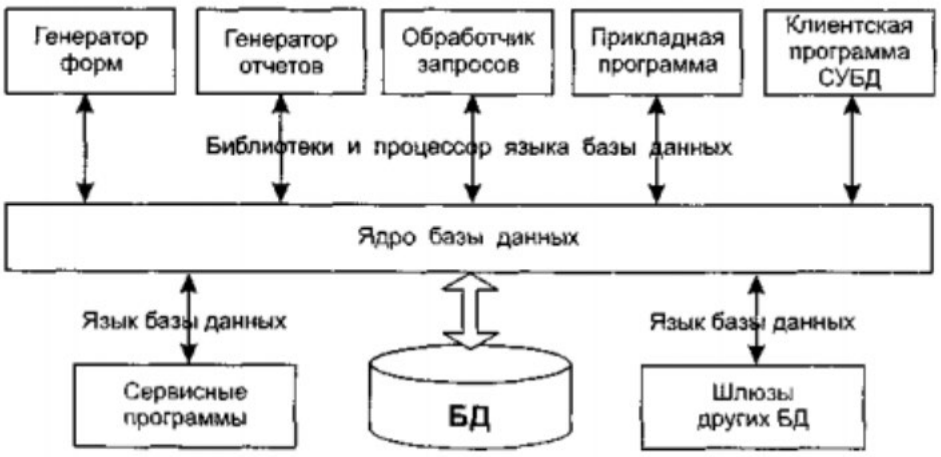


Рис 1 3 Программные средства СУБД

Технические средства

Сегодня большинство банков данных создается и функционирует на основе универсальных вычислительных машин4

Однако для больших баз данных, функционирующих в промышленном режиме, обеспечение эффективной и бесперебойной работы должно основываться на использовании адекватных аппаратных средств.

Устройства ввода-вывода и накопители внешней памяти — традиционно «узкое место» любой базы данных. Объем и быстродействие накопителей являются, очевидно, важными параметрами. Однако столь же значима и отказоустойчивость. Здесь следует отметить необходимость согласованных решений при распределении ролей между аппаратными и программными компонентами управления операциями ввода-вывода. Например, наличие буферной памяти в накопителе, ускоряющей ввод-вывод (аппаратное кэширование) при сбоях системы во время выполнения операции записи в БД может привести к потере данных, переданные для записи данные еще будут находиться в буфере, а так как СУБД отметит операцию записи как уже завершившуюся, откат для восстановления данных станет невозможен.

Для повышения надежности хранения часто используют специализированные дисковые подсистемы — RAID (Redundant Array of Inexpensive Disk) Один логический RAID-диск — это

несколько физических дисков, объединенных в одно устройство, управляемое ' специализированным контроллером, что позволяет распределять основные и системные данные между несколькими носителями (дисками), в том числе дублировать данные. Таким образом, в случае повреждения одного из дисков, можно оперативно восстановить потерянные данные.

Не менее значима роль центрального процессора. Многие промышленные СУБД поддерживают многопроцессорную обработку запросов. Теоретически использование еще одного процессора позволит ускорить обработку. Однако на практике многопроцессорные системы требуют повышенного внимания при приобретении оборудования: надежно работают только сертифицированные системы, использующие соответствующие периферийные устройства.

Для распределенных и удаленно используемых баз данных также важно сетевое окружение" связное оборудование и сетевые протоколы. Здесь важны не только показатели быстродействия, но и поддерживаемые ими возможности обеспечения безопасности.

Организационно-административные подсистемы

Организационно-методические средства не являются технической компонентой системы, однако трудно рассчитывать на устойчивое и долговременное функционирование банка данных, если будут отсутствовать необходимые методические и инструктивные материалы, регламентирующие работу пользователей, различных по своему статусу и уровню подготовленности.

Язык SQL История развития SQL

SQL (Structured Query Language) — Структурированный Язык Запросов — стандартный язык запросов *по* работе с реляционными *БД*. Язык *SQL* появился после *реляционной алгебры*, и его прототип был разработан в конце 70-х годов в компании *IBM* Research. Он был реализован в первом прототипе реляционной *СУБД* фирмы *IBM System* R. В дальнейшем этот язык применялся во многих коммерческих *СУБД* и в силу своего широкого распространения постепенно стал стандартом "де-факто" для языков манипулирования данными в реляционных *СУБД*.

Первый *международный стандарт* языка *SQL* был принят в 1989 г. (далее мы будем называть его *SQL*/89 или SQL1). Иногда стандарт SQL1 также называют стандартом *ANSI*/ISO, и подавляющее большинство доступных на рынке *СУБД* поддерживают этот стандарт полностью. Однако развитие информационных технологий, связанных с базами данных, и необходимость реализации переносимых приложений потребовали в скором времени доработки и расширения первого стандарта *SQL*.

В конце 1992 г. был принят новый *международный стандарт* языка *SQL*, который в дальнейшим будем называть *SQL/92* или *SQL2*. И он не лишен недостатков, но в то же время является существенно более точным и полным, чем *SQL*/89. В настоящий момент большинство производителей *СУБД* внесли изменения в свои продукты так, чтобы они в большей степени удовлетворяли стандарту *SQL2*.

В 1999 году появился новый стандарт, названный *SQL3*. Если отличия между стандартами SQL1 и *SQL2* во многом были количественными, то стандарт *SQL3* соответствует качественным серьезным преобразованиям. В *SQL3* введены новые типы данных, при этом предполагается возможность задания сложных *структурированных типов данных*, которые в большей степени соответствуют объектной ориентации. Наконец, добавлен раздел, который вводит стандарты на события и триггеры, которые ранее не затрагивались в стандартах, хотя давно уже широко использовались в коммерческих *СУБД*. В стандарте определены возможности четкой спецификации триггеров как совокупности события и действия. В качестве действия могут выступать не только *последовательность операторов SQL*, но и *операторы управления ходом выполнения программы*. В рамках управления транзакциями произошел возврат к старой модели транзакций, допускающей *точки сохранения* (*savepoints*), и возможность указания в операторе отката ROLLBACK точек возврата позволит откатывать транзакцию не в начало, а в промежуточную ранее сохраненную точку. Такое решение повышает гибкость реализации сложных алгоритмов обработки информации.

А зачем вообще нужны эти стандарты? Зачем их изобретают и почему надо изучать их? Текст стандарта *SQL2* занимает 600 станиц сухого формального текста, это очень много, и кажется, что это просто происки разработчиков стандартов, а не то, что необходимо рядовым разработчикам. Однако ни один серьезный разработчик, работающий с базами данных, не должен игнорировать стандарт, и для этого существуют весьма веские причины. Разработка любой информационной системы, ориентированной на технологию баз данных (а других информационных систем на настоящий момент и не бывает), является трудоемким процессом, занимающим несколько десятков и даже сотен человеко-месяцев. Следует отдавать себе отчет, что нельзя разработать сколько-нибудь серьезную систему за несколько дней. Кроме того, развитие вычислительной техники, систем телекоммуникаций и программного обеспечения столь стремительно, что проект может устареть еще до момента внедрения. Но развивается не только вычислительная техника, изменяются и реальные объекты, поведение которых моделируется использованием как самой *БД*, так и процедур обработки информации в ней, то есть конкретных приложений, которые составляют реальное наполнение разрабатываемой информационной системы. Именно поэтому проект информационной системы должен быть рассчитан на *расширяемость* и *переносимость* на другие платформы. Большинство поставщиков аппаратуры и программного обеспечения следуют стратегии поддержки стандартов, в противном случае пользователи просто не будут их покупать. Однако каждый поставщик стремится улучшить свой продукт введением дополнительных возможностей, не входящих в стандарт. Выбор разработчиков, следовательно, таков: ориентироваться только на

экзотические особенности данного продукта либо стараться в основном придерживаться стандарта. Во втором случае весь интеллектуальный труд, вкладываемый в разработку, становится более защищенным, так как система приобретает свойства переносимости. И в случае появления более перспективной платформы проект, ориентированный в большей степени на стандарты, может быть легче перенесен на нее, чем тот, который в основном ориентировался на особенности конкретной платформы. Кроме того, стандарты — это верный ориентир для разработчиков, так как все поставщики *СУБД* в своих перспективных разработках обязательно следуют стандарту, и можно быть уверенным, что в конце концов стандарт будет реализован практически во всех перспективных *СУБД*. Так произошло со стандартом SQL1, так происходит со стандартом *SQL2* и так будет происходить со стандартом *SQL3*.

Для поставщиков *СУБД* стандарт — это путеводная *звезда*, которая гарантирует правильное направление *работ*. А вот *эффективность реализации* стандарта — это гарантия успеха.

SQL нельзя в полной мере отнести к традиционным языкам программирования, он не содержит традиционные *операторы*, *управляющие* ходом выполнения программы, *операторы* описания типов и многое другое, он содержит только набор стандартных операторов доступа к данным, хранящимся в базе данных. *Операторы SQL* встраиваются в базовый *язык программирования*, которым может быть любой стандартный язык типа C++, PL, *COBOL* и т. д. Кроме того, *операторы SQL* могут выполняться непосредственно в интерактивном режиме.

Структура SQL

В отличие от *реляционной алгебры*, где были представлены только *операции* запросов к *БД*, *SQL* является полным языком, в нем присутствуют не только *операции* запросов, но и *операторы*, соответствующие *Data Definition Language* (*DDL*) — языку описания данных. Кроме того, язык содержит *операторы*, предназначенные для управления (администрирования) *БД*.

SQL содержит *разделы*, представленные в табл. 5.1:

Таблица 5.1. Операторы определения данных DDL

Оператор	Смысл	Действие
CREATE TABLE	Создать таблицу	Создает новую таблицу в БД
DROP TABLE	Удалить таблицу	Удаляет таблицу из БД
ALTER TABLE	Изменить таблицу	Изменяет структуру существующей таблицы или ограничения целостности, задаваемые для данной таблицы
CREATE VIEW	Создать представление	Создает виртуальную таблицу, соответствующую некоторому SQL-запросу
ALTER VIEW	Изменить представление	Изменяет ранее созданное представление
DROP VIEW	Удалить представление	Удаляет ранее созданное представление
CREATE INDEX	Создать индекс	Создает индекс для некоторой таблицы для обеспечения быстрого доступа по атрибутам, входящим в индекс
DROP INDEX	Удалить индекс	Удаляет ранее созданный индекс

Таблица 5.2. Операторы манипулирования данными *Data Manipulation* Lanquaqe (DML)

Оператор	Смысл	Действие
DELETE	Удалить строки	Удаляет одну или несколько строк, соответствующих условиям фильтрации, из базовой таблицы. Применение оператора согласуется с принципами поддержки целостности, поэтому этот оператор не всегда может быть выполнен корректно, даже если синтаксически он записан правильно

INSERT	Вставить строку	Вставляет одну строку в базовую таблицу. Допустимы модификации оператора, при которых сразу несколько строк могут быть перенесены из одной таблицы или запроса в базовую таблицу
UPDATE	Обновить строку	Обновляет значения одного или нескольких столбцов в одной или нескольких строках, соответствующих условиям фильтрации

Таблица 5.3. Язык запросов Data Query Lanquaqe (DQL)

Оператор	**Смысл**	**Действие**
SELECT	Выбрать строки	Оператор, заменяющий все операторы *реляционной алгебры* и позволяющий сформировать результирующее отношение, соответствующее запросу

Таблица 5.4. Средства управления транзакциями

Оператор	**Смысл**	**Действие**
COMMIT	Завершить транзакцию	Завершить комплексную взаимосвязанную обработку информации, объединенную в транзакцию
ROLLBACK	Откатить транзакцию	Отменить изменения, проведенные в ходе выполнения транзакции
SAVEPOINT	Сохранить промежуточную точку выполнениятить транзакции	Сохранить промежуточное состояние БД, пометить его для того, чтобы можно было в дальнейшем к нему вернуться

Таблица 5.5. Средства администрирования данных

Оператор	**Смысл**	**Действие**
ALTER DATABASE	Изменить БД	Изменить набор основных объектов в базе данных, ограничений, касающихся всей базы данных
ALTER DBAREA	Изменить область хранения БД	Изменить ранее созданную область хранения
ALTER PASSWORD	Изменить пароль	Изменить пароль для всей базы данных
CREATE DATABASE	Создать БД	Создать новую базу данных, определив основные параметры для нее
CREATE DBAREA	Создать область хранения	Создать новую область хранения и сделать ее доступной для размещения данных
DROP DATABASE	Удалить БД	Удалить существующую базу данных (только в том случае, когда вы имеете право выполнить это действие)
DROP DBAREA	Удалить область хранения БД	Удалить существующую область хранения (если в ней на настоящий момент не располагаются активные данные)
GRANT	Предоставить права	Предоставить права доступа на ряд действий над некоторым объектом БД
REVOKE	Лишить прав	Лишить прав доступа к некоторому объекту или некоторым действиям над объектом

Таблица 5.6. Программный SQL

Оператор	**Смысл**	**Действие**

DECLARE	Определяет курсор для запроса	Задает некоторое имя и определяет связанный с ним запрос к БД, который соответствует виртуальному набору данных
OPEN	Открыть курсор	Формирует виртуальный набор данных, соответствующий описанию указанного курсора и текущему состоянию БД
FETCH	Считать строку из множества строк, определенных курсором	Считывает очередную строку, заданную параметром команды из виртуального набора данных, соответствующего *открытому курсору*
CLOSE	Закрыть курсор	Прекращает доступ к виртуальному набору данных, соответствующему указанному курсору
PREPARE	Подготовить оператор SQL к динамическому выполнению	Сгенерировать план выполнения запроса, соответствующего заданному оператору SQL
EXECUTE	Выполнить оператор SQL, ранее подготовленный к динамическому выполнению	Выполняет ранее подготовленный план запроса

В коммерческих *СУБД* набор основных операторов расширен. В большинство *СУБД* включены *операторы* определения и запуска хранимых процедур и *операторы* определения триггеров.

Типы данных

В языке *SQL*/89 поддерживаются следующие типы данных:

- CHARACTER(n) или CHAR(n) — символьные строки постоянной длины в n символов. При задании данного типа под каждое значение всегда отводится n символов, и если реальное значение занимает менее, чем n символов, то СУБД автоматически дополняет недостающие символы пробелами.
- NUMERIC[(n,m)] — точные числа, здесь n — общее количество цифр в числе, m — количество цифр слева от десятичной точки.
- DECIMAL[(n,m)] — точные числа, здесь n — общее количество цифр в числе, m — количество цифр слева от десятичной точки.
- DEC[(n,m)] — то же, что и DECIMAL[(n,m)].
- INTEGER или INT — целые числа.
- SMALLINT — целые числа меньшего диапазона.

Несмотря на то, что в стандарте SQL1 не определяется точно, что подразумевается под типом INT и SMALLINT (это отдано на откуп реализации), указано только соотношение между этими типами данных, в большинстве реализаций тип данных INTEGER соответствует целым числам, хранимым в четырех байтах, а SMALLINT — соответствует целым числам, хранимым в двух байтах. Выбор одного из этих типов определяется размером числа.

- FLOAT[(n)] — числа большой точности, хранимые в форме с плавающей точкой. Здесь n — число байтов, резервируемое под хранение одного числа. Диапазон чисел определяется конкретной реализацией.
- REAL — вещественный тип чисел, который соответствует числам с плавающей точкой, меньшей точности, чем FLOAT.
- DOUBLE PRECISION специфицирует тип данных с определенной в реализации точностью большей, чем определенная в реализации точность для REAL.

В стандарте SQL92 добавлены следующие типы данных:

- VARCHAR(n) — строки символов переменной длины.
- NCHAR(N) — строки локализованных символов постоянной длины.
- NCHAR *VARYING*(n) — строки локализованных символов переменной длины.

- BIT(n) — строка битов постоянной длины.
- BIT *VARYING*(n) — строка битов переменной длины.
- DATE — календарная дата.
- TIMESTAMP(точность) — дата и время.
- INTERVAL — временной интервал.

Большинство коммерческих *СУБД* поддерживают еще дополнительные типы данных, которые не специфицированы в стандарте. Так, например, практически все *СУБД* в том или ином виде поддерживают *тип данных* для представления неструктурированного текста большого объема. Этот тип аналогичен типу MEMO в настольных *СУБД*. Называются эти типы *по*-разному, например в *ORACLE* этот тип называется LONG, в *DB2* — LONG VARCHAR, в SYBASE и MS *SQL Server* — TEXT.

Однако следует отметить, что специфика реализации отдельных типов данных серьезным образом влияет на результаты запросов к *БД*. Особенно это касается реализации типов данных DATE и TIMESTAMP. Поэтому при переносе приложений будьте внимательны, на разных платформах они могут работать *по*-разному, и одной из причин может быть различие в интерпретации типов данных.

При выполнении сравнений в операциях фильтрации могут использоваться *константы* заданных типов. В стандарте определены следующие *константы*. Для числовых типов данных определены *константы* в виде последовательности цифр с необязательным заданием знака числа и десятичной точкой. То есть правильными будут *константы*:

213.314 612.716 + 551.702

Константы с плавающей запятой задаются, как и в большинстве языков программирования, путем задания мантиссы и порядка, разделенных символом E, например:

2.9E-4 -134.235E7 0.54267E18

Строковые *константы* должны быть заключены в одинарные кавычки:

'Крылов Ю.Д.' 'Санкт-Петербург'

В некоторых реализациях, например MS *SQL Server* и Informix, допустимы двойные кавычки в строковых константах:

"Москва" "New York"

Однако следует отметить, что использование двойных кавычек может вызвать дополнительные проблемы при переносе приложений на другую платформу, поэтому мы рекомендуем *по* возможности избегать такого представления символьных констант.

Константы даты, времени и временного интервала в реляционных *СУБД* представляются в виде строковых констант. Форматы этих констант отличаются в различных *СУБД*. Кроме того, формат представления даты различен в разных странах. В большинстве *СУБД* реализованы способы настройки форматов представления дат или специальные функции преобразования форматов дат, как сделано, например, в СУБД *ORACLE*. Приведем примеры констант в MS *SQL Server*:

March 15, 1999 Mar 15 1999 3/15/1999 3-15-99 1999 MAR 15

В *СУБД ORACLE* та же константа запишется как

15-MAR-99

Кроме пользовательских констант в *СУБД* могут существовать и специальные системные *константы*. Стандарт SQL1 определяет только одну системную константу USER, которая соответствует имени пользователя, под которым вы подключились к *БД*.

В операторах *SQL* могут использоваться выражения, которые строятся *по* стандартным правилам применения знаков арифметических операций сложения (+), вычитания (-), умножения (*) и деления (/). Однако в ряде *СУБД* операция деления (/) интерпретируется как *деление* нацело, поэтому при построении сложных выражений вы можете получить результат, не соответствующий традиционной интерпретации выражения. В стандарт *SQL2* включена возможность выполнения операций сложения и вычитания над датами. В большинстве *СУБД* также определена *операция конкатенации* над строковыми данными, обозначается она, к сожалению, *по*-разному. Так, например, для *DB2 операция конкатенации* обозначается двойной вертикальной чертой, в

MS *SQL Server* — знаком сложения (+), поэтому два выражения, созданные в разных *СУБД*, эквивалентны:

'Mr./Mrs. ' || NAME || ' ' LAST_NAME

'Mr./Mrs. ' + NAME + ' ' LAST_NAME

В стандарте SQL1 не были определены встроенные функции, однако в большинстве коммерческих *СУБД* такие функции были реализованы, и в стандарт *SQL2* уже введен ряд стандартных встроенных функций:

- BITLENGTH(строка) — количество битов в строке;
- CAST(значение AS тип данных) — значение, преобразованное в заданный тип данных;
- CHARLENGTH(строка) — длина строки символов;
- CONVERT(строка USING функция) — строка, преобразованная в соответствии с указанной функцией;
- CURRENTDATE — текущая дата;
- CURRENTTIME(точность) — текущее время с указанной точностью;
- CURRENTTIMESTAMP(точность) — текущие дата и время с указанной точностью;
- LOWER(строка) — строка, преобразованная к нижнему регистру;
- OCTEDLENGTH(строка) — число байтов в строке символов;
- POSITION(первая строка IN вторая строка) — позиция, с которой начинается вхождение первой строки во вторую;
- SUBSTRING(строка FROM n FOR длина) — часть строки, начинающаяся с n-го символа и имеющая указанную длину;
- TRANSLATE(строка USING функция) — строка, преобразованная с использованием указанной функции;
- TRIM(BOTH символ FROM строка) — строка, у которой удалены все первые и последние символы;
- TRIM(*LEADING* символ FROM строка) — строка, в которой удалены все первые указанные символы;
- TRIM(TRAILING символ FROM строка) — строка, в которой удалены последние указанные символы;
- UPPER(строка) — строка, преобразованная к верхнему регистру.

Оператор выбора SELECT

Язык запросов (*Data Query Language*) в *SQL* состоит из единственного оператора SELECT. Этот единственный оператор поиска реализует все *операции реляционной алгебры*. Как просто, всего один оператор. Однако писать запросы на языке *SQL* (грамотные запросы) сначала совсем не просто. Надо учиться, так же как надо учиться решать математические задачки или составлять алгоритмы для решения непростых комбинаторных задач. Один и тот же *запрос* может быть реализован несколькими способами, и, будучи все правильными, они, тем не менее, могут существенно отличаться *по* времени исполнения, и это особенно важно для больших баз данных.

Синтаксис оператора SELECT имеет следующий вид:

SELECT[ALL|DISTINCT](<Список полей>|*)
FROM <Список таблиц>
[WHERE <Предикат-условие выборки или соединения>]
[GROUP BY <Список полей результата>]
[HAVING <Предикат-условие для группы>]
[ORDER BY <Список полей, по которым упорядочить вывод>]

Здесь *ключевое слово* ALL означает, что в результирующий набор строк включаются все строки, удовлетворяющие условиям запроса. Значит, в результирующий набор могут попасть одинаковые строки. И это нарушение принципов теории отношений (в отличие от *реляционной алгебры*, где *по* умолчанию предполагается отсутствие дубликатов в каждом результирующем отноше-

нии). *Ключевое слово* DISTINCT означает, что в результирующий набор включаются только различные строки, то есть дубликаты строк результата не включаются в набор.

Символ *. (звездочка) означает, что в результирующий набор включаются все столбцы из исходных таблиц запроса.

В разделе FROM задается перечень исходных отношений (таблиц) запроса.

В разделе WHERE задаются условия отбора строк результата или условия соединения кортежей исходных таблиц, подобно *операции* условного соединения в *реляционной алгебре*.

В разделе GROUP BY задается *список* полей группировки.

В разделе HAVING задаются предикаты-условия, накладываемые на каждую группу.

В части ORDER BY задается *список* полей упорядочения результата, то есть *список* полей, который определяет порядок сортировки в результирующем отношении. Например, если первым полем списка будет указана Фамилия, а вторым Номер группы, то в результирующем отношении сначала будут собраны в алфавитном порядке студенты, и если найдутся однофамильцы, то они будут расположены в порядке возрастания номеров групп.

В выражении условий раздела WHERE могут быть использованы следующие предикаты:

- *Предикаты сравнения* { =, <>, >,<, >=,<= },которые имеют традиционный смысл.
- Предикат Between A and B —принимает значения между A и B. Предикат истинен, когда сравниваемое значение попадает в заданный диапазон, включая границы диапазона. Одновременно в стандарте задан и противоположный предикат Not Between A and B,который истинен тогда, когда сравниваемое значение не попадает в заданный интервал, включая его границы.
- Предикат вхождения в множество IN (множество) истинен тогда, когда сравниваемое значение входит в множество заданных значений. При этом множество значений может быть задано простым перечислением или встроенным подзапросом. Одновременно существует противоположный предикат NOT IN (множество), который истинен тогда, когда сравниваемое значение не входит в заданное множество.
- *Предикаты сравнения* с образцом LIKE и NOT LIKE. Предикат LIKE требует задания шаблона, с которым сравнивается заданное значение, предикат истинен, если сравниваемое значение соответствует шаблону, и ложен в противном случае. Предикат NOT LIKE имеет противоположный смысл.

По стандарту в шаблон могут быть включены специальные символы:

- символ подчеркивания (_) — для обозначения любого одиночного символа;
- символ процента (%) — для обозначения любой произвольной последовательности символов;
- остальные символы, заданные в шаблоне, обозначают самих себя.
- *Предикат сравнения* с неопределенным значением IS NULL. Понятие неопределенного значения было внесено в концепции баз данных позднее. Неопределенное значение интерпретируется в реляционной модели как значение, неизвестное на данный момент времени. Это значение при появлении дополнительной информации в любой момент времени может быть заменено на некоторое конкретное значение. При сравнении неопределенных значений не действуют стандартные правила сравнения: одно неопределенное значение никогда не считается равным другому неопределенному значению. Для выявления равенства значения некоторого атрибута неопределенному применяют специальные стандартные предикаты:

<имя атрибута>IS NULL и <имя атрибута> IS NOT NULL.

Если в данном кортеже (в данной строке) указанный атрибут имеет неопределенное значение, то предикат IS NULL принимает значение "Истина" (TRUE), а предикат IS NOT NULL — "Ложь" (FALSE), в противном случае предикат IS NULL принимает значение "Ложь", а предикат IS NOT NULL принимает значение "Истина".

Введение Null-значений вызвало необходимость модификации классической двузначной логики и превращения ее в трехзначную. Все логические операции, производимые с неопределенными значениями, подчиняются этой логике в соответствии с заданной таблицей истинности:

A **B** **N** $A \wedge$ $A \vee$

ot A				
T	T	F	TR	TR
RUE	RUE	ALSE	UE	UE
T	F	F	FA	TR
RUE	ALSE	ALSE	LSE	UE
T	N	F	Null	TR
RUE	ull	ALSE		UE
F	T	T	FA	TR
ALSE	RUE	RUE	LSE	UE
F	F	T	FA	FA
ALSE	ALSE	RUE	LSE	LSE
F	N	T	FA	Null
ALSE	ull	RUE	LSE	
N	T	N	Null	TR
ull	RUE	ull		UE
N	F	N	FA	Null
ull	ALSE	ull	LSE	
N	N	N	Null	Null
ull	ull	ull		

- Предикаты существования EXISTS и несуществования NOT EXISTS. Эти предикаты относятся к встроенным подзапросам, и подробнее мы рассмотрим их, когда коснемся вложенных подзапросов.

В условиях поиска могут быть использованы все рассмотренные ранее предикаты.

Отложив на время знакомство с группировкой, рассмотрим детально первые три строки оператора SELECT:

- SELECT — ключевое слово, которое сообщает СУБД, что эта команда — запрос. Все запросы начинаются этим словом с последующим пробелом. За ним может следовать способ выборки — с удалением дубликатов (DISTINCT) или без удаления (ALL, подразумевается по умолчанию). Затем следует список перечисленных через запятую столбцов, которые выбираются запросом из таблиц, или символ '*' (звездочка) для выбора всей строки. Любые столбцы, не перечисленные здесь, не будут включены в результирующее отношение, соответствующее выполнению команды. Это, конечно, не значит, что они будут удалены или их информация будет стерта из таблиц, потому что запрос не воздействует на информацию в таблицах — он только показывает данные.
- FROM — ключевое слово, подобно SELECT, которое должно быть представлено в каждом запросе. Оно сопровождается пробелом и затем именами таблиц, используемых в качестве источника информации. В случае если указано более одного имени таблицы, неявно подразумевается, что над перечисленными таблицами осуществляется операция декартова произведения. Таблицам можно присвоить имена-псевдонимы, что бывает полезно для осуществления операции соединения таблицы с самой собою или для доступа из вложенного подзапроса к текущей записи внешнего запроса (вложенные подзапросы здесь не рассматриваются).

Все последующие разделы оператора SELECT являются необязательными.

Самый простой запрос SELECT без необязательных частей соответствует просто декартову произведению. Например, выражение

SELECT * FROM R1, R2

соответствует декартову произведению таблиц R1 и R2. Выражение

SELECT R1.A, R2.B FROM R1, R2

соответствует проекции декартова произведения двух таблиц на два столбца A из таблицы R1 и B из таблицы R2, при этом дубликаты всех строк сохранены, в отличие от операции про-

ектирования в *реляционной алгебре*, где при проектировании по умолчанию все дубликаты кортежей уничтожаются.

- WHERE — ключевое слово, за которым следует предикат — условие, налагаемое -на запись в таблице, которому она должна удовлетворять, чтобы попасть в выборку, аналогично операции селекции в *реляционной алгебре*.

Рассмотрим базу данных, которая моделирует сдачу сессии в некотором учебном заведении. Пусть она состоит из трех отношений R1, R2, R3. Будем считать, что они представлены таблицами R1, R2 и R3 соответственно.

R1 = (ФИО, Дисциплина, Оценка) ; R2 = (ФИО, Группа) ; R_3 = (Группы, Дисциплина)

R1

ФИО	**Дисциплина**	**Оценка**
Петров Ф. И.	Базы данных	5
Сидоров К. А.	Базы данных	4
Миронов А. В.	Базы данных	2
Степанова К. Е.	Базы данных	2
Крылова Т. С.	Базы данных	5
Сидоров К. А.	Теория информации	4
Степанова К. Е.	Теория информации	2
Крылова Т. С.	Теория информации	5
Миронов А. В.	Теория информации	Null
Владимиров В. А.	Базы данных	5
Трофимов П. А.	Сети и телекоммуникации	4
Иванова Е. А.	Сети и телекоммуникации	5
Уткина Н. В.	Сети и телекоммуникации	5
Владимиров В. А.	Английский язык	4
Трофимов П. А.	Английский язык	5
Иванова Е. А.	Английский язык	3
Петров Ф. И.	Английский язык	5

R2

ФИО	**Группа**
Петров Ф. И.	4906

Сидоров К. А. 4906

Миронов А. В. 4906

Крылова Т. С. 4906

Владимиров В. А. 4906

Степанова К. Е. 4906

Трофимов П. А. 4807

Иванова Е. А. 4807

Уткина Н. В. 4807

R3

Группа	**Дисциплина**
4906	Базы данных
4906	Теория информации
4906	Английский язык
4807	Английский язык
4807	Сети и телекоммуникации

Приведем несколько примеров использования оператора SELECT.

- Вывести список всех групп (без повторений), где должны пройти экзамены.

SELECT DISTINCT Группы FROM R3

Результат:

Группа
4906
4807

- Вывести список студентов, которые сдали экзамен по дисциплине "Базы данных" на "отлично".

- SELECT ФИО
- FROM R1

WHERE Дисциплина = "Базы данных" AND Оценка = 5

Результат:

ФИО
Петров Ф. И.
Крылова Т. С.

Владимирова
В. А.

- Вывести список всех студентов, которым надо сдавать экзамены с указанием названий дисциплин, по которым должны проводиться эти экзамены.

- SELECT ФИО,Дисциплина
- FROM R2,R3

WHERE R2.Группа = R3.Группа;

Здесь часть WHERE задает условия *соединения отношений* R2 и R3, при отсутствии условий соединения в части WHERE результат будет эквивалентен расширенному декартову произведению, и в этом случае каждому студенту были бы приписаны все дисциплины из отношения R3, а не те, которые должна сдавать его группа.

Результат:

ФИО	**Дисциплина**
Петров Ф. И.	Базы данных
Сидоров К. А.	Базы данных
Миронов А. В.	Базы данных
Степанова К. Е.	Базы данных
Крылова Т. С.	Базы данных
Владимиров В. А.	Базы данных
Петров Ф. И.	Теория информации
Сидоров К. А.	Теория информации
Миронов А. В.	Теория информации
Степанова К. Е.	Теория информации
Крылова Т. С.	Теория информации
Владимиров В. А.	Теория информации
Петров Ф. И.	Английский язык
Сидоров К. А.	Английский язык
Миронов А. В.	Английский язык
Степанова К. Е.	Английский язык
Крылова Т. С.	Английский язык
Владимиров В. А.	Английский язык
Трофимов	Сети и телекомму-

П. А. никации
Иванова Е. Сети и телекомму-
А. никации
Уткина Н. Сети и телекомму-
В. никации
Трофимов Английский язык
П. А.
Иванова Е. Английский язык
А.
Уткина Н. Английский язык
В.

- Вывести список лентяев, имеющих несколько двоек.

```
SELECT DISTINCT R1.ФИО
FROM R1 a, R1 b
WHERE a.ФИО = b.ФИО AND
a.Дисциплина <> b.Дисциплина AND
a.Оценка <= 2 AND b.Оценка <= 2;
```

Здесь мы использовали псевдонимы для именования отношения R1 a и b, так как для записи условий поиска нам необходимо работать сразу с двумя экземплярами данного отношения.

Результат:

ФИО

Степанова
К. Е.

Из этих примеров хорошо видно, что логика работы оператора выбора (*декартово произведение*—селекция—*проекция*) не совпадает с порядком описания в нем данных (сначала *список* полей для проекции, потом *список* таблиц для декартова произведения, потом условие соединения). Дело в том, что *SQL* изначально разрабатывался для применения конечными пользователями, и его стремились сделать возможно ближе к языку естественному, а не к языку алгоритмическому. *По* этой причине *SQL* на первых порах вызывает путаницу и раздражение у начинающих его изучать профессиональных программистов, которые привыкли разговаривать с машиной именно на алгоритмических языках.

Наличие неопределенных (Null) значений повышает гибкость обработки информации, хранящейся в *БД*. В наших примерах мы можем предположить ситуацию, когда студент пришел на экзамен, но не сдавал его *по* некоторой причине, в этом случае оценка *по* некоторой дисциплине для данного студента имеет неопределенное *значение*. В данной ситуации можно поставить вопрос: "Найти студентов, пришедших на экзамен, но не сдававших его с указанием названия дисциплины". Оператор SELECT будет выглядеть следующим образом:

```
SELECT ФИО, Дисциплина
FROM R1
WHERE Оценка IS NULL
```

Результат:

ФИО **Дисциплина**

Миронов Теория ин-
А. В. формации

Применение агрегатных функций и вложенных запросов в операторе выбора

В *SQL* добавлены дополнительные функции, которые позволяют вычислять обобщенные групповые значения. Для применения *агрегатных функций* предполагается предварительная операция группировки. В чем состоит суть *операции* группировки? При группировке все множество *кортежей отношения* разбивается на группы, в которых собираются кортежи, имеющие одинаковые значения атрибутов, которые заданы в списке группировки.

Например, сгруппируем *отношение* R1 *по* значению столбца Дисциплина. Мы получим 4 группы, для которых можем вычислить некоторые групповые значения, например количество кортежей в группе, максимальное или минимальное *значение* столбца Оценка.

Это делается с помощью *агрегатных функций*. *Агрегатные функции* вычисляют одиночное *значение* для всей группы таблицы. *Список* этих функций представлен в табл. 5.7.

Таблица 5.7. *Агрегатные функции*

Функция	**Результат**
COUNT	Количество строк или непустых значений полей, которые выбрал запрос
SUM	Сумма всех выбранных значений данного поля
AVG	Среднеарифметическое значение всех выбранных значений данного поля
MIN	Наименьшее из всех выбранных значений данного поля
MAX	Наибольшее из всех выбранных значений данного поля

R1

	ФИО	Дисциплина	Оценка
Группа 1	Петров Ф. И.	Базы данных	5
	Сидоров К. А.	Базы данных	4
	Миронов А. В.	Базы данных	2
	Степанова К. Е.	Базы данных	2
	Крылова Т. С.	Базы данных	5
	Владимиров В. А.	Базы данных	5
Группа 2	Сидоров К. А.	Теория информации	4
	Степанова К. Е.	Теория информации	2
	Крылова Т. С.	Теория информации	5
	Миронов А. В.	Теория информации	Null
Группа 3	Трофимов П. А.	Сети и телекоммуникации	4
	Иванова Е. А.	Сети и телекоммуникации	5
	Уткина Н. В.	Сети и телекоммуникации	5
Группа 4	Владимиров В. А.	Английский язык	4
	Трофимов П. А.	Английский язык	5
	Иванова Е. А.	Английский язык	3
	Петров Ф. И.	Английский язык	5

Агрегатные функции используются подобно именам полей в операторе SELECT, но с одним исключением: они берут имя поля как *аргумент*. С функциями SUM и AVG могут использоваться

только числовые поля. С функциями COUNT, MAX и MIN могут использоваться как числовые, так и символьные поля. При использовании с символьными полями MAX и MIN будут транслировать их в эквивалент *ASCII* кода и обрабатывать в алфавитном порядке. Некоторые *СУБД* позволяют использовать вложенные агрегаты, но это является отклонением от стандарта *ANSI* со всеми вытекающими отсюда последствиями.

Например, можно вычислить количество студентов, сдававших экзамены *по* каждой дисциплине. Для этого надо выполнить *запрос* с группировкой *по* полю "Дисциплина" и вывести в качестве результата название дисциплины и количество строк в группе *по* данной дисциплине. Применение символа * в качестве аргумента функции COUNT означает подсчет всех строк в группе.

```
SELECT R1.Дисциплина, COUNT(*)
FROM R1
GROUP BY R1.Дисциплина
```

Результат:

Дисциплина	**COUNT(*)**
Базы данных	6
Теория информации	4
Сети и телекоммуникации	3
Английский язык	4

Если же мы хотим сосчитать количество сдавших экзамен *по* какой-либо дисциплине, то нам необходимо исключить неопределенные значения из исходного отношения перед группировкой. В этом случае *запрос* будет выглядеть следующим образом:

```
SELECT R1.Дисциплина, COUNT(*)
FROM R1
WHERE R1.Оценка IS NOT NULL
GROUP BY R1.Дисциплина
```

Получим результат:

Дисциплина	**COUNT(*)**
Базы данных	6
Теория информации	3
Сети и телекоммуникации	3
Английский язык	4

В этом случае строка со студентом

Миронов А. В.	**Теория информации**	**null**

не попадет в набор кортежей перед группировкой, поэтому количество кортежей в группе для дисциплины "*Теория информации*" будет на 1 меньше.

Можно применять *агрегатные функции* также и без *операции* предварительной группировки, в этом случае все *отношение* рассматривается как одна *группа* и для этой группы можно вычислить одно *значение* на группу.

Обратившись снова к базе данных "Сессия" (таблицы R1, R2, R3), найдем количество успешно сданных экзаменов:

```
SELECT COUNT(*)
FROM R1
WHERE Оценка > 2;
```

Это, конечно, отличается от выбора поля, поскольку всегда возвращается одиночное *значение*, независимо от того, сколько строк находится в таблице. Аргументом *агрегатных*

функций могут быть отдельные столбцы таблиц. Но для того, чтобы вычислить, например, количество различных значений некоторого столбца в группе, необходимо применить *ключевое слово* DISTINCT совместно с именем столбца. Вычислим количество различных оценок, полученных *по* каждой дисциплине:

```
SELECT R1.Дисциплина, COUNT(DISTINCT R1.Оценка)
FROM R1
WHERE R1.Оценка IS NOT NULL
GROUP BY R1.Дисциплина
```

Результат:

Дисциплина	**COUNT(DISTINCT R1.Оценка)**
Базы данных	3
Теория информации	3
Сети и телекоммуникации	2
Английский язык	3

В результат можно включить *значение* поля группировки и несколько *агрегатных функций*, а в условиях группировки можно использовать несколько полей. При этом группы образуются *по* набору заданных полей группировки. *Операции* с агрегатными функциями могут быть применены к объединению *множества* исходных таблиц. Например, поставим вопрос: определить для каждой группы и каждой дисциплины количество успешно сдавших экзамен и средний балл *по* дисциплине.

```
SELECT R1.Оценка, R1.Дисциплина, COUNT(*), AVR(Оценка)
FROM R1,R2
WHERE R1.ФИО = R2.ФИО AND
R1.Оценка IS NOT NULL AND
R1.Оценка > 2
GROUP BY R1.Оценка R1.Дисциплина
```

Результат:

Дисциплина	**COUNT(*)**	***AVR*(Оценка)**
Базы данных	6	3.83
Теория информации	3	3.67
Сети и телекоммуникации	3	4.66
Английский язык	4	4.25

Мы не можем использовать *агрегатные функции* в предложении WHERE, потому что предикаты оцениваются в терминах одиночной строки, а *агрегатные функции* — в терминах групп строк.

Предложение GROUP BY позволяет определять *подмножество* значений в особом *поле* в терминах другого поля и применять функцию агрегата к подмножеству. Это дает возможность объединять поля и *агрегатные функции* в едином предложении SELECT. *Агрегатные функции* могут применяться как в выражении вывода результатов строки SELECT, так и в выражении условия обработки сформированных групп HAVING. В этом случае каждая агрегатная *функция* вычисляется для каждой выделенной группы. Значения, полученные при вычислении *агрегатных функций*, могут быть использованы для вывода соответствующих результатов или для условия отбора групп.

Построим *запрос*, который выводит группы, в которых *по* одной дисциплине на экзаменах получено больше одной двойки:

```
SELECT R2.Группа
```

```
FROM R1,R2
WHERE R1.ФИО = R2.ФИО AND
R1.Оценка = 2
GROUP BY R2.Группа, R1.Дисциплина
HAVING count(*)> 1
```

В дальнейшем в качестве примера будем работать не с *БД* "Сессия", а с *БД* "Банк", состоящей из одной таблицы F, в которой хранится *отношение* F, содержащее информацию о счетах в филиалах некоторого банка:

```
F = (N, ФИО, Филиал, ДатаОткрытия, ДатаЗакрытия, Остаток);
Q = (Филиал, Город);
```

поскольку на этой базе можно ярче проиллюстрировать работу с агрегатными функциями и группировкой.

Например, предположим, что мы хотим найти суммарный *остаток* на счетах в филиалах. Можно сделать раздельный *запрос* для каждого из них, выбрав SUM(Остаток) из таблицы для каждого филиала. GROUP BY, однако, позволит поместить их все в одну команду:

```
SELECT Филиал, SUM(Остаток)
FROM F
GROUP BY Филиал;
```

GROUP BY применяет *агрегатные функции* независимо для каждой группы, определяемой с помощью значения поля Филиал. *Группа* состоит из строк с одинаковым значением поля Филиал, и *функция* SUM применяется отдельно для каждой такой группы, то есть суммарный *остаток* на счетах подсчитывается отдельно для каждого филиала. *Значение* поля, к которому применяется GROUP BY, имеет, *по* определению, только одно *значение* на группу вывода, как и результат работы агрегатной функции. Поэтому мы можем совместить в одном запросе агрегат и *поле*. Вы можете также использовать GROUP BY с несколькими полями.

Предположим, что мы хотели бы увидеть только те суммарные значения остатков на счетах, которые превышают $5000. Чтобы увидеть суммарные остатки свыше $5000, необходимо использовать предложение HAVING. Предложение HAVING определяет критерии, используемые, чтобы удалять определенные группы из вывода, точно так же как предложение WHERE делает это для индивидуальных строк.

Правильной командой будет следующая:

```
SELECT Филиал, SUM(Остаток)
FROM F
GROUP BY Филиал
HAVING SUM(Остаток) > 5000;
```

Аргументы в предложении HAVING подчиняются тем же самым правилам, что и в предложении SELECT, где используется GROUP BY. Они должны иметь одно *значение* на группу вывода.

Следующая *команда* будет запрещена:

```
SELECT Филиал,SUM(Остаток)
FROM F
GROUP BY Филиал
HAVING ДатаОткрытия = 27/12/1999;
```

Поле ДатаОткрытия не может быть использовано в предложении HAVING, потому что оно может иметь больше чем одно *значение* на группу вывода. Чтобы избежать такой ситуации, предложение HAVING должно ссылаться только на агрегаты и поля, выбранные GROUP BY. Имеется правильный способ сделать вышеупомянутый *запрос*:

```
SELECT Филиал,SUM(Остаток) FROM F
WHERE ДатаОткрытия = '27/12/1999' GROUP BY Филиал;
```

Смысл данного запроса следующий: найти сумму остатков *по* каждому филиалу счетов, открытых 27 декабря 1999 года.

Как и говорилось ранее, HAVING может использовать только аргументы, которые имеют одно *значение* на группу вывода. Практически, ссылки на *агрегатные функции* — наиболее общие, но и поля, выбранные с помощью GROUP BY, также допустимы. Например, мы хотим увидеть суммарные остатки на счетах филиалов в Санкт-Петербурге, Пскове и Урюпинске:

```
SELECT Филиал, SUM(Остаток)
FROM F,Q
WHERE F.Филиал = Q.Филиал
GROUP BY Филиал
HAVING Город IN ("Санкт-Петербург", "Псков", "Урюпинск");
```

Поэтому в арифметических выражениях предикатов, входящих в условие выборки раздела HAVING, прямо можно использовать только спецификации столбцов, указанных в качестве столбцов группирования в разделе GROUP BY. Остальные столбцы можно специфицировать только внутри спецификаций *агрегатных функций* COUNT, SUM, AVG, MIN и MAX, вычисляющих в данном случае некоторое агрегатное *значение* для всей группы строк. Аналогично обстоит дело с подзапросами, входящими в предикаты условия выборки раздела HAVING: если в подзапросе используется характеристика текущей группы, то она может задаваться только путем ссылки на столбцы группирования.

Результатом выполнения раздела HAVING является сгруппированная *таблица*, содержащая только те группы строк, для которых результат вычисления *условия поиска* есть TRUE. В частности, если раздел HAVING присутствует в табличном выражении, не содержащем GROUP BY, то результатом его выполнения будет либо пустая *таблица*, либо результат выполнения предыдущих разделов *табличного выражения*, рассматриваемый как одна *группа* без столбцов группирования.

Вложенные запросы

Теперь вернемся к *БД* "Сессия" и рассмотрим на ее примере использование вложенных запросов.

С помощью *SQL* можно вкладывать запросы внутрь друг друга. Обычно внутренний *запрос* генерирует *значение*, которое проверяется в предикате внешнего *запроса (в предложении* WHERE или HAVING), определяющего, верно оно или нет. Совместно с подзапросом можно использовать *предикат* EXISTS, который возвращает истину, если *вывод* подзапроса не пуст.

В сочетании с другими возможностями оператора выбора, такими как группировка, *подзапрос* представляет собой мощное средство для достижения нужного-результата. В части FROM оператора SELECT допустимо применять синонимы к именам таблицы, если при формировании запроса нам требуется более чем один экземпляр некоторого отношения. Синонимы задаются с использованием ключевого слова AS, которое может быть вообще опущено. Поэтому часть FROM может выглядеть следующим образом:

```
FROM R1 AS A, R1 AS B
```

или

```
FROM R1 A, R1 B;
```

оба выражения эквивалентны и рассматриваются как применения оператора SELECT к двум экземплярам таблицы R1.

Например, покажем, как выглядят на *SQL* некоторые запросы к *БД* "Сессия":

- Список тех, кто сдал все положенные экзамены.

```
SELECT ФИО
FROM R1 as a
WHERE Оценка > 2
GROUP BY ФИО
HAVING COUNT(*) = (SELECT COUNT(*)
FROM R2,R3
WHERE R2.Группа=R3.Группа AND ФИО=a.ФИО)
```

Здесь во встроенном запросе определяется общее число экзаменов, которые должен сдавать каждый студент, обучающийся в группе, в которой учится данный студент, и это число сравнивается с числом экзаменов, которые сдал данный студент.

- Список тех, кто должен был сдавать экзамен по БД, но пока еще не сдавал.
- SELECT ФИО
- FROM R2 a, R3
- WHERE R2.Группа=R3.Группа AND Дисциплина = "БД" AND NOT EXISTS (SELECT ФИО
- FROM R1

WHERE ФИО=a.ФИО AND Дисциплина = "БД")

Предикат EXISTS (*SubQuery*) истинен, когда подзапрос *SubQuery* не пуст, то есть содержит хотя бы один кортеж, в противном случае предикат EXISTS ложен.

Предикат NOT EXISTS обратно — истинен только тогда, когда подзапрос *SubQuery* пуст.

Обратите внимание, каким образом NOT EXISTS с вложенным запросом позволяет обойтись без *операции разности* отношений. Например, формулировка запроса со словом "все" может быть выполнена как бы с двойным отрицанием. Рассмотрим пример базы, которая моделирует поставку отдельных деталей отдельными поставщиками, она представлена одним отношением SP "Поставщики—детали" со схемой

SP (Номер_поставщика, номер_детали) P (номер_детали, наименование)

Вот каким образом формулируется ответ на запрос: "Найти поставщиков, которые поставляют все детали".

```
SELECT DISTINCT НОМЕР_ПОСТАВЩИКА FROM SP SP1 WHERE NOT EXISTS
(SELECT номер_детали FROM P WHERE NOT EXISTS
(SELECT * FROM SP SP2
WHERE SP2.номер_поставщика=SP1.номер_поставщика AND
sp2.номер_детали = P.номер_детали));
```

Фактически мы переформулировали этот *запрос* так: "Найти поставщиков таких, что не существует детали, которую бы они не поставляли". Следует отметить, что этот *запрос* может быть реализован и через *агрегатные функции* с подзапросом:

```
SELECT DISTINCT Номер_поставщика
FROM SP
GROUP BY Номер_поставщика
HAVING Count(DISTINCT номер_детали) =
(SELECT Count( номер_детали)
FROM P)
```

В стандарте SQL92 *операторы* сравнения расширены до многократных сравнений с использованием ключевых слов ANY и ALL. Это расширение используется при сравнении значения *определенного столбца* со столбцом данных, возвращаемым вложенным запросом.

Ключевое слово ANY, поставленное в любом предикате сравнения, означает, что *предикат* будет истинен, если хотя бы для одного значения из подзапроса *предикат сравнения* истинен. *Ключевое слово* ALL требует, чтобы *предикат сравнения* был бы истинен при сравнении со всеми строками подзапроса.

Например, найдем студентов, которые сдали все экзамены на оценку не ниже чем "хорошо". Работаем с той же базой "Сессия", но добавим к ней еще одно *отношение* R4, которое характеризует сдачу лабораторных *работ* в течение семестра:

```
R1 = (ФИО, Дисциплина, Оценка);
R2 = (ФИО, Группа);
R3 = (Группы, Дисциплина )
R4 = (ФИО, Дисциплина, Номерлабраб, Оценка);
Select R1.Фио From R1 Where 4 < = All (Select R1.Оценка
From R1 as R11
```

Where R1.Фио = R11.Фио)

Рассмотрим еще один пример:

Выбрать студентов, у которых оценка *по* экзамену не меньше, чем хотя бы одна оценка *по* сданным им лабораторным работам *по* данной дисциплины:

Select R1.ФИО
From R1
Where R1.Оценка >= ANY (Select R4.Оценка
From R4
Where R1.Дисциплина = R4. Дисциплина AND R1.ФИО = R4.ФИО)

Внешние объединения

Стандарт *SQL2* расширил понятие условного объединения. В стандарте SQL1 при *объединении отношений* использовались только условия, задаваемые в части WHERE оператора SELECT, и в этом случае в результирующее *отношение* попадали только сцепленные *по* заданным условиям кортежи исходных отношений, для которых эти условия были определены и истинны. Однако в действительности часто необходимо объединять таблицы таким образом, чтобы в результат попали все строки из первой таблицы, а вместо тех строк второй таблицы, для которых не выполнено условие соединения, в результат попадали бы неопределенные значения. Или наоборот, включаются все строки из правой (второй) таблицы, а отсутствующие части строк из первой таблицы дополняются неопределенными значениями. Такие объединения были названы внешними в противоположность объединениям, определенным стандартом SQL1, которые стали называться внутренними.

В общем случае *синтаксис* части FROM в стандарте *SQL2* выглядит следующим образом:

FROM <список исходных таблиц>
< выражение естественного объединения >
< выражение объединения >
< выражение перекрестного объединения >
< выражение запроса на объединение >

<список исходных таблиц>::= <имя_таблицы_1> [имя синонима таблицы_1] [...] [,<имя_таблицы_n>[<имя синонима таблицы_n>]]

<выражение естественного объединения>:: =
<имя_таблицы_1> NATURAL { INNER | FULL [OUTER] LEFT [OUTER] | RIGHT [OUTER]} JOIN <имя_таблицы_2>

<выражение перекрестного объединения>:: = <имя_таблицы_1> CROSS JOIN <имя_таблицы_2>

<выражение запроса на объединение>::=
<имя_таблицы_1> UNION JOIN <имя_таблицы_2>

<выражение объединения>::= <имя_таблицы_1> { INNER
FULL [OUTER] | LEFT [OUTER] | RIGHT [OUTER]}
JOIN {ON условие | [USING (список столбцов)]} <имя_таблицы_2>

В этих определениях INNER — означает внутреннее *объединение*, LEFT — левое *объединение*, то есть в результат входят все строки таблицы 1, а части результирующих кортежей, для которых не было соответствующих значений в таблице 2, дополняются значениями NULL (неопределено). *Ключевое слово* RIGHT означает правое внешнее *объединение*, и в отличие от левого объединения в этом случае в результирующее *отношение* включаются все строки таблицы 2, а недостающие части из таблицы 1 дополняются неопределенными значениями, *Ключевое слово* FULL определяет полное внешнее *объединение*: и левое и правое. При полном внешнем объединении выполняются и правое и левое внешние объединения и в результирующее *отношение* включаются все строки из таблицы 1, дополненные неопределенными значениями, и все строки из таблицы 2, также дополненные неопределенными значениями.

Ключевое слово OUTER означает внешнее, но если заданы ключевые слова FULL, LEFT, RIGHT, то *объединение* всегда считается внешним.

Рассмотрим примеры выполнения внешних объединений. Снова вернемся к *БД* "Сессия". Создадим *отношение*, в котором будут стоять все оценки, полученные всеми студентами *по* всем экзаменам, которые они должны были сдавать. Если студент не сдавал данного экзамена, то вместо оценки у него будет стоять неопределенное *значение*. Для этого выполним последовательно естественное внутреннее *объединение* таблиц R2 и R3 *по* атрибуту Группа, а полученное *отношение* соединим левым внешним естественным объединением с таблицей R1, используя столбцы ФИО и Дисциплина. При этом в стандарте разрешено использовать скобочную структуру, так как результат объединения может быть одним из аргументов в части FROM оператора SELECT.

```
SELECT R1.ФИО, R1.Дисциплина, R1.Оценка
FROM (R2 NATURAL INNER JOIN R3 ) LEFT JOIN R1 USING ( ФИО, Дисциплина)
```

Результат:

ФИО	**Дисциплина**	**Оценка**
Петров Ф. И.	Базы данных	5
Сидоров К. А.	Базы данных	4
Миронов А. В.	Базы данных	2
Степанова К. Е.	Базы данных	2
Крылова Т. С	Базы данных	5
Владимиров В. А.	Базы данных	5
Петров Ф. И.	Теория информации	Null
Сидоров К. А.	Теория информации	4
Миронов А. В.	Теория информации	Null
Степанова К. Е.	Теория информации	2
Крылова Т. С	Теория информации	5

ФИО	**Дисциплина**	**Оценка**
Владимиров В. А.	Теория информации	Null
Петров Ф. И.	Английский язык	5
Сидоров К. А.	Английский язык	Null
Миронов А. В.	Английский язык	Null
Степанова К. Е.	Английский язык	Null
Крылова Т. С.	Английский язык	Null

Владимиров В. А.	Английский язык	4
Трофимов П. А.	Сети и телекоммуникации	4
Иванова Е. А.	Сети и телекоммуникации	5
Уткина Н. В.	Сети и телекоммуникации	5
Трофимов П. А.	Английский язык	5
Иванова Е. А.	Английский язык	3
Уткина Н. В.	Английский язык	Null

Рассмотрим еще один пример, для этого возьмем *БД* "Библиотека". Она состоит из трех отношений, имена атрибутов здесь набраны латинскими буквами, что является необходимым в большинстве коммерческих *СУБД*.

BOOKS(ISBN, TITLE, AUTOR, COAUTOR, YEARIZD, PAGES)
READER(NUM_READER, NAME_READER, ADRESS,
HOOM_PHONE, WORK_PHONE, BIRTH_DAY)
EXEMPLARE(INV, ISBN, YES_NO, NUM_READER, DATE_IN, DATE_OUT)

Здесь *таблица* BOOKS описывает все книги, присутствующие в библиотеке, она имеет следующие атрибуты:

- ISBN — уникальный шифр книги;
- TITLE — название книги;
- AUTOR — фамилия автора;
- COAUTOR — фамилия соавтора;
- YEARIZD — год издания;
- PAGES — число страниц.

Таблица READER хранит сведения обо всех читателях библиотеки, и она содержит следующие атрибуты:

- NUM_READER — уникальный номер читательского билета;
- NAME_READER — фамилию и инициалы читателя;
- ADRESS — адрес читателя;
- HOOM_PHONE — номер домашнего телефона;
- WORK_PHONE — номер рабочего телефона;
- BIRTH_DAY — дату рождения читателя.

Таблица EXEMPLARE содержит сведения о текущем состоянии всех экземпляров всех книг. Она включает в себя следующие столбцы:

- INV — уникальный инвентарный номер экземпляра книги;
- ISBN — шифр книги, который определяет, какая это книга, и ссылается на сведения из первой таблицы;
- YES_NO — признак наличия или отсутствия в библиотеке данного экземпляра в текущий момент;
- NUM_READER — номер читательского билета, если книга выдана читателю, и Null в противном случае;
- DATE_IN — если книга у читателя, то это дата, когда она выдана читателю;
- DATE_OUT — дата, когда читатель должен вернуть книгу в библиотеку.

Определим перечень книг у каждого читателя; если у читателя нет книг, то номер экземпляра книги равен NULL. Для выполнения этого поиска нам надо использовать левое внеш-

нее *объединение*, то есть мы берем все строки из таблицы READER и соединяем со строками из таблицы *EXEMPLARE*, если во второй таблице нет строки с соответствующим номером читательского билета, то в строке результирующего отношения *атрибут EXEMPLARE*.INV будет иметь неопределенное *значение* NULL:

```
SELECT READER.NAME_READER, EXEMPLARE.INV
FROM READER LEFT JOIN EXEMPLARE
ON READER.NUM_READER=EXEMPLARE.NUM_READER
```

Операция внешнего объединения, как мы уже упоминали, может использоваться для формирования источников в предложении FROM, поэтому допустимым будет, например, следующий текст запроса:

```
SELECT *
FROM ( BOOKS LEFT JOIN EXEMPLARE)
LEFT JOIN (READER NATURAL JOIN EXEMPLARE)
USING (ISBN)
```

При этом для книг, ни один экземпляр которых не находится на руках у читателей, значения номера читательского билета и дат взятия и возврата книги будут неопределенными.

Перекрестное *объединение* в трактовке стандарта *SQL2* соответствует *операции* расширенного декартова произведения, то есть *операции* соединения двух таблиц, при которой каждая строка первой таблицы соединяется с каждой строкой второй таблицы.

Операция *запроса на объединение эквивалентна операции* теоретико-множественного объединения в алгебре. При этом требование эквивалентности схем исходных отношений сохраняется. *Запрос* на *объединение* выполняется *по* следующей схеме:

```
SELECT — запрос
UNION
SELECT — запрос
UNION
SELECT — запрос
```

Все запросы, участвующие в *операции* объединения, не должны содержать выражений, то есть вычисляемых полей.

Например, нужно вывести *список* читателей, которые держат на руках книгу "Идиот" или книгу "Преступление и наказание". Вот как будет выглядеть *запрос*:

```
SELECT READER.NAME_READER
FROM READER, EXEMPLARE,BOOKS
WHERE EXEMPLARE.NUM_READER= READER.NUM_READER AND
EXEMPLARE.ISBN = BOOKS.ISBN AND
BOOKS.TITLE = "Идиот"
UNION
SELECT READER.NAME_READER
FROM READER, EXEMPLARE,BOOKS
WHERE EXEMPLARE.NUM_READER= READER.NUM_READER AND
EXEMPLRE.ISBN = BOOKS.ISBN AND
BOOKS.TITLE = "Преступление и наказание"
```

По умолчанию при выполнении запроса на *объединение* дубликаты кортежей всегда исключаются. Поэтому, если найдутся читатели, у которых находятся на руках обе книги, то они все равно в результирующий *список* попадут только один раз.

Запрос на *объединение* может объединять любое число исходных запросов.

Так, к предыдущему запросу можно добавить еще читателей, которые держат на руках книгу "Замок":

```
UNION
SELECT READER.NAME_READER
```

```
FROM READER, EXEMPLARE,BOOKS
WHERE EXEMPLARE.NUM_READER= READER.NUM_READER AND
EXEMPLARE.ISBN = BOOKS.ISBN AND
BOOKS.TITLE = "Замок"
```

В том случае, когда вам необходимо сохранить все строки из исходных отношений, необходимо использовать *ключевое слово* ALL в *операции* объединения. В случае сохранения дубликатов кортежей схема выполнения запроса на *объединение* будет выглядеть следующим образом:

```
SELECT — запрос
  UNION   ALL
SELECT -  запрос
  UNION   ALL
SELECT -  запрос
```

Однако тот же результат можно получить простым изменением фразы WHERE первой части исходного запроса, соединив локальные условия логической операцией ИЛИ и исключив дубликаты кортежей.

```
SELECT DISTINCT READER.NAME_READER
FROM READER, EXEMPLARE,BOOKS
WHERE EXEMPLARE.NUM_READER= READER.NUM_READER AND
EXEMPLARE.ISBN = BOOKS.ISBN AND
BOOKS.TITLE = "Идиот" OR
BOOKS.TITLE = "Преступление и наказание" OR
BOOKS.TITLE = "Замок"
```

Ни один из исходных запросов в *операции* UNION не должен содержать предложения упорядочения результата ORDER BY,однако результат объединения может быть упорядочен, для этого предложение ORDER BY с указанием списка столбцов упорядочения записывается после текста последнего исходного SELECT- запроса.

Операторы манипулирования данными

В *операции* манипулирования данными входят три *операции*: операция удаления записей — ей соответствует оператор DELETE, операция добавления или ввода новых записей — ей соответствует оператор INSERT и операция изменения (обновления записей) — ей соответствует оператор UPDATE. Рассмотрим каждый из операторов подробнее.

Все *операторы* манипулирования данными позволяют изменить данные только в одной таблице.

Оператор ввода данных INSERT имеет следующий *синтаксис*:

```
INSERT INTO имя_таблицы [(<список столбцов>) ]
VALUES (<список значений>)
```

Подобный *синтаксис* позволяет ввести только одну строку в таблицу. Задание списка столбцов необязательно тогда, когда мы вводим строку с заданием значений всех столбцов. Например, введем новую книгу в таблицу BOOKS

```
INSERT INTO BOOKS ( ISBN,TITLE,AUTOR,COAUTOR,YEARIZD,PAGES)
  VALUES ("5-88782-290-2",
       "Аппаратные средства IBM PC. Энциклопедия",
       "Гук М. ","",2000,816)
```

В этой книге только один *автор*, нет соавторов, но мы в списке столбцов задали столбец COAUTOR, поэтому мы должны были ввести соответствующее *значение*

в разделе VALUES. Мы ввели пустую строку, потому что мы знаем точно, что нет соавтора. Мы могли бы ввести *неопределенное значение* NULL.

Так как мы вводим полную строку, то мы можем не задавать *список* столбцов, ограничиться только заданием перечня значений, в этом случае оператор ввода будет выглядеть следующим образом:

```
INSERT INTO BOOKS VALUES ("5-88782-290-2",
```

"Аппаратные средства IBM PC. Энциклопедия",
"Гук М.","",2000,816)

Результаты работы обоих операторов одинаковые.

Наконец, мы можем ввести неполный перечень значений, то есть не вводить соавтора, так как он отсутствует для данного издания. Но в этом случае мы должны задать *список* вводимых столбцов, тогда оператор ввода будет выглядеть следующим образом:

```
INSERT INTO BOOKS ( ISBN,TITLE,AUTOR,YEARIZD,PAGES)
  VALUES ("5-88782-290-2",
        "Аппаратные средства IBM PC. Энциклопедия",
        "Гук М.",2000,816)
```

Столбцу COAUTOR будет присвоено в этом случае *значение* NULL.

Какие столбцы должны быть заданы при вводе данных? Это определяется тем, как описаны эти столбцы при описании соответствующей таблицы, и будет рассмотрено более подробно при описании языка *Data Definition Language* (*DDL*) в "лекции 8" . Здесь мы пока отметим, что если столбец или *атрибут* имеет признак обязательный (NOT NULL) при описании таблицы, то оператор INSERT должен обязательно содержать данные для ввода в каждую строку данного столбца. Поэтому если в таблице все столбцы обязательные, то каждая вводимая строка должна содержать полный перечень вводимых значений, а указание имен столбцов в этом случае необязательно. В противном случае, если имеется хотя бы один необязательный столбец и вы не вводите в него значений, задание списка имен столбцов — обязательно.

В набор значений могут быть включены специальные функции и выражения. Ограничением здесь является то, что значения этих функций должны быть определены на момент ввода данных. Поэтому, например, мы можем сформировать оператор ввода данных в таблицу EXEMPLARE следующим образом:

```
INSERT INTO EXEMPLARE (INV,ISBN,YES_NO,NUM_READER,DATE_IN, DATE_OUT)
VALUES (1872, "5-88782-290-2",NO,
344,GetDate(),DateAdd(d,GetDate(),14))
```

И это означает, что мы выдали экземпляр книги с инвентарным номером 1872 читателю с номером читательского билета 344, отметив, что этот экземпляр не присутствует с этого момента в библиотеке, и определили дату выдачи книги как текущую дату (*функция* GetDate()), а дату возврата задали двумя неделями позднее, использовав при этом функцию DateAdd (), которая позволяет к одной дате добавить заданное количество интервалов даты и тем самым получить новое *значение* типа "дата". Мы добавили 14 дней к текущей дате.

Оператор ввода данных позволяет ввести сразу множество строк, если их можно выбрать из некоторой другой таблицы. Допустим, что у нас есть *таблица* со студентами и в ней указаны основные данные о студентах: их фамилии, адреса, домашние телефоны и даты рождения. Тогда мы можем сделать всех студентов читателями нашей библиотеки одним оператором:

```
INSERT INTO READER (NAME_READER, ADRESS, HOOM_PHONE, BIRTH_DAY)
SELECT (NAME_STUDENT, ADRESS, HOOM_PHONE, BIRTH_DAY)
FROM STUDENT
```

При этом номер читательского билета может назначаться автоматически, поэтому мы не вводим значения этого столбца в таблицу. Кроме того, мы предполагаем, что у студентов дневного отделения еще нет работы и поэтому нет рабочего телефона, и мы его не вводим .

Оператор удаления данных позволяет удалить одну или несколько строк из таблицы в соответствии с условиями, которые задаются для удаляемых строк.

Синтаксис оператора DELETE следующий:

```
DELETE FROM имя_таблицы [WHERE условия_отбора]
```

Если условия отбора не задаются, то из таблицы удаляются все строки, однако это не означает, что удаляется вся *таблица*. Исходная *таблица* остается, но она остается пустой, незаполненной.

Например, если нам надо удалить результаты прошедшей сессии, то мы можем удалить все строки из отношения R1 командой

```
DELETE FROM R1
```

Условия отбора в части WHERE имеют тот же вид, что и условия фильтрации в операторе SELECT.Эти условия определяют, какие строки из исходного отношения будут удалены. Например, если мы исключим студента Миронова А. В., то мы должны написать следующую команду:

```
DELETE FROM R2
WHERE ФИО = 'Миронов А.В.'
```

В части WHERE может находиться встроенный *запрос*. Например, если нам надо исключить неуспевающих студентов, то *по* закону о высшем образовании неуспевающим считается студент, имеющий две и более задолженности *по* последней сессии. Тогда нам в условиях отбора надо найти студентов, имеющих либо две или более двоек, либо два и более несданных экзамена из числа тех, которые студент сдавал. Для поиска таких горе-студентов нам надо выбрать из отношения R1 все строки с оценкой 2 или с неопределенным значением, потом надо сгруппировать полученный результат *по* атрибуту ФИО и, подсчитав количество строк в каждой группе, которое соответствует количеству несданных экзаменов каждым студентом, отобрать те группы, у которых количество строк не менее двух. Теперь попробуем просто записать эту сложную конструкцию на *SQL* и убедимся, что этот сложный *запрос* записывается достаточно компактно.

```
DELETE FROM R2 WHERE R2.ФИО IN (SELECT R1.ФИО FROM R1
WHERE Оценка = 2 OR Оценка IS NULL
GROUP BY R1.ФИО HAVING COUNT(*) >= 2
```

Однако при выполнении *операции* DELETE, включающей сложный *подзапрос*, в подзапросе нельзя упоминать таблицу, из которой удаляются строки, поэтому *СУБД* отвергнет такой красивый *подзапрос*, который попытается удалить всех не только сдававших, но и несдававших студентов, которые имеют более двух задолженностей.

```
DELETE FROM R2 WHERE R2.ФИО IN (SELECT R1.ФИО
FROM (R2 NATURAL INNER JOIN R3 )
LEFT JOIN R1 USING ( ФИО, Дисциплина)
WHERE Оценка = 2 OR Оценка IS NULL
GROUP BY R1.ФИО
HAVING COUNT(*) >= 2
```

Все *операции* манипулирования данными связаны с понятием целостности *базы данных*, которое будет рассматриваться далее в "лекции 9" . В настоящий момент мне бы хотелось отметить только то, что *операции* манипулирования данными не всегда выполнимы, даже если синтаксически они написаны правильно. Действительно, если мы бы захотели удалить какую-нибудь группу из отношения R3, то *СУБД* не позволила бы нам это сделать, так как в отношениях R1 и R2 есть строки, связанные с удаляемой строкой в отношении R3. Почему так делается, мы узнаем позднее, а пока просто примем к сведению, что не все *операторы* манипулирования выполнимы.

Операция обновления данных UPDATE требуется тогда, когда происходят изменения во внешнем мире и их надо адекватно отразить в базе данных, так как надо всегда помнить, что *база данных* отражает некоторую предметную область. Например, в нашем учебном заведении произошло счастливое событие, которое связано с тем, что госпожа Степанова К. Е. пересдала экзамен *по* дисциплине "*Базы данных*" с двойки сразу на четверку. В этом случае нам надо срочно выполнить соответствующую корректировку таблицы R1. Операция обновления имеет следующий формат:

```
UPDATE имя_таблицы
SET имя_столбца = новое_значение [WHERE условие_отбора]
```

Часть *WHERE* является необязательной, так же как и в операторе DELETE. Она играет здесь ту же роль, что и в операторе DELETE, — позволяет отобрать строки, к которым будет применена

операция модификации. Если условие отбора не задается, то операция модификации будет применена ко всем строкам таблицы.

Для решения ранее поставленной задачи нам необходимо выполнить следующую операцию

```
UPDATE R1
SET R1.Оценка  = 4
WHERE R1.ФИО = "Степанова  К.Е." AND R1.Дисциплина  = "Базы данных"
```

В каких случаях требуется провести изменение в нескольких строках? Это не такая уж редкая задача. Например, если мы расширим нашу учебную базу данных еще одним отношением, которое содержит перечень курсов, на которых учатся наши студенты, то можно с помощью *операции* обновления промоделировать операцию перевода групп на следующий курс. Пусть новое *отношение* R4 имеет следующую схему:

R4 = <Группа, Курс>

R4

Г
руппа урс

4
906

4
807

В этом случае перевод на следующий курс можно выполнить следующей операцией обновления:

```
UPDATE R4
SET R4.Курс  = R4.Курс  + 1
```

И результат будет выглядеть следующим образом:

Г
руппа урс

49
06

48
07

Операция модификации, так же как и операция удаления, может использовать сложные подзапросы. Расширим нашу базу еще одним отношением, которое будет содержать перечень студентов, получающих стипендию с указанием надбавки, которую они получают за отличную учебу. Исходно там могут находиться все студенты с указанием неопределенного размера стипендии. *По* мере анализа отношения R1 мы можем постепенно заменять неопределенные значения на конкретные размеры стипендии. *Отношение* R5 имеет вид:

R5

ФИО	Группа	Стипендия
Петров Ф. И.	4906	<Null>
Сидоров К. А.	4906	<Null>
Миронов А. В.	4906	<Null>
Крылова Т. С.	4906	<Null>
Владимиров В. А.	4906	<Null>
Трофимов	4	<Null

П. А. 807 >
Иванова Е. 4 <Null
А. 807 >
Уткина Н. 4 <Null
В. 807 >

Будем считать наличие трех пятерок *по* сессии признаком повышенной стипендии, + 50% к основной, наличие двух пятерок из сданных экзаменов и отсутствие двоек и троек на сданных экзаменах — признаком повышения стипендии на 25%, наличие хотя бы одной двойки среди сданных экзаменов — признаком снятия или отсутствия стипендии вообще, то есть –100% надбавки. При отсутствии троек на сданных экзаменах назначим обычную стипендию с надбавкой 0%. Однако все эти изменения мы должны будем сделать отдельными операциями обновления.

Назначение повышенной стипендии:

```
UPDATE R5
SET R5.Стипендия = 50% WHERE R5.ФИО IN
(SELECT R1.ФИО
FROM R1
WHERE R1.Оценка = 5
GROUP BY R1.ФИО
HAVING COUNT(*) =3 )
```

Назначение стипендии с надбавкой 25%:

```
UPDATE R5
SET R5.Стипендия = 25% WHERE R5.ФИО
IN (SELECT R1.ФИО FROM R1
WHERE R1.ФИО NOT
IN (SELECT A.ФИО FROM R1 A
WHERE A.Оценка <=3 OR A.Оценка IS NULL)
GROUP BY R1.ФИО HAVING COUNT(*)>=2 )
```

Назначение обычной стипендии:

```
UPDATE R5
SET R5.Стипендия = 0%
WHERE R5.ФИО IN (SELECT R1.ФИО FROM R1
WHERE R1.Оценка >=4 AND R1.ФИО NOT IN (SELECT A.ФИО FROM R1 A
WHERE A.Оценка <= 3 OR A.Оценка IS NULL) )
```

Снятие стипендии:

```
UPDATE R5
SET R5.Стипендия = —100% WHERE R5.ФИО IN
(SELECT R1.ФИО FROM R1
WHERE R1.Оценка <= 2 OR
R1.Оценка IS NULL)
```

Почему мы в первом запросе на обновление не использовали дополнительную проверку на отсутствие двоек, троек и несданных экзаменов, как мы сделали это при назначении следующих видов стипендии? Просто мы учли особенности нашей *предметной области*: у нас в соответствии с исходными данными только 3 экзамена. Но если мы можем предположить, что число экзаменов может быть произвольным и изменяться от семестра к семестру, то нам надо изменить наш *запрос*. *Запрос* — это некоторый *алгоритм* решения конкретной задачи, которую мы формулируем заранее на естественном языке. И оттого, что наша задача решается всего одним оператором языка *SQL*, она не становится примитивной. *Мощность* языка *SQL* и состоит в том, что он позволяет одним предложением сформулировать ответы на достаточно сложные запросы, для реализации которых на традиционных языках понадобилось бы писать большую программу. Итак, подумаем, как нам надо изменить текст нашего запроса на обновление для назначения повышен-

ной стипендии при любом количестве сданных экзаменов. Прежде всего, каждая *группа* может иметь свое число экзаменов в сессию, это зависит от специальности и учебного плана, *по* которому учится данная *группа*. Поэтому для каждого студента нам надо знать, сколько экзаменов он должен был сдавать и сколько экзаменов он сдал на пять, и в том случае, когда эти два числа равны, мы можем назначить ему повышенную стипендию.

Будем решать нашу задачу *по* шагам. В конечном счете нам все равно надо знать, сколько экзаменов должен сдавать каждый конкретный студент, поэтому сначала сосчитаем количество экзаменов, которые должна сдавать *группа*, в которой учится этот студент.

Это мы делать умеем, для этого надо сделать *запрос* SELECT над отношением R3, сгруппировав его *по* атрибуту Группа, и вывести для каждой группы количество дисциплин, *по* которым должны сдаваться экзамены. Если мы учтем, что в одной сессии *по* одной дисциплине не бывает более одного экзамена, то можно просто подсчитывать количество строк в каждой группе.

```
SELECT R3.Группа, число_экзаменов = COUNT(*)
FROM R3 GROUP BY R3.Группа
```

Однако нам нужен не этот *запрос*, нам нужен *запрос*, в котором мы определяем для каждого студента количество экзаменов. Этот *запрос* мы должны строить *по* схеме встроенного запроса:

```
SELECT COUNT(*)
  FROM R3
  WHERE R2.Группа = R3.Группа
  GROUP BY R3.Группа
```

А почему мы здесь в части FROM не написали имя второго отношения R2? Мы имя этого отношения укажем для связи с вышестоящим запросом, когда будем формировать *запрос* полностью. Теперь попробуем сформулировать полностью *запрос*. Нам надо объединить отношения R1 и R2 *по* атрибуту ФИО, нам надо знать группу, в которой учится каждый студент, далее надо выбрать все строки с оценкой 5 и сгруппировать их *по* фамилии студента, сосчитав количество строк в каждой группе, а выбирать мы будем те группы, в которых число строк в группе равно числу строк во встроенном запросе, рассмотренном ранее, при условии равенства количества строк в группе результату подзапроса, который выводит только одно число.

```
SELECT R1.ФИО FROM R1,R2
  WHERE R1. ФИО = R2.ФИО AND R1.Оценка = 5
  GROUP BY R1.ФИО HAVING COUNT(*) = (SELECT  COUNT(*)
    FROM R3 WHERE R2.Группа = R3.Группа
    GROUP BY R3.Группа)
```

Ну а теперь нам осталась последняя простейшая операция: надо заменить старый вложенный *запрос*, определявший отличников, получивших три пятерки на сессии, на новый универсальный *запрос*:

```
UPDATE R5
SET R5.Стипендия = 50%
  WHERE R5.ФИО IN (SELECT R1.ФИО
    FROM R1,R2
    WHERE R1. ФИО = R2.ФИО AND
        R1.Оценка = 5 GROUP BY R1.ФИО
    HAVING COUNT(*) = (SELECT COUNT(*) FROM R3
       WHERE R2.Группа = R3.Группа GROUP BY R3.Группа))
```

Вот какой сложный *запрос* мы построили. Это ведь практически один оператор, а какую сложную задачу он решает. Действительно, *мощность* языка *SQL* иногда удивляет даже профессионалов, кажется невозможно построить один *запрос* для решения конкретной задачи, но когда начинаешь поэтапно его конструировать — все получается. Самое сложное — это сделать переход от словесной формулировки задачи к представлению ее в терминах нашего *SQL*, но этот процесс сродни процессу алгоритмизации при решении задач традиционного программирования, а он всегда был самым трудным, творческим и неформализуемым процессом. Недаром на заре развития

программирования известный американский специалист *по* программированию Дональд Е. Кнут озаглавил свой многотомный капитальный труд *по* теории и практике программирования "Искусство программирования для ЭВМ" ("The *art* of *computer programming*").

Обобщенная структура СУБД

Мы рассмотрели отдельные аспекты работы *СУБД*. Теперь попробуем кратко обобщить все, что узнали, и построим некоторую условную обобщенную структуру *СУБД*. На рис. 14.1 изображена такая структура. Здесь условно показано, что *СУБД* должна управлять внешней памятью, в которой расположены файлы с данными, файлы журналов и файлы системного каталога.

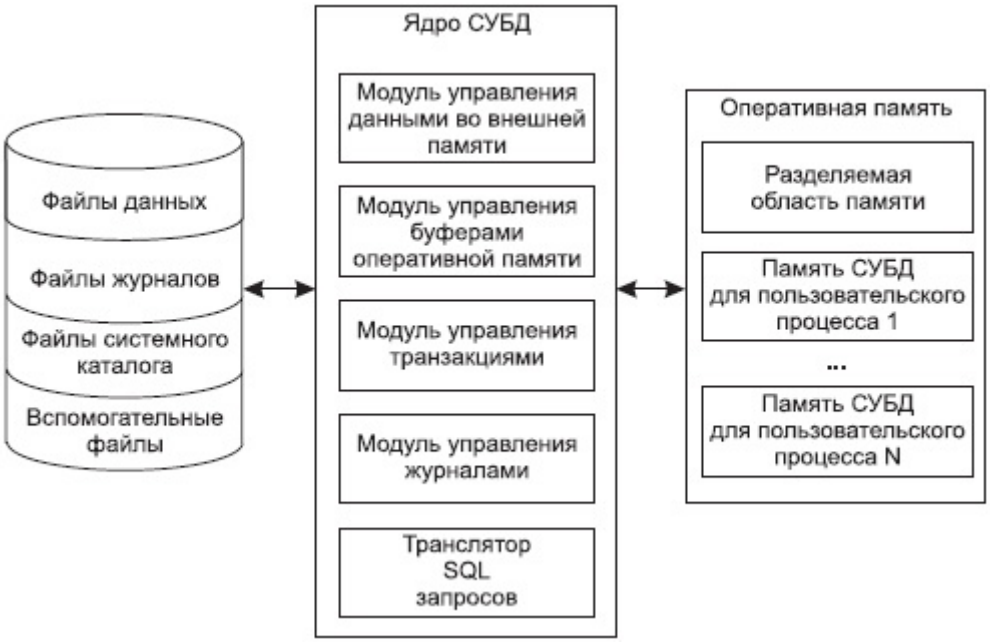


Рис. 14.1. Обобщенная структура СУБД

С другой стороны, *СУБД* управляет и оперативной памятью, в которой располагаются буфера с данными, буфера журналов, данные системного каталога, которые необходимы для поддержки целостности и проверки привилегий пользователей. Кроме того, в оперативной памяти во время работы *СУБД* располагается *информация*, которая соответствует текущему состоянию обработки запросов, там хранятся планы выполнения скомпилированных запросов и т. д.

Модуль управления внешней памятью обеспечивает создание необходимых структур внешней памяти как для хранения данных, непосредственно входящих в *БД*, так и для служебных целей, например для ускорения доступа к данным в некоторых случаях (обычно для этого используются индексы). Как мы рассматривали ранее, в некоторых реализациях *СУБД* активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня *устройств внешней памяти*. Но подчеркнем, что в развитых *СУБД* пользователи в любом случае не обязаны знать, использует ли *СУБД* файловую систему, и если использует, то как организованы файлы. В частности, *СУБД* поддерживает собственную систему именования объектов *БД*.

Модуль управления буферами оперативной памяти предназначен для решения задач эффективной буферизации, которая используется практически для выполнения всех остальных *функций СУБД*.

Условно оперативную *память*, которой управляет *СУБД*, можно представить как совокупность буферов, хранящих страницы данных, буферов, хранящих страницы журналов транзакций и область совместно используемого пула (см. рис. 14.2). Последняя область содержит фрагменты системного каталога, которые необходимо постоянно держать в оперативной памяти, чтобы ускорить обработку запросов пользователей, и область операторов *SQL* с курсорами. Фрагменты си-

стемного каталога в некоторых реализациях называются словарем данных. В стандарте *SQL2* определены общие требования к системному каталогу.

Рис. 14.2. Оперативная память, управляемая СУБД

Системный каталог в реляционных *СУБД* представляет собой совокупность специальных таблиц, которыми владеет сама *СУБД*. Таблицы системного каталога создаются автоматически при установке программного обеспечения сервера *БД*. Все системные таблицы обычно объединяются некоторым специальным "системным идентификатором пользователя". При обработке *SQL*-запросов *СУБД* постоянно обращается к этим таблицам. В некоторых *СУБД* разрешен ограниченный *доступ* пользователей к ряду системных таблиц, однако только в режиме чтения. Только *системный администратор* имеет некоторые *права* на модификацию данных в некоторых системных таблицах.

Каждая *таблица* системного каталога содержит информацию об отдельных структурных элементах *БД*. В стандарте *SQL2* определены следующие системные таблицы:

Таблица 14.1. Содержание системного каталога по стандарту *SQL2*

Системная таблица	**Содержание**
USERS	Одна строка для каждого идентификатора пользователя с зашифрованным паролем
SCHEMA	Одна строка для каждой информационной схемы
DATA TYPE DESCRIPTION	Одна строка для каждого домена или столбца, имеющего определенный тип данных
DOMAINS	Одна строка для каждого домена
DOMAIN *CONSTRAINS*	Одна строка для каждого ограничивающего условия, наложенного на домен
TABLES	Одна строка для каждой таблицы с указанием имени, владельца, количества столбцов, размеров данных столбцов, и т. д.
VIEWS	Одна строка для каждого представления с указанием имени, имени владельца, запроса, который определяет представление и т. д.
COLUMNS	Одна строка для каждого столбца с указанием имени столбца, имени таблицы или представления, к которому он относится, *типа данных столбца*, его размера, допустимости или недопустимости неопределенных значений (NULL) и т. д.
VIEW TABLE USAGE	Одна строка для каждой таблицы, на которую имеется ссылка в каком-либо представлении (если представление многотабличное, то для каждой таблицы заносится одна строка)
VIEW COLUMN USAGE	Одна строка для каждого столбца, на который имеется ссылка в некотором представлении
TABLE *CONSTRAINS*	Одна строка для каждого условия ограничения, заданного в каком-либо определении таблицы
KEY COLUMN USAGE	Одна строка для каждого столбца, на который наложено условие

	уникальности и который присутствует в определении первичного или внешнего ключа (если первичный или внешний ключ заданы несколькими столбцами, то для каждого из них задается отдельная строка)
REFERENTIAL CONSTRAINTS	Одна строка для каждого внешнего ключа, присутствующего в определении таблицы
CHECK CONSTRAINTS	Одна строка для каждого условия проверки, заданного в определении таблицы
CHECK TABLE USAGE	Одна строка для каждой таблицы, на которую имеется ссылка в условиях проверки, ограничительном условии для домена или всей таблицы
CHECK COLUMN USAGE	Одна строка для каждого столбца, на который имеется ссылка в условии проверки, ограничительном условии для домена или ином ограничительном условии
ASSERTIONS	Одна строка для каждого декларативного утверждения целостности
TABLE PRIVILEGES	Одна строка для каждой привилегии, предоставленной на какую-либо таблицу
COLUMN PRIVILEGES	Одна строка для каждой привилегии, предоставленной на какой-либо столбец
USAGE PRIVILEGES	Одна строка для каждой привилегии, предоставленной на какой-либо домен, набор символов и т. д.
CHARACTER SETS	Одна строка для каждого заданного набора символов
COLLATIONS	Одна строка для заданной последовательности
TRANSLATIONS	Одна строка для каждого заданного преобразования
SQL LAGUAGES	Одна строка для каждого заданного языка, поддерживаемого СУБД

Стандарт *SQL2* не требует, чтобы *СУБД* в точности поддерживала требуемый набор системных таблиц. Стандарт ограничивается требованием того, чтобы для рядовых пользователей были доступны некоторые специальные представления системного каталога. Поэтому системные таблицы организованы по-разному в разных *СУБД* и имеют различные имена, но большинство *СУБД* предо-ставля-ют ряд основных представлений рядовым пользователям.

Кроме того, системный каталог отражает некоторые дополнительные возможности, предоставляемые конкретными *СУБД*. Так, например, в системном каталоге *Oracle* присутствуют таблицы синонимов.

Область *SQL* содержит данные связывания, временные буферы, *дерево* разбора и *план выполнения* для каждого оператора *SQL*, переданного серверу *БД*. Область разделяемого пула ограничена в размере, поэтому, возможно, в ней не могут поместиться все *операторы SQL*, которые были выполнены с момента запуска сервера *БД*. *Ядро СУБД* удаляет старые, давно не используемые *операторы*, освобождая *память* под новые *операторы SQL*. Если *пользователь* выполняет *запрос*, *план выполнения* которого уже хранится в разделяемом пуле, то *СУБД* не производит его разбор и построение нового плана, она сразу запускает его на выполнение, возможно, с новыми параметрами.

Модуль управления транзакциями поддерживает *механизмы* фиксации и отката транзакций, он связан с модулем управления буферами оперативной памяти и обеспечивает сохранение всей информации, которая требуется после мягких или жестких сбоев в системе. Кроме того, *модуль* управления транзакциями содержит специальный механизм поиска *тупиковых ситуаций* или взаимоблокировок и реализует одну из принятых стратегий принудительного завершения транзакций для развязывания *тупиковых ситуаций*.

Особое внимание надо обратить на *модуль* поддержки *SQL*. Это практически *транслятор* с языка *SQL* и блок *оптимизации запросов*.

В общем, *оптимизация запросов* может быть разделена на синтаксическую и семантическую.

Методы синтаксической оптимизации запросов

Методы *синтаксической оптимизации* запросов связаны с построением некоторой эквивалентной формы, называемой иногда канонической формой, которая требует меньших затрат на выполнение запроса, но дает результат, полностью эквивалентный исходному запросу.

К методам, используемым при *синтаксической оптимизации* запросов, относятся следующие:

- **Логические преобразования запросов**.Прежде всего это относится к преобразованию предикатов, входящих в условие выборки. Предикаты, содержащие операции сравнения простых значений. Такой предикат имеет вид арифметическое выражение ОС арифметическое выражение, где ОС — операция сравнения, а арифметические выражения левой и правой частей в общем случае содержат имена полей отношений и константы (в языке SQL среди констант могут встречаться и имена переменных объемлющей программы, значения которых становятся известными только при реальном выполнении запроса).

Канонические представления могут быть различными для предикатов разных типов. Если предикат включает только одно имя поля, то его каноническое представление может, например, иметь вид имя поля ОС константное арифметическое выражение (эта форма предиката — простой предикат селекции — очень полезна при выполнении следующего этапа оптимизации). Если начальное представление предиката имеет вид

(n+12)*R.B ОС 100

здесь n — переменная языка, R.B — имя столбца B отношения R, *ОС* - допустимая операция сравнения.

Каноническим представлением такого предиката может быть

R.B ОС 100/(n+12)

В этом случае мы один раз для заданного значения переменной n вычисляем выражение в скобках и правую часть операции сравнения 100/(n +12), а потом каждую строку можем сравнивать с полученным значением.

Если предикат включает в точности два имени поля разных отношений (или двух разных вхождений одного отношения), то его каноническое представление может иметь вид имя поля ОС арифметическое выражение, где арифметическое выражение в правой части включает только константы и второе имя поля (это тоже форма, полезная для выполнения следующего шага оптимизации, — предикат соединения; особенно важен случай *эквисоединения*, когда ОС — это равенство). Если в начальном представлении предикат имеет вид:

12*(R1.A)-n*(R2.B) ОС m,

то его каноническое представление:

R1.A ОС (m+n*(R2.B)/12

В общем случае желательно приведение предиката к каноническому представлению вида арифметическое выражение ОС константное арифметическое выражение, где выражения правой и левой частей также приведены к каноническому представлению. В дальнейшем можно произвести поиск общих арифметических выражений в разных предикатах запроса. Это оправдано, поскольку при выполнении запроса вычисление арифметических выражений будет производиться при выборке каждого очередного кортежа, то есть потенциально большое число раз.

При приведении предикатов к каноническому представлению можно вычислять константные выражения и избавляться от логических отрицаний.

Еще один класс логических преобразований связан с приведением к каноническому виду логического выражения, задающего условие выборки запроса. Как правило, используются либо дизъюнктивная, либо *конъюнктивная нормальные формы*. Выбор канонической формы зависит от общей организации оптимизатора.

При приведении логического условия к каноническому представлению можно производить поиск общих предикатов (они могут существовать изначально, могут появиться после приведения предикатов к каноническому виду или в *процессе нормализации* логического условия) и

упрощать логическое выражение за счет, например, выявления конъюнкции взаимно противоречащих предикатов.

- **Преобразования запросов с изменением порядка реляционных операций**.В традиционных оптимизаторах распространены логические преобразования, связанные с изменением порядка выполнения *реляционных операций*.

Например, имеем следующий запрос:

R1 NATURAL JOIN R2
  WHERE R1.A OC a AND R2.B C b

Здесь a и b некоторые константы, которые ограничивают значение атрибутов отношений R1 и R2.

Если мы его рассмотрим в терминах *реляционной алгебры*, то это *естественное соединение* отношений R1 и R2, в которых заданы внутренние ограничения на кортежи каждого отношения.

Для уменьшения числа соединяемых кортежей резоннее сначала произвести операции выборки на каждом отношении и только после этого перейти в операции естественного соединения.

Поэтому данный запрос будет эквивалентен следующей последовательности операций *реляционной алгебры*:

R3 = R1[R1.A OC a]
R4 = R2[R2.B C b]
R5 = R3*[ ]*R4

Хотя немногие реляционные системы имеют языки запросов, основанные в чистом виде на реляционной алгебре, правила преобразований алгебраических выражений могут быть полезны и в других случаях. Довольно часто реляционная алгебра используется в качестве основы внутреннего представления запроса. Естественно, что после этого можно выполнять и алгебраические преобразования.

В частности, существуют подходы, связанные с преобразованием запросов на языке SQL к алгебраической форме. Особенно важно то, что реляционная алгебра более проста, чем язык SQL. Преобразование запроса к алгебраической форме упрощает дальнейшие действия оптимизатора по выборке оптимальных планов. Вообще говоря, развитый оптимизатор запросов системы, ориентированной на SQL, должен выявить все возможные планы выполнения любого запроса, но "пространство поиска" этих планов в общем случае очень велико; в каждом конкретном оптимизаторе используются свои эвристики для сокращения пространства поиска. Некоторые, возможно, наиболее оптимальные планы никогда не будут рассматриваться. Разумное преобразование запроса на SQL к алгебраическому представлению сокращает пространство поиска планов выполнения запроса с гарантией того, что оптимальные планы потеряны не будут.

- **Приведение запросов с вложенными подзапросами к запросам с соединениями**.Основным отличием языка SQL от языка *реляционной алгебры* является возможность использовать в логическом условии выборки предикаты, содержащие вложенные подзапросы. Глубина вложенности не ограничивается языком, то есть, вообще говоря, может быть произвольной. Предикаты с вложенными подзапросами при наличии общего синтаксиса могут обладать весьма различной семантикой. Единственным общим для всех возможных се-мантик вложенных подзапросов алгоритмом выполнения запроса является вычисление вложенного подзапроса всякий раз при вычислении значения предиката. Поэтому естественно стремиться к такому преобразованию запроса, содержащего предикаты со вложенными подзапросами, которое сделает семантику подзапроса более явной, предоставив тем самым в дальнейшем оптимизатору возможность выбрать способ выполнения запроса, наиболее точно соответствующий семантике подзапроса.

Каноническим представлением запроса на n отношениях называется *запрос*, содержащий n–1 *предикат* соединения и не содержащий предикатов с вложенными подзапросами. Фактически каноническая форма — это алгебраическое *представление* запроса.

Например, *запрос* с вложенным подзапросом:

(SELECT R1.A FROM R1

```
  WHERE R1.B IN
    (SELECT R2.B FROM R2
      WHERE R1.C = R2.D) )
```

эквивалентен

```
(SELECT R1.A
  FROM R1, R2
  WHERE R1.A = R2.B AND R1.C = R2.D)
```

Второй *запрос*:

```
(SELECT R1.A FROM R1
 WHERE R1.K = (SELECT AVG (R2.B)
           FROM R2 WHERE R1.C = R2.D)
```

или

```
(SELECT R1.A
  FROM R1, R3
  WHERE R1.C = R3.D AND R1.K = R3.L)
  R3 = SELECT R2.D, L AVG (R2.B)
       FROM R2 GROUP BY R2.D
```

При использовании подобного подхода в оптимизаторе запросов не обязательно производить формальные преобразования запросов. Оптимизатор должен в большей степени использовать семантику обрабатываемого запроса, а каким образом она будет распознаваться — это вопрос техники.

Заметим, что в кратко описанном нами подходе имеются некоторые тонкие семантические некорректности. Известны исправленные методы, но они слишком сложны технически, чтобы рассматривать их в данном пособии.

Методы семантической оптимизации запросов

Рассмотренные ранее методы никак не связаны с семантикой конкретной *БД*, они применимы к любой *БД*, вне зависимости от ее конкретного содержания. Семантические методы оптимизации основаны как раз на учете семантики конкретной *БД*. Таких методов в различных реализациях может быть множество, мы с вами коснемся лишь некоторых из них:

- **Преобразование запросов с учетом семантической информации**.Это прежде всего относится к запросам, которые выполняются над представлениями. Само представление представляет собой запрос. В БД представление хранится в виде скомпилированного плана выполнения запроса, то есть в нем в некоторой канонической форме представлены уже все предикаты и сам план выполнения запроса. При преобразовании внешнего запроса производится объединение внешнего запроса с внутренней формой запроса, составляющего основу представления, и строится обобщенная каноническая форма, объединяющая оба запроса. Для этой новой формы проводится анализ и преобразование предикатов. Поэтому при выполнении запроса над представлением будет выполнено не два, а только один запрос, оптимизированный по обобщенным параметрам запроса.
- **Использование ограничений целостности при анализе запросов**.Ограничения целостности связаны с условиями, которые накладываются на значения столбцов таблицы. При выполнении запросов над таблицами условия запросов объединяются специальным образом с условиями ограничений таблицы и полученные обобщенные предикаты уже анализируются. Допустим, что мы ищем в нашей библиотеке читателей с возрастом более 100 лет, но если у нас есть ограничение, заданное для таблицы READERS, которое ограничивает дату рождения наших читателей, так чтобы читатель имел дату рождения в пределах от 17 до 100 лет включительно. Поэтому оптимизатор запроса, сопоставив два эти предиката, может сразу определить, что результатом запроса будет пустое множество.

После оптимизации *запрос* имеет непроцедурный вид, то есть в нем не определен жесткий порядок выполнения элементарных операций над исходными объектами. На следующем этапе строятся все возможные планы выполнения запросов и для каждого из них производятся стои-

мостные оценки. Оценка планов выполнения запроса основана на анализе текущих объемов данных, хранящихся в отношениях *БД*, и на статистическом анализе хранимой информации. В большинстве *СУБД* ведется учет диапазона значений отдельного столбца с указанием процентного содержания для каждого диапазона. Поэтому при построении плана запроса *СУБД* может оценить объем промежуточных отношений и построить план таким образом, чтобы на наиболее ранних этапах выполнения запроса минимизировать количество строк, включаемых в промежуточные отношения.

Кроме ядра *СУБД* каждый поставщик обеспечивает специальные инструментальные средства, облегчающие *администрирование БД* и разработку новых проектов *БД* и пользовательских приложений для данного сервера. В последнее время практически все утилиты и инструментальные средства имеют развитый графический *интерфейс*.

Для разработки приложений пользователи могут применять не только инструментальные средства, поставляемые вместе с сервером *БД*, но и средства сторонних поставщиков. Так, в нашей стране получила большую популярность инструментальная среда Delphi, которая позволяет разрабатывать приложения для различных серверов *БД*. За рубежом более популярными являются инструментальные системы быстрой разработки приложений (RAF Rapid *Application Foundation*) продукты компании *Advanced Information System*, инструментальной среды *Power* Builder фирмы *Power Soft*, системы *SQL Windows* фирмы Gupta (Taxedo).

Защита информации в базах данных

В современных *СУБД* поддерживается один из двух наиболее общих подходов к вопросу обеспечения безопасности данных: избирательный подход и обязательный подход. В обоих подходах единицей данных или "объектом данных", для которых должна быть создана система безопасности, может быть как вся *база данных* целиком, так и любой *объект* внутри *базы данных*.

Эти два подхода отличаются следующими свойствами:

- В случае избирательного управления некоторый пользователь обладает различными правами (привилегиями или полномочиями) при работе с данными объектами. Разные пользователи могут обладать разными правами доступа к одному и тому же объекту. Избирательные права характеризуются значительной гибкостью.
- В случае неизбирательного управления, наоборот, каждому объекту данных присваивается некоторый классификационный уровень, а каждый пользователь обладает некоторым уровнем допуска. При таком подходе доступом к определенному объекту данных обладают только пользователи с соответствующим уровнем допуска.
- Для реализации избирательного принципа предусмотрены следующие методы. В базу данных вводится новый тип объектов БД — это пользователи. Каждому пользователю в БД присваивается уникальный идентификатор. Для дополнительной защиты каждый пользователь кроме уникального идентификатора снабжается уникальным паролем, причем если идентификаторы пользователей в системе доступны системному администратору, то пароли пользователей хранятся чаще всего в специальном кодированном виде и известны только самим пользователям.
- Пользователи могут быть объединены в специальные группы пользователей. Один пользователь может входить в несколько групп. В стандарте вводится понятие группы PUBLIC, для которой должен быть определен минимальный стандартный набор прав. По умолчанию предполагается, что каждый вновь создаваемый пользователь, если специально не указано иное, относится к группе PUBLIC.
- Привилегии или полномочия пользователей или групп — это набор действий (операций), которые они могут выполнять над объектами БД.
- В последних версиях ряда коммерческих СУБД появилось понятие "роли". Роль — это поименованный набор полномочий. Существует ряд стандартных ролей, которые определены в момент установки сервера баз данных. И имеется возможность создавать новые роли, группируя в них произвольные полномочия. Введение ролей позволяет упростить управление привилегиями пользователей, структурировать этот процесс. Кроме того, введение ролей не связано с

конкретными пользователями, поэтому роли могут быть определены и сконфигурированы до того, как определены пользователи системы.

- Пользователю может быть назначена одна или несколько ролей.
- Объектами БД, которые подлежат защите, являются все объекты, хранимые в БД: таблицы, представления, хранимые процедуры и триггеры. Для каждого типа объектов есть свои действия, поэтому для каждого типа объектов могут быть определены разные права доступа.

На самом элементарном уровне концепции обеспечения безопасности баз данных исключительно просты. Необходимо поддерживать два фундаментальных принципа: проверку полномочий и проверку подлинности (аутентификацию).

Проверка полномочий основана на том, что каждому пользователю или процессу информационной системы соответствует набор действий, которые он может выполнять *по* отношению к определенным объектам. Проверка подлинности означает достоверное подтверждение того, что *пользователь* или процесс, пытающийся выполнить санкционированное действие, действительно тот, за кого он себя выдает.

Система назначения полномочий имеет в некотором роде иерархический характер. Самыми высокими правами и полномочиями обладает *системный администратор* или *администратор* сервера *БД*. Традиционно только этот тип пользователей может создавать других пользователей и наделять их определенными полномочиями.

СУБД в своих системных каталогах хранит как описание самих пользователей, так и описание их привилегий *по* отношению ко всем объектам.

Далее схема предоставления полномочий строится *по* следующему принципу. Каждый *объект* в *БД* имеет владельца — пользователя, который создал данный *объект*. *Владелец объекта* обладает всеми правами-полномочиями на данный *объект*, в том числе он имеет право предоставлять другим пользователям полномочия *по* работе с данным объектом или забирать у пользователей ранее предоставленные полномочия.

В ряде *СУБД* вводится следующий уровень иерархии пользователей — это *администратор БД*. В этих *СУБД* один *сервер* может управлять множеством *СУБД* (например, MS *SQL Server*, Sybase).

В *СУБД Oracle* применяется однобазовая *архитектура*, поэтому там вводится понятие подсхемы — части общей схемы *БД* и вводится *пользователь*, имеющий *доступ* к подсхеме.

В стандарте *SQL* не определена *команда* создания пользователя, но практически во всех коммерческих *СУБД* создать пользователя можно не только в интерактивном режиме, но и программно с использованием специальных хранимых процедур. Однако для выполнения этой *операции пользователь* должен иметь право на *запуск* соответствующей системной процедуры.

В стандарте *SQL* определены два оператора: GRANT и *REVOKE* соответственно предоставления и *отмены привилегий*.

Оператор *предоставления привилегий* имеет следующий формат:

```
GRANT {<список действий> | ALL PRIVILEGES }
  ON <имя_объекта>
  TO {<имя_пользователя> | PUBLIC }
  [WITH GRANT OPTION ]
```

Здесь *список* действий определяет набор действий из общедопустимого перечня действий над объектом данного типа.

Параметр ALL PRIVILEGES указывает, что разрешены все действия из допустимых для объектов данного типа.

<имя_объекта> — задает имя конкретного объекта: таблицы, представления, хранимой процедуры, триггера.

<имя_пользователя> или PUBLIC определяет, кому предоставляются данные привилегии.

Параметр WITH GRANT OPTION является необязательным и определяет режим, при котором передаются не только *права* на указанные действия, но и право передавать эти *права* другим

пользователям. Передавать *права* в этом случае *пользователь* может только в рамках разрешенных ему действий.

Рассмотрим пример, пусть у нас существуют три пользователя с абсолютно уникальными именами user1, user2 и user3. Все они являются пользователями одной *БД*.

User1 создал *объект* Tab1, он является владельцем этого объекта и может передать *права* на работу с эти объектом другим пользователям. Допустим, что *пользователь* user2 является оператором, который должен вводить данные в Tab1 (например, таблицу новых заказов), а *пользователь* user 3 является большим начальником (например, менеджером отдела), который должен регулярно просматривать введенные данные.

Для объекта типа *таблица* полным допустимым перечнем действий является набор из четырех операций: SELECT, INSERT, DELETE, UPDATE. При этом операция обновление может быть ограничена несколькими столбцами.

Общий формат оператора назначения привилегий для объекта типа *таблица* будет иметь следующий *синтаксис*:

```
GRANT {[SELECT][,INSERT][,DELETE][,UPDATE (<список столбцов>)]}
  ON <имя_таблицы>
  TO {<имя_пользователя> | PUBLIC } [WITH GRANT OPTION ]
```

Тогда резонно будет выполнить следующие назначения:

```
GRANT INSERT
  ON Tab1
  TO user2
GRANT SELECT
  ON Tab1
  TO user3
```

Эти назначения означают, что *пользователь* user2 имеет право только вводить новые строки в *отношение* Tab1, а *пользователь* user3 имеет право просматривать все строки в таблице Tab1.

При назначении прав доступа на операцию модификации можно уточнить, *значение* каких столбцов может изменять *пользователь*. Допустим, что *менеджер* отдела имеет право изменять цену на предоставляемые услуги. Предположим, что цена задается в столбце COST таблицы Tab1. Тогда операция назначения привилегий пользователю user3 может измениться и выглядеть следующим образом:

```
GRANT SELECT, UPDATE (COST)
   ON Tab1
   TO user3
```

Если наш *пользователь* user1 предполагает, что *пользователь* user4 может его замещать в случае его отсутствия, то он может предоставить этому пользователю все *права по* работе с созданной таблицей Tab1.

```
GRANT ALL PRIVILEGES
   ON Tab1
   TO user4
WITH GRANT OPTION
```

В этом случае *пользователь* user4 может сам назначать привилегии *по* работе с таблицей Tab1 в отсутствие владельца объекта пользователя user1. Поэтому в случае появления нового оператора пользователя user5 он может назначить ему *права* на ввод новых строк в таблицу командой

```
GRANT INSERT
   ON Tab1
   TO user5
```

Если при передаче полномочий набор операций над объектом ограничен, то *пользователь*, которому переданы эти полномочия, может передать другому пользователю только те полномо-

чия, которые есть у него, или часть этих полномочий. Поэтому если пользователю user4 были делегированы следующие полномочия:

```
GRANT SELECT, UPDATE, DELETE
  ON Tab1
  TO user4
WITH GRANT OPTION,
```

то *пользователь* user4 не сможет передать полномочия на ввод данных пользователю user5, потому что эта операция не входит в *список* разрешенных для него самого.

Кроме непосредственного назначения прав *по* работе с таблицами эффективным методом защиты данных может быть создание представлений, которые будут содержать только необходимые столбцы для работы конкретного пользователя и *предоставление прав* на работу с данным представлением пользователю.

Так как представления могут соответствовать итоговым запросам, то для этих представлений недопустимы *операции* изменения, и, следовательно, для таких представлений набор допустимых действий ограничивается операцией SELECT. Если же представления соответствуют выборке из базовой таблицы, то для такого представления допустимыми будут все 4 *операции*: SELECT, INSERT, UPDATE и DELETE.

Для отмены ранее назначенных привилегий в стандарте *SQL* определен оператор *REVOKE*. Оператор *отмены привилегий* имеет следующий *синтаксис*:

```
REVOKE {<список операций> | ALL PRIVILEGES}
  ON <имя_объекта>
  FROM {<список пользователей> | PUBLIC }
     {CASCADE | RESTRICT }
```

Параметры CASCADE или RESTRICT определяют, каким образом должна производиться *отмена привилегий*. *Параметр* CASCADE отменяет привилегии не только пользователя, который непосредственно упоминался в операторе GRANT при предоставлении ему привилегий, но и всем пользователям, которым этот *пользователь* присвоил привилегии, воспользовавшись параметром WITH GRANT OPTION.

Например, при использовании *операции*:

```
REVOKE ALL PRIVILEGES
  ON Tab1
  FROM user4 CASCADE
```

будут отменены привилегии и пользователя user5, которому *пользователь* user4 успел присвоить привилегии.

Параметр RESTRICT ограничивает отмену привилегий только пользователю, непосредственно упомянутому в операторе *REVOKE*. Но при наличии делегированных привилегий этот оператор не будет выполнен. Так, например, операция:

```
REVOKE ALL PRIVILEGES
  ON Tab1 TO user4 RESTRICT
```

не будет выполнена, потому что *пользователь* user4 передал часть своих полномочий пользователю user5.

Посредством оператора *REVOKE* можно отобрать все или только некоторые из ранее присвоенных привилегий *по* работе с конкретным объектом. При этом из описания синтаксиса оператора *отмены привилегий* видно, что можно отобрать привилегии одним оператором сразу у нескольких пользователей или у целой группы PUBLIC.

Поэтому корректным будет следующее использование оператора *REVOKE*:

```
REVOKE INSERT    ON Tab1    TO user2,user4 CASCADE
```

При работе с другими объектами изменяется *список операций*, которые используются в операторах GRANT и *REVOKE*.

По умолчанию действие, соответствующее запуску (исполнению) хранимой процедуры, назначается всем членам группы PUBLIC.

Если вы хотите изменить это условие, то после *создания хранимой процедуры* необходимо записать оператор *REVOKE*.

```
REVOKE EXECUTE
  ON COUNT_EX
  TO PUBLIC CASCADE
```

И теперь мы можем назначить новые *права* пользователю user4.

```
GRANT EXECUTE
  ON COUNT_EX
  TO user4
```

Системный администратор может разрешить некоторому пользователю создавать и изменять таблицы в некоторой *БД*. Тогда он может записать оператор предоставления прав следующим образом:

```
GRANT CREATE TABLE,
      ALTER TABLE,
      DROP TABLE
  ON DB_LIB
  TO user1
```

В этом случае *пользователь* user1 может создавать, изменять или удалять таблицы в *БД* DB_LIB, однако он не может разрешить создавать или изменять таблицы в этой *БД* другим пользователям, потому что ему дано разрешение без *права* делегирования своих возможностей.

В некоторых *СУБД пользователь* может получить *права* создавать *БД*. Например, в MS *SQL Server системный администратор* может предоставить пользователю main_user право на создание своей *БД* на данном сервере. Это может быть сделано следующей командой:

```
GRANT CREATE DATABASE
  ON SERVER_0
  TO main_user
```

По принципу иерархии *пользователь* main_user, создав свою *БД*, теперь может предоставить *права* на создание или изменение любых объектов в этой *БД* другим пользователям.

В *СУБД*, которые поддерживают однобазовую архитектуру, такие разрешения недопустимы. Например, в *СУБД Oracle* на сервере создается только одна *БД*, но пользователи могут работать на уровне подсхемы (части таблиц *БД* и связанных с ними объектов). Поэтому там вводится понятие системных привилегий. Их очень много, 80 различных привилегий.

Для того чтобы разрешить пользователю создавать объекты внутри этой *БД*, используется понятие системной привилегии, которая может быть назначена одному или нескольким пользователям. Они выдаются только на действия и конкретный *тип объекта*. Поэтому если вы, как *системный администратор*, предоставили пользователю право создания таблиц (CREATE TABLE), то для того чтобы он мог создать *триггер* для таблицы, ему необходимо предоставить еще одну *системную привилегию* CREATE TRIGGER. Система защиты в *Oracle* считается одной из самых мощных, но это имеет и обратную сторону — она весьма сложная. Поэтому задача администрирования в *Oracle* требует хорошего знания как семантики принципов поддержки прав доступа, так и физической реализации этих возможностей

Реализация системы защиты в MS SQL Server

SQL server 6.5 поддерживает 3 режима проверки при определении прав пользователя:

1. Стандартный (standard).
2. Интегрированный (integrated security).
3. Смешанный (mixed).

Стандартный режим защиты предполагает, что каждый *пользователь* должен иметь учетную *запись* как *пользователь* домена NT *Server*. Учетная *запись* пользователя домена включает *имя пользователя* и его индивидуальный *пароль*. Пользователи доменов могут быть объединены в группы. Как *пользователь* домена *пользователь* получает *доступ* к определенным ресурсам домена. В качестве одного из ресурсов домена и рассматривается *SQL Server*. Но для доступа

к *SQL Server пользователь* должен иметь учетную *запись* пользователя MS *SQL Server*. Эта учетная *запись* также должна включать уникальное *имя пользователя* сервера и его *пароль*. При подключении к операционной среде *пользователь* задает свое имя и *пароль* пользователя домена. При подключении к серверу баз данных *пользователь* задает свое уникальное *имя пользователя SQL Server* и свой *пароль*.

Интегрированный режим предполагает, что для пользователя задается только одна учетная *запись* в операционной системе, как пользователя домена, а *SQL Server* идентифицирует пользователя *по* его данным в этой учетной записи. В этом случае *пользователь* задает только одно свое имя и один *пароль*.

В случае смешанного режима часть пользователей может быть подключена к серверу с использованием стандартного режима, а часть с использованием интегрированного режима.

В MS *SQL Server* 7.0 оставлены только 2 режима: интегрированный, называемый *Windows* NT *Authentication Mode* (*Windows* NT *Authentication*), и смешанный, Mixed *Mode* (*Windows* NT *Authentication and SQL Server Authentication*). *Алгоритм* проверки аутентификации пользователя в MS *SQL Server* 7.0 приведен на рис. 13.1.

При попытке подключения к серверу *БД* сначала проверяется, какой метод аутентификации определен для данного пользователя. Если определен *Windows* NT *Authentication Mode*, то далее проверяется, имеет ли данный *пользователь* домена *доступ* к ресурсу *SQL Server*, если он имеет *доступ*, то выполняется попытка подключения с использованием имени пользователя и пароля, определенных для пользователя домена; если данный *пользователь* имеет *права* подключения к *SQL Server*, то подключение выполняется успешно, в противном случае *пользователь* получает сообщение о том, что данному пользователю не разрешено подключение к *SQL Server*. При использовании смешанного режима аутентификации средствами *SQL Server* проводится последовательная проверка имени пользователя (*login*) и его пароля (*password*); если эти параметры заданы корректно, то подключение завершается успешно, в противном случае *пользователь* также получает сообщение о невозможности подключиться к *SQL Server*.

Для *СУБД Oracle* всегда используется в *дополнение* к имени пользователя и пароля в операционной среде его имя и *пароль* для работы с сервером *БД*.

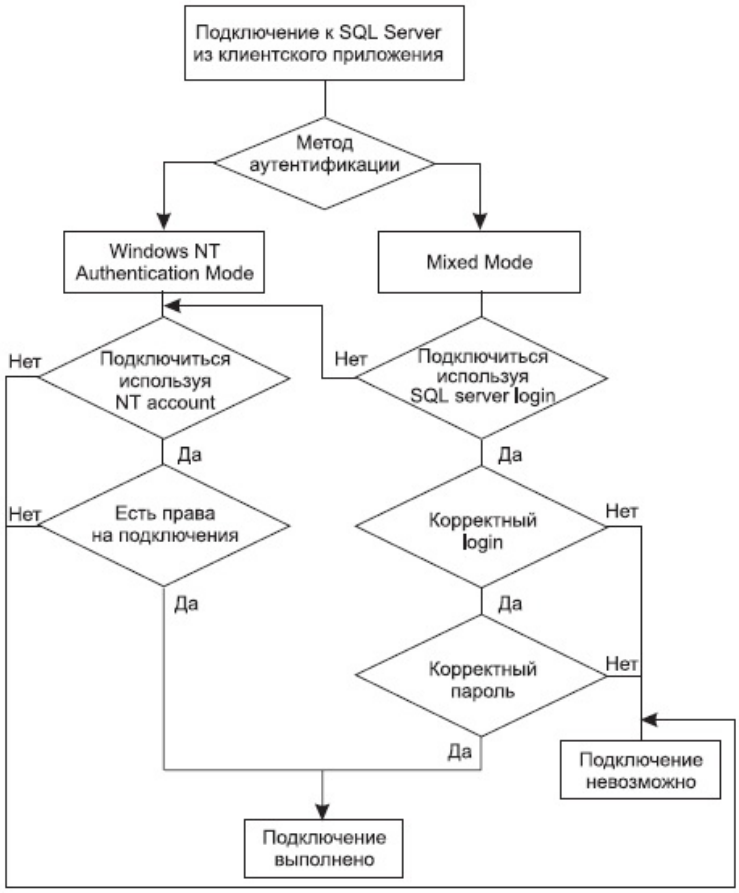


Рис. 13.1. Алгоритм проверки аутентификации пользователя в MS SQL Server 7.0

Проверка полномочий

Второй задачей при работе с *БД*, как указывалось ранее, является проверка полномочий пользователей. Полномочия пользователей хранятся в специальных системных таблицах, и их проверка осуществляется ядром *СУБД* при выполнении каждой *операции*. Логически для каждого пользователя и каждого объекта в *БД* как бы строится некоторая условная *матрица*, где *по* одному измерению расположены объекты, а *по* другому — пользователи. На пересечении каждого столбца и каждой строки расположен перечень разрешенных операций для данного пользователя над данным объектом. С первого взгляда кажется, что эта модель проверки достаточно устойчивая. Но сложность возникает тогда, когда мы используем косвенное обращение к объектам. Например, пользователю **user_N** не разрешен *доступ* к таблице Tab1, но этому пользователю разрешен *запуск* хранимой процедуры SP_N, которая делает выборку из этого объекта. *По* умолчанию все хранимые процедуры запускаются под именем их владельца.

Такие проблемы должны решаться организационными методами. При разрешении доступа некоторых пользователей необходимо помнить о возможности косвенного доступа.

В любом случае проблема защиты никогда не была чисто технической задачей, это комплекс организационно-технических мероприятий, которые должны обеспечить максимальную *конфиденциальность* информации, хранимой в *БД*.

Кроме того, при работе в сети существует еще проблема проверки подлинности полномочий.

Эта проблема состоит в следующем. Допустим, процессу 1 даны полномочия *по* работе с *БД*, а процессу 2 такие полномочия не даны. Тогда напрямую процесс 2 не может обратиться к *БД*, но он может обратиться к процессу 1 и через него получить *доступ* к информации из *БД*.

Поэтому в безопасной среде должна присутствовать модель проверки подлинности, которая обеспечивает подтверждение заявленных пользователями или процессами идентификаторов. Проверка полномочий приобрела еще большее *значение* в условиях массового распространения рас-

пределенных вычислений. При существующем высоком уровне связности вычислительных систем необходимо контролировать все обращения к системе.

Проблемы проверки подлинности обычно относят к сфере безопасности коммуникаций и сетей, поэтому мы не будем их здесь более обсуждать, за исключением следующего замечания. В целостной системе компьютерной безопасности, где четкое выполнение программы защиты информации обеспечивается за счет взаимодействия соответствующих средств в операционных системах, сетях, базах данных, проверка подлинности имеет прямое *отношение* к безопасности баз данных.

Модель безопасности, основанная на базовых механизмах проверки полномочий и проверки подлинности, не решает таких проблем, как украденные пользовательские идентификаторы и пароли или злонамеренные действия некоторых пользователей, обладающих полномочиями, — например, когда программист, работающий над учетной системой, имеющей полный *доступ* к учетной базе данных, встраивает в *код программы* "Троянского коня" с целью хищения или намеренного изменения информации, хранимой в *БД*. Такие вопросы выходят за рамки нашего обсуждения средств защиты баз данных, но следует тем не менее представлять себе, что *программа* обеспечения информационной безопасности должна охватывать не только технические области (такие как защита сетей, баз данных и операционных систем), но и проблемы физической защиты, надежности персонала (скрытые проверки), *аудит*, различные процедуры поддержки безопасности, выполняемые вручную или частично автоматизированные.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К ПРАКТИЧЕСКИМ ЗАНЯТИЯМ

Практическое занятие 3. Нормализация таблиц

Цель. Изучить вопросы, связанные с нормализацией данных, созданием таблиц в БД и заданием схемы данных.

Описание базы данных "Склад"

База данных предназначена для учета товаров, которые поступают по заказам на оптовый склад. Сотрудник оформляет закупку товаров нескольких наименований у одного поставщика.

Общая закупка включает в себя несколько сделок, отображаемых набором сведений о закупке каждого товара. Каждый товар относится к тому или иному типу товаров. Закупленные товары должны прибыть на склад при помощи одного из возможных способов доставки.

При создании базы выделяем следующие сущности:

Товары - содержит сведения о товарах;

Типы - справочник групп (типов) товаров;

Сделки - содержит сведения о заказах (проведенных операциях по закупке) каждого из товаров;

Закупки - содержит сведения о заказах нескольких товаров от одного поставщика;

Сотрудники - содержит сведения о сотрудниках, оформивших заказ;

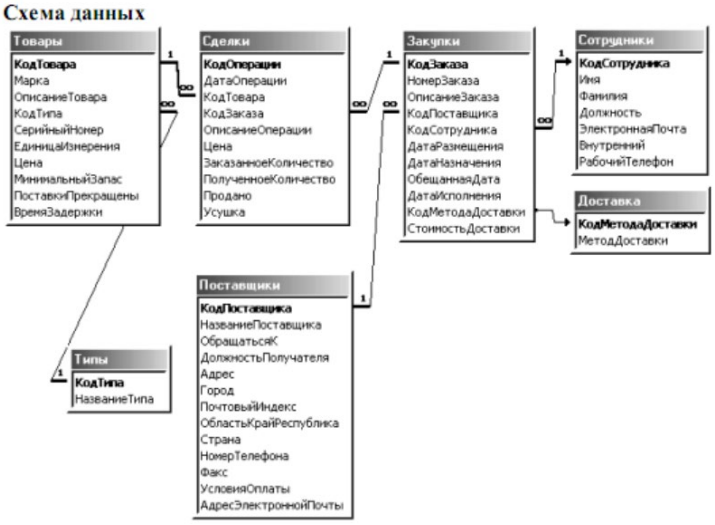
Поставщики - содержит сведения о каждой организации - поставщике товаров

Доставка - справочник видов доставки.

Порядок выполнения.

1. Создать таблицы БД (список таблиц приведен ниже). Задать требуемые свойства полей.
2. Создать схему данных.
3. Сделать подстановки в связанных полях
4. Ввести тестовые данные (по 10-20 строк в таблицы "Товары" и др., по 5 строк в таблицы "Типы" и др.

Схема данных



Список таблиц и свойств полей таблиц

Таблица "Товары"

Имя поля	Тип данных	Размер/Формат	Подпись	Индексир. поле	Обязат. поле	Другие
КодТовара	Счетчик	Дл целое	Код товара	Да (Совп. не доп.)		
Марка	Текстовый	50	Марка	Да (Совп. доп.)	Нет	
ОписаниеТовара	Текстовый	255	Описание товара	Нет	Нет	
КодТипа	Числовой	Дл целое	Код типа	Да (Совп. доп.)	Нет	
СерийныйНомер	Текстовый	50	Серийный номер	Да (Совп. доп.)	Нет	
Цена	Денежный	Денежный	Цена	Нет	Нет	
ЕдиницаИзмерения	Текстовый	50	Единица измерения	Нет	Нет	
МинимальныйЗапас	Числовой	Дл целое	Минимальный запас	Нет	Нет	
ПоставкиПрекращены	Логический	Да/Нет	Поставки прекращены	Нет	Нет	
ВремяЗадержки	Текстовый	30	Срок	Нет	Нет	

Таблица "Типы"

Имя поля	Тип данных	Размер/Формат	Подпись	Индексир. поле	Обязат. поле	Другие
КодТипа	Счетчик	Дл целое	Код типа	Да (Совп. не доп.)		
НазваниеТипа	Текстовый	50	Категория	Да (Совп. доп.)	Нет	

Таблица "Сделки"

Имя поля	Тип данных	Размер/Формат	Подпись	Индексир. поле	Обязат. поле	Другие
КодОперации	Счетчик	Дл целое	Код операции	Да (Совп. не доп.)		
ДатаОперации	Дата/время	Краткий формат даты	Дата операции	Да (Совп. доп.)	Нет	Маска ввода 99.99.00;0
КодТовара	Числовой	Дл целое	Код товара	Да (Совп. доп.)	Нет	
КодЗаказа	Числовой	Дл целое	Код заказа	Да (Совп. доп.)	Нет	
ОписаниеОперации	Текстовый	255	Описание операции	Нет	Нет	
Цена	Денежный	Денежный	Цена	Нет	Нет	
ЗаказанноеКоличество	Числовой	Длинное целое	Заказанное количество	Нет	Нет	
ПолученноеКоличество	Числовой	Длинное целое	Полученное количество	Нет	Нет	
Продано	Числовой	Длинное целое	Продано	Нет	Нет	
Усушка	Числовой	Длинное целое	Усушка	Нет	Нет	

Таблица "Закупки"

Имя поля	Тип данных	Размер/ Формат	Подпись	Индексир. поле	Обязат. поле	Другие
КодЗаказа	Счетчик	Дл целое	Код заказа	Да (Совп. не доп.)		
НомерЗаказа	Текстовый	30	НомерЗаказа	Нет	Нет	
ОписаниеЗаказа	Текстовый	255	Описание Заказа	Нет	Нет	
КодПоставщика	Числовой	Дл целое	Код поставщика	Да (Совп. доп.)	Нет	
КодСотрудника	Числовой	Дл целое	Код сотрудника	Да (Совп. доп.)	Нет	
ДатаРазмещения	Дата/время	Краткий формат даты	Дата размещения	Да (Совп. доп.)	Нет	Маска ввода 99.99.00;0
ДатаНазначения	Дата/время	Краткий формат даты	Дата назначения	Нет	Нет	Маска ввода 99.99.00;0
ОбещаннаяДата	Дата/время	Краткий формат даты	Обещанная дата	Нет	Нет	Маска ввода 99.99.00;0
ДатаИсполнения	Дата/время	Краткий формат даты	Дата исполнения	Нет	Нет	Маска ввода 99.99.00;0
КодМетодаДоставки	Числовой	Длинное целое	Код доставки	Да (Совп. доп.)	Нет	
СтоимостьДоставки	Денежный	Денежный	Цена доставки	Нет	Нет	

Таблица "Поставщики"

Имя поля	Тип данных	Размер/ Формат	Подпись	Индексир. поле	Обязат. поле	Другие
КодПоставщика	Счетчик	Дл целое	Код поставщика	Да (Совп. не доп.)		
НазваниеПоставщика	Текстовый	50	Название	Да (Совп. доп.)	Нет	
ОбращатьсяК	Текстовый	50	Обращаться к	Да (Совп. доп.)	Нет	
ДолжностьПолучателя	Текстовый	50	Должность	Нет	Нет	
Адрес	Текстовый	255		Нет	Нет	
Город	Текстовый	50		Нет	Нет	
ПочтовыйИндекс	Текстовый	20	Индекс	Да (Совп. доп.)	Нет	
ОбластьКрайРеспублика	Текстовый	20	Регион	Нет	Нет	
Страна	Текстовый	50		Нет	Нет	
НомерТелефона	Текстовый	30	Телефон	Нет	Нет	
Факс	Текстовый	30		Нет	Нет	
УсловияОплаты	Текстовый	255	Условия оплаты	Нет	Нет	
АдресЭлектроннойПочты	Текстовый	50	Электронная почта	Да (Совп. доп.)	Нет	

Таблица "Доставка"

Имя поля	Тип данных	Размер/ Формат	Подпись	Индексир. поле	Обязат. поле	Другие
КодМетодаДоставки	Счетчик	Дл целое	Код доставки	Да (Совп. не доп.)		
МетодДоставки	Текстовый	20	Метод доставки	Нет	Нет	

Таблица "Сотрудники"

Имя поля	Тип данных	Размер/ Формат	Подпись	Индексир. поле	Обязат. поле	Другие
КодСотрудника	Счетчик	Дл целое	Код сотрудника	Да (Совп. не доп.)		
Имя	Текстовый	50		Нет	Нет	
Фамилия	Текстовый	50		Да (Совп. доп.)	Нет	
Должность	Текстовый	50		Нет	Нет	
ЭлектроннаяПочта	Текстовый	50	Имя электронной почты	Да (Совп. доп.)	Нет	
Внутренний	Текстовый	30		Нет	Нет	
РабочийТелефон	Текстовый	30		Нет	Нет	

Содержание отчета.

1.Описание сущностей и их атрибутов.

2.Описание таблиц и определение свойств полей таблиц.

3.Модель данных «сущность – связь».

4.Описание типов связей и связываемых полей.

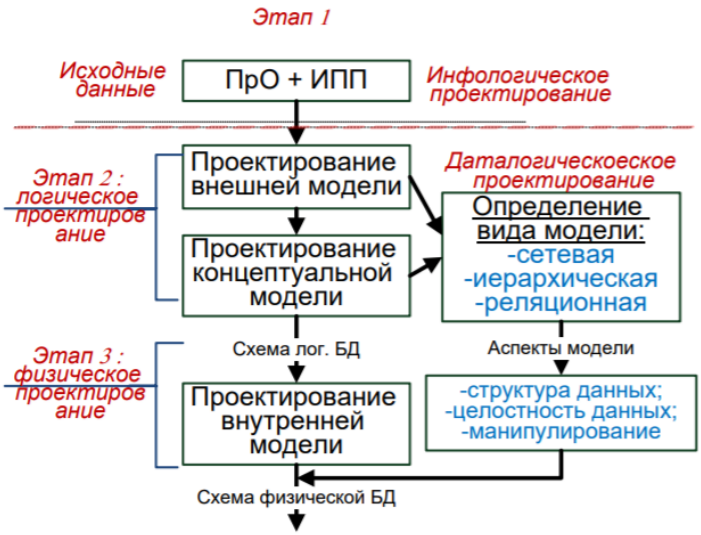
Контрольные вопросы

1. Основные понятия информационно-логических моделей: сущность, атрибут (реквизит), информационный объект, информационный элемент.
2. Основные понятия баз данных: поле, запись, таблица.
3. Что такое нормализация данных?
4. Ключи в БД (простые, составные, первичные, вторичные, внешние) и их назначение.
5. Индексация таблиц и ее назначение.
6. Какими свойствами обладают таблицы, которые находятся в1-й, во 2-й и в 3-й нормальной форме?
7. Какими свойствами обладают связи "один-к-одному", "один-ко-многим", "много-ко-многим"?
8. Каковы свойства внутреннего объединения таблиц, левого и правого внешнего объединений?
9. Реализация отношения "много-ко-многим" в базе данных.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К ЛАБОРАТОРНЫЙ ЗАНЯТИЯМ

Лабораторная работа 2. «Создание базы данных в среде разработки»

Цель: создать точную и защищенную БД, на основе которой можно гарантировать эффективное построение прикладных программ.



Процесс проектирования БД

Соотношение ПрО и ИПП

ИПП – информационные потребности пользователя; ПрО – предметная область Процесс проектирования БД состоит из 2-х основных этапов:

1) проектирование логической БД;
2) проектирование физической БД.

При проектировании логической БД производится анализ ПрО и ИПП. Пользовательские требования выражаются рядом внешних моделей – представлений. Проектирование внешней модели заключается в формализации этих представлений.

Концептуальная модель данных (КМД) соответствует общему представлению о БД, то есть она включает представление о структуре данных, их целостности и манипулировании данными.

Преобразование внешней модели (ВМД) в КМД определяется выбором СУБД. Как внешняя, так и концептуальная модель может быть 3-х видов:

1) сетевая;

2) иерархическая;

3) реляционная.

Физическое проектирование связано с фактической реализацией БД. Оно определяет рациональный выбор структуры хранения данных и методов доступа к ним. Результат физического проектирования - внутренняя модель данных.

Реляционная БД – это набор экземпляров конечных отношений. Схема реляционной БД может быть представлена в виде совокупности следующих отношений:

$$\begin{cases} R1(A11, A12, ..., A1n) \\ ... \\ Rm(Am1, Am2, ..., Amk), \end{cases}$$

где m, n, k – любые целые числа, определяющие размерность схемы БД.

Главная задача проектирования реляционной БД: понизить избыточность и повысить надежность БД. При этом основное место отводится целостности данных, то есть ограничениям на зависимости между атрибутами отношений.

Нормализация необходима для решения проблемы рационального выбора вариантов схем отношений из возможного множества альтернативных вариантов.

Схема отношений должна удовлетворять следующим требованиям:

1) выбранные для отношения первичные ключи должны быть минимальными;

2) выбранный состав отношений БД должен быть минимален то есть отличаться минимальной избыточностью атрибутов;

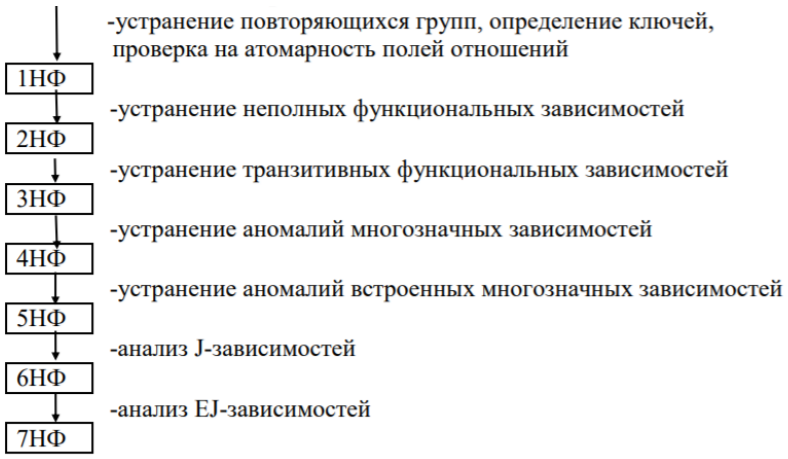
3) при выполнении операций модификации, удаления и включения не должно быть трудностей;

4) перестройка набора отношений, приведение новых типов данных должна быть минимальной;

5) разброс времени ответа на различные запросы к БД должен быть небольшим.

Методы нормализации базируются на понятиях функциональных зависимостей, многозначных зависимостей и зависимостей соединения.

Выделяют 7 этапов нормализации, соответственно 7 НФ:



Определение 1НФ: чтобы отношение соответствовало 1НФ необходимо привести это отношение в соответствие трем условиям:

1) определить первичные ключи;

2) устранить повторяющиеся группы;

3) привести поля отношения к атомарности.

Основой этого этапа нормализации является определение ключей. На рисунке 8 приведена схема определения ключевых атрибутов в древовидной форме, которая называется правилом распространения ключей.

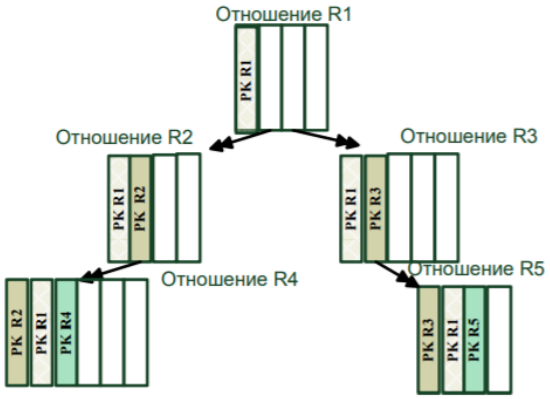


Схема распространения ключей в древовидной структуре:
PK Ri – первичный ключ отношения i

Таблица находится во 2НФ, если она удовлетворяет определению первой и все ее поля не входящие в первичный ключ связаны полной функциональной зависимостью с первичным ключом. Рейс (№ самолета, дата вылета, время вылета, пункт назначения, тип самолета, грузоподъемность, число членов экипажа, обслуживаемая фирма, ФИО командира экипажа, звание командира экипажа)

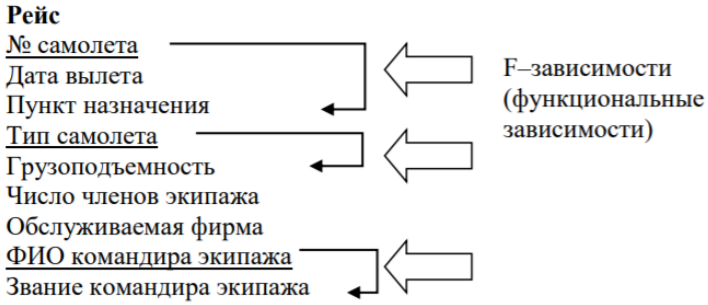


Таблица находится в 3НФ, если она удовлетворяет 2НФ и ни одно из ее не ключевых полей не зависит функционально от другого не ключевого поля, т.е. существуют зависимости между неключевыми атрибутами:

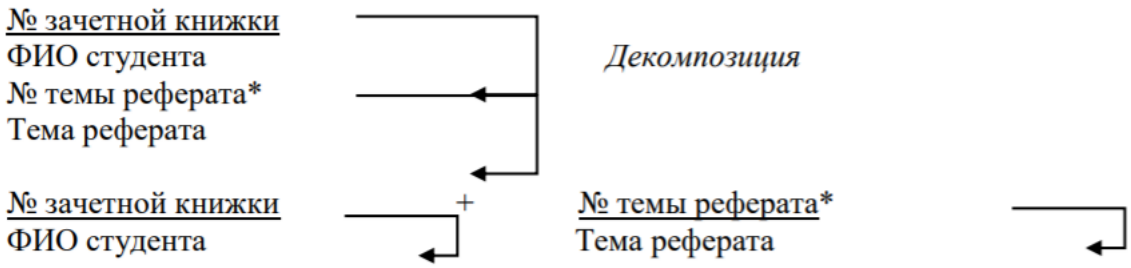
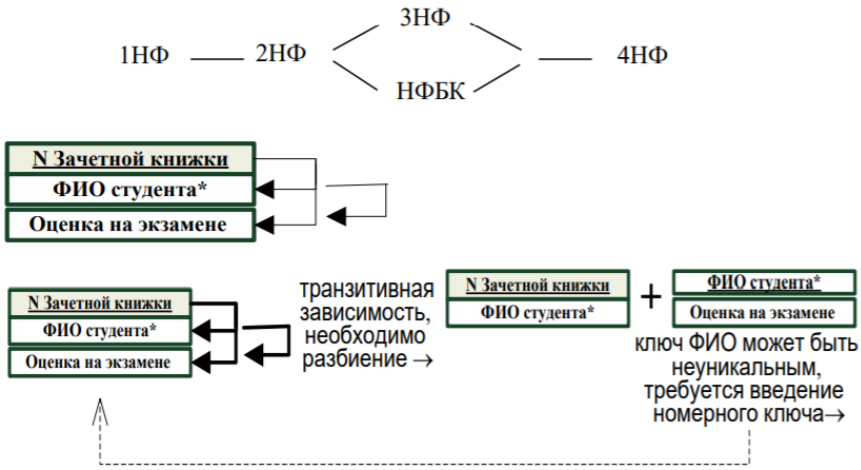


Таблица находится в НФ Бойса-Кодда (НФБК) тогда и только тогда, когда любая функциональная зависимость между полями сводится к полной функциональной зависимости от возможного ключа. В соответствии с этим таблицы Блюда и Продукты, имеющие по паре возможных ключей (№ блюда, блюдо, № продукта, продукт) находятся в НФБК или в 3НФ.



Доказательство отсутствия декомпозиции в НФБК

В следующих НФ (4 и 5) учитываются не только функциональные многозначные связи между полями таблицы. Полная декомпозиция таблицы – это совокупность любого числа ее проекций, соединение которых полностью совпадает с содержимым таблицы. Обычно на практике достаточно приведения БД к 3НФ или НФБК.

Контрольные вопросы

1. Назовите этапы проектирования БД.
2. Сформулировать цель проектирования БД.
3. Что такое предметная область и информационные потребности пользователя?
4. В чем отличия логической и физической моделей БД?
5. Что такое концептуальная и внешняя модели данных?
6. С чем связано физическое проектирование БД?
7. Записать формальное представление БД в виде совокупности отношений.
8. Дать определение реляционной БД.
9. Сформулировать главную задачу проектирования БД.
10. Сформулировать требования к отношениям БД.
11. Что такое нормализация БД?
12. Сколько существует основных нормальных форм?
13. Сформулировать основные этапы приведения к НФ.
14. Какова цель «распространения» ключей при приведении к 1НФ?
15. Почему при приведении к НФБК декомпозиция отношений не требуется?

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К САМОСТОЯТЕЛЬНОЙ РАБОТЕ СТУДЕНТОВ

Самостоятельная работа развивает мотивационную составляющую образовательной деятельности студентов, акцентируясь на самообразовании и самовоспитании, осуществляемых в интересах повышения профессиональной компетенции. Она развивает систему общеучебных умений, способствующих ее рациональной организации:

- планировать собственную образовательную деятельность,
- четко ставить систему задач,
- вычленять среди них главные направления работы,
- избирать способы наиболее быстрого и экономного решения поставленных задач,
- осуществлять оперативный контроль за выполнением задания,

• оперативно вносить коррективы в самостоятельную работу, анализировать промежуточные и общие итоги работы,

• сравнивать полученные результаты с намеченными в начале работы целями, выявлять причины отклонений и определять пути их коррекции в дальнейшей работе.

Самостоятельная работа включает два вида – аудиторную и внеаудиторную. В первом случае она выполняется на учебных занятиях под руководством преподавателя и по его заданию. Студенты обеспечиваются необходимым учебным материалом и дидактическими материалами.

Внеаудиторная самостоятельная работа выполняется по заданию преподавателя, но без его непосредственного участия. Видами заданий для внеаудиторной работы являются: изучение текста учебной литературы, конспектирование текста, работа с конспектом лекции, ответы на контрольные вопросы при выполнении индивидуального задания, тестирование, решение задач, продумывание алгоритма будущей программы, работа с компьютером, а именно, кодирование и отладка программы, подготовка отчета по лабораторным заданиям, подготовка к сдаче экзамена.

Основные формы самостоятельной учебной работы:

работа над конспектом лекции: лекции – основной источник информации по многим предметам, позволяющий не только изучить материал, но и получить представление о наличии других источников, сопоставить разные взгляды на основные проблемы данного курса. Лекции предоставляют возможность «интерактивного» обучения, когда есть возможность задавать преподавателю вопросы и получать на них ответы. Поэтому имеет смысл находить время для хотя бы беглого просмотра информации по материалу лекций (учебники, справочники и пр.) и непонятные, а также дискуссионные моменты обсуждать с преподавателем, другими студентами;

подготовка к практическому занятию: производится, как правило, с использованием методических пособий, состоит в теоретической подготовке (особенно для семинаров) и выполнении практических заданий (решение задач, ответы на вопросы и т.д.).

Существует ряд форм практических занятий:

- лабораторные занятия с оборудованием (иногда с постановкой опытов);
- практикум по освоению тех или иных навыков, методик;
- семинар (с разбором теоретических вопросов в рамках какой-либо темы);
- коллоквиум (семинар но итогам изучения нескольких родственных тем); □

подготовка к семинарскому занятию производится по правилам выполнения задания практической работы, обычно по определенному вопросу и более или менее узкому кругу литературы (часто всего два-три учебных пособия);

доработка конспекта лекции с применением учебника, методической литературы, дополнительной литературы: этот вид самостоятельной работы студентов особенно важен в том случае, когда изучаемый предмет содержит много неоднозначно трактуемых вопросов, проблем. Тогда преподаватель заведомо не может успеть изложить различные точки зрения, и студент должен ознакомиться с ними по имеющейся литературе.

Кроме того, рабочая программа предметов предполагает рассмотрение некоторых относительно несложных тем только во время самостоятельных занятий, без чтения лектором; подбор, изучение, анализ и конспектирование рекомендованной литературы; самостоятельное изучение отдельных тем, параграфов; консультации по сложным, непонятным вопросам лекций, семинаров, зачетов;

подготовка к зачету: данная форма СРС может быть весьма разнообразной по своей сути, так как сам зачет бывает различным. Он проводится обычно по итогам семестра перед сессией в письменной или устной форме, причем преподаватель может включать в него вопросы как практических занятий, так и лекционных (что особенно уместно, когда по данному предмету не сдается экзамен).

Главное отличие зачета от экзамена – почти всегда не пяти-, а двухбалльная система оценки (сдал – не сдал), что делает его получение несколько более простым делом. С другой стороны, порой процедура его сдачи достаточно сложна, а иногда применяется и пятибалльная оценка (так

называемый дифференцированный зачет).

Таким образом, для сдачи зачета необходимо, прежде всего, выполнить все требования преподавателя, что предполагает знание этих требований. Нужно как можно раньше выяснить, какие вопросы предстоит готовить и каковы правила самой процедуры (учитывается ли посещаемость, надо ли пропущенные занятия отрабатывать, а если надо, то каким образом и т.д.). Практика показывает, что хорошее посещение занятий является почти полной гарантией получения зачета, так как тогда можно быть в курсе всех требований преподавателя. И, напротив, большое количество пропусков может осложнить жизнь даже сильному студенту.

Кроме того, необходимо учитывать, что проблемы могут появиться при распространенном подходе студента к практическим занятиям, когда многие работают первые месяцы вполсилы, накапливая задолженности по выполнению рефератов, практических заданий, конспектов и пр., а перед сессией пытаются все это сделать за одну неделю. Старайтесь распределять силы равномерно по всей дистанции семестра, и тогда зачетная неделя перед сессией будет не самой напряженной, а самой разгрузочной.

ЛИТЕРАТУРА

Лазицкас Е.А. Базы данных и системы управления базами данных [Электронный ресурс] : учебное пособие / Е.А. Лазицкас, И.Н. Загумённикова, П.Г. Гилевский. — Электрон. текстовые данные. — Минск: Республиканский институт профессионального образования (РИПО), 2016. — 268 c. — 978-985-503-558-0. — Режим доступа: http://www.iprbookshop.ru/67612.html

Баженова, И. Ю. Основы проектирования приложений баз данных : учебное пособие для СПО / И. Ю. Баженова. — Саратов : Профобразование, 2019. — 325 c. — ISBN 978-5-4488-0361-1. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/86200.html

Разработка и защита баз данных в Microsoft SQL Server 2005 : учебное пособие для СПО / . — Саратов : Профобразование, 2019. — 148 c. — ISBN 978-5-4488-0366-6. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/86207.html

Сети и телекоммуникации : учебник и практикум для среднего профессионального образования / К. Е. Самуйлов [и др.] ; под редакцией К. Е. Самуйлова, И. А. Шалимова, Д. С. Кулябова. — Москва : Издательство Юрайт, 2019. — 363 с. — (Профессиональное образование). — ISBN 978-5-9916-0480-2. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://www.biblio-online.ru/bcode/430406

Дибров, М. В. Компьютерные сети и телекоммуникации. Маршрутизация в ip-сетях в 2 ч. Часть 1 : учебник и практикум для среднего профессионального образования / М. В. Дибров. — Москва : Издательство Юрайт, 2019. — 333 с. — (Профессиональное образование). — ISBN 978-5-534-04638-0. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://www.biblio-online.ru/bcode/437357

СОДЕРЖАНИЕ