Конспект лекций по предмету Базы данных

Разработчик: Калинкина Е. А - преподаватель общепрофессиональных дисциплин Областного государственного бюджетного профессионального образовательного учреждения «Рязанский колледж электроники»

Лекции 1,2.
Раздел 1 Введение в Базы и банки Данных

Введение

В настоящее время жизнь человека настолько сильно насыщена различного рода информацией, что для ее обработки требуется создание огромного количества хранилищ и банков данных различного назначения. Практически все системы в той или иной степени связаны с функциями долговременного хранения и обработки информации. Фактически информация становится фактором, определяющим эффективность любой сферы деятельности. Увеличились информационные потоки и повысились требования к скорости обработки данных, и теперь уже большинство операций не может быть выполнено вручную, они требуют применения наиболее перспективных компьютерных технологий. Любые административные решения требуют четкой и точной оценки текущей ситуации и возможных перспектив ее изменения. И если раньше в оценке ситуации участвовало несколько десятков факторов, которые могли быть вычислены вручную, то теперь таких факторов сотни и сотни тысяч, и ситуация меняется не в течение года, а через несколько минут, а обоснованность принимаемых решений требуется большая, потому что и реакция на неправильные решения более серьезная, более быстрая и более мощная, чем раньше. И, конечно, обойтись без информационной модели производства, хранимой в базе данных, в этом случае невозможно. По этой причине в последние годы появилось множество различных компьютерных систем, называемых системами управления базами данных, которые предназначены именно для этих целей. Для принятия обоснованных и эффективных решений в производственной деятельности, в управлении экономикой и в политике современный специалист должен уметь с помощью компьютеров и средств связи получать, накапливать, хранить и обрабатывать данные, представляя результат в виде наглядных документов. Поэтому, в данном реферате рассмотрим работу с базами данных.

Цель базы данных — помочь людям и организациям вести учет определенных вещей. На первый взгляд, эта цель кажется скромной, и вы, возможно, удивитесь, зачем нам нужна такая сложная технология и целый курс, посвященный этому предмету. Большинство из нас может вспомнить ситуации, в которых нам требуется отслеживать некоторые вещи. Многие, например, составляю список дел, которые нужно сделать на этой неделе, список покупок в магазине, список расходов для налоговой декларации и так далее. Почему не делать то же самое для информационных систем? На самых ранних стадиях развития информационных технологий использовались списки — набитые на перфокарте и написанные на магнитной ленте. Со временем, однако, стало ясно, что только немногие проблемы можно решить с помощью таких списков. В следующем разделе мы обсудим такие проблемы, а затем опишем, как построить базы данных для их решения.

Базы данных и информационные системы. Основные понятия.

Для облегчения обработки информации создаются информационные системы (ИС). Автоматизированными называются ИС, в которых применяют технические средства, в частности ЭВМ. Большинство существующих ИС являются автоматизированными, поэтому для краткости просто будем называть их ИС.

В широком понимании под определение ИС попадает любая система обработки информации. По области применения ИС можно разделить на системы: используемые в производстве, образовании, здравоохранении, науке, военном деле, социальной сфере, торговле и других отраслях. По целевой функции ИС можно условно разделить на следующие основные категории: управляющие, информационно-справочные, поддержки принятия решений.

Заметим, что иногда используется более узкая трактовка понятия ИС как совокупности аппаратно-програмных средств, задействованных для решения некоторой прикладной задачи. В организации например, могут существовать ИС, на которых соответственно возложены следующие задачи: учет кадров и материально-технических средств, расчет с поставщиками и заказчиками, бухгалтерский учет и т. п.

Банк данных является разновидностью ИС, в которой реализованы функции централизованного хранения и накопления обрабатываемой информации организованной в одну или несколько баз данных.

Банк данных в общем случае состоит из следующих компонентов: базы (нескольких баз) данных, системы управления базами данных, словаря данных, администратора, вычислительной системы и

обслуживающего персонала. Вкратце рассмотрим названные компоненты и некоторые связанные с ними важные понятия.

Ядром банка данных является база данных (БД), ориентированная на удовлетворение информационных потребностей многих пользователей.

База данных (БД) – представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние и их взаимосвязей в рассматриваемой предметной области.

Из определения следует, что база данных является проекцией предметной области, ее отражением. В этом отражении мы видим объекты предметной области, их состояние и отношения между ними. Отсюда следует, что для любой предметной области, для любой сферы может быть создана база данных, отражающая ее состояние. Одна из задач курса как раз научиться строить такие проекции, такие отражения, а также создавать средства для работы с ними.

Итак, база данных состоит из объектов, информации об их состоянии и отношений между ними.

Под объектом понимается некоторое целое (явление, понятие, предмет, процесс, действие), обладающее рядом неотъемлемых свойств (качеств).

Логическую структуры хранимых в базе данных называют моделью представления данных. К основным моделям представления данных (или просто моделям данных) относятся следующие: иерархическая сетевая, реляционная, постреляционная, многомерная и объектно-ориентированная. (подробнее мы это будем рассматривать в следующем разделе нашего курса).

Требования к базам данных. Свойства баз данных.

- Прежде всего БД должны удовлетворять актуальным информационным потребностям пользователей.
- БД должна также отвечать требованиям достоверности и непротиворечивости.
- Требуемая пользователям информация должна выдаваться достаточно быстро,
- к тому же БД в целом должны отвечать требованиям по производительности. В них следует предусмотреть возможность реорганизации и расширения при изменении предметной области.
- Более того, БД должны быть универсальными в плане изменения аппаратной и программной среды.
- Важными факторами являются также разграничение доступа и обеспечение независимости приложений от организации данных.

Наиболее широко БД используется в управленческой деятельности, а также в других сферах благодаря следующим свойствам.

- Важным и очевидным свойством является скорость получения необходимой информации.
- Вычислительная техника позволяет осуществить оперативный доступ к информации.
- Другим очевидным свойством БД является полная доступность. Вся информация, содержащаяся в ней, доступна для использования.
- Важным свойством, обеспечившим широкое применение БД, является гибкость. Имеется возможность получать ответы на те вопросы, которые ранее оставались без ответа.
- Изменения в БД должны вносится сравнительно легко.
- Последним не менее важным свойством БД является ее целостность, как логическая, так и физическая. В результате использования БД уменьшилось дублирование данных, появилась возможность упорядочить проведение обновления, что привело к согласованности данных.

Управляет работой БД специальная программа, которая организует и хранит данные таким образом, чтобы можно было ввести данные с клавиатуры или другого устройства ввода информации, снова найти и извлечь их при необходимости. Фактически это не одна программа, а целый пакет программ, обеспечивающий доступ к данным, централизованное управление данными и получивший название система управления базами данных (так называемая СУБД).

Т.е СУБД играет центральную роль в функционировании банка данных. *Система управления базой данных (СУБД)* – это комплекс языковых и программных средств, предназначенных для создания, ведения и совместимости использования БД многими пользователями.

Таким образом, введение СУБД отделяет логическую структуру данных от физической структуры данных в памяти ЭВМ. Не всякая управляющая программа работы с БД является СУБД. СУБД- это пакет программ, позволяющий:

- обеспечить пользователей (прикладные программы) языковыми средствами описания и манипулирования данными;
- обеспечить поддержку логических моделей данных. Модель данных определяет логическое представление физических данных;
- обеспечивать операции создания и манипулирования логическими данными (выбор, вставка, обновление, удаление и т.п.) и одновременное выполнение этих операций над физическими данными;
- обеспечить защиту и согласованность данных, поскольку при коллективном режиме работы многих пользователей возможно использование общих физических данных.

Обычно СУБД различают по используемой модели данных. Так, СУБД, основанные на использовании реляционной модели данных, называют реляционными СУБД.

СУБД дают возможность пользователям осуществлять непосредственное управление данными, а программистам быстро разрабатывать более совершенные программные средства их обработки. Первые СУБД были разработаны для больших и мини-ЭВМ. (Например: IMS (IBM, 1968 г.), ИНЭС (ВНИИСИ АН СССР, 1976 г.)). Они достаточно громоздки, сложны для освоения и использования. СУБД для ПК отличаются более простой архитектурой, значительно проще для освоения и использования, имеют развитые языки программирования БД. Количество современных систем управления базами данных исчисляется тысячами.

(Следующий объект банка данных): *Приложение* представляет собой программу или комплекс программ, обеспечивающих автоматизацию обработки информации для прикладной задачи. Нами рассматриваются приложения, использующие БД. Приложения могут создаваться в среде или вне среды СУБД – с помощью системы программирования, использующей средства к БД, к примеру Delphi. Приложения разработанные в среде СУБД, часто называют *приложениями СУБД*, а приложения вне СУБД, – *внешними приложениями.*

Для работы с базой данных зачастую достаточно СУБД и не нужно использовать приложения, создание которых требует программирования. Приложения разрабатывают главным образом в случаях, когда требуется обеспечить удобство работы с БД неквалифицированным пользователям или интерфейс СУБД не устраивает пользователей.

Словарь данных (СД) представляет собой подсистему БнД, предназначенную для централизованного хранения информации о структурах данных, взаимосвязях файлов БД друг с другом, типа данных и форматах их представления, принадлежности данных пользователям, кодах защиты и разграничения доступа и т.п.

Функционально СД присутствует во всех БнД, но не всегда выполняющий эти функции компонент имеет такое название. Чаще всего функции СД выполняются СУБД и вызываются из основного меню системы или реализуются с помощью ее утилит.

Администратор БД (АБД) есть лицо или группа лиц, отвечающее за выработку требований к БД, ее проектирование, создание, эффективное использование и сопровождение. В процессе эксплуатации АБД обычно за функционированием информационной системы, обеспечивает защиту от несанкционированного доступа, контролирует избыточность, непротиворечивость, сохранность и достоверность хранимой в БД информации. Для однопользовательских информационных систем функции АБД обычно возлагаются на лиц, непосредственно работающих с приложениями БД.

В вычислительной сети АБД, как правило, взаимодействует с *администратором сети*. В обязанности последнего входит контроль за функционированием аппаратно-программных средств сети, реконфигурация сети, восстановление программного обеспечения после сбоев и отказов оборудования, профилактические мероприятия и обеспечение разграничения доступа.

Вычислительная система (ВС) представляет собой совокупность взаимосвязанных и согласованно действующих ЭВМ или процессоров и других устройств, обеспечивающих автоматизацию процессов приема, обработки и выдачи информации потребителям. Поскольку основными функциями БнД являются хранение и обработка данных, то используемая ВС наряду с приемлемой мощностью центральных процессоров (ЦП) должна иметь достаточный объем оперативной и внешней памяти прямого доступа.

Обслуживающий персонал выполняет функции поддержания технических и программных средств в работоспособном состоянии. Он проводит профилактические, регламентные, восстановительные и другие работы по планам, а также по мере необходимости.

История развития баз данных

В истории вычислительной техники можно проследить развитие двух основных областей ее использования. Первая область — применение вычислительной техники для выполнения численных расчетов, которые слишком долго или вообще невозможно производить вручную. Развитие этой области способствовало расширению методов численного решения сложных математических задач, появлению языков программирования, ориентированных на удобную запись численных алгоритмов, становлению обратной связи с разработчиками новых архитектур ЭВМ.

Вторая область, которая непосредственно относится к нашей теме, — это использование средств вычислительной техники в автоматических или автоматизированных информационных системах. Информационная система представляет собой программно-аппаратный комплекс, обеспечивающий выполнение следующих функций:

- надежное хранение информации в памяти компьютера;
- выполнение специфических для данного приложения преобразований информации и вычислений;
- предоставление пользователям удобного и легко осваиваемого интерфейса.

Обычно такие системы имеют дело с большими объемами информации, имеющей достаточно сложную структуру. Классическими примерами информационных систем являются банковские системы, автоматизированные системы управления предприятиями, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т. д.

Вторая область использования вычислительной техники возникла несколько позже первой. Это связано с тем, что на заре вычислительной техники возможности компьютеров по хранению информации были очень ограниченными. Говорить о надежном и долговременном хранении информации можно только при наличии запоминающих устройств, сохраняющих информацию после выключения электрического питания. Оперативная (основная) память компьютеров этим свойством обычно не обладает. В первых компьютерах использовались два вида устройств внешней памяти — магнитные ленты и барабаны. Емкость магнитных лент была достаточно велика, но по своей физической природе они обеспечивали последовательный доступ к данным. Магнитные же барабаны (они ближе всего к современным магнитным дискам с фиксированными головками) давали возможность произвольного доступа к данным, но имели ограниченный объем хранимой информации.

Наличие сравнительно медленных устройств хранения данных, к которым относятся магнитные ленты и барабаны, было недостаточным.

Следующими появились съемные магнитные диски с подвижными головками, что явилось революцией в истории вычислительной техники. Эти устройства внешней памяти обладали существенно большей емкостью, чем магнитные барабаны, обеспечивали удовлетворительную скорость доступа к данным в режиме произвольной выборки, а возможность смены дискового пакета на устройстве позволяла иметь практически неограниченный архив данных.

С появлением магнитных дисков началась история систем управления данными во внешней памяти.

Важным шагом в развитии именно информационных систем явился переход к использованию централизованных систем управления файлами. С точки зрения прикладной программы, файл — это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные.

Главное, что следует отметить, это то, что структура записи файла была известна только программе, которая с ним работала, система управления файлами не знала ее. И поэтому для того, чтобы извлечь некоторую информацию из файла, необходимо было точно знать структуру записи файла с точностью до бита. Каждая программа, работающая *с* файлом, должна была иметь у себя внутри структуру данных, соответствующую структуре этого файла. Поэтому при изменении структуры файла требовалось изменять структуру программы, а это требовало новой компиляции, то есть процесса перевода программы в исполняемые машинные коды. Такая ситуации характеризовалась как зависимость программ от данных. Для информационных систем характерным является наличие большого числа различных пользователей (программ), каждый из которых имеет свои специфические алгоритмы обработки информации, хранящейся в одних и тех же файлах. Изменение структуры файла,

которое было необходимо для одной программы, требовало исправления и дополнительной отладки всех остальных программ, работающих с этим же файлом. Это было первым существенным недостатком файловых систем, который явился толчком к созданию новых систем хранения и управления информацией.

Отсутствие централизованных методов управления доступом к информации послужило еще одной причиной разработки СУБД.

Следующей причиной стала необходимость обеспечения эффективной параллельной работы многих пользователей с одними и теми же файлами. В общем случае системы управления файлами обеспечивали режим многопользовательского доступа. Если операционная система поддерживает многопользовательский режим, вполне реальна ситуация, когда два или более пользователя одновременно пытаются работать с одним и тем же файлом. Если все пользователи собираются только читать файл, ничего страшного не произойдет. Но если хотя бы один из них будет изменять файл, для корректной работы этих пользователей требуется взаимная синхронизация их действий по отношению к файлу.

При подобном способе организации одновременная работа нескольких пользователей, связанная с модификацией данных в файле, либо вообще не реализовывалась, либо была очень замедлена.

Эти недостатки послужили тем толчком, который заставил разработчиков информационных систем предложить новый подход к управлению информацией. Этот подход был реализован в рамках новых программных систем, названных впоследствии Системами Управления Базами Данных (СУБД), а сами хранилища информации, которые работали под управлением данных систем, назывались базами или банками данных (БД и БнД).

Первый этап — базы данных на больших ЭВМ

История развития СУБД насчитывает более 30 лет.

Базы данных хранились во внешней памяти центральной ЭВМ, пользователями этих баз данных были задачи, запускаемые в основном в пакетном режиме.

Особенности этого этапа развития выражаются в следующем:

- Все СУБД базируются на мощных мультипрограммных операционных системах (MVS, SVM, RTE, OSRV, RSX, UNIX), поэтому в основном поддерживается работа с централизованной базой данных в режиме распределенного доступа.
- Функции управления распределением ресурсов в основном осуществляются операционной системой (ОС).
- Поддерживаются языки низкого уровня манипулирования данными, ориентированные на навигационные методы доступа к данным.
- Значительная роль отводится администрированию данных.
- Проводятся серьезные работы по обоснованию и формализации реляционной модели данных, и была создана первая система (System R), реализующая идеологию реляционной модели данных.
- Проводятся теоретические работы по оптимизации запросов и управлению распределенным доступом к централизованной БД.
- Результаты научных исследований открыто обсуждаются в печати, идет мощный поток общедоступных публикаций, касающихся всех аспектов теории и практики баз данных, и результаты теоретических исследований активно внедряются в коммерческие СУБД.

Появляются первые языки высокого уровня для работы с реляционной моделью данных. Однако отсутствуют стандарты для этих первых языков.

Второй Этап: Эпоха персональных компьютеров

С ПК появилось множество программ, предназначенных для работы неподготовленных пользователей. Эти программы были просты в использовании и интуитивно понятны: это прежде всего различные редакторы текстов, электронные таблицы и другие. Простыми и понятными стали операции копирования файлов и перенос информации с одного компьютера на другой, распечатка текстов, таблиц и других документов. Системные программисты были отодвинуты на второй план. Каждый пользователь мог себя почувствовать полным хозяином этого мощного и удобного устройства. И, конечно, это сказалось и на работе с базами данных. Появились программы, которые назывались

системами управления базами данных и позволяли хранить значительные объемы информации, они имели удобный интерфейс для заполнения данных, встроенные средства для генерации различных отчетов. Эти программы позволяли автоматизировать многие учетные функции, которые раньше велись вручную. Постоянное снижение цен на персональные компьютеры сделало их доступными не только для организаций и фирм, но и для отдельных пользователей. Компьютеры стали инструментом для ведения документации и собственных учетных функций. Это все сыграло как положительную, так и отрицательную роль в области развития баз данных.

Особенности этого этапа следующие:

- Все СУБД были рассчитаны на создание БД в основном с монопольным доступом. И это понятно. Компьютер персональный, он не был подсоединен к сети, и база данных на нем создавалась для работы одного пользователя. В редких случаях предполагалась последовательная работа нескольких пользователей, например, сначала оператор, который вводил бухгалтерские документы, а потом главбух, который определял проводки, соответствующие первичным документам.
- Большинство СУБД имели развитый и удобный пользовательский интерфейс. В большинстве существовал интерактивный режим работы с БД как в рамках описания БД, так и в рамках проектирования запросов.
- Во всех настольных СУБД поддерживался только внешний уровень представления реляционной модели, то есть только внешний табличный вид структур данных.
- При наличии высокоуровневых языков манипулирования данными типа реляционной алгебры и SQL в настольных СУБД поддерживались низкоуровневые языки манипулирования данными на уровне отдельных строк таблиц.
- В настольных СУБД отсутствовали средства поддержки ссылочной и структурной целостности базы данных. Эти функции должны были выполнять приложения, однако скудость средств разработки приложений иногда не позволяла это сделать, и в этом случае эти функции должны были выполняться пользователем, требуя от него дополнительного контроля при вводе и изменении информации, хранящейся в БД.
- Наличие монопольного режима работы фактически привело к вырождению функций администрирования БД и в связи с этим — к отсутствию инструментальных средств администрирования БД.
- И, наконец, последняя и в настоящий момент весьма положительная особенность — это сравнительно скромные требования к аппаратному обеспечению со стороны настольных СУБД.

Третий Этап: Распределенные базы данных

Хорошо известно, что история развивается по спирали, поэтому после процесса «персонализации» начался обратный процесс — интеграция. Множится количество локальных сетей, все больше информации передается между компьютерами, остро встает задача согласованности данных, хранящихся и обрабатывающихся в разных местах, но логически друг с другом связанных, возникают задачи, связанные с параллельной обработкой *транзакций* — последовательностей операций над БД, переводящих ее из одного непротиворечивого состояния в другое непротиворечивое состояние. Успешное решение этих задач приводит к появлению *распределенных баз данных,* сохраняющих все преимущества настольных СУБД и в то же время позволяющих организовать параллельную обработку информации и поддержку целостности БД.

Особенности данного этапа:

- Практически все современные СУБД обеспечивают поддержку полной реляционной модели, а именно:
  - структурной целостности — допустимыми являются только данные, представленные в виде отношений реляционной модели;
  - языковой целостности, то есть языков манипулирования данными высокого уровня (в основном SQL);
  - ссылочной целостности, контроля за соблюдением ссылочной целостности в течение всего времени функционирования системы, и гарантий невозможности со стороны СУБД нарушить эти ограничения.
- Большинство современных СУБД рассчитаны на многоплатформенную архитектуру, то есть они могут работать на компьютерах с разной архитектурой и под разными операционными

системами, при этом для пользователей доступ к данным, управляемым СУБД на разных платформах, практически неразличим.

- Необходимость поддержки многопользовательской работы с базой данных и возможность централизованного хранения данных потребовали развития средств защиты данных.
- Потребность в новых реализациях вызвала создание серьезных теоретических трудов по оптимизации реализаций распределенных БД и запросами с внедрением полученных результатов в коммерческие СУБД.
- Для того чтобы не потерять клиентов, которые ранее работали на настольных СУБД, практически все современные СУБД имеют средства подключения клиентских приложений, разработанных с использованием настольных СУБД, и средства экспорта данных из форматов настольных СУБД второго этапа развития.
- Именно к этому этапу можно отнести разработку ряда стандартов в рамках языков описания и манипулирования данными начиная с SQL89, SQL92, SQL99 и технологий по обмену данными между различными СУБД.
- Именно к этому этапу можно отнести начало работ, связанных с концепцией объектно-ориентированных БД — ООБД. Представителями СУБД, относящимся ко второму этапу, можно считать MS Access 97 и все современные серверы баз данных Oracle7.3,Oracle 8.4 MS SQL6.5, MS SQL7.0, System 10, System 11, Informix, DB2, SQL Base и другие современные серверы баз данных, которых в настоящий момент насчитывается несколько десятков.

И четвертый этап: Перспективы развития систем управления базами данных

Этот этап характеризуется появлением новой технологии доступа к данным — *интранет.* Основное отличие этого подхода от технологии клиент-сервер состоит в том, что отпадает необходимость использования специализированного клиентского программного обеспечения. Для работы с удаленной базой данных используется стандартный браузер Интернета и для конечного пользователя процесс обращения к данным происходит аналогично скольжению по Всемирной Паутине. При этом встроенный в загружаемые пользователем HTML-страницы код, написанный обычно на языке Java, Java-script, Perl и других, отслеживает все действия пользователя и транслирует их в низкоуровневые SQL-запросы к базе данных, выполняя, таким образом, ту работу, которой в технологии клиент-сервер занимается клиентская программа. Удобство данного подхода привело к тому, что он стал использоваться не только для удаленного доступа к базам данных, но и для пользователей локальной сети предприятия. В этом случае для подключения нового пользователя к возможности использовать данную задачу не требуется установка дополнительного клиентского программного обеспечения. Однако алгоритмически сложные задачи рекомендуется реализовывать в архитектуре «клиент-сервер» с разработкой специального клиентского программного обеспечения.

У каждого из вышеперечисленных подходов к работе с данными есть свои достоинства и свои недостатки, которые и определяют область применения того или иного метода, и в настоящее время все подходы широко используются.

Лекция 4

Архитектура базы данных. Физическая и логическая независимость.

Терминология в СУБД, да и сами термины «база данных» и «банк данных» частично заимствованы из финансовой деятельности. Это заимствование — не случайно и объясняется тем, что работа с информацией и работа с денежными массами во многом схожи: например, две банкноты достоинством в сто рублей столь же неотличимы и взаимозаменяемы, как два одинаковых байта (естественно, за исключением серийных номеров). Вы можете положить деньги на некоторый счет и предоставить возможность вашим родственникам или коллегам использовать их для иных целей. Вы можете поручить банку оплачивать ваши расходы с вашего счета или получить их наличными в другом банке, и это будут уже другие денежные купюры, но их ценность будет эквивалентна той, которую вы имели, когда клали их на ваш счет.

Одним из важнейших аспектов развития СУБД стала идея отделения логической структуры БД и манипуляций данными, необходимых пользователям, от физического представления, требуемого компьютерным оборудованием. И идея эта должна быть заложена в фундамент, на котором будет строиться все здание информационной системы.

Одна и та же БД в зависимости от точки зрения может иметь различные уровни описания. По числу уровней описания данных, поддерживаемых СУБД, различают одно-, двух- и трехуровневые системы. В настоящее время чаще всего поддерживается трехуровневая архитектура описания БД (рис.), с тремя уровнями абстракции, на которых можно рассматривать базу данных.

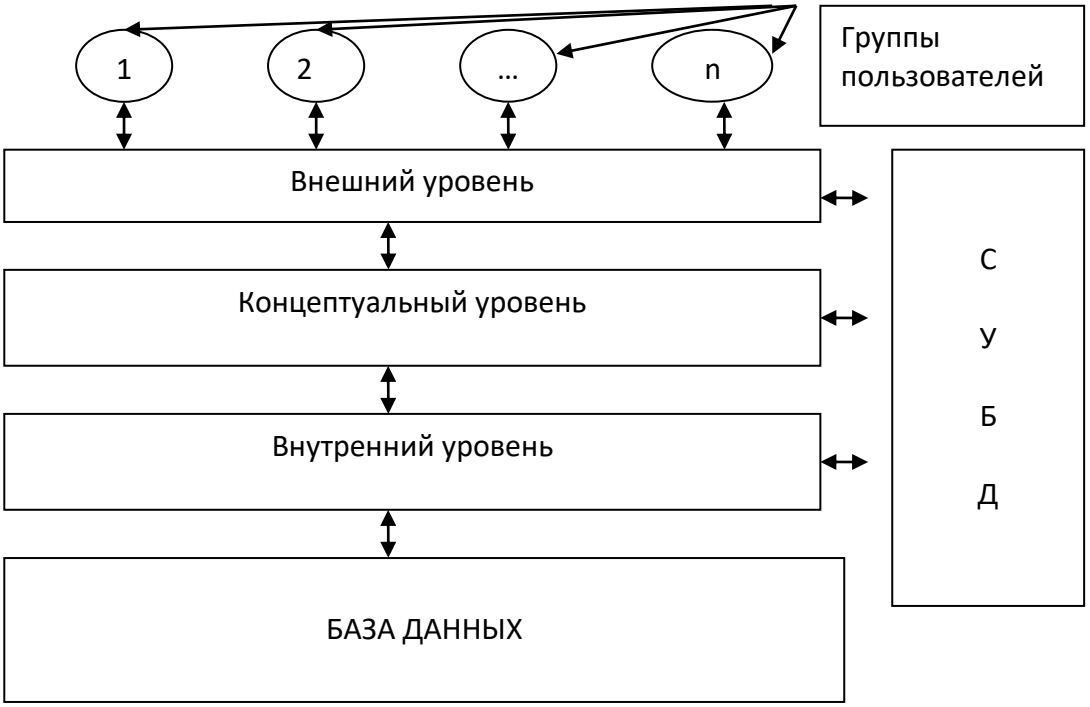


Рис. *Трехуровневая модель системы управления базой данных, предложенная ANSI*

Такая архитектура включает:

1. Внешний уровень, на котором пользователи воспринимают данные, где отдельные группы пользователей имеют свое представление на базу данных, и каждая группа видит и обрабатывает только те данные, которые необходимы именно ей. Например, система распределения работ использует сведения о квалификации сотрудника, но ее не интересуют сведения об окладе, домашнем адресе и телефоне сотрудника, и наоборот, именно эти сведения используются в подсистеме отдела кадров.
2. Внутренний уровень. На котором СУБД и операционная система воспринимают данные;
3. Концептуальный уровень — центральное управляющее звено, здесь база данных представлена в наиболее общем виде, который объединяет данные, используемые всеми приложениями, работающими с данной базой данных. Фактически концептуальный уровень отражает обобщенную

модель предметной области (объектов реального мира), для которой создавалась база данных. Как любая модель, концептуальная модель отражает только существенные, с точки зрения обработки, особенности объектов реального мира.

Данная архитектура СУБД вызревала не сразу, а постепенно, в течение ряда лет. Первые предложения поступили в 1971 году от рабочей группы участвовавшей на конференции по языкам и системам данных, которая обосновала необходимость использования двухуровневого подхода, построенного на основе выделения системного представления и пользовательских представлений.

В 1975 году Комитет планирования стандартов и норм Американского национального института стандартов предложил обобщенную структуру систем баз данных, признав необходимость использования трехуровневой архитектуры, которая и была официально признана в 1978 году.

Описание структуры данных на любом уровне называется *схемой.*

Основным назначением трехуровневой архитектуры является обеспечение независимости от данных. Суть этой независимости заключается в том, что изменения на нижних уровнях никак не влияют на верхние уровни. Различают два типа независимости от данных: логическую и физическую.

Логическая независимость от данных означает полную защищенность внешних схем от изменений, вносимых в концептуальную схему. Такие изменения концептуальной схемы, как добавление или удаление новых сущностей, атрибутов или связей, должны осуществляться без необходимости внесения изменений в уже существующие внешние схемы для других групп пользователей. Таким образом, тем группам пользователей, которых эти изменения не касаются, не потребуется вносить изменения в свои программы.

Физическая независимость от данных означает защищенность концептуальной схемы от изменений, вносимых во внутреннюю схему. Такие изменения внутренней схемы, как использование различных файловых систем или структур хранения, разных устройств хранения, модификация индексов или хеширование, должны осуществляться без необходимости внесения изменений в концептуальную или внешнюю схемы. Пользователем могут быть замечены изменения только в общей производительности системы.

Далее рассмотрим каждый из трех названных уровней.

Внешний уровень — это пользовательский уровень. Пользователем может быть программист, или конечный пользователь, или администратор базы данных. Представление базы данных с точки зрения пользователей называется *внешним представлением.* Каждая группа пользователей выделяет в моделируемой предметной области, общей для всей организации, те сущности, атрибуты и связи, которые ей интересны. Выражая их в наиболее удобной для себя форме, она формирует свое пользовательское представление, причем одни и те же данные могут отображаться по-разному в разных пользовательских представлениях. Например, в информационной системе СУЗ пользователя из бухгалтерии будет интересовать информация о студентах, которым должна быть начислена стипендия, но его совершенно не будет интересовать информация об успеваемости всех студентов и многое другое.

Эти частичные или переопределенные описания БД для отдельных групп пользователей или ориентированные на отдельные аспекты предметной области называют *подсхемой.*

При рассмотрении внешнего уровня БД важно отметить, что каждый тип пользователей может применять для работы с БД свой язык общения. Конечные пользователи употребляют либо язык запросов, либо специальный язык, поддерживаемый приложениями и вызывающий определенные для пользователя экранные формы и пользовательские меню. Прикладные программисты чаще применяют либо языки высокого уровня, например С, Pascal и т. п., либо специальные языки СУБД. Языки последнего типа относят к языкам четвертого поколения.

Какой бы язык высокого уровня не использовался, он должен включать в себя и подъязык для работы с данными. Система может поддерживать любое количество подъязыков данных, любое количество базовых языков. Но язык SQL поддерживается практически всеми системами. Он может использоваться и как встроенный в другие языки, и как отдельный самостоятельный язык запросов.

Концептуальный уровень является промежуточным уровнем в трехуровневой архитектуре и обеспечивает представление всей информации базы данных в абстрактной форме. Описание базы данных на этом уровне называется концептуальной схемой, которая является результатом концептуального проектирования.

Концептуальное проектирование базы данных включает анализ информационных потребностей пользователей и определение нужных им элементов данных. Таким образом, *концептуальная схема* — это единое логическое описание всех элементов данных и отношений между ними, логическая структура всей базы данных. Для каждой базы данных имеется только одна концептуальная схема.

Концептуальная схема должна содержать:

- объекты и их атрибуты; связи между объектами;
- ограничения, накладываемые на данные;
- семантическую информацию о данных;
- обеспечение безопасности и поддержки целостности данных.

Концептуальный уровень поддерживает каждое внешнее представление, в том смысле, что любые доступные пользователю данные должны содержаться (или могут быть вычислены) на этом уровне. Однако этот уровень не содержит никаких сведений о методах хранения данных.

Внутренний уровень является третьим уровнем архитектуры БД. На внутреннем уровне область хранения представляется как бесконечное линейное адресное пространство.

На нижнем уровне находится внутренняя схема, которая является полным описанием внутренней модели данных. Для каждой базы данных существует только одна внутренняя схема.

Внутренняя схема описывает физическую реализацию базы данных и предназначена для достижения оптимальной производительности и обеспечения экономного использования дискового пространства. Именно на этом уровне осуществляется взаимодействие СУБД с методами доступа операционной системы (вспомогательными функциями хранения и извлечения записей данных) с целью размещения данных на запоминающих устройствах, создания индексов, извлечения данных и т. д. На внутреннем уровне хранится следующая информация:

- распределение дискового пространства для хранения данных и индексов;
- описание подробностей сохранения записей (с указанием реальных размеров сохраняемых элементов данных);
- сведения о размещении записей;
- сведения о сжатии данных и выбранных методах их шифрования.

СУБД отвечает за установление соответствия между всеми тремя типами схем разных уровней, а также за проверку их непротиворечивости.

Ниже внутреннего уровня находится *физический уровень,* который контролируется операционной системой, но под руководством СУБД. Физический уровень учитывает, каким образом данные будут представлены в машине. Он обеспечивает физический взгляд на базу данных: дисководы, физические адреса, индексы, указатели и т. д. За этот уровень отвечают проектировщики физической базы данных, которые работают только с известными операционной системе элементами.

Итак, подведем итоги. Реализация трехуровневой архитектуры БД требует, чтобы СУБД переводила информацию с одного уровня на другой. Выгодой такого перевода является независимость логического и физического представления данных, но и плата за эту независимость немалая — большая системная задержка.

Для установления соответствия между любыми внешней и внутренней схемами СУБД должна использовать информацию из концептуальной схемы. Концептуальная схема связана с внутренней схемой посредством концептуально-внутреннего отображения. Оно позволяет СУБД найти фактическую запись или набор записей на физическом устройстве хранения, которые образуют логическую запись в концептуальной схеме.

В то же время каждая внешняя схема связана с концептуальной схемой с помощью внешне-концептуального отображения. С его помощью СУБД может отображать имена пользовательского представления на соответствующую часть концептуальной схемы.

Процесс прохождения пользовательского запроса

Рисунок иллюстрирует взаимодействие пользователя, СУБД и ОС при обработке запроса на получение данных. Цифрами помечена последовательность взаимодействий:

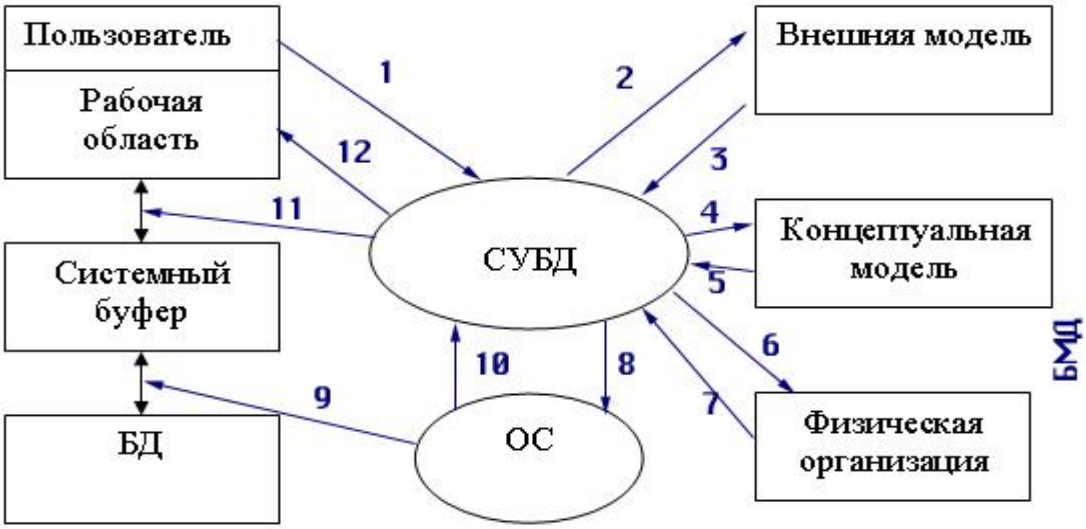


Рис. 2.2. *Схема прохождения запроса к БД*

1. Пользователь посылает СУБД запрос на получение данных из БД.
2. Анализ прав пользователя и внешней модели данных, соответствующей данному пользователю, подтверждает или запрещает доступ данного пользователя к запрошенным данным.
3. В случае запрета на доступ к данным СУБД сообщает пользователю об этом (стрелка 12) и прекращает дальнейший процесс обработки данных, в противном случае СУБД определяет часть концептуальной модели, которая затрагивается запросом пользователя.
4. СУБД получает информацию о запрошенной части концептуальной модели.
5. СУБД запрашивает информацию о местоположении данных на физическом уровне (файлы или физические адреса).
6. В СУБД возвращается информация о местоположении данных в терминах операционной системы.
7. СУБД вежливо просит операционную систему предоставить необходимые данные, используя средства операционной системы.
8. Операционная система осуществляет перекачку информации из устройств хранения и пересылает ее в системный буфер.
9. Операционная система оповещает СУБД об окончании пересылки.
10. СУБД выбирает из доставленной информации, находящейся в системном буфере, только то, что нужно пользователю, и пересылает эти данные в рабочую область пользователя.

БМД — это *База Метаданных,* именно здесь и хранится вся информация об используемых структурах данных, логической организации данных, правах доступа пользователей и, наконец, физическом расположении данных. Для управления БМД существует специальное программное обеспечение администрирования баз данных, которое предназначено для корректного использования единого информационного пространства многими пользователями.

Всегда ли запрос проходит полный цикл? Конечно, нет. СУБД обладает достаточно развитым интеллектом, который позволяет ей не повторять бессмысленных действий. И поэтому, например, если этот же пользователь повторно обратится к СУБД с новым запросом, то для него уже не будут проверяться внешняя модель и права доступа, а если дальнейший анализ запроса покажет, что данные могут находиться в системном буфере, то СУБД осуществит только 11 и 12 шаги в обработке запроса.

Разумеется, механизм прохождения запроса в реальных СУБД гораздо сложнее, но и эта упрощенная схема показывает, насколько серьезными и сложными должны быть механизмы обработки запросов, поддерживаемые реальными СУБД.

Архитектура информационной системы.

Эффективность функционирования информационной системы (ИС) во многом зависит от ее архитектуры. В настоящее время перспективной является архитектура клиент-сервер. В достаточно распространенном варианте она предполагает наличие компьютерной сети и распределенной базы данных, включающей корпоративную базу данных (КБД) и персональные базы данных (ПБД). *Распределенная база данных* – это совокупность логически взаимосвязанных баз данных, распределенных в компьютерной сети. КБД размещается на компьютере-сервере, ПБД размещаются на компьютерах сотрудников подразделений, являющихся клиентами корпоративной БД.

Сервером определенного ресурса в компьютерной сети называется компьютер (программа), управляющий этим ресурсом, ***клиентом*** — компьютер (программа), использующий этот ресурс. В качестве ресурса компьютерной сети могут выступать, к примеру, базы данных, файловые системы, службы печати, почтовые службы. Тип сервера определяется видом ресурса, которым он управляет. Например, если управляемым ресурсом является база данных, то соответствующий сервер называется *сервером базы данных.*

Достоинством организации информационной системы по архитектуре клиент-сервер является удачное сочетание *централизованного* хранения, обслуживания **и** коллективного доступа к общей корпоративной информации с индивидуальной работой пользователей над персональной информацией. Архитектура клиент-сервер допускает различные варианты реализации.

Исторически первыми появились распределенные ИС с применением ***файл-сервера*** (рис. 1).

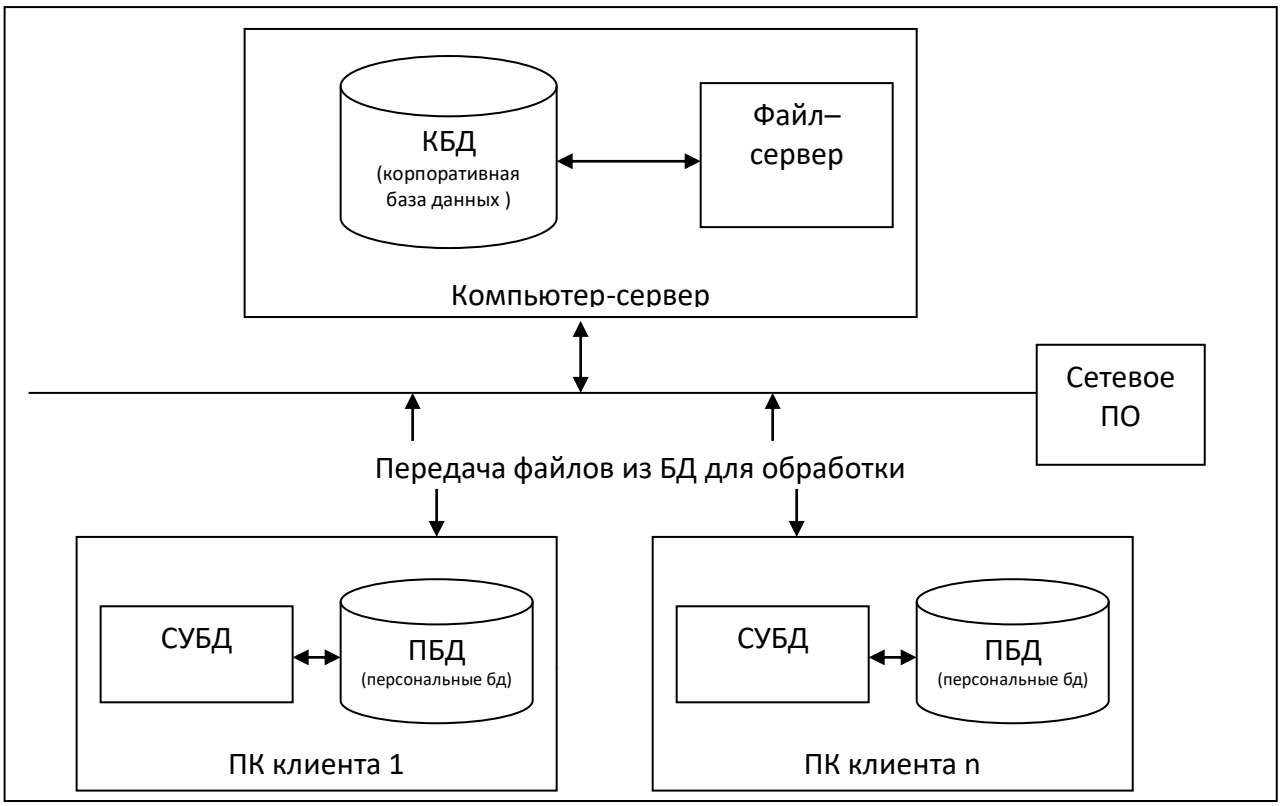
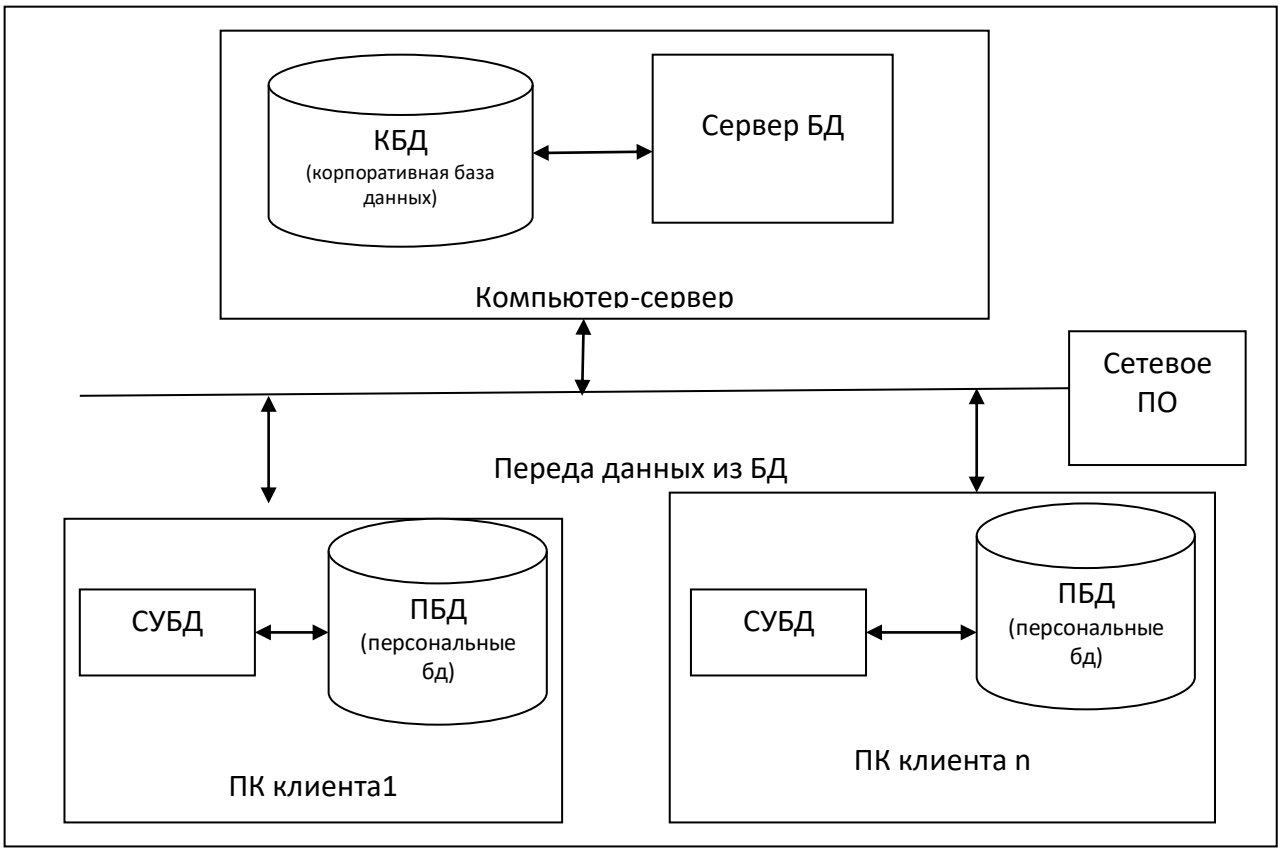


Рис. 1 Структура ИС с файл-сервером

В таких ИС по запросам пользователей файлы базы данных передаются на персональные компьютеры (ПК), где и производится их обработка. *Недостатком* такого варианта архитектуры является высокая интенсивность передачи обрабатываемых данных. Причем зачастую передаются избыточные данные: вне зависимости от того сколько записей из базы данных требуется пользователю, файлы базы данных передаются целиком.

Теперь рассмотрим структуру распределенной ИС, построенной по архитектуре клиент-сервер с использованием сервера баз данных, рис. 2.



При такой архитектуре сервер базы данных обеспечивает выполнение основного объема обработки данных. Формируемые пользователем или приложением запросы поступают к серверу БД в виде инструкций языка SQL. Сервер базы данных выполняет поиск и извлечение нужных данных, которые затем передаются на компьютер пользователя. *Достоинством* такого подхода в сравнении предыдущим является заметно меньший объем передаваемых данных.

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

• компьютер-сервер, содержащий корпоративную и персональные базы;

• компьютер-сервер и персональные компьютеры с ПБД;

• несколько компьютеров-серверов и персональных компьютеров с ПБД.

Использование архитектуры клиент-сервер дает возможность постепенного наращивания информационной системы предприятия, во-первых, по мере развития предприятия; во-вторых, по мере развития самой информационной системы.

Разделение общей БД на корпоративную БД и персональные БД позволяет уменьшить сложность проектирования БД по сравнению с централизованным вариантом, а значит, снизить вероятность ошибок при проектировании и стоимость проектирования.

Важнейшим *достоинством* применения БД в информационных системах является обеспечение независимости данных от прикладных программ. Это даёт возможность пользователям не заниматься проблемами представления данных на физическом уровне: размещения данных в памяти, методов доступа к ним и т. д.

Классификация СУБД.

В качестве основных классификационных признаков СУБД можно использовать следующие: вид программы, характер использования, модель данных. Названные признаки существенно влияют на целевой выбор СУБД и эффективность использования разрабатываемой информационной системы.

В общем случае под СУБД можно понимать любой программный продукт, поддерживающий процессы создания, ведения и использования БД. Рассмотрим, какие из имеющихся на рынке программ имеют отношение к БД и в какой мере они связаны с базами данных.

К СУБД относятся следующие основные виды программ:

•полнофункциональные СУБД;

•серверы БД;

•клиенты БД;

•средства разработки программ работы с БД.

Полнофункциональные СУБД (ПФСУБД) представляют собой традиционные СУБД, которые сначала появились для больших машин, затем для мини-машин и для ПЭВМ. Из числа всех СУБД современные ПФСУБД являются наиболее многочисленными и мощными по своим возможностям.

Обычно ПФСУБД имеют развитый интерфейс, позволяющий с помощью команд меню выполнять основные действия с БД: создавать и модифицировать структуры таблиц, вводить данные, формировать запросы, разрабатывать отчеты, выводить их на печать и т. п.

Серверы БД предназначены для организации центров обработки данных в сетях ЭВМ. Эта группа БД в настоящее время менее многочисленна, но их количество постепенно растет. Серверы БД реализуют функции управления базами данных, запрашиваемые другими (клиентскими) программами обычно с помощью операторов SQL.

В роли ***клиентских программ*** для серверов БД в общем случае могут использоваться различные программы: ПФСУБД, электронные таблицы, текстовые процессоры, программы электронной почты и т. д. При этом элементы пары «клиент — сервер» могут принадлежать одному или разным производителям программного обеспечения.

В случае, когда клиентская и серверная части выполнены одной фирмой, естественно ожидать, что распределение функций между ними выполнено рационально. В остальных случаях обычно преследуется цель обеспечения доступа к данным «любой ценой». Примером такого соединения является случай, когда одна из полнофункциональных СУБД играет роль сервера, а вторая СУБД (другого производителя) — роль клиента.

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

•клиентских программ;

•серверов БД и их отдельных компонентов;

•пользовательских приложений.

Программы первого и второго вида довольно малочисленны, так как предназначены, главным образом, для системных программистов. Пакетов третьего вида гораздо больше, но меньше, чем полнофункциональных СУБД.

По характеру использования СУБД делят на *персональные* и *многопользовательские.*

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

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

В зависимости от способа хранения и обработки БД (централизованного или децентрализованного) СУБД можно разделить на два класса: *централизованные* (или обычные) СУБД и *децентрализованные* (или распределенные) СУБД. В обычных СУБД данные хранятся в том же месте, где и программы их управления. В распределенных СУБД как программное обеспечение, так и данные распределены по узлам сети. Распределенные СУБД могут быть *однородными* или *неоднородными.*

Неоднородность СУБД может проявлятся в отличии поддерживаемых модели данных, типов данных, языков запросов, фирм-разработчиков и т. д.

Одной из разновидностей распределённых СУБД являются *мультибазовые системы,* в которых управление каждым из узлов осуществляется автономно. В мультибазовых СУБД производится такая интеграция локальных систем, при которой не требуется изменение существующих СУБД и в то же время конечным пользователям предоставляется доступ к совместно используемым данным. Пользователи локальных СУБД получают возможность управлять данными собственных узлов без централизованного контроля, который присутствует в обычных распределенных СУБД.

В зависимости от возможности распараллеливания процесса обработки данных выделяют *СУБД с последовательной и параллельной обработкой (параллельные СУБД).* Параллельные СУБД функционируют в многопроцессорной вычислительной системе (как правило, со множеством устройств хранения данных) или в сети компьютеров.

С точки зрения пользователя, СУБД реализует ***функции*** хранения, изменения (пополнения, редактирования и удаления) и обработки информации, а также разработки и получения различных выходных документов:

Для работы с хранящейся в базе данных информацией СУБД предоставляет программам и пользователям следующие два типа *языков:*

•язык описания данных — высокоуровневый непроцедурный язык декларативного типа, предназначенный для описания логической структуры данных;

•язык манипулирования данными — совокупность конструкций, обеспечивающих выполнение основных операций по работе с данными: ввод, модификацию и выборку данных по запросам.

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

Перечисленные выше функции СУБД, в свою очередь, используют следующие основные функции более низкого уровня, которые назовем ***низкоуровневыми:***

1. управление данными во внешней памяти;
2. управление буферами оперативной памяти;
3. управление транзакциями;
4. ведение журнала изменений в БД;
5. обеспечение целостности и безопасности

Дадим краткую характеристику необходимости и особенностям реализации перечисленных функций в современных СУБД.

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

Необходимость буферизации данных и как следствие реализации функции *управления буферами* оперативной памяти обусловлено тем, что объем оперативной памяти меньше объема внешней памяти.

Буферы представляют собой области оперативной памяти, предназначенные для ускорения обмена между внешней и оперативной памятью. В буферах временно хранятся фрагменты БД, данные из которых предполагается использовать при обращении к СУБД или планируется записать в базу после обработки.

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

В зависимости от времени, требуемого для выполнения, выделяют *обычные и продолжительные транзакции.* Продолжительные транзакции могут охватывать часы, дни и даже месяцы. Такие транзакции могут возникать в процессе проектирования и разработки сложных систем крупным коллективом людей. Кроме того, помимо обычных *плоских транзакций,* используется модель *вложенных транзакций.* В этом случае транзакция рассматривается как набор взаимосвязанных подзадач (субтранзакций), каждая из которых также может состоять из произвольного количества субтранзакций.

Говорят, что транзакции присущи три основных свойства:

• атомарность (выполняются все входящие в транзакцию операции или ни одна);

• серализуемость (отсутствует взаимное влияние выполняемых в одно и то же время транзакций);

•долговечность (даже крах системы не приводит к утрате результатов зафиксированной транзакции).

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

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

При параллельном выполнении смеси транзакций возможно возникновение конфликтов (блокировок), разрешение которых является функцией СУБД. При обнаружении таких случаев обычно производится «откат» путем отмены изменений, произведенных одной или несколькими транзакциями.

Ведение журнала изменений в БД (журнализация изменений) выполняется СУБД для обеспечения надежности хранения данных в базе при наличии аппаратных сбоев и отказов, а также ошибок в программном обеспечении.

Журнал СУБД — это особая БД или часть основной БД, непосредственно недоступная пользователю и используемая для записи информации обо всех изменениях базы данных.

Обеспечение целостности БД составляет необходимое условие успешного функционирования БД, особенно для случая использования БД в сетях. ***Целостность БД*** есть свойство базы данных, означающее, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную область информация. Поддержание целостности БД включает проверку целостности и ее восстановление в случае обнаружения противоречий в базе данных. Целостное состояние БД описывается с помощью *ограничений целостности* в виде условий, которым должны удовлетворять хранимые в базе данные. Примером таких условий может служить ограничение диапазонов возможных значений атрибутов объектов, сведения о которых хранятся в БД, или отсутствие повторяющихся записей в таблицах реляционных БД.

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

Лекция 9-10

МОДЕЛИ ДАННЫХ

Цель лекции: знакомство с существующими моделями баз данных. Изучение особенностей и принципов организации наиболее популярных и современных из них.

Данные и информация

Одними из основополагающих в концепции баз данных являются обобщенные категории «данные» и «модель данных».

Понятие **«данные»** в концепции баз данных — это набор конкретных значений, параметров, характеризующих объект, условие, ситуацию или любые другие факторы в числовой, текстовой, графической, звуковой форме.

Примеры данных:

«Иванов Иван Иванович $30»

«Арматура Д10 10 40»

«Труба Д40 30 80»

и т.д.

Можно ли сказать, что представленные в примере данные являются информацией? Что означают данные «Иванов Иван Иванович $30»? Что означает «Арматура Д10 10 40»? Ответить на этот вопрос сейчас невозможно, т.к. данные «Иванов» могут оказаться фамилией сотрудника, а могут фамилией поставщика; это также может быть фамилия соседа по лестничной площадке или название фирмы. То же можно сказать и о второй строке: «Арматура» это может быть наименование товара, а может наименование строительного материала, который используется при строительстве здания. «10» -может означать количество, но в то же время сумму или цену, а возможно и еще что-то, например, остаток товара на складе.

Из приведенного примера видно, что значение этих данных, их смысл нам неизвестен. Однако если мы придадим этим данным определенную форму и впишем их в некоторую структуру все станет ясно.

Примеры данных;

Ф.И.О. сотрудника	Сумма выплаченного аванса, $
Иванов Иван Иванович	30

Наименование товара	Остаток на складе, т.	Цена, тыс. руб.
Арматура Д10	10	40
Труба Д40	30	80

Данные не обладают определенной структурой, данные становятся информацией тогда, когда пользователь задает им определенную структуру, то есть осознаёт их смысловое содержание. Поэтому **центральным понятием** в области баз данных является понятие модели данных.

Модель — упрощенная рабочая схема какого-либо объекта, явления, процесса или его части.

Модель данных - схема (порядок, совокупность принципов, система) организации данных в единое целое для создания, накопления, обработки и управления. Это некоторая абстракция, которая будучи приложена к конкретным данным, позволяет пользователям и разработчикам трактовать их уже как информацию, то есть сведения, содержащие не только данные, но и взаимосвязи между ними.

Применение той или иной модели данных позволяет придать данным конкретную форму (структуру) и смысловое значение. Данные, вписанные в определенную модель, называют **информацией.**

Более простое определение: *модель данных* - совокупность структур данных и операций по их обработке.

Классификация моделей данных

В соответствии с рассмотренной ранее трехуровневой архитектурой мы сталкиваемся с понятием модели данных по отношению к каждому уровню. И действительно, физическая модель данных оперирует категориями, касающимися организации внешней памяти и структур хранения, используемых в данной операционной среде. В настоящий момент в качестве физических моделей используются различные методы размещения данных, основанные на файловых структурах: это организация файлов

прямого и последовательного доступа, индексных файлов и инвертированных файлов, файлов, использующих различные методы хэширования, взаимосвязанных файлов. Кроме того, современные БД широко используют страничную организацию данных. Физические модели данных, основанные на страничной организации, являются наиболее перспективными.

Наибольший интерес вызывают модели данных, используемые на концептуальном уровне. По отношению к ним внешние модели называются подсхемами и используют те же абстрактные категории, что и концептуальные модели данных.

На рис.16 представлена классификация моделей данных.

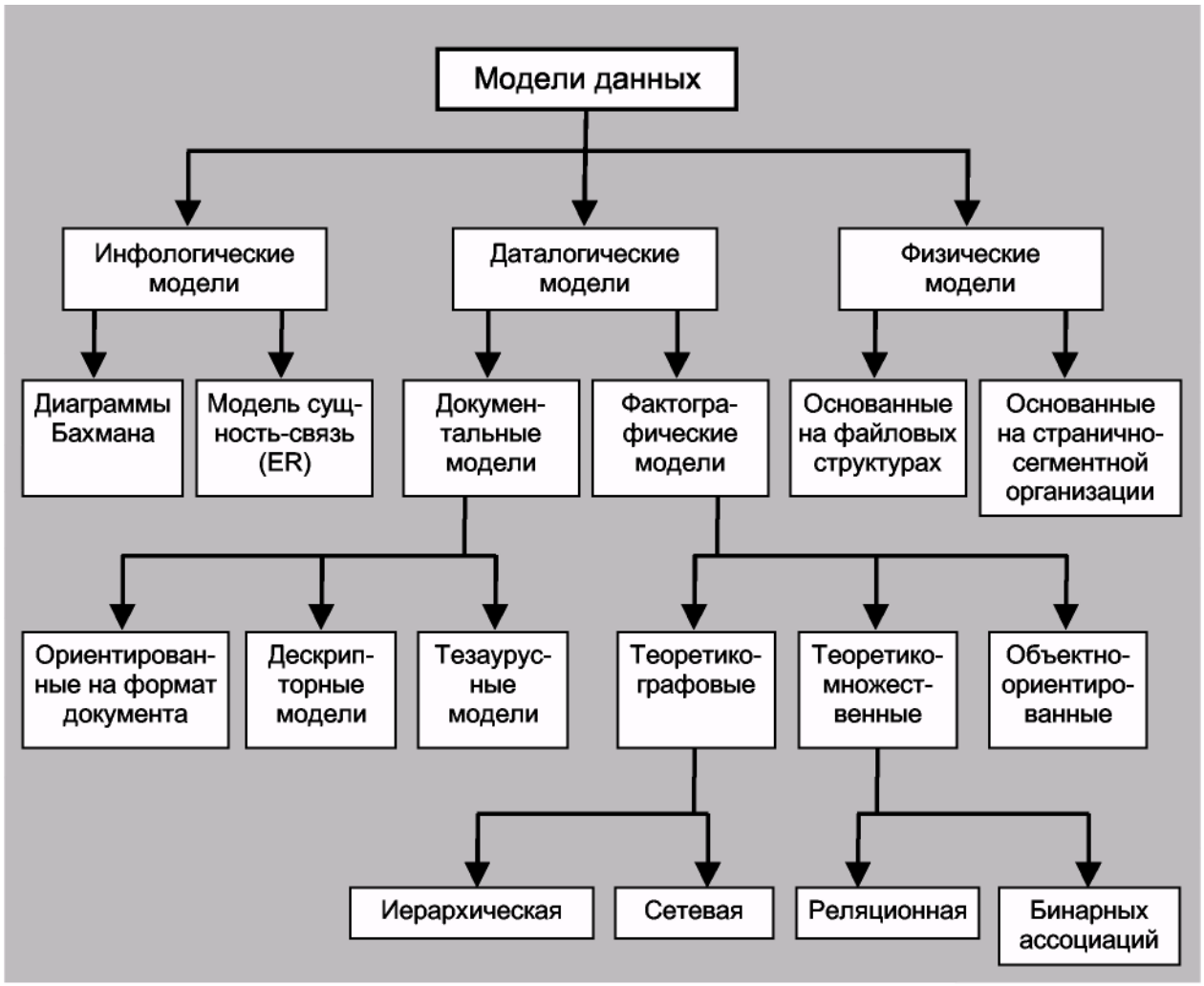


Рис. 16. ***Классификация моделей данных***

Кроме трех рассмотренных уровней абстракции, при проектировании БД существует еще один уровень, предшествующий им. Модель этого уровня должна выражать информацию о предметной области в виде, независимом от используемой СУБД. Эти модели называются *инфологическими,* или *семантическими,* и отражают в естественной и удобной для разработчиков и других пользователей форме информационно-логический уровень абстрагирования, связанный с фиксацией и описанием объектов предметной области, их свойств и их взаимосвязей.

Инфологические модели данных используются на ранних стадиях проектирования для описания структур данных в процессе разработки приложения, а ***даталогические модели*** уже поддерживаются конкретной СУБД. ***Физические модели*** описывают структуры и принципы их хранения во внешней памяти, а также доступа к ним в зависимости от используемых аппаратных средств и программного обеспечения низкого уровня.

Среди разновидностей инфологических моделей наибольшее распространение получили ***модели сущность-связь,*** подробное рассмотрение которых будет приведено позже.

Документальные модели *данных* соответствуют представлению о слабоструктурированной информации, ориентированной в основном на свободные форматы документов, текстов на естественном языке.

Модели, ориентированные на формат документа, основаны на языках разметки документов и связаны прежде всего со стандартным общим языком разметки - SGML (Standart Generalised Markup Language), который был утвержден ISO в качестве стандарта еще в 80-х годах. Этот язык предназначен для создания других языков разметки, он определяет допустимый набор тегов (ссылок), их атрибуты и внутреннюю структуру документа. Контроль за правильностью использования тегов осуществляется при помощи специального набора правил, называемых DTD-описаниями, которые используются программой клиента при разборе документа. Для каждого класса документов определяется свой набор правил, описывающих грамматику соответствующего языка разметки. С помощью SGML можно описывать структурированные данные, организовывать информацию, содержащуюся в документах, представлять эту информацию в некотором стандартизованном формате. Но ввиду некоторой своей сложности SGML использовался в основном для описания синтаксиса других языков (наиболее известным из которых является HTML), и немногие приложения работали с SGML-документами напрямую.

Гораздо более простой и удобный, чем SGML, язык HTML позволяет определять оформление элементов документа и имеет некий ограниченный набор инструкций — тегов, при помощи которых осуществляется процесс разметки. Инструкции HTML в первую очередь предназначены для управления процессом вывода содержимого документа на экране программы-клиента и определяют этим самым способ представления документа, но не его структуру. В качестве элемента гипертекстовой базы данных, описываемой HTML, используется текстовый файл, который может легко передаваться по сети с использованием протокола HTTP. Эта особенность, а также то, что HTML является открытым стандартом и огромное количество пользователей имеет возможность применять возможности этого языка для оформления своих документов, безусловно, повлияли на рост популярности HTML и сделали его сегодня главным механизмом представления информации в Интернете.

Однако HTML сегодня уже не удовлетворяет в полной мере требованиям, предъявляемым современными разработчиками к языкам подобного рода. И ему на смену был предложен новый язык гипертекстовой разметки, мощный, гибкий и, одновременно с этим, удобный язык XML. В чем же заключаются его достоинства?

XML (Extensible Markup Language) - это язык разметки, описывающий целый класс объектов данных, называемых XML-документами. Он используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов. То есть сам по себе XML не содержит никаких тегов, предназначенных для разметки, он просто определяет порядок их создания.

Тезаурусные модели основаны на принципе организации словарей, содержат определенные языковые конструкции и принципы их взаимодействия в заданной грамматике. Эти модели эффективно используются в системах-переводчиках, особенно многоязыковых переводчиках. Принцип хранения информации в этих системах и подчиняется тезаурусным моделям.

Дескрипторные модели — самые простые из документальных моделей, они широко использовались на ранних стадиях использования документальных *баз* данных. В этих моделях каждому документу соответствовал дескриптор — описатель. Этот дескриптор имел жесткую структуру и описывал документ в соответствии с теми характеристиками, которые требуются для работы с документами в разрабатываемой документальной БД. Например, для БД, содержащей описание патентов, дескриптор содержал название области, к которой относился патент, номер патента, дату выдачи патента и еще ряд ключевых параметров, которые заполнялись для каждого патента. Обработка информации в таких базах данных велась исключительно по дескрипторам, то есть по тем параметрам, которые характеризовали документ, а не по самому тексту документа.

Рассмотрим особенности наиболее распространенных моделей данных более подробно.

Теоретико-графовые модели данных

Эти модели отражают совокупность объектов реального мира в виде графа взаимосвязанных информационных объектов.

Граф - система, которая интуитивно может быть рассмотрена как множество кружков и множество соединяющих их линий (геометрический способ задания графа) (рис.17). Кружки называются вершинами графа, линии со стрелками - дугами, без стрелок - ребрами. Граф, в котором направление линий не выделяется (все линии являются ребрами), называется неориентированным; граф, в котором направление линий принципиально (линии являются дугами) называется ориентированным.

Язык графов оказывается удобным для описания многих физических, технических, экономических, биологических, социальных и других систем.

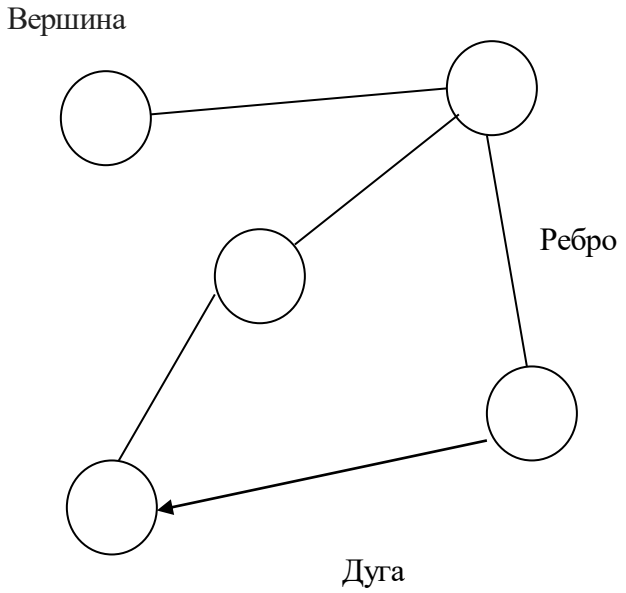


Рис. ***17. Графическое изображение графа***

К *теоретико-графовым моделям* относятся две разновидности:

- сетевые модели;
- иерархические модели.

В таких моделях данных предусматриваются характерные для подобного рода структур операции навигации и манипулирования данными. Принципиальное значение при этом имеет то обстоятельство, что предусматривается одновременная обработка только одиночных объектов данных из БД — записей, сегментов или полей записей. Интерактивный доступ к БД поддерживается только путем создания соответствующих прикладных программ с собственным интерфейсом. Пользовательские приложения этих систем, используя языки программирования, расширенные функциями СУБД, осуществляют явную навигацию в БД. В связи с этим системы, построенные на этих моделях данных, называют *навигационными.*

Аппарат навигации в ТГ-моделях служит для установки тех объектов данных, к которым будет применяться очередная операция манипулирования данными. Такие объекты называются *текущими.* Механизмы доступа к данным и навигации по структуре данных в таких моделях достаточно сложны, особенно в сетевой модели, и опираются на концепцию текущего состояния механизма доступа.

Иерархическая модель данных

В иерархической модели связи между данными можно описать с помощью упорядоченного графа (или дерева). Упрощенно представление связей между данными в иерархической модели показано на рис.18.

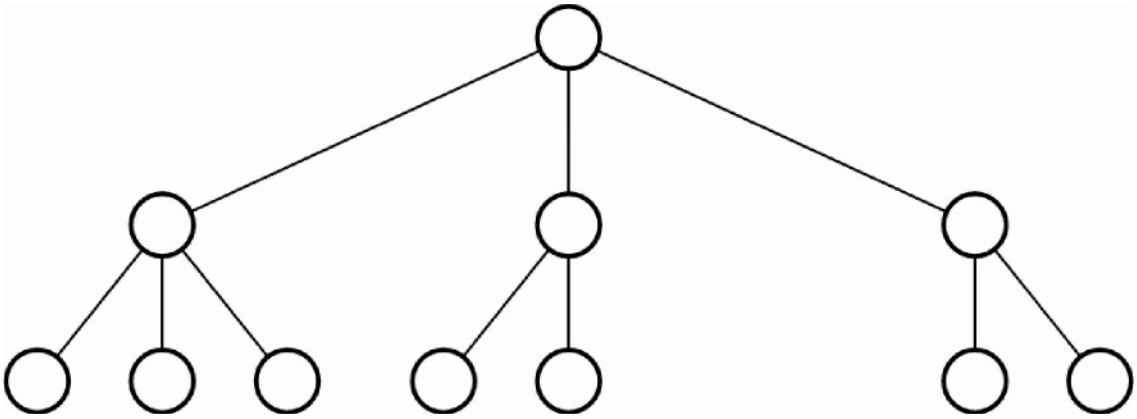


Рис. 18. ***Представление связей в иерархической модели***

Для описания структуры (схемы) иерархической БД на некотором языке программирования используется тип данных «дерево».

Основными информационными единицами в иерархической модели данных являются сегменты поля.

Поле данных – определяется как наименьшая неделимая единица данных доступная пользователю.

Для сегмента определяется тип и экземпляр сегмента. Экземпляр сегмента образуется из конкретных значений полей данных.

Тип сегмента – именованная совокупность входящих в него типов данных.

В иерархической модели данных вершине графа соответствует тип сегмента или просто сегмент, а дугам – типы связи (предок-потомок). В данных моделях сегмент, являющийся потомком должен иметь в точности одного предка.

Тип «дерево» является составным. Он включает в себя подтипы («поддеревья»), каждый из которых, в свою очередь, является типом «дерево». Каждый из типов «дерево» состоит из одного «корневого» типа и упорядоченного набора (возможно пустого) подчиненных типов. Каждый из элементарных типов, включенных в тип «дерево», является простым или составным типом «запись». Простая запись состоит из одного типа, например числового, а составная «запись» объединяет некоторую совокупность типов, например целое, строку символов и указатель (ссылку). Пример типа «дерево» показан на рис.19.

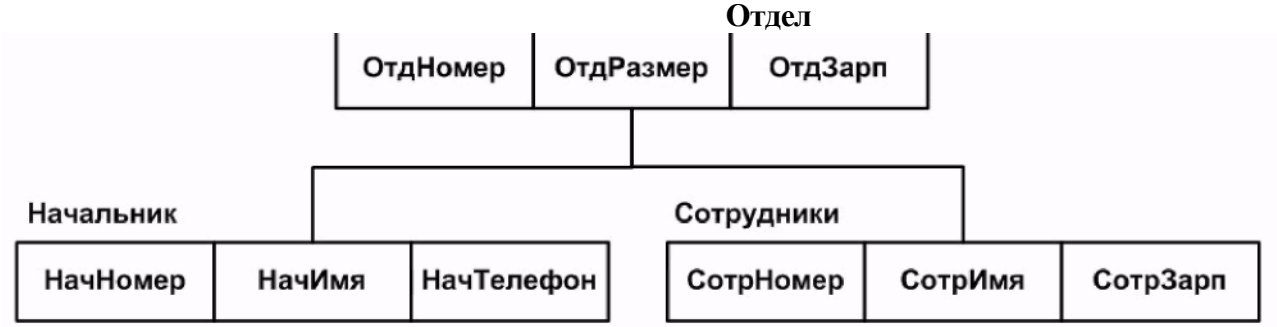


Рис. 19. ***Пример типа «дерево»***

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

Корневым называется тип, который имеет подчиненные типы и сам не является подтипом. **Подчиненный** тип (подтип) является потомком по отношению к типу, который выступает для него в роли предка (родителя). Потомки одного и того же типа являются близнецами по отношению друг к другу.

В целом тип «дерево» представляет собой иерархически организованный набор типов «запись».

Иерархическая БД представляет собой упорядоченную совокупность экземпляров данных типа «дерево» (деревьев), содержащих экземпляры типа «запись» (записи). Часто отношения родства между типами переносят на отношения между самими записями. Поля записей хранят собственно числовые или

символьные значения, составляющие основное содержание БД. Обход всех элементов иерархической БД обычно производится сверху вниз слева направо.

Данные в базе с приведенной схемой (рис.19) могут выглядеть например так, как показано на рис.20 (мы показываем один экземпляр дерева).

Рис. 20. ***Пример данных типа «дерево»***

Для организации физического размещения иерархических данных в памяти ЭВМ могут использоваться следующие группы методов:

- представление линейным списком с последовательным распределением памяти (адресная арифметика, левосписковые структуры);
- представление связными линейными списками (методы, использующие указатели и справочники).

В соответствии с определением типа «дерево», можно заключить, что между предками и потомками автоматически поддерживается контроль целостности связей. Основное правило контроля целостности звучит следующим образом: потомок не может существовать без родителя, а у некоторых родителей может не быть потомков. Механизмы поддержания целостности связей между записями различных деревьев отсутствуют.

К достоинствам иерархической модели данных относятся эффективное использование памяти ЭВМ и неплохие показатели времени выполнения основных операций над данными. Иерархическая модель удобна для работы с иерархически упорядоченной информацией.

Недостатком иерархической модели является

- частично дублируется информация между записями (такие записи называются парными). В иерархической модели не предусмотрена поддержка соответствия между парными записями
- иерархическая модель реализует отношения между исходной и дочерней записями по схеме 1:M, т.е. одной родительской записи может соответствовать любое число дочерних. Возникает связь «1 ко многим»
- ее громоздкость для обработки информации с достаточно сложными логическими связями, а также сложность понимания для обычного пользователя.

Сетевая модель данных

Сетевая модель данных позволяет отображать разнообразные взаимосвязи элементов данных в виде произвольного графа, обобщая тем самым иерархическую модель данных.

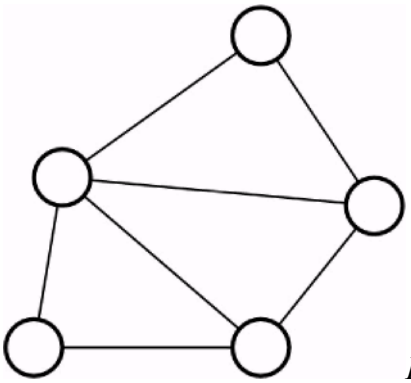


Рис. 21. ***Представление связей в сетевой модели***

Для описания схемы сетевой БД используется две группы типов: «запись» и «связь». Тип «связь» определяется для двух типов «запись»: предка и потомка. Переменные типа «связь» являются экземплярами связей.

Сетевая БД состоит из набора записей и набора соответствующих связей. На формирование связи особых ограничений не накладывается. Если в иерархических структурах запись-потомок могла иметь только одну запись-предка, то в сетевой модели данных запись-потомок может иметь произвольное число записей-предков.

Пример схемы простейшей сетевой БД показан на рис.22. Типы связей здесь обозначены надписями на соединяющих типы записей линиях.

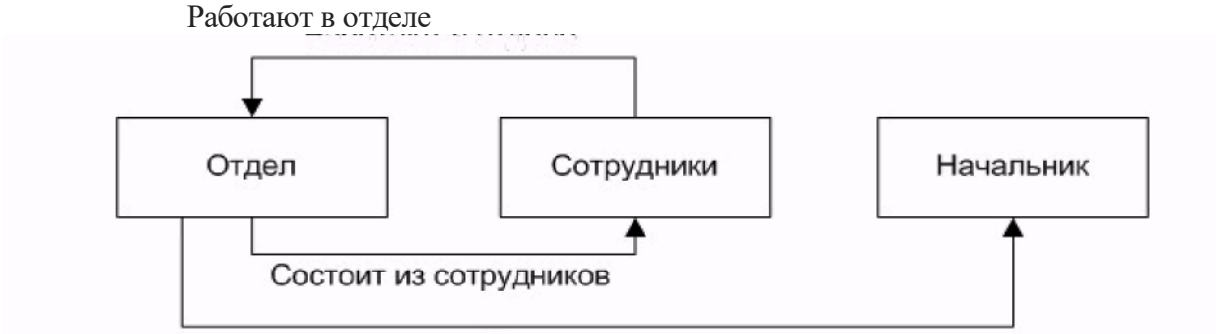


Рис. 22. ***Пример схемы сетевой БД***

В различных СУБД сетевого типа для обозначения одинаковых по сути понятий зачастую используются различные термины. Например, такие как элементы и агрегаты данных, записи, наборы, области и т.д.

Физическое размещение данных в базах сетевого типа может быть организовано практически теми же методами, что и в иерархических БД.

Достоинством сетевой модели данных является возможность эффективной реализации по показателям затрат памяти и оперативности. В сравнении с иерархической моделью сетевая модель предоставляет большие возможности в смысле допустимости образования произвольных связей.

Недостатком сетевой модели данных является высокая сложность и жесткость схемы БД, построенной на ее основе, а также сложность для понимания и выполнения обработки информации в БД обычным пользователем. Кроме того, в сетевой модели данных ослаблен контроль целостности связей вследствие допустимости установления произвольных связей между записями.

Теоретико-множественные модели данных

Реляционная модель данных

Реляционная модель данных предложена сотрудником фирмы IBM Эдгаром Коддом и основывается на понятии отношение (relation).

Реляционная модель данных (РМД) - это способ рассмотрения данных, при котором данные воспринимаются пользователем как взаимосвязанные таблицы и в распоряжении пользователя имеются некоторые операторы, которые генерируют новые таблицы из старых.

Под таблицами здесь понимается структура данных, состоящая из строк и столбцов. В этой структуре каждый столбец содержит данные только одного типа, каждая строка состоит из набора значений составляющих ее столбцов.

Под операторами понимаются операции выборки, группировки, соединения и некоторые другие, результатом которых являются новые таблицы, полученные на основании старых.

Реляционная модель данных (РМД) некоторой предметной области представляет собой набор отношений, изменяющихся во времени. При создании информационной системы совокупность отношений позволяет хранить данные об объектах предметной области и моделировать связи между ними.

Отношение представляет собой множество элементов, называемых **кортежами.**

Элементы РМД и формы их представления приведены в таблице.

Основные элементы реляционной модели данных и формы их представления

Элементы РМД	*Форма представления (описание)*
1. Отношение	Представляет собой двумерную таблицу, содержащую некоторые данные
2. Схема отношения	Строка заголовков столбцов таблицы (заголовок таблицы)
3. Кортеж	Строка таблицы, характеризующая отдельный объект
4. Сущность	Объект любой природы данные о котором хранятся в БД. Данные о сущности хранятся в одном или нескольких отношениях.
5. Атрибут	Представляет собой свойство характеризующее сущность (заголовок столбца таблицы)
6. Домен	Множество допустимых значений определенного атрибута отношений
7. Значение атрибута	Значение в поле записи
8. Первичный ключ	Один или несколько атрибутов отношения, однозначно определяющих каждый из кортежей
9. Потенциальный ключ	Один или несколько атрибутов отношения, однозначно характеризующий каждый из кортежей, но не выбранный в качестве первичного ключа
10. Внешний ключ	Если отношение R_1 содержит не ключевой атрибут А, значения которого являются значениями ключевого атрибута В другого отношения R_2, тогда атрибут А является внешним ключом
11. Тип данных	Тип значений элементов таблицы. Каждый домен образует значение одного типа данных. Например, числовые или символьные

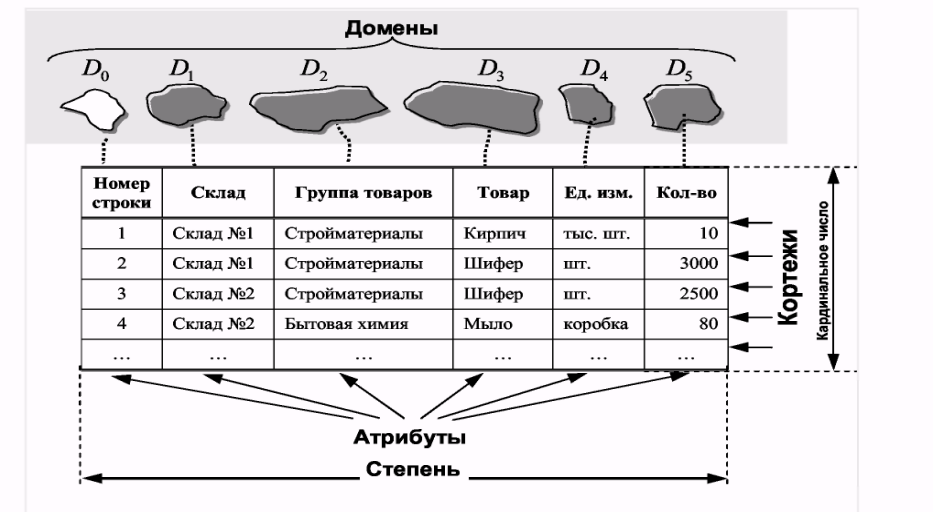


Рис. 23. ***Представление отношения «Остатки товаров на складах»***

Математически отношение можно записать следующим образом. Пусть даны n множеств *D1,D2,D3,...,Dn,* тогда отношение *R* есть множество упорядоченных кортежей < *d1,d2,d3,...,dn* >, где *dk € Dk, dk* - атрибут, а *Dk* -домен отношения *R*. На рис.23 приведен пример представления отношения «Остатки товаров на складах». В общем случае порядок кортежей в отношении, как и в любом множестве не определен. Однако в реляционных СУБД для удобства кортежи все же упорядочивают. Чаще всего для этого выбирают некоторый атрибут, по которому система автоматически сортирует кортежи по возрастанию или убыванию. Если пользователь не назначает атрибута упорядочения, система автоматически присваивает номер кортежам в порядке их ввода.

Ключи обычно используют для достижения следующих целей:

1. Исключения дублирования значений в ключевых атрибутах (остальные атрибуты в расчет не принимаются);
2. Упорядочения кортежей. Возможно упорядочение по возрастанию или убыванию значений всех ключевых атрибутов, а также смешанное упорядочение (по одним - возрастание, а по другим - убывание);
3. Ускорения работы с кортежами отношения;
4. Организации связанных таблиц.

Реляционная модель накладывает на внешние ключи ограничение для обеспечения целостности данных, называемое **ссылочной целостностью.** Это означает, что каждому значению внешнего ключа должны соответствовать строки в связываемых отношениях.

Поскольку не всякой таблице можно поставить в соответствие отношение, то приведем условия, выполнение которых позволяет считать таблицу отношением:

1. все строки таблицы должны быть уникальны, т.е. не может быть строк с одинаковыми первичными ключами;
2. имена столбцов таблицы должны быть различны, а значения их простыми, т.е. недопустима группа значений в одном столбце одной строки;
3. все строки одной таблицы должны иметь одну структуру, соответствующую именам и типам столбцов;
4. порядок размещения строк в таблице может быть произвольным.

В некоторых СУБД одна таблица размещается в отдельном файле. В некоторых СУБД одна таблица считается БД, в других — БД может содержать несколько таблиц.

В общем случае можно считать, что БД включает одну или несколько таблиц, объединенных смысловым содержанием, а также процедурами контроля целостности и обработки информации в интересах решения некоторой прикладной задачи.

К отношениям можно применять систему операций, позволяющую создавать новые таблицы на основании старых. Например, результатом запроса к реляционной БД может быть новое отношение, вычисленное на основе имеющихся отношений. Поэтому можно разделить обрабатываемые данные на хранимую и вычисляемую части.

Основной единицей обработки данных в РБД является отношение, а не отдельные его кортежи (записи).

Индексирование.

Определение ключа для таблицы означает автоматическую сортировку записей, контроль отсутствия повторений значений в ключевых полях записей и повышение скорости выполнения операций поиска в таблице. Для реализации этих функций в СУБД применяют *индексирование*.

Под **индексом** понимают средство *ускорения* операции поиска записей в таблице, а следовательно, и других операций, использующих поиск: извлечение, модификация, сортировка и т.д. таблицу для которой используется индекс, называют *индексированной*.

Индекс выполняет роль оглавления таблицы, просмотр которого предшествует обращению к записям таблицы. В некоторых системах, например, Paradox, индексы хранятся в индексных файлах, хранимых отдельно от табличных файлов.

В поле ключа индексного файла можно хранить значения ключевых полей индексируемой таблицы либо свертку ключа (так называемый хеш-код). *Преимущество* хранения хеш-кода вместо значения состоит в том, что длина свертки независимо от длины исходного значения ключевого поля всегда имеет некоторую постоянную и достаточно малую величину (например, 4 байта), что существенно снижает время поисковых операций. *Недостатком* хеширования является необходимость выполнения операции свертки (требует определенного времени), а также борьба с возникновением коллизий (свертка различных значений может дать одинаковый хеш-код).

На практике для создания индекса для некоторой таблицы БД пользователь указывает поле таблицы, которое требует индексации. Ключевые поля таблицы во многих СУБД как правило индексируются автоматически. Индексные файлы, создаваемые по ключевым полям таблицы, часто называются *файлами первичных индексов*.

Индексы создаваемые пользователем для не ключевых полей, иногда называются ***вторичными (пользовательскими) индексами***. Введение таких индексов не изменяет физического расположения записей таблицы, но влияет на последовательность просмотра записей. Индексные файлы, создаваемые для поддержания вторичных индексов таблицы, обычно называются *файлами вторичных индексов*.

Некоторыми СУБД, например Access, деление индексов на первичные и вторичные не производится. В этом случае используются автоматически создаваемые индексы и индексы, определяемые пользователем по любому из не ключевых полей.

Главная причина повышения скорости выполнения различных операций в индексированных таблицах состоит в том, что основная часть работы производится с небольшими индексными файлами, а не самими таблицами. Наибольший эффект повышения производительности работы с индексированными таблицами достигается для значительных по объёму таблиц.

Индексы могут изменяться перед выполнением запросов к БД, после выполнения запросов к БД, по специальным командам пользователя или программным вызовам приложений.

Связывание таблиц. Основные виды связей

При проектировании реальных БД информацию обычно размещают в нескольких таблицах. Таблицы при этом связаны семантикой информации. В реляционных СУБД для указания связей таблиц производят операцию из связывания.

Связывание таблиц позволяет повысить достоверность хранимой информации за счет того, что многие СУБД автоматически проводят контроль целостности вводимой в БД информации при связывании, кроме того, связи между таблицами облегчают доступ к данным за счет возможности обращения к произвольным полям связанных записей, что уменьшает количество явных обращений к таблицам данных.

Между двумя таблицами в общем случае могут устанавливаться следующие четыре основных вида связи:

- один к одному (1:1);
- один ко многим (1 :M);
- многие к одному (M: 1);
- многие ко многим (M:M);

Число сущностей, которые могут быть ассоциированы через набор связей, называются *степенью связи.*

Набор связей – это ассоциация между N сущностями, причём N всегда больше или равен 2, каждая из которых относится к некоторому набору сущностей. Если связь осуществляется между двумя сущностями, то связь называется *бинарной.*

Связь вида 1:1

Данный вид связи образуется в случае, когда все поля связи основной и дополнительной таблиц являются ключевыми. Поскольку значения в ключевых полях обеих таблиц не повторяются, обеспечивается взаимнооднозначное соответствие записей из этих таблиц. Связанные таблицы при этом являются равноправными.

Например, между таблицами «Сотрудник» и «Адреса сотрудников» могут существовать отношения один к одному при условии, что сотрудник может иметь не более одного адреса. Это может быть оправдано в случае, когда адрес сотрудника является необязательной информацией и может не вноситься в БД, однако необходимо предусмотреть возможность его ввода при наличии адреса.

На практике связи вида 1:1 используются достаточно редко, т.к. хранимую в двух таблицах информацию легко объединить в одну таблицу, которая занимает меньше места в памяти ЭВМ. Но существуют случаи, когда удобнее иметь не одну, а две таблицы, как например из-за соображений безопасности, если необходимо ограничить *доступ* к секретным данным.

Связь вида 1:М

Этот вид связи имеет место в случае, когда одной записи основной таблицы соответствует несколько записей вспомогательной таблицы. Это наиболее распространенный вид связей в РМД.

Пример. Предположим существуют следующие таблицы:

Поставщик

Наименование	Код поставщика
Поставщик 1	1
Поставщик 2	2
Поставщик 3	3

Товар

Код	Наименование товара	Код товара
1	Товар 1	1
2	Товар 2	2
1	Товар 3	3
3	Товар 4	4

Зависимым называется отношение, если каждую запись данного отношения можно полностью идентифицировать только при установлении связи с основной таблицей.

Связь вида М:1

Является разновидность связи 1:М. Так если в предыдущем примере рассматривать не отношение «Поставщик» - «Товар», а «Товар» - «Поставщик», получим связь вида М:1.

Связь вида М:М

Так, в приведенном выше примере, может сложиться ситуация, при которой один товар может поставляться несколькими поставщиками, а один поставщик может поставлять несколько наименований товаров. В этом случае между таблицами имеет место отношение вида М:М.

Реляционная модель данных эту связь не поддерживает. Поэтому при возникновении данного вида связи в этот вид связи вовлекается более чем 2 таблицы, т.к. это наиболее общий вид связи. Возникает в случае, если нескольким записям основной таблицы соответствует несколько записей дополнительной таблицы. В случае возникновения такой связи ни одна из таблиц не является подчиненной. Подобный вид связей образует иерархию (дерево) связей.

Ещё одной важной характеристикой связи является класс принадлежностей входящих в неё сущностей (*кардинальность связи*). Если сущность имеет обязательный класс принадлежностей, то этот факт обозначается указанием интервала числа возможных вхождений сущности в связь. В данном случае это одна или много записей. Необязательно класс принадлежностей обозначается числом 0, 1 или много.

Лекция 17.

Объектно-ориентированная модель данных (самостоятельно)

Направление объектно-ориентированных баз данных (ООБД) возникло сравнительно давно. Публикации появлялись уже в середине 1980-х. Однако наиболее активно это направление развивается в последние годы. С каждым годом увеличивается число публикаций и реализованных коммерческих и экспериментальных систем.

Возникновение направления ООБД определяется прежде всего потребностями практики: необходимостью разработки сложных информационных прикладных систем, для которых технология предшествующих систем БД не была вполне удовлетворительной.

Общие понятия объектно-ориентированного подхода и их преломление в ООБД

В наиболее общей и классической постановке объектно-ориентированный подход базируется на концепциях:

- объекта и идентификатора объекта;
- атрибутов и методов;
- классов;
- иерархии и наследования классов.

Любая сущность реального мира в объектно-ориентированных языках и системах моделируется в виде объекта. Любой объект при своем создании получает генерируемый системой уникальный идентификатор, который связан с объектом во все время его существования и не меняется при изменении состояния объекта.

Каждый объект имеет состояние и поведение. Состояние объекта - набор значений его атрибутов. Поведение объекта - набор методов (программный код), оперирующих над состоянием объекта. Значение атрибута объекта - это тоже некоторый объект или множество объектов. Состояние и поведение объекта инкапсулированы в объекте; взаимодействие между объектами производится на основе передачи сообщений и выполнении соответствующих методов.

Множество объектов с одним и тем же набором атрибутов и методов образует класс объектов. Объект должен принадлежать только одному классу (если не учитывать возможности наследования). Допускается наличие примитивных предопределенных классов, объекты-экземляры которых не имеют атрибутов: целые, строки и т.д. Класс, объекты которого могут служить значениями атрибута объектов другого класса, называется доменом этого атрибута.

Допускается порождение нового класса на основе уже существующего класса - наследование. В этом случае новый класс, называемый подклассом существующего класса (суперкласса) наследует все атрибуты и методы суперкласса. В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы.

Одной из более поздних идей объектно-ориентированного подхода является идея возможного переопределения атрибутов и методов суперкласса в подклассе (перегрузки методов). Эта возможность увеличивает гибкость, но порождает дополнительную проблему: при компиляции объектно-ориентированной программы могут быть неизвестны структура и программный код методов объекта, хотя его класс (в общем случае - суперкласс) известен. Для разрешения этой проблемы применяется так называемый метод позднего связывания, означающий, по сути дела, интерпретационный режим выполнения программы с распознаванием деталей реализации объекта во время выполнения посылки сообщения к нему.

Как видно, при таком наборе базовых понятий, если не принимать во внимание возможности наследования классов и соответствующие проблемы, объектно-ориентированный подход очень близок к подходу языков программирования с абстрактными (или произвольными) типами данных.

Видимо, наиболее важным новым качеством ООБД, которое позволяет достичь объектно-ориентированный подход, является поведенческий аспект объектов. В прикладных информационных системах, основывавшихся на БД с традиционной организацией (вплоть до тех, которые базировались на семантических моделях данных), существовал принципиальный разрыв между структурной и поведенческой частями. Структурная часть системы поддерживалась всем аппаратом БД, ее можно было моделировать, верифицировать и т.д., а поведенческая часть создавалась изолированно. В частности, отсутствовали формальный аппарат и системная поддержка совместного моделирования и гарантирования согласованности этих структурной (статической) и поведенческой (динамической) частей. В среде ООБД проектирование, разработка и сопровождение прикладной системы становится процессом, в котором интегрируются

структурный и поведенческий аспекты. Конечно, для этого нужны специальные языки, позволяющие определять объекты и создавать на их основе прикладную систему.

Объектно-ориентированные модели данных

Первой формализованной и общепризнанной моделью данных была реляционная модель Кодда. В этой модели, как и во всех следующих, выделялись три аспекта - структурный, целостный и манипуляционный. Структуры данных в реляционной модели основываются на плоских нормализованных отношениях, ограничения целостности выражаются с помощью средств логики первого порядка и, наконец, манипулирование данными осуществляется на основе реляционной алгебры или равносильного ей реляционного исчисления. Как отмечают многие исследователи, своим успехом реляционная модель данных во многом обязана тому, что опиралась на строгий математический аппарат теории множеств, отношений и логики первого порядка.

Основные трудности объектно-ориентированного моделирования данных проистекают из того, что такого развитого математического аппарата, на который могла бы опираться общая объектно-ориентированная модель данных, не существует.

Во-первых, следуя практике многих ООБД, предлагается выделить два уровня моделирования объектов: нижний (структурный) и верхний (поведенческий). База данных - это набор элементов данных, связанных отношениями
"входит в класс" или "является атрибутом". Таким образом, БД может рассматриваться как ориентированный граф. Важным моментом является поддержание наряду с понятием объекта понятия значения.

Важным аспектом является четкое разделение схемы БД и самой БД. В качестве первичных концепций схемного уровня ООБД выступают типы и классы. Отмечается, что во всех системах, использующих только одно понятие (либо тип, либо класс) это понятие неизбежно перегружено: тип предполагает наличие некоторого множества значений, определяемого структурой данных этого типа; класс также предполагает наличие множества объектов, но это множество определяется пользователем. Таким образом, типы и классы играют разную роль, и для строгости и недвусмысленности требуются одновременное поддержание обоих понятий.

Объектно-ориентированные СУБД

В настоящее время ведется очень много экспериментальных и производственных работ в области объектно-ориентированных СУБД. Больше всего университетских работ, которые в основном носят исследовательский характер. Но уже год назад отмечалось существование по меньшей мере тринадцати коммерчески доступных систем ООБД. Среди них O2 (французский консорциум Altair), ORION (американская компания MCC), GemStone (американская фирма Servio Logic) и Iris (Hewlett-Packard). К сожалению, по поводу многих коммерческих систем практически отсутствуют доступные публикации, но и имеющейся информации достаточно, чтобы охарактеризовать типовую организацию современной объектно-ориентированной СУБД.

Рассмотрим особенности организации двух систем - ORION и O2.

Проект ORION осуществлялся с 1985 по 1989 г. фирмой MCC под руководством известного еще по работам в проекте System R Бона Кима. Под названием ORION на самом деле скрывается семейство трех СУБД: ORION-1 - однопользовательская система; ORION-1SX, предназначенная для использования в качестве сервера в локальной сети рабочих станций; ORION-2 -полностью распределенная объектно-ориентированная СУБД. Реализация всех систем производилась с использованием языка Common Lisp на рабочих станциях (и их локальных сетях) Symbolics 3600 с ОС Genera 7.0 и SUN-3 в среде ОС UNIX. Описание реализации ORION-2 пока не опубликовано, поэтому мы рассмотрим только ORION-1 и ORION-1SX.

Проект O2 реализуется французской компанией Altair, образованной специально для целей проектирования и реализации объектно-ориентированной СУБД. Начало проекта датируется сентябрем 1986 г., и он был рассчитан на пять лет: три года на прототипирование и два года на разработку промышленного образца. Текущий прототип системы функционирует в режиме клиент/сервер в локальной сети рабочих станций SUN с соответствующим разделением функций между сервером и клиентами.

Даже приведенное краткое описание особенностей двух объектно-ориентированных СУБД показывает прагматичность современного подхода к организации таких систем. Их разработчики не стремятся к полному соблюдению чистоты объектно-ориентированного подхода

и применяют наиболее простые решения проблем, которые на самом деле еще не решены. Пока в сообществе разработчиков объектно-ориентированных систем БД не видно работы, которая могла бы сыграть в этом направлении решающую роль. Правда, и проблемы ООБД гораздо более сложны, чем решаемые в реляционных системах.

Лекция 18.

Постреляционная модель данных. (самостаятельно)

Классическая реляционная модель предполагает неделимость данных, хранящихся в полях записей таблиц. Это означает, что информация в таблице представляется в первой нормальной форме Существует ряд случаев, когда это ограничение мешает эффективной реализации приложений.

Постреляционная модель данных представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся в записях таблиц. Постреляционная модель данных допускает многозначные поля - поля, значения которых состоят из подзначений. Набор значений многозначных полей считается самостоятельной таблицей, встроенной в основную таблицу. На примере информации о накладных и товарах (рис. 1) для сравнения приведено представление одних и тех же данных с помощью реляционной (а) и постреляционной (б) моделей. Таблица **Накладные** содержит данные о номерах накладных (**Н_No**) и номерах покупателей (**П_No**). В таблице **Накладные.Товары** содержатся данные о каждой из накладных: номер накладной (**Н_No**), название товара (**Назв_Т**) и количество товара (**Кол-во_Т**). Таблица **Накладные** связана с таблицей **Накладные.Товары** по полю **Н_No.**

а) Накладные

Н_No	П_No
0373	8723
8374	8232
7364	8723

Накладные.Товары

Н_No	Назв_Т	Кол-во_Т
0373	Сыр	3
0373	Рыба	2
8374	Лимонад	1
8374	Сок	6
8374	Печенье	2
7364	Йогурт	1

б) Накладные

Н_No	П_No	Назв_Т	Кол-во_Т
0373	8723	Сыр	3
0373		Рыба	2
8374	8232	Лимонад	1
8374		Сок	6
8374		Печенье	2
7364	8723	Йогурт	1

Рис. 1. Структуры данных реляционной и постреляционной моделей

Как видно из рисунка, по сравнению с реляционной моделью в постреляционной модели данные хранятся более эффективно, а при обработке не требуется выполнять операцию соединения данных из двух таблиц.

Помимо обеспечения вложенности полей постреляционная модель поддерживает ассоциированные многозначные поля (множественные группы). Совокупность ассоциированных полей называется **ассоциацией**. При этом в строке первое значение одного столбца ассоциации соответствует первым значениям всех других столбцов ассоциации. Аналогичным образом связаны все вторые значения столбцов и т. д.

На длину полей и количество полей в записях таблицы не накладывается требование постоянства. Это означает, что структура данных и таблиц имеют большую гибкость.

Поскольку постреляционная модель допускает хранение в таблицах ненормализованных данных, возникает проблема обеспечения целостности и непротиворечивости данных. Эта проблема решается включением в СУБД механизмов, подобных хранимым процедурам в клиент-серверных системах.

Для описания функций контроля значений в полях имеется возможность создавать процедуры (коды конверсии и коды корреляции), автоматически вызываемые до или после обращения к данным. Коды корреляции выполняются сразу после чтения данных, перед их обработкой. Коды конверсии, наоборот, выполняются после обработки данных.

Достоинством постреляционной модели является возможность представления совокупности связанных реляционных таблиц одной постреляционной таблицей. Это обеспечивает высокую наглядность представления информации и повышение эффективности ее обработки.

Недостатком постреляционной модели является сложность решения проблемы обеспечения целостности и непротиворечивости хранимых данных.
Рассмотренная нами постреляционная модель данных поддерживается СУБД uniVers. К числу других СУБД, основанных на постреляционной модели данных, относятся также системы Bubba и Dasdb.

Многомерная модель (самостоятельно).

Многомерный подход к представлению данных в базе появился практически одновременно с реляционным, но реально работающих многомерных СУБД (МСУБД) до настоящего времени было очень мало. С середины 90-х годов интерес к ним стал приобретать массовый характер. Толчком послужила в 1993 году программная статья одного из основоположников реляционного подхода Э. Кодда. В ней сформулированы 12 основных требований к системам класса OLAP (Online Analytical Processing - оперативная аналитическая обработка), важнейшие из которых связаны с возможностями концептуального представления и обработки многомерных данных. Многомерные системы позволяют оперативно обрабатывать информацию для проведения анализа и принятия решения.
В развитии концепций ИС можно выделить следующие два направления:

- системы оперативной (транзакционной) обработки;
- системы аналитической обработки (системы поддержки принятия решений).

Реляционные СУБД предназначались для информационных систем оперативной обработки информации и в этой области были весьма эффективны. В системах аналитической обработки они показали себя несколько неповоротливыми и недостаточно гибкими. Более эффективными здесь оказываются многомерные СУБД (МСУБД).
Многомерные СУБД являются узкоспециализированными СУБД, предназначенными для интерактивной аналитической обработки информации.
Основные понятия, используемые в этих СУБД: агрегируемость, историчность и прогнозируемость данных.
Агрегируемость данных означает рассмотрение информации на различных уровнях ее обобщения. В информационных системах степень детальности представления информации для пользователя зависит от его уровня: аналитик, пользователь-оператор, управляющий, руководитель.
Историчность данных предполагает обеспечение высокого уровня статичности (неизменности) как самих данных, так и их взаимосвязей, а также обязательность привязки данных ко времени. Статичность данных позволяет использовать при их обработке специализированные методы загрузки, хранения, индексации и выборки.
Временная привязка данных необходима для частого выполнения запросов, имеющих значения времени и даты в составе выборки. Необходимость упорядочения данных по времени в процессе обработки и представления данных пользователю накладывает требования на механизмы хранения и доступа к информации. Так, для уменьшения времени обработки запросов желательно, чтобы данные всегда были отсортированы в том порядке, в котором они наиболее часто запрашиваются.
Прогнозируемость данных подразумевает задание функций прогнозирования и применение их к различным временным интервалам.
Многомерность модели данных означает не многомерность визуализации цифровых данных, а многомерное логическое представление структуры информации при описании и в операциях манипулирования данными.
По сравнению с реляционной моделью многомерная организация данных обладает более высокой наглядностью и информативностью. Для иллюстрации на рис. 2 приведены реляционное (а) и многомерное (б) представления одних и тех же данных об объемах продаж автомобилей.
Если речь идет о многомерной модели с мерностью больше двух, то не обязательно визуально информация представляется в виде многомерных объектов (трех-, четырех- и более мерных гиперкубов). Пользователю и в этих случаях более удобно иметь дело с двухмерными таблицами или графиками. Данные при этом представляют собой "срезы" из многомерного хранилища данных, выполненные с разной степенью детализации.

а)

Модель	Месяц	Объем
"Жигули"	июнь	12
"Жигули"	июль	24
"Жигули"	август	5
"Москвич"	июнь	2
"Москвич"	июль	18
"Волга"	июль	19

б)

Модель	Июнь	Июль	Август
"Жигули"	12	24	5
"Москвич"	2	18	N
"Волга"	N	19	N

Рис. 2. Реляционное и многомерное представление данных

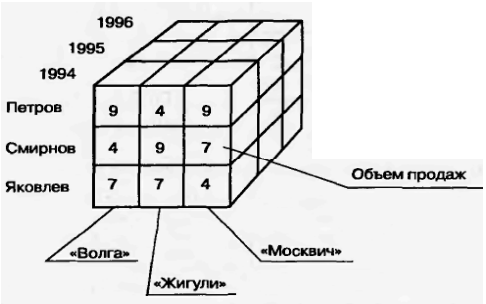
Рассмотрим основные понятия многомерных моделей данных, к числу которых относятся измерение и ячейка.

Измерение (Dimension) - это множество однотипных данных, образующих одну из граней гиперкуба. Примерами наиболее часто используемых временных измерений являются Дни, Месяцы, Кварталы и Годы. В качестве географических измерений широко употребляются Города, Районы, Регионы и Страны. В многомерной модели данных измерения играют роль индексов, служащих для идентификации конкретных значений в ячейках гиперкуба.

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

В примере на рис. 2(б) каждое значение ячейки Объем продаж однозначно определяется комбинацией временного измерения (Месяц продаж) и модели автомобиля. На практике зачастую требуется большее количество измерений. Пример трехмерной модели данных приведен на рис. 3.

В существующих МСУБД используются два основных варианта (схемы) организации данных: гиперкубическая и поликубическая.



Измерения:
Время (год)—1994, 1995, 1996
Менеджер — Петров, Смирнов, Яковлев
Модель — «Волга», «Жигули», «Москвич»
Показатель: Объем продаж

Рис.3. Пример трехмерной модели

В поликубической схеме предполагается, что в БД может быть определено несколько гиперкубов с различной размерностью и с различными измерениями в качестве граней. Примером системы, поддерживающей поликубический вариант БД, является сервер Oracle Express Server.

В случае гиперкубической схемы предполагается, что все показатели определяются одним и тем же набором измерений. Это означает, что при наличии нескольких гиперкубов БД все они имеют одинаковую размерность и совпадающие измерения. Очевидно, в некоторых случаях информация в БД может быть избыточной (если требовать обязательное заполнение ячеек).

В случае многомерной модели данных применяется ряд специальных операций, к которым относятся: формирование "среза", "вращение", агрегация и детализация.

"Срез" (Slice) представляет собой подмножество гиперкуба, полученное в результате фиксации одного или нескольких измерений. Формирование "срезов" выполняется для ограничения используемых пользователем значений, так как все значения гиперкуба практически никогда одновременно не используются. Например, если ограничить значения измерения Модель автомобиля в гиперкубе (рис. 3) маркой "Жигули", то получится двухмерная таблица продаж этой марки автомобиля различными менеджерами по годам.

Операция "вращение" (Rotate) применяется при двухмерном представлении данных. Суть ее заключается в изменении порядка измерений при визуальном представлении данных. Так, "вращение" двумерной таблицы, показанной на рис. 2(б), приведет к изменению ее вида таким образом, что по оси X будет марка автомобиля, а по оси Y - время.
Операцию "вращение" можно обобщить и на многомерный случай, если под ней понимать процедуру изменения порядка следования измерений. В простейшем случае, например, это может быть взаимная перестановка двух произвольных измерений.
Операции "агрегация" (Drill Up) и "детализация" (Drill Down) означают соответственно переход к более общему и к более детальному представлению информации пользователю из гиперкуба.
Для иллюстрации смысла операции "агрегация" предположим, что у нас имеется гиперкуб, в котором помимо измерений гиперкуба, приведенного на рис. 3, имеются еще измерения: Подразделение, Регион, Фирма, Страна. Заметим, что в этом случае в гиперкубе существует иерархия (снизу вверх) отношений между измерениями: Менеджер, Подразделение, Регион, Фирма, Страна.
Пусть в описанном гиперкубе определено, насколько успешно в 1995 году менеджер Петров продавал автомобили "Жигули" и "Волга". Тогда, поднимаясь на уровень выше по иерархии, с помощью операции "агрегация" можно выяснить, как выглядит соотношение продаж этих же моделей на уровне подразделения, где работает Петров.
Основным достоинством многомерной модели данных является удобство и эффективность аналитической обработки больших объемов данных, связанных со временем. При организации обработки аналогичных данных на основе реляционной модели происходит нелинейный рост трудоемкости операций в зависимости от размерности БД и существенное увеличение затрат оперативной памяти на индексацию.
Недостатком многомерной модели данных является ее громоздкость для простейших задач обычной оперативной обработки информации.
Примерами систем, поддерживающими многомерные модели данных, являются Essbase (Arbor Software), Media Multi-matrix (Speedware), Oracle Express Server (Oracle) и Cache (InterSystems). Некоторые программные продукты, например Media/ MR (Speedware), позволяют одновременно работать с многомерными и с реляционными БД. В СУБД Cache, в которой внутренней моделью данных является многомерная модель, реализованы три способа доступа к данным: прямой (на уровне узлов многомерных массивов), объектный и реляционный.

Реляционная алгебра.

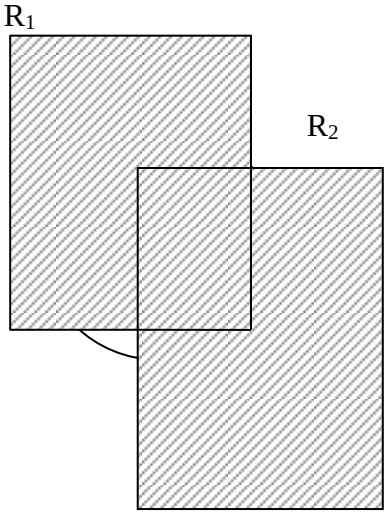
Реляционная алгебра – это множество отношений с замкнутым на нём множеством операций над отношениями.

Основным множеством в реляционной алгебре является множество отношений. Всё множество операций над отношениями можно разделить на 2 группы:

1. *Базовые теоретико-множественные операции* (объединение, пересечение, разность, расширенное декартово произведение).
2. *Специальные реляционные операции* (горизонтальный выбор – фильтрация, вертикальный выбор – проекция, условное соединение, деление).

Теоретико-множественные операции.

1) *Объединением* двух отношений называется отношение, содержащее множество кортежей, принадлежащих либо первому, либо второму, либо обоим отношениям одновременно.



$R_1=\{r_1\}$
$R_2=\{r_2\}$

$R_3= R_1 U R_2=\{r \backslash r \in R_1 \vee r \in R_2\}$

Пример. Имеются 2 отношения R_1 и R_2, которые содержат перечни деталей, изготовленных на разных участках одного цеха. Нужно получить полный перечень деталей, изготавливаемых в данном цехе.

R_1

Шифр	Наименование
11073	Гайка M_1
11075	Гайка M_2
11076	Гайка M_3

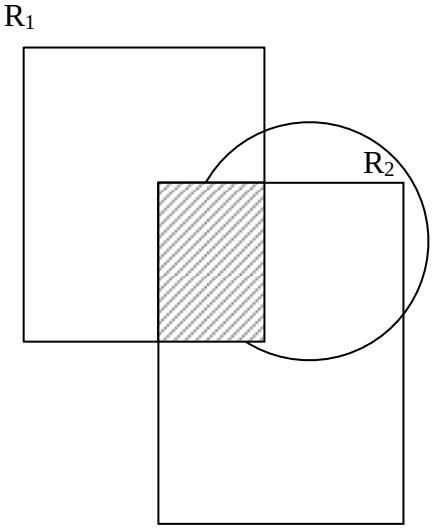
R_2

Шифр	Наименование
11073	Гайка M_1
11076	Гайка M_3
11077	Гайка M_4
11006	Болт M_3

$R_3= R_1 U R_2$

Шифр	Наименование
11073	Гайка M_1
11075	Гайка M_2
11076	Гайка M_3
11077	Гайка M_4
11006	Болт M_3

2) *Пересечением* отношений называется отношение, которое содержит множество кортежей, принадлежащих одновременно и первому и второму отношениям.



$R_4=R_1 \cap R_2=\{r \backslash r \in R_1 \vee r \in R_2\}$

$R_4=R_1 \cap R_2$

Шифр	Наименование
11073	Гайка M_1
11076	Гайка M_3

3) Разностью двух отношений называется отношение, содержащее кортежи, которые принадлежат первому отношению и не принадлежат второму. Операция разности не является коммутативной, т.е. результат этой операции зависит от порядка следования аргумента.

$R_5=R_1/R_2=\{r \backslash r \in R_1 \vee r \notin R_2\}$

$R_5=R_1/R_2$

Шифр	Наименование
11075	Гайка M_2

4) *Расширенное декартово произведение* – это сцепление реальных множеств, т.е. множество упорядоченных пар кортежей.

Сцеплением (конкатенацией) *кортежей* называется кортеж, полученный добавлением значений второго в конце первого.

$c=<c_1, c_2, \dots, c_n>$, где n – число элементов в первом кортеже.

$q=<q_1, q_2, \dots, q_m>$, где m – число элементов во втором кортеже.

Тогда сцеплением $(c,q) = < c_1, c_2, \dots, c_n, q_1, q_2, \dots, q_m >$

Расширенным декартовым произведением отношения R_1 степени n со схемой отношения $S_{R_1}=(A_1, A_2, A_3, \dots, A_n)$ и отношения R_2 степени m со схемой отношения $S_{R_2}=(B_1, B_2, B_3, \dots, B_m)$ называется отношение R_3 степени n+m со схемой отношений $S_{R_3}=(A_1, A_2, A_3, \dots, A_n, B_1, B_2, B_3, \dots, B_m)$, содержащее кортежи, полученные сцеплением кортежа c с каждым кортежем q отношения R_2.

$R_1=\{c\}$ $R_2=\{q\}$

$R_3=R_1 \otimes R_2= \{(c,q) / c \in R_1, q \in R_2\}$

Отношение декартова произведения используется для получения универсального отношения. Универсальное отношение характеризует все возможные комбинации между элементами отдельных множеств.

Пример: Пусть существует отношение R_6, содержащее обязательную номенклатуру деталей для все цехов и отношение R_7, содержащее перечень всех цехов. Требуется получить отношение, в котором была бы отражена ситуация, когда каждый цех изготавливает все эти детали.

R_6

Шифр	Наименование
11073	Гайка M_1
11076	Гайка M_3
11006	Болт M_3

R_7

Цех
Цех 1
Цех 2
Цех 3

Тогда отношение $R_8=R_6 \otimes R_7$

Шифр детали	Наименование	Цех
11073	Гайка M_1	Цех 1
11076	Гайка M_3	Цех 1
11006	Болт M_3	Цех 1
11073	Гайка M_1	Цех 2
11076	Гайка M_3	Цех 2
11006	Болт M_3	Цех 2
11073	Гайка M_1	Цех 3
11076	Гайка M_3	Цех 3
11006	Болт M_3	Цех 3

Универсальное умножение не имеет самостоятельного значения, но может использоваться в операциях над данными (нужно использовать во всех ситуациях, которые характеризуются словом «все»).

Специальные операции.

I. *Горизонтальный выбор (фильтрация)*. Представляет собой набор тех кортежей исходного отношения, который удовлетворяет некоторому условию.

Сравнение – элементарное логическое выражение, состоящее из сравниваемых выражений левой и правой части, разделённых операцией сравнения.

Пусть α – логическое выражение, состоящее из термов сравнения с помощью связок ∩ (объединение), U (пересечение), — (разность). В качестве термов сравнения допускаются следующие варианты:

1. A ос a, где A – выражение принадлежащее значению из домена D

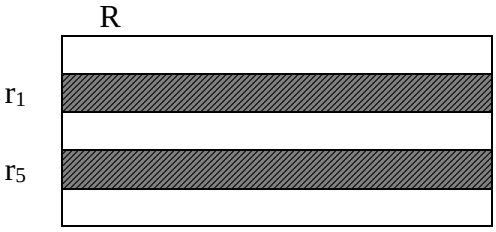
   a – константа или выражение, областью значения которого является домен

   ос – одна из допустимых для домена D операция сравнения.

2. A ос B, где A и B – имена сравниваемых атрибутов, которые также принадлежат значениям из одного домена.

Тогда результатом выбора или фильтрации, заданной на отношении R в виде выражения α, называется отношение R[α], включающее только те кортежи из исходного отношения, для которого истинно условие выбора или фильтрации.

$R[\alpha(r)]=\{r/r \in R \ V \ \alpha(r) = «истина»\}$



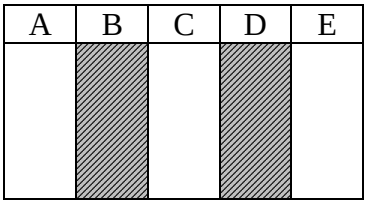
Пример: Необходимо выбрать из отношения R_8 детали с шифром 11073.

$R_9 = R_8$ [шифр детали = «11073»]

II. Вертикальный выбор – проекция.

Проекцией отношения R на набор атрибутов B называется отношение со схемой $S_{R_{(B)}}$=B, содержащее кортежи, получаемые из исходного отношения R путём удаления из них значений, не принадлежащих атрибутам из набора B. Данная операция позволяет получить только требуемые характеристики объекта.

R



Пример: Необходимо выбрать из номенклатуры изготовления деталей все цеха, которые изготавливают деталь Гайка M_1.

R_{10}

Шифр детали	Наименование	Цех
11073	Гайка M_1	Цех 1
11075	Гайка M_2	Цех 1
11003	Болт M_1	Цех 1
11006	Болт M_3	Цех 1
11073	Гайка M_1	Цех 2
11075	Гайка M_2	Цех 2
11076	Гайка M_3	Цех 3
11003	Болт M_1	Цех 3
11006	Болт M_3	Цех 3

Тогда $R_{11} = (R_{10} \bullet$[название = «Гайка M_1»]$) \bullet$ [цех]

Цех
Цех 1
Цех 2

III. Операция условного соединения

Операция условного соединения представляет собой получение нового отношения на основании двух исходных, соединённых друг с другом на основании какого-либо условия.

Пример: предположим, что имеется некоторое отношение R_{12}, которое содержит перечень деталей с указанием материалов, из которых они изготавливаются, и рассмотренное отношение R_{10}, содержащее список деталей, которые фактически производятся в цехах. Требуется определить перечень деталей, которые изготавливаются в Цехе 1 из материала Сталь 1.

R_{12}

Шифр детали	Наименование	Материал
11073	Гайка M_1	Сталь 1
11075	Гайка M_2	Сталь 2
11076	Гайка M_3	Сталь 1
11003	Болт M_1	Сталь 3
11006	Болт M_3	Сталь 3

Тогда $R_{13} = R_{12}$ [R_{12}• Шифр = R_{10}• Шифр детали] R_{10}

Шифр детали	Наименование	Материал	Цех
11073	Гайка M_1	Сталь 1	Цех 1
11073	Гайка M_1	Сталь 1	Цех 2
11075	Гайка M_2	Сталь 2	Цех 1
11075	Гайка M_2	Сталь 2	Цех 2
11076	Гайка M_3	Сталь 1	Цех 3
11003	Болт M_1	Сталь 3	Цех 1
11003	Болт M_1	Сталь 3	Цех 3
11006	Болт M_3	Сталь 3	Цех 1
11006	Болт M_3	Сталь 3	Цех 3

$R_{14} = R_{13}$ [(R_{13}• Наименование = «Сталь 1») ∩ (R_{13}• Цех = «цех1»)]

Шифр детали	Наименование	Материал	Цех
11073	Гайка M_1	Сталь 1	Цех 1

$R_{15} = R_{14}$ [Цех]

Наименование
Гайка M_1

IV. Деление

Это бинарная несимметричная операция, причём степень результирующего отношения не совпадает со степенью ни одного из операторов, а вычисляется как разность между степенью отношения делимого и степенью отношения делителя.

Пусть отношение R, называется делимым, содержащим атрибуты A_1, A_2,…A_n и отношение S, называется делителем и содержит подмножество атрибутов A, при этом $k<n$. (k- степень делителя, а n-степень делимого). Результирующее отношение C определено на атрибутах отношения R, которых нет в S, т.е. (A_{k+1}, A_{k+2}, …, A_n), кортежи включаются в результирующее отношение C только тогда, когда его декартово произведение с отношением S содержится в делимом R.

Пример: Даны отношения, содержащие сведения об экзаменах, которые должны были сдавать студенты (S) и сведения о результатах сдачи экзаменов (R). Необходимо определить сведения о студентах, сдавших все экзамены, определенные в отношении S.

R

№ зачёта	Фамилия	Дисциплина	Дата
02-Э-01	Иванов	Физика	10.01.09
02-Э-01	Иванов	Химия	14.01.09
02-Э-02	Сидоров	Физика	10.01.09
02-Э-02	Сидоров	Химия	14.01.09
02-Э-05	Петров	Физика	10.01.09

S

Дисциплина	Дата
Химия	14.01.09
Физика	10.01.09

C

№ зачёта	Фамилия
02-Э-01	Иванов
02-Э-02	Сидоров

Дополнительные операции реляционной алгебры.

К дополнительным операциям реляционной алгебры относятся: группировка, удаление, изменение, переименование. Наиболее часто используется операция группировка.

Группировка представляет собой такое обобщение исходного отношения, при котором происходит объединение кортежей значениями атрибутов группировки, а по другим атрибутам рассчитывается какое-либо арифметическое отношение, позволяющее оценить значение группы кортежей.

Пример: Пусть содержится отношение R, в котором отражена информация о проданных товарах. Необходимо получить список покупателей.

R

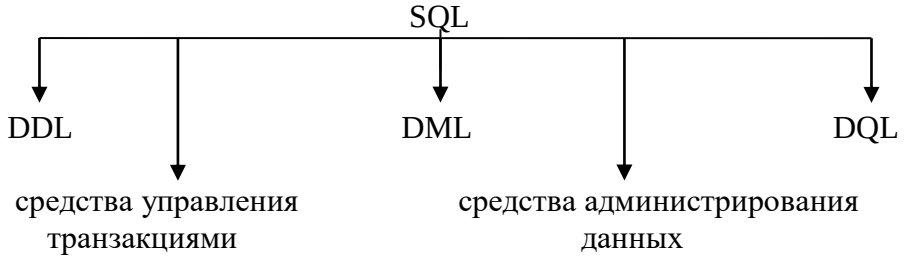
Дата	Покупатель	Наименование товара	Кол-во	Цена
04.05.09	ООО «Орион»	Товар 1	5000	7500
04.05.09	ООО «Орион»	Товар 2	12000	6000
04.05.09	ООО «Орион»	Товар 3	2000	20000
05.05.09	ООО «Дельта»	Товар 3	30000	200000
09.05.09	ООО «Пингвин»	Товар 1	1600	3000
09.05.09	ООО «Пингвин»	Товар 2	5000	4000

R_1=Группировка по атрибуту «Покупатель» (R)

R_1

Покупатель
ООО «Орион»
ООО «Дельта»
ООО «Пингвин»

Структура языка SQL.



DDL (язык описания данных)

Основная задача это – единое описание данных и их структур вне зависимости от выбранной СУБД. Язык DDL вводит единые типы данных и единые языковые конструкции (операторы языка) для создания любых структур данных.

Тип данных – это способ кодирования информации в двоичную форму и способ декодирования информации в форму понятую человеку.

Т.е. тип данных – это множество операций над информацией. Можно выделить 3 основных типа данных: строковый, числовой и дата/время.

Типы данных определенные в стандарте SQL2.

Тип данных	Описание
CHAR (до 8000)	Строки символов постоянной длины
VARCHAR (длина)	Строки символов переменной длины, до 8000 символов
NCHAR (длина)	Строки локализованных символов постоянной длины. До 4000 символов Unicode.
NCHAR VARYING (длина)	Строки локализованных символов переменной длины. До 4000 символов Unicode.
INTEGER или INT	[-2 000 000 … 2 000 000]. Целые числа.
SMALLINT	Малые целые числа. [≈-32 000 … 32 000].
BIT (длина)	Цепочки битов постоянной длины.
BIT VARYING (длина)	Цепочки битов переменной длины.
NUMERIG (точность, степень)	Вещественное число, где точность – количество разрядов целой части, а степень – количество разрядов дробной части.
DECIMAL	Вещественное число, где точность – количество разрядов целой части, а степень – количество разрядов дробной части.
FLOAT (точность)	Числа большой точности, хранимые в форме с плавающей точкой [-3.40 E+308; +3.40 E+38]
DOUBLE PRECISION	Вещественное число высокой точности [-1.79 E+308; 1.79 E+308]
DATE	Дата
TIME (точность)	Время
TIME STAMP (точность)	Дата и время
INTERVAL	Временной интервал

Основные операторы языка DDL.

Оператор	Назначение	Действие
CREATE TABLE	Создать таблицу	Создаёт новую таблицу в БД
DROP TABLE	Удалить таблицу	Удаляет таблицу из БД
ALTER TABLE	Изменить таблицу	Изменяет структуру существующей таблицы или ограничивает целостность, задаваемое для данной таблицы

CREATE VIEW	Создать представление	Создаёт виртуальную таблицу, соответствующую некоторому SQL – запросу.
ALTER VIEW	Изменить представление	Изменяет ранее созданное представление
DROP VIEW	Удалить представление	Удаляет созданное представление
CREATE INDEX	Создать индекс по определенному полю	Создает индекс для некоторого атрибута таблицы, для обеспечения быстрого доступа по атрибутам, входящим в индекс.
DROP INDEX	Удалить индекс	Удаляет созданный индекс

Оператор определения таблицы может иметь следующий вид:

```
CREATE TABLE <имя таблицы> (<список полей>);
(<имя поля 1> <тип данных 1> <ключевые слова-признаки>,
 <имя поля 2> <тип данных 2> <ключевые слова-признаки>,
…
);
```

Пример1:

```
CREATE TABLE Продажа
        (Дата операции DATE TIME NOT NULL,
        Номер сделки INTEGER NOT NULL,
        Наименование клиента INTEGER NOT NULL,
        Наименование товара INTEGER NOT NULL,
        Количество NUMERIC (6,2),
        Цена NUMERIC (6,2),
        Сумма NUMERIC (10,2),
        PRIMARY KEY (Номер сделки)
        );
CREATE TABLE Товар
        (Код товара INTEGER NOT NULL,
        Наименование VARCHAR (50) NOT NULL,
        PRIMARY KEY (Код товара)
        );
CREATE TABLE Клиент
        (Код клиента INTEGER NOT NULL,
        Наименование VARCHAR (50) NOT NULL,
        Адрес        VARCHAR (100),
        PRIMARY KEY (Код клиента)
        );
ALTER TABLE Продажа
    ADD CONSTRAINT Состоит_из
        FOREING KEY (Наименование товара)
        REFERENCES Товар (Код товара)
        ON DELETE CASCADE,
        ON UPDATE CASCADE
        );
```

Оператор создания индекса имеет формат вида:

CREATE INDEX <Наименование индекса> ON <наименование таблицы> (<список полей для индексирования>);
Пример 2:

CREATE INDEX Индекс Дата Операции ON Продажа (Дата операции);
DROP TABLE Продажа;

Операторы манипулирования данными:

Операторы	**Пояснение**
DELETE	Удаляет одну или несколько строк, соответствующих условиям фильтрации, из базовой таблицы
INSERT	Вставляет одну строку в базовую таблицу
UPDATE	Обновляет значения одного или нескольких столбцов в одной или нескольких строках, соответствующих условиям фильтрации

Язык запросов DQL.

Оператор	**Пояснение**
SELECT	Выбирает строки; оператор, позволяющий сформировать результирующую таблицу, соответствующую запросу

Назначение этого оператора состоит в выборке и отображении данных одной или нескольких таблиц БД. Он реализует все операции реляционной алгебры. Один и тот же запрос может быть реализован несколькими способами, которые могут существенно отличаться по времени исполнения, что важно для объёмных БД.

Синтаксис оператора SELECT:

SELECT [DISTINCT /ALL] {*/ [<список полей>]} FROM <список таблиц>
[WHERE <условие выборки или соединение>]
[GROUP BY <список полей группировки>]
[HAVING <условие для групп>]
[ORDER BY <список полей сортировки> ASC/DESC]

Распределенная обработка данных.

Свойства идеальной системы управления распределенными базами данных:

1) Прозрачность относительно расположения данных СУБД должна представлять все данные так, как если бы они были локальными.
2) Гетерогенность системы: СУБД должна работать с данными, которые хранятся с различной архитектурой и производительностью.
3) Прозрачность относительно сети: система управления БД должна одинаково работать в условиях разнородных сетей.
4) Поддержка распределенных запросов: пользователь должен иметь возможность объединять данные из любых баз.
5) Поддержка распределенных изменений: возможность изменять данные в любых базах, на доступ к которым есть права, даже если эти базы размещены в разных системах.
6) Безопасность.
7) Универсальность доступа: СУБД должна обеспечивать единую методику ко всем данными точками.

Режимы работы с базой данных.

1) Многозадачность – однопользовательский или многопользовательский.

2) Правила обслуживания запросов – последовательное или параллельное.
3) Схема размещения данных – централизованные или распределенные БД.

Извлечение данных. Команда SELECT.

Основное назначение раздела SELECT – задание набора столбцов, возвращаемых после выполнения запроса, т.е. внешнего вида результата. С помощью раздела SELECT можно ограничить количество строк, которое будет включено в результат выборки. Синтаксис раздела SELECT следующий:

SELECT [ALL/DISTINCT]

[TOPn [PERCENT][WITH TIES]]

< Список_выбора >

Ключевые слова: ALL и DISTINCT

При указании ключевого слова ALL в результат запроса выводятся все строки, удовлетворяющие сформулированным условиям, тем самым разрешается включение в результат одинаковых строк. Параметр ALL используется по умолчанию. Если в запросе SELECT указывается ключевое слово DISTINCT, то в результат выборки не будет включаться более одной повторяющейся строки. Т.е. каждая возвращающаяся строка будет уникальной.

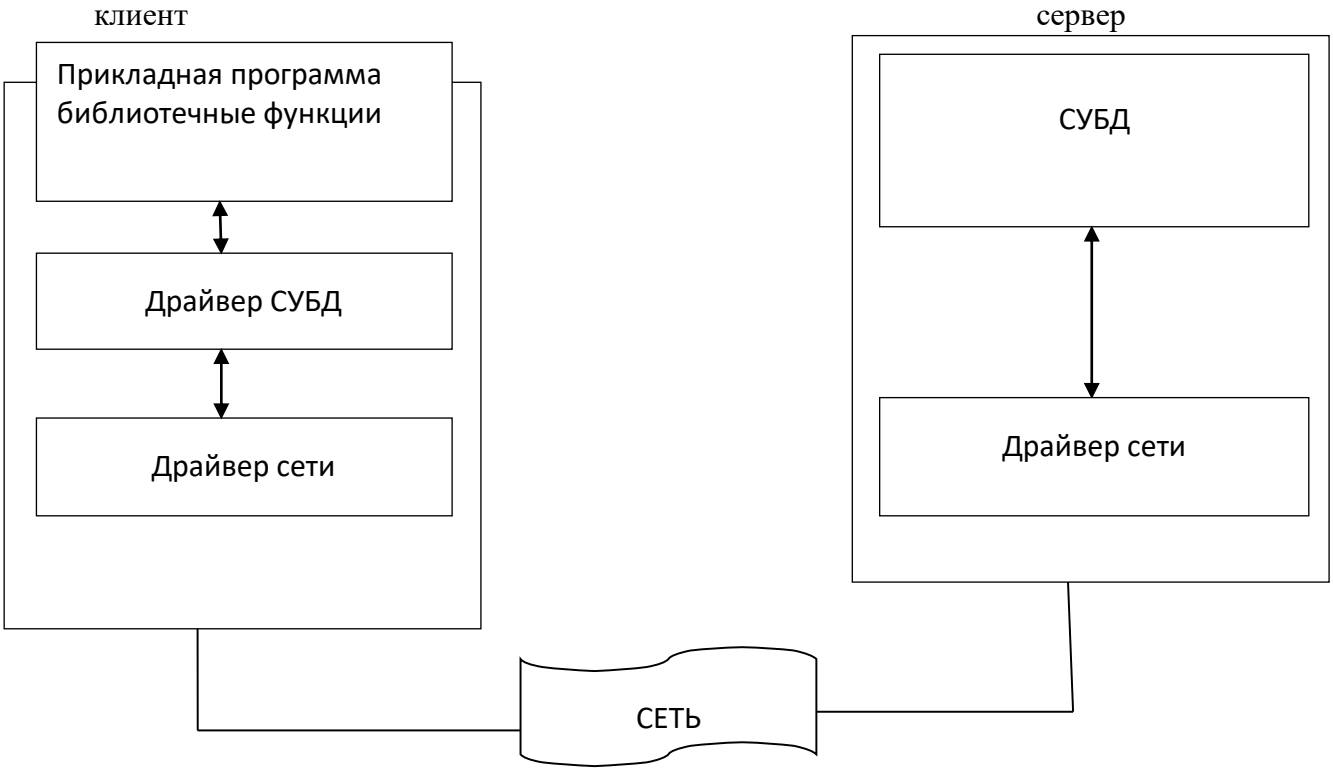
Ключевое слово TOPn.

Использование TOPn, где n – числовое значение, позволяет отобрать в результат не все строки, а только n первых. При этом выбираются первые строки результата выборки, а не исходных данных.

Использование библиотек доступа и встраиваемого SQL.

Каждая СУБД помимо интерактивной SQL-утилиты обязательно имеет библиотеку процедур доступа и набор драйверов СУБД для различных ОС.

Схема взаимодействия с использованием библиотек процедур доступа.



Функции библиотеки доступа.

1. Соединение с БД.
2. Запрос к БД на выполнение SQL-выражения
3. Запрос на извлечение данных.
4. Запрос на изменение данных.
5. Закрытие соединения с БД.

Реализация операций реляционной алгебры средствами оператора SELECT.

Для выбора кортежей, отвечающих некоторому условию необходимо выполнить запрос вида:

SELECT [DISTINCT/ALL] {[<список полей>]} FROM <список таблиц>
[WHERE <условие выборки или соединения>]

Товары (код товара, наименование товара, цена)

Контрагенты (код контрагента, наименование контрагента)

Закупки (код операции, дата операции, код товара, код контрагента, количество, цена)

- Символ «*» определяет очень часто встречаемую ситуацию, когда в результирующий набор включаются все столбцы из исходной таблицы запроса.
- Во фразе FROM задается перечень исходных таблиц запроса.
- Во фразе WHERE определяются условия отбора строк результата или условия соединения строк исходных таблиц.

Пример. Выбрать из таблицы «Закупки» все операции по товару «Товар 1»

SELECT *
FROM Закупки
WHERE Код товара =1

SELECT Закупка *
FROM Закупка
WHERE (((Закупка. [код товара]) =1));

Пример 2:

SELECT Закупка. *, Закупка. [дата операции], Закупка. [дата операции]
FROM Закупка
WHERE (((Закупка. [дата операции])>=#1/1/2005#) AND ((Закупка. [дата операции])<=# 1/15/2005#));

Пример 3:

SELECT Закупка. *
FROM Закупка
WHERE (((Закупка. [дата операции]) >=#1/1/2005#) AND ((Закупка. [код товара]) = 1 or (Закупка. [код товара]) = 2));

Вертикальный выбор (проекция) представляет собой операцию выбора определённых столбцов из множества столбцов таблицы. Если какой-либо столбец выбран то можно говорить, что по нему достроена проекция.

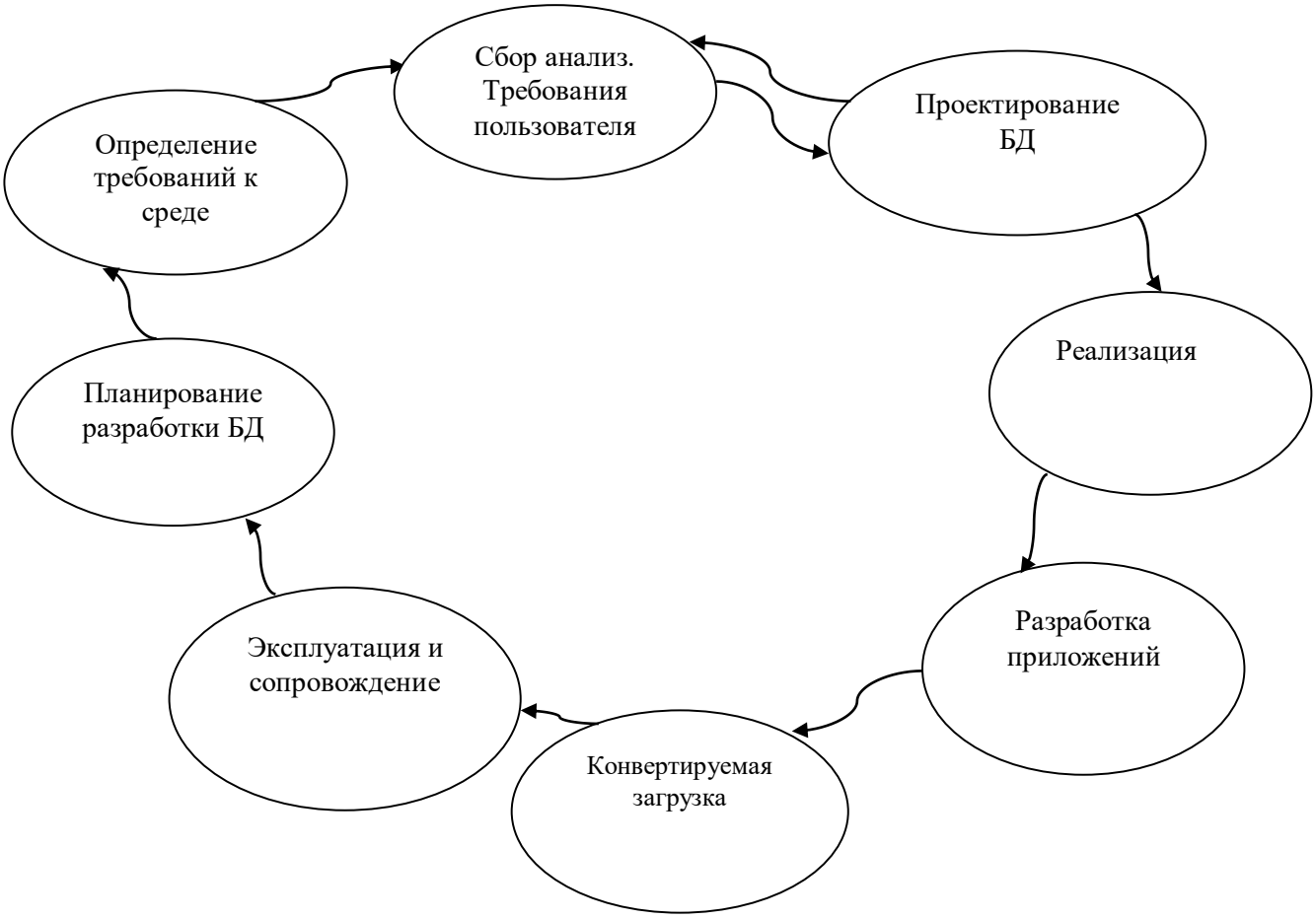
SELECT [Дата операции], [Код товара], [количество], [цена]
FROM Закупки.

Условное соединение позволяет соединять кортежи наборов данных, для которых выполняются некоторые условия. Соединение может быть внутренним и внешним.

Внутреннее соединение может быть реализовано либо за счёт помещения условия соединения в секции WHERE оператора SELECT, либо с использованием оператора INNER JOIN в секции FROM.

Внешние соединения – реализуется за счёт помещения условия соединения в селекции FROM оператора SELECT. Внешнее соединение получаются сцеплением записей первой таблицы с записями второй таблицы при соблюдения условий соединения. Естественно, что при соединении таблиц будут иметь место ситуации, когда либо первой строке таблицы не будет соответствовать ни одна.

Проектирование БД.
Жизненный цикл БД.



Планы проектирования.

Основные цели проектирования БД: это представление данных и связей между ними, необходимых для всех основных областей применения данного приложения и существующих групп его пользователей.

1. Создание модели данных, способных поддержать выполнение любых требуемых транзакций обработки данных.

Транзакция – это последовательная операция над БД, рассматриваемая СУБД как единое целое, т.е. представляет собой набор действий с целью доступа или изменения содержимого БД.

Проектирование простой транзакции:

1) Добавление
2) Обновление
3) Удаление

Проектирование сложной транзакции: образуется в том случае, когда в БД необходимо внести сразу несколько изменений (изменения в нескольких таблицах).

Если во время выполнения транзакции происходит сбой (выход компьютера из строя, сбой приложений), то БД попадает в противоречивое положение, т.к. частично изменения были уже внесены, поэтому все частичные изменения отменяются, и БД возвращается в прежнее состояние.

Транзакция выполняется полностью или не выполняется вообще.

2. Разработка предварительного варианта проекта, структура которого позволяет удовлетворить требования, предъявляемые к производительности системы.

Создание БД как модели предметной области выделяют следующие:

1. Объектную или предметную систему, представляющую фрагмент реального мира.
2. Информационная система, описывающая некоторую объектную систему.
3. Даталогическое представление информации с помощью данных.

Оптимальная модель данных удовлетворяет следующие критерии:

Структурная достоверность, простота, выразительность, отсутствие избыточности, размеренность, целостность, способность к совместимому использованию.

Проектирование можно разделить на несколько подэтапов:

(1) Концептуальное

(2) Логическое

(3) Нормализация

(4) Физическое проектирование БД

Концептуальное проектирование.

Результатом является создание для анализируемой части предприятия концептуальной модели БД.

Построение данной модели осуществляется в определенном порядке:

1) Создаются подробные модели пользовательских представлений данных
2) Они интегрируются в концептуальную модель данных.

Существуют два основных подхода проектирования системы БД:

1) Нисходящий
2) Восходящий

Восходящий подход применяется для проектирования простых БД с относительно необходимым количеством атрибутов. При данном подходе работа начинается с самого нижнего уровня, который на основе анализа существующих между ними связей группируется в отношениях. Полученное отношение в дальнейшем подвергается процессу нормализации, который приводит к созданию нормализованных взаимосвязанных таблиц, основанных на функциональных зависимостях между атрибутами.

Нисходящий применяется при проектировании сложных БД с большим количеством атрибутов. Так как установить между атрибутами все зависимости довольно сложно. Этот подход начинается с разработки БД, которая содержит несколько высокоуровневых сущностей и связей. Дальше работа продолжится в виде нескольких нисходящихся уточнений, которые содержат низкоуровневые сущности и связи относящихся к ним атрибутов.

Нисходящийся подход демонстрируется в концепции «сущность-связь» (ER-модель).

Модель «сущность-связь» относится к семантическим моделям, т.е. модели, которые связаны с символьным содержанием данных независимо от их представлений в ЭВМ.

Построение опций концептуальной модели выделяет ряд этапов:

1.Выделение локальных представлений соответствующих относительно независимым данным.

2.Формирование объектов, описывающих локальную предметную область, проектирование БД и описание атрибутов, составляющих структуры каждого объекта.

3.Выделение ключевых атрибутов.

4.Выделение ключевых связей между объектами удалений.
5.Анализ и добавление не ключевых атрибутов.
6.Объединение локальных атрибутов.

Построение концептуальной модели осуществляется на основе анализа предметной области на естественном языке. В процессе разработки концептуальной модели данных постоянно подвергаются тестированию и проверке на соответствие требованиям пользователей. Концептуальная модель является источником для логического проектирования БД.

Логический этап.

Целью является создание логической модели БД. Для создания общей модели предпринимаются существующие 2 подхода:

1. Централизованный

2. На основе приложений

При централизованном подходе, который применяется только для сложных БД образуется единый список требований для всех типов пользователей путём объединения этих требований.

При использовании метода интеграции представлений осуществляется слияние отдельных локальных моделей данных, отражающих единую общую модель.

В дальнейшем проектирование данных осуществляется на определенной модели данных (сетевая и др.), которая определяется типом предполагаемым для реализации СУБД. В процессе проектирование постепенно подвергается проверке на устранение избыточных данных.

Построение моделей будет источником для проектирования.

Процесс нормализации.

Процесс приведения таблиц БД к строгой форме путём последовательного преобразования таблиц к состоянию, в котором они удовлетворяют первым трем условиям нормальных форм. Происходит последовательное улучшение логической модели данных с тем, чтобы обеспечить ей устойчивость к операциям добавления, удаления и изменения БД.

Физическое проектирование БД.

Целью является создание, описания СУБД ориентированной модели БД. Действия выполняемые на этом этапе, специфичны для разных моделей БД.

Проектирование реляционных БД с использованием нормализации.

Исходной точкой является представление предметной области в виде одного или нескольких отношений, и на каждом шаге проектирования производится набор схем отношений, обладающих лучшими свойствами. Каждой нормальной форме соответствует некоторый определённый набор ограничений, и отношение находящееся в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

В теории реляционных БД обычно выделяется следующая последовательность нормальных форм:

- Первая нормальная форма (1NF);
- Вторая нормальная форма (2NF);
- Третья нормальная форма (3NF);
- Нормальная форма Бойса-Кодда (BCNF);
- Четвёртая нормальная форма (4NF);
- Пятая нормальная форма (51NF или PJ/NF).

Основные свойства нормальных форм:

- Каждая следующая нормальная форма в некотором смысле лучше предыдущей
- При переходе к следующей нормальной форме свойства предыдущей сохраняются.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных БД понятии *функциональной зависимости*.

1) Функциональная зависимость

В отношении R атрибут Y функционально зависит от атрибута X (X и Y могут быть составными) в том и только том случае, если каждому значению X соответствует в точности одно значение Y:R.X(r)R.Y.

2) Полная функциональная зависимость

Функциональная зависимость R.X(r)R.Y называется полной, если атрибут Y не зависит функционально от любого точного подмножества X.

3) Транзитивная функциональная зависимость

Функциональная зависимость R.X→R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R.X→R.Z и R.Z→R.Y и отсутствует функциональная зависимость R.Z→ R.X.

4) Неключевой атрибут

Неключевым атрибутом называется любой атрибут отношения, не входящий в состав первичного ключа.

5) Взаимно независимые атрибуты

Два или более атрибута взаимно независимы, если ни один из этих атрибутов не является функционально зависимым от других.

Пример схемы отношения:

СОТРУДНИКИ – ОТДЕЛЫ – ПРОЕКТЫ

(СОТР_НОМЕР, СОТР_ЗАРП, ОТД_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)

Первичный ключ: СОТР_НОМЕР, ПРО_НОМЕР

Функциональная зависимость:

СОТР_НОМЕР→СОТР_ЗАРП

СОТР_НОМЕР→ОТД_НОМЕР

ОТД_НОМЕР→СОТР_ЗАРП

СОТР_НОМЕР, ПРО_НОМЕР→СОТР_ЗАДАН

6) Вторая нормальная форма (единственным ключом отношения является первичный ключ)

Отношение R находится во второй нормальной форме (2NF) в том и только том случае, когда находится в 1NF (первой нормальной форме), и каждый не ключевой атрибут полностью зависит от первичного ключа.

Пример: СОТРУДНИКИ – ОТДЕЛЫ (СОТР_НОМЕР, СОТР_ЗАРП, ОТД_НОМЕР)

Первичный ключ: СОТР_НОМЕР

Функциональная зависимость:

СОТР_НОМЕР→СОТР_ЗАРП

СОТР_НОМЕР→ОТД_НОМЕР

ОТД_НОМЕР→СОТР_ЗАРП

СОТРУДНИКИ – ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)

Первичный ключ: СОТР_НОМЕР, ПРО_НОМЕР

Функциональная зависимость:

СОТР_НОМЕР, ПРО_НОМЕР→СОТР_ЗАДАН

7) Третья нормальная форма

Отношение R находится в 3NF (в третьей нормальной форме) в том и только том случае, если оно находится в 2NF (во второй нормальной форме), и каждый не ключевой атрибут не транзитивно зависит от первичного ключа.

8) Детерминант

Детерминант – любой атрибут, от которого полностью функционально зависит некоторый другой атрибут.

9) Нормальная форма Бойса-Кодда

Отношение R находится в нормальной форме Бойса-Кодда (BCNF) в том и только том случае, если каждый детерминант является возможным ключом.

10) Многозначные зависимости

В отношении R(A,B,C) существует многозначная зависимость R.A→→R.B в том и только том случае, если множество значений B, соответствующее паре значение A и C, зависит только от A и не зависит от C.

11) Четвёртая нормальная форма

Отношение R находится в 4NF (четвёртой нормальной форме) в том и только том случае, если в случае существования многозначной зависимости A→→B все остальные атрибуты R функционально зависят от A.

12) Зависимость соединения

Отношение R(X,Y,…,Z) удовлетворяет зависимости соединения *(X,Y,…,Z) в том и только том случае, когда R восстанавливается без потерь путём соединения своих проекций на X,Y,…,Z.

13) Пятая нормальная форма

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