|  |
| --- |
| Тема 1. Понятие БД и СУБД  /\*Сейчас информационные системы трудно представить без БД. Большинство организаций используют их для повседневной работы: от приказов и распоряжений до систем бухгалтерского учета. Различные справочники также часто распространяются в виде БД. Информационной основой сайтов в Internet часто являются БД. Вначале посмотрим, что было, когда не было БД Во-первых это нужно для того, чтобы понять все плюсы использования БД. А во-вторых еще не редко встречается мнение среди программистов, зачем мне СУБД, если я прекрасно все напишу на Pascal. В курсе выясним, что нужно написать на Pascal \*/  Раздел 1. Традиционные файловые системы  /\*Это не имеет отношения к операционным системам, это имеет отношение к способу хранения информации \*/  Файловая система – набор программ, которые выполняют для пользователей некоторые операции, например создание отчетов. Каждая программа определяет свои собственные данные и управляет ими.  В этом случае каждая функция автоматизируемой организации может быть реализована одной или несколькими программами, и каждая функция имеет свои собственные данные. Для доступа к каждому файлу необходимо создавать свои –программы.  /\*Пусть имеются два отдела. Первый занимается учетом производства товаров, а другой – их продажей. И один и другой отдел будет манипулировать информацией о товарах. Однако каждый будет иметь свой, независимый от другого, файл товаров. \*/  Недостатки файловых систем.  1 Дублирование данных.  Из-за децентрализованной работы с данными, проводимой в каждом подразделении независимо от других, в файловой системе фактически поощряется бесконтрольное дублирование данных. Такое дублирование нежелательно по двум причинам.  А) Дублирование данных сопровождается неэкономным расходованием ресурсов, поскольку на ввод избыточных данных требуется затрачивать дополнительное время, денежные ресурсы и дисковое пространство.  Б) Дублирование данных может привести к нарушению целостности системы. Например, человека могут повысить в должности, что отразится в файле отдела кадров, но не в файле бухгалтерии, что приведет к начислению ему заработной платы по старой должности. /\*Произведен товар, что зафиксировано в файле отдела производства, но эта информация не попала в файл отдела реализации, следовательно, товар не продается. \*/  2 Зависимость от данных.  Физическая структура и способ хранения информации жестко зафиксирован в коде программ. Это значит, что изменить существующую структуру. Например, изменение типа длины строки с 40 на 41 повлечет следующие действия. Пишется программа, которая создает временный файл с новой структурой, перекачивает данные в новый файл, удаляет старый файл, создает новый файл с новой структурой, но со старым именем, перекачивает данные, разрушает временный файл. Кроме этого, необходимо переписать все приложения, обрабатывающие этот файл.  3 Несовместимость форматов файлов.  Поскольку структура файлов определяется кодом приложений, она также зависит от языка программирования этого приложения. Таким образом, обратиться к файлу, созданному приложением, написанном на другом языке, может быть затруднительно. /\*как минимум могут отсутствовать нужные типы данных \*/  4 Фиксированные запросы.  Файловые системы очень зависимы от программиста, потому что все требуемые запросы создаются именно им. Это происходит из-за того, что в файловых системах отсутствуют инструменты для создания произвольных запросов к данным, а также получения информации о структурах файлов.  Все перечисленные недостатки файловых систем являются следствием двух факторов:  1 Определение данных содержится внутри приложений, а не храниться отдельно и независимо от них.  2 Помимо приложений не существует никаких других инструментов доступа к данным.  Раздел 2. БД.  База данных – это совместно используемый набор логически связанных данных и описаний этих данных, предназначенный для удовлетворения информационных потребностей организации.  Таким образом, БД представляет собой единое хранилище информации организации. При этом БД должна быть снабжена описанием структуры, называемым словарем данных или системным каталогом. Элементы такого описания называются метаданными, то есть данными о данных.  Хранение данных отдельно от их описания позволяет использовать абстрагирование данных. Оно заключается в том, что можно изменить внутреннее определение объекта, не изменяя внешнего, то есть из вне объект остается тем же самым, что позволяет не изменять его обработки. /\*Вообще это характерно для объектно-ориентированного программирования. \*/  Логическая связь данных заключается в том, что информация описывается некоторой логической моделью, например, сущность-связь. /\*Можно пояснить \*/  Раздел 3. Система управления базами данных  СУБД – это программное обеспечение, с помощью которого пользователи могу определять, создавать и поддерживать БД, а также осуществлять к ней контролируемый доступ.  Функции СУБД. /\*Если это выполняется – СУБД, иначе, еще неизвестно \*/  1 СУБД должна предоставлять пользователям возможность сохранять, извлекать и обновлять данные в БД.  2 СУБД должна иметь доступный конечным пользователям каталог, в котором хранится описание элементов данных.  Обычно в системном каталоге храниться следующая информация.  а) имена, типы и размеры элементов данных  б) имена связей  в) накладываемые на данные ограничения поддержки целостности  г) имена санкционированных пользователей, которым предоставлено право доступа к данным  д) статистические данные, например счетчики обращений к объектам БД  3 СУБД должна иметь механизм выполнения транзакций. /\*пояснить, что такое транзакция, противоречивое состояние БД, то есть зачем нужны, а так же, что не везде присутствует \*/  4 СУБД должна иметь механизм, которые гарантирует корректное обновление БД при параллельном выполнении операций обновления несколькими пользователями. То есть должен быть механизм решения конфликтов при одновременном обращении к одному и тому же элементу данных.  5 СУБД должна предоставлять средства восстановления БД на случай какого-либо ее повреждения или разрушения. /\*Начиная от некорректных действий пользователей, заканчивая повреждением физических носителей \*/  6 СУБД должна иметь механизм, гарантирующий возможность доступа к БД только санкционированных пользователей. То есть мало хранить в системном каталоге список этих пользователей, но нужно еще иметь механизм допуска к информации только санкционированных пользователей. /\*Таким образом СУБД должна обеспечивать возможность хранения конфиденциальной информации \*/  7 СУБД должна обладать способностью к интеграции с коммуникационным программным обеспечением. Проще говоря БД должны быть доступны по сети.  8 СУБД должна обладать инструментами контроля за тем, чтобы данные и их изменения соответствовали заданным правилам. /\*Привести примеры правил целостности \*/  Современные СУБД могут иметь вспомогательные службы:  1 Утилиты импорта и экспорта данных из других СУБД и плоских файлов.  2 Утилиты мониторинга, которые позволяют оценить текущее состояние БД, то есть процессы, выполняемые в текущий момент. /\*Какие соединения открыты, какие запросы выполняются, какие данные заблокированы \*/  3 Программы статистического анализа, позволяющие оценить производительность системы и степень использования БД.  4 Инструменты реорганизации физического хранения данных /\*индексы \*/  5 Инструменты сборки мусора и перераспределения памяти. /\*пояснить когда возникает \*/  /\*Не всегда все реализовано, но следует знать, что может быть включено. Чем больше функций, тем дороже. Необходимо анализировать насколько дорогой продукт необходим, что вообще не нужно, что можно дописать самостоятельно \*/  **1. Лекция: Введение: версия для печати и PDA**  Лекция посвящена истории возникновения области знаний, связанной с базами данных (БД). Выделяются основные этапы развития теории и практики БД, даются сравнительные характеристики этих этапов |
| http://www.intuit.ru/img/empty.gif |
| **http://www.intuit.ru/img/empty.gif** |
| http://www.intuit.ru/img/empty.gif |
| **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 интенсификации методов численного решения сложных математических задач, появлению языков программирования, ориентированных на удобную запись численных алгоритмов, становлению обратной связи с разработчиками новых архитектур ЭВМ. Характерной особенностью данной области применения вычислительной техники является наличие сложных алгоритмов обработки, которые применяются к простым по структуре данным, объем которых сравнительно невелик.  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:   1. надежное хранение информации в памяти компьютера; 2. выполнение специфических для данного приложения преобразований информации и вычислений; 3. 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.  Эти ограничения не являлись слишком существенными для чисто численных расчетов. Даже если программа должна обработать (или произвести) большой объем информации, при программировании можно продумать расположение этой информации во внешней памяти (например, на последовательной магнитной ленте), обеспечивающее эффективное выполнение этой программы. Однако в информационных системах совокупность взаимосвязанных информационных объектов фактически отражает модель объектов реального мира. А потребность пользователей в информации, адекватно отражающей состояние реальных объектов, требует сравнительно быстрой реакции системы на их запросы. И в этом случае наличие сравнительно медленных устройств хранения данных, к которым относятся магнитные ленты и барабаны, было недостаточным.  Можно предположить, что именно требования нечисловых приложений вызвали появление съемных *магнитных дисков 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. До этого каждая прикладная программа, которой требовалось хранить данные во внешней памяти, сама определяла расположение каждой порции данных на магнитной ленте или барабане и выполняла обмены между оперативной памятью и*устройствами внешней памяти* с помощью программно-аппаратных средств низкого уровня (машинных команд или вызовов соответствующих программ операционной системы). Такой режим работы не позволяет или очень затрудняет поддержание на одном внешнем носителе нескольких архивов долговременно хранимой информации. Кроме того, каждой прикладной программе приходилось решать проблемы именования частей данных и структуризации данных во внешней памяти.  **Файлы и файловые системы**  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. Правила именования файлов, способ доступа к данным, хранящимся в файле, и структура этих данных зависят от конкретной *системы управления файлами* и, возможно, от типа файла. *Система управления файлами* берет на себя распределение внешней памяти, отображение имен файлов в соответствующие адреса во внешней памяти и обеспечение доступа к данным.  Конкретные модели файлов, используемые в системе управления файлами, мы рассмотрим далее, когда перейдем к физическим способам организации баз данных, а на этом этапе нам достаточно знать, что пользователи видят файл как линейную последовательность записей и могут выполнить над ним ряд стандартных операций:   * создать файл (требуемого типа и размера); * открыть ранее созданный файл; * прочитать из файла некоторую запись (текущую, следующую, предыдущую, первую, последнюю); * записать в файл на место текущей записи новую, добавить новую *запись в конец файла*.   В разных файловых системах эти операции могли несколько отличаться, но общий смысл их был именно таким. 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 ситуация 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 перекомпиляции 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.  Далее, поскольку файловые системы являются общим хранилищем файлов, принадлежащих, вообще говоря, разным пользователям, *системы управления файлами* должны обеспечивать *авторизацию доступа* к файлам. В общем виде подход состоит в том, что по отношению к каждому зарегистрированному пользователю данной вычислительной системы для каждого существующего файла указываются действия, которые разрешены или запрещены данному пользователю. В большинстве современных систем управления файлами применяется подход к защите файлов, впервые реализованный в ОС UNIX. В этой ОС каждому зарегистрированному пользователю соответствует пара целочисленных идентификаторов: *идентификатор группы*, к которой относится этот пользователь, и его собственный идентификатор в группе. При каждом файле хранится полный идентификатор пользователя, который создал этот файл, и фиксируется, какие  действия с файлом может производить его создатель, какие действия с файлом доступны для других пользователей той же группы и что могут делать с файлом пользователи других групп. Администрирование режимом доступа к файлу в основном выполняется его создателем-владельцем. Для множества файлов, отражающих информационную модель одной предметной области, такой децентрализованный принцип управления доступом вызывал дополнительные трудности. И 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.  В системах управления файлами обычно применялся следующий подход. В операции открытия файла (первой и обязательной операции, с которой должен начинаться сеанс работы с файлом) среди прочих параметров указывался режим работы (чтение или изменение). Если к моменту выполнения этой операции некоторым пользовательским процессом PR1 файл был уже открыт другим процессом PR2 в режиме изменения, то в зависимости от особенностей системы процессу PR1 либо сообщалось о невозможности открытия файла, либо он блокировался до тех пор, пока в процессе PR2 не выполнялась операция закрытия файла.  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. В 1968 году была введена в эксплуатацию первая промышленная СУБД система *IMS* фирмы IBM. В 1975 году появился первый стандарт ассоциации по языкам систем обработки данных — *Conference* of *Data System* Languages (CODASYL), который определил ряд фундаментальных понятий в теории систем баз данных, которые и до сих пор являются основополагающими для *сетевой модели данных*.  В дальнейшее развитие теории баз данных большой вклад был сделан американским математиком Э. Ф. Коддом, который является создателем реляционной модели данных. В 1981 году Э. Ф. Кодд получил за создание реляционной модели и *реляционной алгебры* престижную премию Тьюринга Американской ассоциации по вычислительной технике.  Менее двух десятков лет прошло с этого момента, но стремительное развитие вычислительной техники, изменение ее принципиальной роли в жизни общества, обрушившийся бум персональных ЭВМ и, наконец, появление мощных рабочих станций и сетей ЭВМ повлияло также и на развитие технологии баз данных. Можно выделить четыре этапа в развитии данного направления в обработке данных. Однако необходимо заметить, что все же нет жестких временных ограничений в этих этапах: они плавно переходят один в другой и даже сосуществуют параллельно, но тем не менее выделение этих этапов позволит более четко охарактеризовать отдельные стадии развития технологии баз данных, подчеркнуть особенности, специфичные для конкретного этапа.  Первый этап развития СУБД связан с организацией баз данных на больших машинах типа IBM 360/370, ЕС-ЭВМ и мини-ЭВМ типа PDP11 (фирмы Digital Equipment Corporation — DEC), разных моделях HP (фирмы Hewlett Packard).  Базы данных хранились во внешней памяти центральной ЭВМ, пользователями этих баз данных были задачи, запускаемые в основном в пакетном режиме. *Интерактивный режим*доступа обеспечивался с помощью консольных терминалов, которые не обладали собственными вычислительными ресурсами (процессором, внешней памятью) и служили только устройствами ввода-вывода для центральной ЭВМ. Программы доступа к БД писались на различных языках и запускались как обычные числовые программы. Мощные операционные системы обеспечивали возможность условно параллельного выполнения всего множества задач. Эти системы можно было отнести к системам распределенного доступа, потому что база данных была централизованной, хранилась на устройствах внешней памяти одной центральной ЭВМ, а доступ к ней поддерживался от многих пользователей-задач.  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, было введено понятие транзакции. * 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.  **Эпоха персональных компьютеров**  Персональные компьютеры стремительно ворвались в нашу жизнь и буквально перевернули наше представление о месте и роли вычислительной техники в жизни общества. Теперь компьютеры стали ближе и доступнее каждому пользователю. Исчез благоговейный страх рядовых пользователей перед непонятными и сложными языками программирования. 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, позволяющего автоматизировать многие аспекты деятельности. 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. Кажущаяся простота и доступность персональных компьютеров и их программного обеспечения породила множество дилетантов. Эти разработчики, считая себя знатоками, стали проектировать недолговечные базы данных, которые не учитывали многих особенностей объектов реального мира. Много было создано систем-однодневок, которые не отвечали законам развития и взаимосвязи реальных объектов. Однако доступность персональных компьютеров заставила пользователей из многих областей знаний, которые ранее не применяли вычислительную технику в своей деятельности, обратиться к ним. И спрос на развитые удобные программы обработки данных заставлял поставщиков программного обеспечения поставлять все новые системы, которые принято называть настольными (desktop) СУБД. Значительная конкуренция среди поставщиков заставляла совершенствовать эти системы, предлагая новые возможности, улучшая интерфейс и быстродействие систем, снижая их стоимость. Наличие на рынке большого числа СУБД, выполняющих сходные функции, потребовало разработки методов *экспорта-импорта данных* для этих систем и открытия форматов хранения данных.  Но и в этот период появлялись любители, которые вопреки здравому смыслу разрабатывали собственные СУБД, используя стандартные языки программирования. Это был тупиковый вариант, потому что дальнейшее развитие показало, что перенести данные из нестандартных форматов в новые СУБД было гораздо труднее, а в некоторых случаях требовало таких трудозатрат, что легче было бы все разработать заново, но данные все равно надо было переносить на новую более перспективную СУБД. И это тоже было результатом недооценки тех функций, которые должна была выполнять СУБД.  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. Кроме того, большинство СУБД предлагали развитый и удобный инструментарий для разработки готовых приложений без программирования. Инструментальная среда состояла из готовых элементов приложения в виде шаблонов экранных форм, отчетов, этикеток (Labels), графических конструкторов запросов, которые достаточно просто могли быть собраны в единый комплекс. * 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. Вполне работоспособные приложения, разработанные, например, на Clipper, работали на PC 286. * В принципе, их даже трудно назвать полноценными СУБД. Яркие представители этого семейства — очень широко использовавшиеся до недавнего времени СУБД Dbase (DbaseIII+, DbaseIV), FoxPro, Clipper, Paradox.   **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:   + О структурной целостности — допустимыми являются только данные, представленные в виде отношений реляционной модели;   + О языковой целостности, то есть языков манипулирования данными высокого уровня (в основном SQL);   + О 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 децентрализованного хранения данных потребовали развития средств администрирования БД с реализацией общей концепции средств защиты данных. * Потребность в новых реализациях вызвала создание серьезных теоретических трудов по оптимизации реализаций распределенных БД и работе с распределенными транзакциями 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, к которым можно отнести и протокол ODBC (Open DataBase Connectivity), предложенный фирмой Microsoft. * Именно к этому этапу можно отнести начало работ, связанных с концепцией объектно-ориентированных БД — ООБД. Представителями СУБД, относящимся к 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.   **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, например Microsoft Internet Explorer или Netscape Navigator, и для конечного пользователя процесс обращения к данным происходит аналогично скольжению по Всемирной Паутине (см. [рис. 1.1](http://www.intuit.ru/department/database/dbmdi/1/dbmdi_1.html#image.1.1)). 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. Простые задачи обработки данных, не связанные со сложными алгоритмами, требующими согласованного изменения данных во многих взаимосвязанных объектах, достаточно просто и эффективно могут быть построены по данной архитектуре. 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.  Взаимодействие с базой данных в технологии интранет  **Рис. 1.1.**  Взаимодействие с базой данных в технологии интранет  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.  **Контрольные вопросы**   1. Найдите сходства первого и четвертого этапов развития. 2. Найдите отличия первого и третьего этапов развития. 3. Если при использовании файловых систем для параллельного доступа пользователей создавать копии файлов для каждого пользователя, может ли это ускорить параллельную работу с информацией? |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://www.intuit.ru/img/empty.gif | |  | | --- | | **Базы данных** | | http://www.intuit.ru/img/empty.gif | | **http://www.intuit.ru/img/empty.gif** | | http://www.intuit.ru/img/empty.gif | | **1. Лекция: Введение в базы данных. Общая характеристика основных понятий: версия для печати и PDA**  Лекция посвящена рассмотрению развития основных понятий обработки данных, связанного с постоянным *расширением классов* решаемых на ЭВМ задач. Показывается необходимость *интеграции данных*при решении несколькими пользователями задач, использующих общие данные. Вводится понятие базы данных. | | http://www.intuit.ru/img/empty.gif | | **http://www.intuit.ru/img/empty.gif** | | http://www.intuit.ru/img/empty.gif | | Цель лекции: показать, что с изменением вида решаемых на ЭВМ задач в программировании возникают новые виды представления данных, в том числе такой вид, как базы данных.  **1.1. Развитие основных понятий представления данных**  Любой вычислительный процесс представляет собой отображение (по определенному алгоритму) некоторых входных данных в выходные.  http://www.intuit.ru/department/database/databases/1/01_01.gif  Соотношение сложности *представления обрабатываемых данных* и алгоритма вычислений определяет два класса задач:   * вычислительные задачи – достаточно простое представление данных и сложный, многооперационный процесс вычислений; * задачи обработки данных (невычислительные задачи) – простой алгоритм обработки данных и сложное *представление обрабатываемых данных*.   На начальной стадии обучения программированию основное внимание уделяется разработке алгоритма решения задачи. Однако часто оказывается, что возможность (или невозможность) решения конкретной задачи зависит не только от выбранного алгоритма, но и от того, какие понятия используются для представления обрабатываемых данных.  Рассмотрим простейший пример вычисления по формуле:  Y=X2+5X,  где X и Y – определенные числа, которые являются здесь элементарными единицами данных ( *элементами данных* ).  При программировании алгоритма решения этой задачи (программирование формулы) используется простейший вид данных – простая переменная ( X и Y представляются в программе простыми переменными). Заметим, что простая переменная в системах программирования характеризуется определенным типом ее значений, которые должны выбираться при программировании. Даже в этом простейшем случае необходимо правильно выбрать тип переменной, причем от этого выбора может зависеть возможность или невозможность решения конкретной прикладной задачи (например, для представления конкретных данных не хватит отведенных разрядов).  Рассмотрим другой пример:  S=a1+a2+...+aN.  Решение этой задачи в общем случае невозможно получить используя только простые переменные. Здесь обрабатывается не отдельное число, а последовательность чисел. В этом случае при программировании используется такой вид данных, как массив – совокупность элементов, с каждым из которых связан упорядоченный набор целых чисел, называемых индексами. Все элементы должны иметь одинаковый тип их значений, который и будет *типом массива*. В этом случае числа a1, a2,..., aN представляются в программе массивом A(1), A(2),..., A(N).  Приведенные примеры показывают, что изменение вида задач обусловливает необходимость использования других видов данных.  Ранние языки программирования (ФОРТРАН, АЛГОЛ-60) были предназначены для решения научно-технических вычислительных задач. В этих языках использовались только вышеуказанные виды данных (простые переменные и массивы) что было вполне достаточно.  Начиная с конца 60-х годов компьютеры начинают интенсивно использоваться для решения так называемых невычислительных задач, связанных с обработкой различного рода документов. Рассмотрим появление новых видов данных на примере упрощенных задач обработки данных.  **Задача 1**. Начисление заработной платы.  Рассматриваем задачу при двух упрощающих предположениях:   * сотруднику начисляется заработная плата на основе его оклада; * никакие налоги и вычеты не учитываются.   Необходимые для решения этой задачи сведения о сотруднике представлены в следующей карточке НАЧИСЛЕНИЕ:   |  |  |  |  | | --- | --- | --- | --- | | **Фамилия, имя, отчество** | **Оклад** | **Количество отработанных дней в месяц** | **Начисленная сумма** | | **FIO** | **O** | **Ko** | **S** |   Для каждого работника начисленная сумма за определенный месяц рассчитывается по следующей формуле:  S=KoO/Kr,  где Kr – количество рабочих дней в данном месяце.  Для каждого сотрудника соответствующие данные имеют конкретное значение, например:   |  |  |  |  | | --- | --- | --- | --- | | Иванов Иван Иванович | 1800 | 24 | 1800 |   Эти значения имеют смысл только во взаимосвязи друг с другом. Отдельно выбранное число 1800 теряет свой содержательный смысл, поэтому использовать такой вид данных, как простая переменная, здесь нельзя. В то же время набор соответствующих значений, характеризующих конкретного сотрудника, имеет разные типы (символьный и числовой), т.е. использовать для его представления такой вид данных, как массив, также нельзя. Таким образом, понятий "простая переменная" и "массив" недостаточно, чтобы представить соответствующую карточку.  Для описания аналогичных представлений данных в предметной области невычислительных задач вводится ряд новых понятий [[[1]](http://www.intuit.ru/department/database/databases/popup.lit.html#1)].  *Элемент данных* (поле) – наименьшая единица поименованных данных.  Для данного примера элементами данных являются FIO, O, Ko, S.  Для описания карточки сотрудника используется понятие " *Логическая запись* ".  *Логическая запись* – поименованная совокупность элементов данных (полей).  *Экземпляр логической записи* – текущее значение элементов записи.  Для представления всего набора карточек сотрудников используется понятие " *Логический файл* "  *Логический файл* - поименованная совокупность всех *экземпляров записей* заданного типа.  Пример *логического файла* НАЧИСЛЕНИЕ:  http://www.intuit.ru/department/database/databases/1/01_02.gif  Таким образом, с помощью введенных понятий можно описывать соответствующие данные. Для отображения этих понятий в современных языках программирования, предназначенных как для вычислительных задач, так и для задач обработки данных, введены новые виды данных.  В алгоритмическом языке Паскаль вводится такой вид данных, как запись (RECORD) – сложная переменная с несколькими компонентами, которые могут иметь разные типы. Кроме того, доступ к компонентам записи (полям) осуществляется не по индексу, а по имени. При программировании задачи 1 на языке Паскаль *логическая запись*НАЧИСЛЕНИЕ представляется видом данных RECORD, набор экземпляров логических записей сотрудников (*логический файл*) представляется "физическим" файлом, формируемым средствами языка Паскаль и операционной системы.  Salary = RECORD  FIO: string;  O: real;  Ko: real;  S: real;  END;  Отметим важную специфику таких невычислительных задач. Для этих задач характерны большие объемы данных (большое количество сотрудников, большое количество производимых изделий и т. п.). Указанные данные, как правило, используются для решения задачи многократно (зарплата начисляется постоянно каждый месяц), поэтому данные должны достаточно долго храниться в памяти ЭВМ. Для длительного хранения всегда используется внешняя память.  В связи с этим решение задачи 1 состоит из двух этапов.  1. Ввод исходных данных и занесение их во внешнюю память.  2. Чтение исходных данных из внешней памяти, расчет начисленных сумм и вывод на печать.  Представленные программы решают поставленную задачу при сделанных предположениях. Необходимые для этого данные хранятся в файле MyFile.*fsf*, предназначенном только для решения этой задачи. Отметим, что в этом случае описание данных включено в прикладную программу. При изменении формата записей файла необходимо изменение прикладной программы. Таким образом, программная система, решающая поставленную задачу, определяет свои собственные данные и управляет ими. Такие программные системы называются файловыми системами [[[2]](http://www.intuit.ru/department/database/databases/popup.lit.html#2)], [[[3]](http://www.intuit.ru/department/database/databases/popup.lit.html#3)].  **Задача 2**. Учет кадрового состава.  Здесь обрабатываются сведения о сотруднике, представленные в карточке СОТРУДНИК:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Фамилия, имя, отчество** | **Должность** | **Год рождения** | **Оклад** | **Место жительства** | | **FIO** | **D** | **G** | **O** | **M** |   Решение задачи состоит из следующих этапов:  Ввод исходных данных и занесение их во внешнюю память.  Чтение исходных данных из внешней памяти с целью удаления, корректировки или добавления записи.  ...  { Чтение данных из внешней памяти }  Assign(F, 'MyFile.fsf');  Reset(F);  IsFound := False;  For I := 1 to N do  Begin  Read(F, Sotr);  If Sotr.FIO = KeyFio Then  Begin  IsFound := True;  Sotr.D := 'Начальник отдела';  Seek(F, FilePos(F)-1);  Write(F, Sotr);  Break;  End;  If IsFound Then  WriteLn('Корректировка успешно произведена')  Else WriteLn('Сотрудника ', KeyFio, 'не обнаружено');  Close(F);  ...  В рассматриваемом случае задача 2 решается независимо от задачи 1.  **Задача 3**. Учет экономии фонда оплаты труда (ФОТ) в связи с болезнью сотрудников.  Обрабатываются сведения, представленные записями ЭКОНОМИЯ ФОТ:   |  |  |  |  | | --- | --- | --- | --- | | **Фамилия, имя, отчество** | **Оклад** | **Количество дней на больничном листе** | **Невыплаченная сумма** | | **FIO** | **O** | **Kдв** | **SN** |   SN=KдвO/Kr.  Программа решения задачи 3 аналогична программе решения задачи 1.  Рассмотрим типичный случай, когда все три вышеуказанные программные системы функционируют в одной организации. Отметим следующие принципиальные эксплуатационные недостатки:  Информация дублируется. В трех файлах присутствуют поля FIO, O, что приводит к существенному перерасходу памяти. При внесении изменений (например, изменении фамилии) приходится вносить одно и то же значение несколько раз в разные файлы, что приводит к увеличению затрат машинного времени. Существует потенциальная возможность противоречивости данных (в один файл изменения внесены, в другой – нет).  Устранить перечисленные недостатки можно, объединив соответствующие записи и создав единую информационную базу для всех вышеназванных задач. На первый взгляд наиболее естественно объединить все записи в одну, убрав дублирующие поля. Получаем возможный вариант объединения:   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | FIO | D | O | G | Ko | M | Kдв | S | SN |   Дублирование информации полностью убрано. Расход памяти минимален. Недостатки устранены. Рассмотрим, как в этом случае изменится время решения задач 1–3. Время решения задачи прямо пропорционально объему считываемых из внешней памяти данных.  Обозначим Ti, li, Ni соответственно время решения, длину записи, число записей i -й задачи ( i = 1, 2, 3) при использовании отдельных файлов для каждой задачи:  http://www.intuit.ru/img/tex/a2c5b095d49bd195f6777a1fdcc267ac.png  где C – некоторый коэффициент пропорциональности.  Обозначим Ri, d, N соответственно время решения i -й задачи ( i = 1, 2, 3) при использовании файла объединенных записей, длину записи, число записей:  http://www.intuit.ru/img/tex/cb855a389d0f84f57461c68512fd9e16.png  Заметим, что N1 = N2 = N, N3 << N.  Тогда время решения i -й задачи ( i = 1, 2) при использовании объединенного файла увеличится в http://www.intuit.ru/img/tex/e71c68eb7b796d87c08dadd55b8fcf23.png раз. Для нашего примера время решения задач в зависимости от выбранной длины полей может изменяться в 2–3 раза. Таким образом, платой за исключение дублирования информации является увеличение времени решаемых задач. Заметим, что такое увеличение, как правило, допустимо.  Время решения задачи 3 увеличится в http://www.intuit.ru/img/tex/53ce203e478e68560440efc5e1b65c4d.png раз. Так как для данного примера N3 << N, то R3 >> T3. Время решения задачи 3 может увеличиться на несколько порядков, что совершенно недопустимо.  Рассмотрим другой вариант построения единой информационной базы. Объединим записи задач 1 и 2, запись задачи 3 оставим отдельно. Получим два типа записей:   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | FIO | D | O | G | | Ko | | S | M | | FIO | O | Kдв | | SN | |   В этом случае дублирование остается (дублируются поля FIO, O ). Но так как N3<<N, то общий объем дублирования незначителен. Время решения задачи 1 и 2 в этом случае незначительно возрастет по сравнению с вариантом отдельных файловых систем, время решения задачи 3 такое же, как и в начальном варианте отдельного файла. Такое объединение позволяет значительно уменьшить влияние недостатков и в то же время существенно увеличивает время решения всех задач. Все три задачи можно решать, используя общую информационную базу из двух типов записей. Отметим, что два приведенных типа записей связаны друг с другом по полю FIO (находятся в некотором отношении). Отметим, что приведенные варианты интеграции не исчерпывают все возможные способы *интеграции данных* для приведенных задач и к вопросу выбора наилучшего варианта вернемся в последующих лекциях.  Здесь очень важно, что в этом случае для решения вышеуказанных задач используется некоторый новый вид данных, формируемый на основе интеграции записей.  Для описания этого вида данных вводится новое понятие " *База данных* " [[[1]](http://www.intuit.ru/department/database/databases/popup.lit.html#1)].  *База данных* – совокупность экземпляров различных типов записей и отношений между записями и элементами.  *Базу данных* можно определить как совокупность взаимосвязанных хранящихся вместе данных при наличии такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений.  Таким образом, появление понятия "Базы данных" обусловлено возникновением нового класса невычислительных задач, при решении которых используются общие данные. В качестве основного критерия оптимальности функционирования базы данных,как правило, используются временные характеристики реализации запросов пользователей прикладными программами.  **Краткие итоги**. Рассмотрено развитие основных понятий представления данных. Описаны классические понятия программирования, связанные с данными (переменная, массив) и появление новых понятий программирования (поле, запись, файл) как следствие расширения круга решаемых задач и их отражения в системах программирования. Поставлена задача интегрирования данных при использовании несколькими задачами общих данных. Определено понятие базы данных. | |

|  |
| --- |
| **2. Лекция: Системы управления базами данных: версия для печати и PDA**  Вводится понятие системы управления базами данных (СУБД). Дается характеристика основных функций системы управления базами данных |
| http://www.intuit.ru/img/empty.gif |
| **http://www.intuit.ru/img/empty.gif** |
| http://www.intuit.ru/img/empty.gif |
| **Цель лекции**: показать необходимость создания программного интерфейса между прикладными программами и базой данных, определить понятие системы управления базами данных и сформулировать основные функции СУБД, вытекающие из задачи взаимодействия многих пользователей с базой данных.  В прикладной программе, использующей при решении задачи один или несколько отдельных файлов, за сохранность и достоверность данных отвечал программист, работающий с этой задачей. Использование базы данных предполагает работу с ней нескольких прикладных программ, решающих задачи разных пользователей.  Естественно, что за сохранность и достоверность интегрированных данных программист, решающий одну из прикладных задач, отвечать уже не может. Кроме того, расширение круга решаемых с использованием базы данных задач может приводить к появлению новых типов записей и отношений между ними. Такое изменение структуры базы данных не должно вести к изменению множества ранее разработанных и успешно функционирующих прикладных программных систем, работающих с базой данных. С другой стороны, возможное изменение любой из прикладных программ, в свою очередь, не должно приводить к изменению структуры данных. Все вышесказанное обусловливает необходимость отделения данных от прикладных программ.  Роль интерфейса между прикладными программами и базой данных, обеспечивающего их независимость, играет программный комплекс – *система управления базами данных* (*СУБД* ) ([рис. 2.1](http://www.intuit.ru/department/database/databases/2/databases_2.html#image.2.1)).  *СУБД* – программный комплекс поддержки интегрированной совокупности данных, предназначенный для создания, ведения и использования базы данных многими пользователями (прикладными программами).  Обеспечение независимости прикладных программ и базы данных  **Рис. 2.1.**  Обеспечение независимости прикладных программ и базы данных  Определим еще одно понятие.  *Банк данных* – система языковых, алгоритмических, программных, технических и организационных средств поддержки интегрированной совокупности данных, а также сами эти данные, представленные в виде баз данных.  Перечислим основные *функции системы управления базами данных*.  1. Определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки.  Как правило, создание структуры **базы данных** происходит в режиме диалога. *СУБД* последовательно запрашивает у пользователя необходимые данные. В большинстве современных *СУБД* **база данных** представляется в виде совокупности таблиц. Рассматриваемая функция позволяет описать и создать в памяти структуру таблицы, провести начальную загрузку данных в таблицы.  2. Предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация).  Такие возможности в *СУБД* представляются либо на основе использования специального языка программирования, входящего в состав *СУБД*, либо с помощью графического интерфейса.  Для клиент-серверных *СУБД* существуют средства, позволяющие выполнять запросы, и программные средства, позволяющие создавать графический интерфейс пользователя.  3. Обеспечение независимости прикладных программ и данных (логической и физической независимости).  Важнейшим свойством *СУБД* является возможность поддерживать два независимых взгляда на базу данных – "взгляд пользователя", воплощаемый в логическом представлении данных, и его отражения в прикладных программах; и "взгляд системы" – физическое представление данных в памяти ЭВМ. Обеспечение логической независимости данных предоставляет возможность изменения (в определенных пределах) логического представления базы данных без необходимости изменения физических структур хранения данных. Таким образом, изменение логического представления данных в прикладных программах не приводит к изменению структур хранения данных. Обеспечение физической независимости данных предоставляет возможность изменять (в определенных пределах) способы организации базы данных в памяти ЭВМ не вызывая необходимости изменения "логического" представления данных. Таким образом, изменение способов организации базы данных не приводит к изменению прикладных программ.  4. Защита логической целостности **базы данных**.  Основной целью реализации этой функции является повышение достоверности данных в базе данных. Достоверность данных может быть нарушена при их вводе в БД или при неправомерных действиях процедур обработки данных, получающих и заносящих в БД неправильные данные. Для повышения достоверности данных в системе объявляются так называемые ограничения целостности, которые в определенных случаях "отлавливают" неверные данные. Так, во всех современных СУБД проверяется соответствие вводимых данных их типу, описанному при создании структуры. Система не позволит ввести символ в поле числового типа, не позволит ввести недопустимую дату и т.п. В развитых системах ограничения целостности описывает программист, исходя из содержательного смысла задачи, и их проверка осуществляется при каждом обновлении данных. Более подробно разные аспекты логической целостности базы данных будут рассматриваться в последующих разделах.  5. Защита физической целостности.  При работе ЭВМ возможны сбои в работе (например, из-за отключения электропитания), повреждение машинных носителей данных. При этом могут быть нарушены связи между данными, что приводит к невозможности дальнейшей работы. Развитые СУБД имеют средства восстановления базы данных. Важнейшим используемым понятием является понятие " *транзакции* ". *Транзакция* – это единица действий, производимых с базой данных. В состав транзакции может входить несколько операторов изменения базы данных, но либо выполняются все эти операторы, либо не выполняется ни один. СУБД, кроме ведения собственно базы данных, ведет также журнал *транзакций*.  Необходимость использования *транзакций* в базах данных проиллюстрируем на упрощенном примере. Предположим, что база данных используется в некотором банке и один из клиентов желает перевести деньги на счет другого клиента банка. В базе данных хранится информация о количестве денег у каждого из клиентов. Нам нужно сделать два изменения в базе данных – уменьшить сумму денег на счете одного из клиентов и, соответственно, увеличить сумму денег на другом счете. Конечно, реальный перевод денег в банке представляет собой гораздо более сложный процесс, затрагивающий много таблиц, а возможно, и много баз данных. Однако суть остается та же – нужно либо совершить все действия (увеличить счет одного клиента и уменьшить счет другого), либо не выполнить ни одно из этих действий. Нельзя уменьшить сумму денег на одном счете, но не увеличить сумму денег на другом. Предположим также, что после выполнения первого из действий (уменьшения суммы денег на счете первого клиента) произошел сбой. Например, могла прерваться связь клиентского компьютера с базой данных или на клиентском компьютере мог произойти системный сбой, что привело к перезагрузке операционной системы. Что в этом случае стало с базой данных? Команда на уменьшение денег на счете первого клиента была выполнена, а вторая команда – на увеличение денег на другом счете – нет, что привело бы к противоречивому, неактуальному состоянию базы данных.  Использование механизма *транзакций* позволяет находить решение в этом и подобных случаях. Перед выполнением первого действия выдается команда начала транзакции. В транзакцию включается операция снятия денег на одном счете и увеличения суммы на другом счете. Оператор завершения транзакций обычно называется COMMIT. Поскольку после выполнения первого действия транзакция не была завершена, изменения не будут внесены в базу данных. Изменения вносятся (фиксируются) только после завершения транзакции. До выдачи данного оператора сохранения данных в базе не произойдет.  В нашем примере, поскольку оператор фиксации транзакции не был выдан, база данных "откатится" в первоначальное состояние – иными словами, суммы на счетах клиентов останутся те же, что и были до начала транзакции. Администратор базы данных может отслеживать состояние транзакций и в необходимых случаях вручную "откатывать" транзакции. Кроме того, в очевидных случаях СУБД самостоятельно принимает решение об "откате" транзакции.  Транзакции не обязательно могут быть короткими. Бывают транзакции, которые длятся несколько часов или даже несколько дней. Увеличение количества действий в рамках одной транзакции требует увеличения занимаемых системных ресурсов. Поэтому желательно делать транзакции по возможности короткими. В журнал транзакций заносятся все транзакции – и зафиксированные, и завершившиеся "откатом". Ведение журнала транзакций совместно с созданием резервных копий базы данных позволяет достичь высокой надежности базы данных.  Предположим, что база данных была испорчена в результате аппаратного сбоя компьютера, на котором был установлен сервер СУБД. В этом случае нужно использовать последнюю сделанную резервную копию базы данных и журнал транзакций. Причем применить к базе данных нужно только те транзакции, которые были зафиксированы после создания резервной копии. Большинство современных СУБД позволяют администратору воссоздать базу данных исходя из резервной копии и журнала транзакций. В таких системах в определенный момент БД копируется на резервные носители. Все обращения к БД записываются программно в журнал изменений. Если база данных разрушена, запускается процедура восстановления, в процессе которой в резервную копию из журнала изменений вносятся все произведенные изменения.  6. Управление полномочиями пользователей на доступ к базе данных.  Разные пользователи могут иметь разные полномочия по работе с данными (некоторые данные должны быть недоступны; определенным пользователям не разрешается обновлять данные и т.п.). В СУБД предусматриваются механизмы разграничения полномочий доступа, основанные либо на принципах паролей, либо на описании полномочий.  7. Синхронизация работы нескольких пользователей.  Достаточно часто может иметь место ситуация, когда несколько пользователей одновременно выполняют операцию обновления одних и тех же данных. Такие коллизии могут привести к нарушению логической целостности данных, поэтому система должна предусматривать меры, не допускающие обновление данных другим пользователям, пока работающий с этими данными пользователь полностью не закончит с ними работать. Основным используемым здесь понятием является понятие " *блокировка* ". *Блокировки*необходимы для того, чтобы запретить различным пользователям возможность одновременно работать с базой данных, поскольку это может привести к ошибкам.  Для реализации этого запрета СУБД устанавливает блокировку на объекты, которые использует транзакция. Существуют разные типы блокировок – табличные, страничные, строчные и другие, которые отличаются друг от друга количеством заблокированных записей. Чаще других используется строчная блокировка – при обращении транзакции к одной строке блокируется только эта строка, остальные строки остаются доступными для изменения.  Таким образом, процесс внесения изменений в базу данных состоит из следующей последовательности действий: выдается оператор начала транзакции, выдается оператор изменения данных, СУБД анализирует оператор и пытается установить блокировки, необходимые для его выполнения, в случае успешной блокировки оператор выполняется, затем процесс повторяется для следующего оператора транзакции. После успешного выполнения всех операторов внутри транзакции выполняется оператор фиксации транзакции. СУБД фиксирует изменения, сделанные транзакцией, и снимает блокировки. В случае неуспеха выполнения какого-либо из операторов транзакция "откатывается", данные получают прежние значения, блокировки снимаются.  8. Управление ресурсами среды хранения.  БД располагается во внешней памяти ЭВМ. При работе в БД заносятся новые данные (занимается память) и удаляются данные (освобождается память). СУБД выделяет ресурсы памяти для новых данных, перераспределяет освободившуюся память, организует ведение очереди запросов к внешней памяти и т.п.  9. Поддержка деятельности системного персонала.  При эксплуатации базы данных может возникать необходимость изменения параметров СУБД, выбора новых методов доступа, изменения (в определенных пределах) структуры хранимых данных, а также выполнения ряда других общесистемных действий. СУБД предоставляет возможность выполнения этих и других действий для поддержки деятельности БД обслуживающему БД системному персоналу, называемому администратором БД.  **Краткие итоги**. Рассмотрено понятие системы управления базами данных как интерфейса между прикладными программами и базами данных. Введено понятие банка данных. Дана характеристика основных функций систем управления базами данных, вытекающих из задачи взаимодействия многих пользователей с базой данных:   * Определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки * Предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация). * Обеспечение независимости прикладных программ (логической и физической независимости). * Защита логической целостности базы данных. * Защита физической целостности. * Управление полномочиями пользователей на доступ к базе данных. * Синхронизация работы нескольких пользователей. * Управление ресурсами среды хранения. * Поддержка деятельности системного персонала. |

|  |
| --- |
| **Лекция: Различные архитектурные решения, используемые при реализации многопользовательских СУБД. Краткий обзор СУБД: версия для печати и PDA**  В лекции рассматриваются различные варианты технологии работы с базой данных в многопользовательском режиме (*централизованная архитектура*, компьютерная сеть с файловым сервером, клиент-серверная архитектура). Дается краткий обзор современных СУБД. |
| http://www.intuit.ru/img/empty.gif |
| **http://www.intuit.ru/img/empty.gif** |
| http://www.intuit.ru/img/empty.gif |
| **Цель лекции**: показать основные варианты технологии работы нескольких пользователей с одной базой данных, связанные как с основными свойствами вычислительной техники, так и с развитием программного обеспечения.  Как уже отмечалось, понятие базы данных изначально предполагало возможность решения многих задач несколькими пользователями. В связи с этим, важнейшей характеристикой *современных СУБД* является наличие многопользовательской технологии работы. Разная реализация таких технологий в разное время была связана как с основными свойствами вычислительной техники, так и с развитием программного обеспечения. Дадим краткую характеристику этих технологий в хронологическом порядке. 3.1. Централизованная архитектура При использовании этой технологии база данных, СУБД и прикладная программа (приложение) располагаются на одном компьютере (мэйнфрейме или персональном компьютере) ([рис.3.1](http://www.intuit.ru/department/database/databases/3/databases_3.html#image.3.1).). Для такого способа организации не требуется поддержки сети и все сводится к автономной работе. Работа построена следующим образом:   * База данных в виде набора файлов находится на жестком диске компьютера. * На том же компьютере установлены СУБД и приложение для работы с БД . * Пользователь запускает приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к БД на выборку/обновление информации. * Все обращения к БД идут через СУБД, которая инкапсулирует внутри себя все сведения о физической структуре БД. * СУБД инициирует обращения к данным, обеспечивая выполнение запросов пользователя (осуществляя необходимые операции над данными). * Результат СУБД возвращает в приложение. * Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.   Централизованная архитектура  **Рис. 3.1.**  Централизованная архитектура  Подобная архитектура использовалась в первых версиях СУБД DB2, Oracle, Ingres [[[5]](http://www.intuit.ru/department/database/databases/popup.lit.html#5)].  Многопользовательская технология работы обеспечивалась либо режимом мультипрограммирования (одновременно могли работать процессор и внешние устройства – например, пока в прикладной программе одного пользователя шло считывание данных из внешней памяти, программа другого пользователя обрабатывалась процессором), либо режимом разделения времени (пользователям по очереди выделялись кванты времени на выполнение их программ). Такая технология была распространена в период "господства" больших ЭВМ (IBM-370, ЕС-1045, ЕС-1060). Основным недостатком этой модели является резкое снижение производительности при увеличении числа пользователей. 3.2. Технология с сетью и файловым сервером (архитектура "файл-сервер") Увеличение сложности задач, появление персональных компьютеров и локальных вычислительных сетей явились предпосылками появления новой архитектуры *файл-сервер*. Эта архитектура баз данных с сетевым доступом предполагает назначение одного из компьютеров сети в качестве выделенного сервера, на котором будут храниться файлы базы данных [[[6]](http://www.intuit.ru/department/database/databases/popup.lit.html#6)]. В соответствии с запросами пользователей файлы с *файл-сервера* передаются на рабочие станции пользователей, где и осуществляется основная часть обработки данных. Центральный сервер выполняет в основном только роль хранилища файлов, не участвуя в обработке самих данных ([рис. 3.2](http://www.intuit.ru/department/database/databases/3/databases_3.html#image.3.2).).  Архитектура "файл-сервер"  **Рис. 3.2.**  Архитектура "файл-сервер"  Работа построена следующим образом:   * База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (файлового сервера). * Существует локальная сеть, состоящая из клиентских компьютеров, на каждом из которых установлены СУБД и приложение для работы с БД. * На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к БД на выборку/обновление информации. * Все обращения к БД идут через СУБД, которая инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на файловом сервере. * СУБД инициирует обращения к данным, находящимся на файловом сервере, в результате которых часть файлов БД копируется на клиентский компьютер и обрабатывается, что обеспечивает выполнение запросов пользователя (осуществляются необходимые операции над данными). * При необходимости (в случае изменения данных) данные отправляются назад на файловый сервер с целью обновления БД. * Результат СУБД возвращает в приложение. * Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.   В рамках архитектуры " *файл-сервер* " были выполнены первые версии популярных так называемых настольных СУБД, таких, как dBase и Microsoft Access.  В литературе [[[6]](http://www.intuit.ru/department/database/databases/popup.lit.html#6)] указываются следующие основные недостатки данной архитектуры:   * При одновременном обращении множества пользователей к одним и тем же данным производительность работы резко падает, т.к. необходимо дождаться пока пользователь, работающий с данными, завершит свою работу. В противном случае возможно затирание исправлений, сделанных одними пользователями, изменениями других пользователей. * Вся тяжесть вычислительной нагрузки при доступе к БД ложится на приложение клиента, так как при выдаче запроса на выборку информации из таблицы вся таблица БД копируется на клиентскую машину и выборка осуществляется на клиенте. Таким образом, неоптимально расходуются ресурсы клиентского компьютера и сети. В результате возрастает сетевой трафик и увеличиваются требования к аппаратным мощностям пользовательского компьютера. * Как правило, используется навигационный подход, ориентированный на работу с отдельными записями. * В БД на *файл-сервере* гораздо проще вносить изменения в отдельные таблицы, минуя приложения, непосредственно из инструментальных средств (например, из утилиты Database Desktop фирмы Borland для файлов Paradox и dBase); подобная возможность облегчается тем обстоятельством, что фактически у таких СУБД база данных – понятие более логическое, чем физическое, поскольку под БД понимается набор отдельных таблиц, сосуществующих в отдельном каталоге на диске. Все это позволяет говорить о низком уровне безопасности – как с точки зрения хищения и нанесения вреда, так и с точки зрения внесения ошибочных изменений. * Недостаточно развитый аппарат транзакций служит потенциальным источником ошибок в плане нарушения смысловой и ссылочной целостности информации при одновременном внесении изменений в одну и ту же запись.  3.3. Технология "клиент – сервер" Использование технологии " *клиент – сервер* " предполагает наличие некоторого количества компьютеров, объединенных в сеть, один из которых выполняет особые управляющие функции (является сервером сети).  Так, архитектура " *клиент – сервер* " разделяет функции приложения пользователя (называемого клиентом) и сервера. Приложение-клиент формирует запрос к серверу, на котором расположена БД, на структурном *языке запросов SQL* (Structured *Query Language*), являющемся промышленным стандартом в мире реляционных БД. Удаленный сервер принимает запрос и переадресует его SQL-серверу БД. SQL-сервер – специальная программа, управляющая удаленной базой данных. SQL-сервер обеспечивает интерпретацию запроса, его выполнение в базе данных, формирование результата выполнения запроса и выдачу его приложению-клиенту. При этом ресурсы клиентского компьютера не участвуют в физическом выполнении запроса; клиентский компьютер лишь отсылает запрос к серверной БД и получает результат, после чего интерпретирует его необходимым образом и представляет пользователю. Так как клиентскому приложению посылается результат выполнения запроса, по сети "путешествуют" только те данные, которые необходимы клиенту. В итоге снижается нагрузка на сеть. Поскольку выполнение запроса происходит там же, где хранятся данные (на сервере), нет необходимости в пересылке больших пакетов данных. Кроме того, SQL-сервер, если это возможно, оптимизирует полученный запрос таким образом, чтобы он был выполнен в минимальное время с наименьшими накладными расходами [[[6]](http://www.intuit.ru/department/database/databases/popup.lit.html#6), [[7]](http://www.intuit.ru/department/database/databases/popup.lit.html#7)]. Архитектура системы представлена на [рис. 3.3](http://www.intuit.ru/department/database/databases/3/databases_3.html#image.3.3).  Все это повышает быстродействие системы и снижает время ожидания результата запроса. При выполнении запросов сервером существенно повышается степень безопасности данных, поскольку правила целостности данных определяются в базе данных на сервере и являются едиными для всех приложений, использующих эту БД. Таким образом, исключается возможность определения противоречивых правил поддержания целостности. Мощный аппарат транзакций, поддерживаемый SQL-серверами, позволяет исключить одновременное изменение одних и тех же данных различными пользователями и предоставляет возможность откатов к первоначальным значениям при внесении в БД изменений, закончившихся аварийно [[[6]](http://www.intuit.ru/department/database/databases/popup.lit.html#6), [[7]](http://www.intuit.ru/department/database/databases/popup.lit.html#7)].  Архитектура "клиент – сервер"  **Рис. 3.3.**  Архитектура "клиент – сервер"  Итак, в результате работа построена следующим образом:   * База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (сервера сети). * СУБД располагается также на сервере сети. * Существует локальная сеть, состоящая из клиентских компьютеров, на каждом из которых установлено клиентское приложение для работы с БД. * На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к СУБД, расположенной на сервере, на выборку/обновление информации. Для общения используется специальный *язык запросов SQL*, т.е. по сети от клиента к серверу передается лишь текст запроса. * СУБД инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на сервере. * СУБД инициирует обращения к данным, находящимся на сервере, в результате которых на сервере осуществляется вся обработка данных и лишь результат выполнения запроса копируется на клиентский компьютер. Таким образом СУБД возвращает результат в приложение. * Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.   Рассмотрим, как выглядит разграничение функций между сервером и клиентом.   * Функции приложения-клиента:   + Посылка запросов серверу.   + Интерпретация результатов запросов, полученных от сервера.   + Представление результатов пользователю в некоторой форме (интерфейс пользователя). * Функции серверной части:   + Прием запросов от приложений-клиентов.   + Интерпретация запросов.   + Оптимизация и выполнение запросов к БД.   + Отправка результатов приложению-клиенту.   + Обеспечение системы безопасности и разграничение доступа.   + Управление целостностью БД.   + Реализация стабильности многопользовательского режима работы.   В архитектуре " *клиент – сервер* " работают так называемые "промышленные" СУБД. Промышленными они называются из-за того, что именно СУБД этого класса могут обеспечить работу информационных систем масштаба среднего и крупного предприятия, организации, банка. К разряду промышленных СУБД принадлежат MS SQL Server, Oracle, Gupta, *Informix*, *Sybase*, DB2, *InterBase* и ряд других [[[6]](http://www.intuit.ru/department/database/databases/popup.lit.html#6)].  Как правило, SQL-сервер обслуживается отдельным сотрудником или группой сотрудников (администраторы SQL-сервера). Они управляют физическими характеристиками баз данных, производят оптимизацию, настройку и переопределение различных компонентов БД, создают новые БД, изменяют существующие и т.д., а также выдают привилегии (разрешения на доступ определенного уровня к конкретным БД, SQL-серверу) различным пользователям [[[6]](http://www.intuit.ru/department/database/databases/popup.lit.html#6)].  Рассмотрим основные достоинства данной архитектуры по сравнению с архитектурой "файл-сервер":   * Существенно уменьшается сетевой трафик. * Уменьшается сложность клиентских приложений (большая часть нагрузки ложится на серверную часть), а, следовательно, снижаются требования к аппаратным мощностям клиентских компьютеров. * Наличие специального программного средства – SQL-сервера – приводит к тому, что существенная часть проектных и программистских задач становится уже решенной. * Существенно повышается целостность и безопасность БД.   К числу недостатков можно отнести более высокие финансовые затраты на аппаратное и программное обеспечение, а также то, что большое количество клиентских компьютеров, расположенных в разных местах, вызывает определенные трудности со своевременным обновлением клиентских приложений на всех компьютерах-клиентах. Тем не менее, архитектура " *клиент – сервер* " хорошо зарекомендовала себя на практике, в настоящий момент существует и функционирует большое количество БД, построенных в соответствии с данной архитектурой. 3.4. Трехзвенная (многозвенная) архитектура "клиент – сервер". *Трехзвенная* (в некоторых случаях *многозвенная* ) **архитектура** (N-*tier* или multi-*tier*). представляет собой дальнейшее совершенствование технологии " *клиент – сервер* ". Рассмотрев архитектуру " *клиент – сервер* ", можно заключить, что она является 2-звенной: первое звено – клиентское приложение, второе звено – сервер БД + сама БД. В*трехзвенной архитектуре* вся бизнес-логика (деловая логика), ранее входившая в клиентские приложения, выделяется в отдельное звено, называемое сервером приложений. При этом клиентским приложениям остается лишь пользовательский интерфейс. Так, в качестве клиентского приложения в описанном выше примере выступает Web-браузер.  Что улучшается при использовании *трехзвенной архитектуры*? Теперь при изменении бизнес-логики более нет необходимости изменять клиентские приложения и обновлять их у всех пользователей. Кроме того, максимально снижаются требования к аппаратуре пользователей.  Итак, в результате работа построена следующим образом:   * База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (сервера сети). * СУБД располагается также на сервере сети. * Существует специально выделенный сервер приложений, на котором располагается программное обеспечение (ПО) делового анализа (бизнес-логика) [[[5]](http://www.intuit.ru/department/database/databases/popup.lit.html#5)]. * Существует множество клиентских компьютеров, на каждом из которых установлен так называемый "тонкий клиент" – клиентское приложение, реализующее интерфейс пользователя. * На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение – тонкий клиент. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к ПО делового анализа, расположенному на сервере приложений. * Сервер приложений анализирует требования пользователя и формирует запросы к БД. Для общения используется специальный *язык запросов SQL*, т.е. по сети от сервера приложений к серверу БД передается лишь текст запроса. * СУБД инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на сервере. * СУБД инициирует обращения к данным, находящимся на сервере, в результате которых результат выполнения запроса копируется на сервер приложений. * Сервер приложений возвращает результат в клиентское приложение (пользователю). * Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.  3.5. Краткий обзор СУБД Многие авторы классифицируют СУБД на две большие категории: так называемые "настольные" и "серверные". 3.5.1. Настольные СУБД Настольные СУБД используются для сравнительно небольших задач (небольшой объем обрабатываемых данных, малое количество пользователей). С учетом этого, указанные СУБД имеют относительно упрощенную архитектуру, в частности, функционируют в режиме файл-сервер, поддерживают не все возможные *функции СУБД* (например, не ведется журнал транзакций, отсутствует возможность автоматического восстановления базы данных после сбоев и т. п.). Тем не менее, такие системы имеют достаточно обширную область применения. Прежде всего, это государственные (муниципальные) учреждения, сфера образования, сфера обслуживания, малый и средний бизнес. Специфика возникающих там задач заключается в том, что объемы данных не являются катастрофически большими, частота обновлений не бывает слишком высокой, организация территориально обычно расположена в одном небольшом здании, количество пользователей колеблется от одного до 10–15 человек. В подобных условиях использование настольных СУБД для управления информационными системами является вполне оправданным, и они с успехом применяются.  Одними из первых СУБД были так называемые dBase-совместимые программные системы, разработанные разными фирмами. Первой широко распространенной системой такого рода была система dBase III – PLUS (фирма Achton-Tate). Развитый язык программирования, удобный интерфейс, доступный для массового пользователя, способствовали широкому распространению системы. В то же время работа системы в режиме интерпретации обусловливала низкую производительность на стадии выполнения. Это привело к появлению новых систем-компиляторов, близких к системе dBase III – PLUS: *Clipper* (фирма Nantucket Inc.), FoxPro (фирма Fox Software), FoxBase+ (фирма Fox Software), *Visual FoxPro* (фирма Microsoft). Одно время достаточно широко использовалась СУБД PARADOX (фирма Borland International).  В последние годы очень широкое распространение получила система управления базами данных Microsoft Access, которая входит в целый ряд версий пакета Microsoft Office(фирма Microsoft). 3.5.2. Серверные СУБД Для крупных организаций ситуация принципиально меняется. Там использование *файл-серверных* технологий является неудовлетворительным по описанным выше причинам. Поэтому на передний край борьбы за автоматизацию выходят так называемые серверные СУБД.  Основными производителями таких систем обработки и хранения данных являются 3 корпорации: Oracle, Microsoft и IBM. Диаграмма соотношения объемов продаж соответствующих систем (источник: *IDC* Report, Май 2006) приводится на [рис. 3.4](http://www.intuit.ru/department/database/databases/3/databases_3.html#image.3.4).  Продажи ПО систем хранения данных в мире  **Рис. 3.4.**  Продажи ПО систем хранения данных в мире  Наиболее распространенными *клиент-серверными* системами здесь соответственно являются системы Oracle (разработчик *компания Oracle*), MS SQL Server (разработчик компания Microsoft), DB2 , *Informix* Dynamic Server (компания IBM).  Дадим краткую характеристику этим системам. MS SQL Server К настоящему времени разработано несколько версий систем: MS SQL Server-2000, MS SQL Server -2005, MS SQL Server-2008. Приведем информацию о системе MS SQL Server-2008 с сервера Microsoft (<http://www.microsoft.com/rus/SQL/2008/default.mspx>)  Microsoft http://www.intuit.ru/img/tex/e84b10a7d82119700986af593ec4c84c.png SQL Server http://www.intuit.ru/img/tex/e8a955d697e8d2170c1c8c3ee20067bb.png 2008 - это законченное предложение в области баз данных и анализа данных для быстрого создания *масштабируемых решений* электронной коммерции, бизнес-приложений и хранилищ данных. Оно позволяет значительно сократить время выхода этих решений на рынок, одновременно обеспечивая масштабируемость, отвечающую самым высоким требованиям. В SQL Server включена поддержка языка XML и протокола HTTP, средства повышения быстродействия и доступности, позволяющие распределить нагрузку и обеспечить бесперебойную работу, функции для улучшения управления и настройки, снижающие совокупную стоимость владения.  Платформа бизнес-анализа SQL Server 2008, тесно интегрированная с Microsoft Office, предоставляет развитую масштабируемую инфраструктуру для внедрения мощных возможностей бизнес-анализа в рабочий процесс всех бизнес-подразделений вашей компании, открывая доступ к нужной бизнес-информации через знакомый интерфейс MS Excel и MS Word.  MS SQL Server-2008 поддерживает создание и работу с корпоративным хранилищем данных, объединяющим информацию со всех систем и приложений, позволяющим получить единую комплексную картину бизнеса вашей компании.  MS SQL Server-2008 предоставляет масштабируемый и высокопроизводительный "процессор данных" - для самых ответственных и требовательных бизнес-приложений, тем, кому необходим высочайший уровень надежности и защиты, позволяя при этом снизить совокупную стоимость владения за счет расширенных возможностей по управлению серверной инфраструктурой.  MS SQL Server-2008 предлагает разработчикам развитую, удобную и функциональную среду программирования, включая средства работы с веб службами, инновационные технологии доступа к данным – все, что необходимо для эффективной работы с данными любых типов и форматов.  Отдельные аспекты MS SQL Server – 2008 будут описаны в лекциях [10](http://www.intuit.ru/department/database/databases/10/) и ["Направления развития баз данных"](http://www.intuit.ru/department/database/databases/14/)14. Oracle К настоящему времени разработано несколько версий систем, каждая из которых включает целую линейку продуктов, например Oracle 8, Oracle 9i, Oracle 10g.  Соответствующие линейки продуктов включают как собственно СУБД (например Oracle Database 10g, Oracle Database 11g) , так и средства разработки и анализа данных.  Приведем информацию о системе с сервера Oracle <http://www.oracle.com/global/ru/mid/oracle_products/database.html>).  Oracle предлагает комплексные, открытые, доступные и удобные в использовании технологические решения. Готовые пакетируемые решения автоматически включают в свою стоимость базу данных, сервер приложений, интеграционную платформу, инструменты аналитики и управления неструктурированными данными. Масштабируемые бизнес-приложения Oracle могут быть легко интегрированы с ИТ-инфраструктурой предприятия без потери уже вложенных в IT инвестиций.  СУБД Oracle Database 11g обеспечивает улучшенные характеристики за счет автоматизации задач администрирования и обеспечения лучших в отрасли возможностей по безопасности и соответствию нормативно-правовым актам в области защиты информации. Появилось больше функций автоматизации, самодиагностики и управления. Среди характеристик системы можно отметить управление большими объемами данных с использованием распределенных таблиц и компрессии, эффективную защиту данных, возможность полного восстановления, возможность интеграции геофизических данных медиа-контента в бизнес-процеcc и т.д. Серверы баз данных компании IBM К настоящему времени разработаны линейки продуктов DB2 и *Informix*, включающие как собственно СУБД так и средства разработки и анализа данных (DB2 Universal Database DB2 Personal Edition, DB2 Enterprise 9 и др., а также *Informix* Dynamic Server, *Informix* Dynamic Server Express, *Informix* Extended Parallel Server и др.  Приведем информацию о части таких систем с сервера (<http://www-01.ibm.com/software/ru/data/?pgel=ibmhzn>)  Универсальный сервер баз данных DB2 Universal Database - это масштабируемая, обьектно-реляционная система управления базами данных с интегрированной поддержкой мультимедиа и Web, работающая на системах от персональных компьютеров и серверов на процессорах Intel до Unix, от однопроцессорных систем до симметричных многопроцессорных систем (SMP) и систем с массовым параллелизмом (MPP), на хостах AS/400 и мейнфреймах. DB2 Universal Database объединяет в себе высокую производительность систем обработки транзакций в режиме on-line, объектно-реляционные расширения, усовершенствованные средства оптимизации с возможностями параллельной обработки и поддержкой очень больших баз данных. DB2 Universal Database также имеет новые встроенные средства для облегчения переноса на свою базу приложений, разработанных на других системах управления базами данных, таких как Oracle, Microsoft, *Sybase* и *Informix*. Помимо этого, DB2 Universal Database включает в себя дополнительные средства поддержки систем аналитической обработки в реальном времени (OLAP) и систем поддержки принятия решений, множество простых в использовании расширений (DB2 extenders). DB2 Universal Database доступна на абсолютном большинстве ключевых платформ, что дает заказчикам ту гибкость, которая им необходима.  Кроме вышеуказанных зарубежных систем отметим и отечественную разработку – СУБД НИКА, преемницу широко распространенной в Советском Союзе СУБД ИНЕС для ЕС ЭВМ.  **Краткие итоги**. В лекции рассмотрены различные архитектурные решения, используемые при реализации *многопользовательских СУБД*. *Централизованная архитектура*. Технология с сетью и файловым сервером (архитектура " *файл-сервер* "). Архитектура " *клиент – сервер* " (распределенная модель вычислений). Трехзвенная (*многозвенная) архитектура* клиент – сервер. Дан обзор *современных СУБД* (настольные СУБД, серверные СУБД). |

|  |
| --- |
| **4. Лекция: Различные представления о данных в базах данных. Основные этапы проектирования баз данных: версия для печати и PDA**  В лекции рассматриваются различные представления о данных в базах данных. Описываются модели данных (внешнее представление, концептуальная модель, структура хранения) и основные этапы проектирования базы данных. Рассматривается жизненный цикл проектирования базы данных. |
| http://www.intuit.ru/img/empty.gif |
| **http://www.intuit.ru/img/empty.gif** |
| http://www.intuit.ru/img/empty.gif |
| **Цель лекции**: Показать существование различных представлений о данных (различных моделей) у разных групп лиц, работающих с данными. Рассмотреть отражение этих представлений в трехуровневой архитектуре базы данных (внешний уровень, концептуальный уровень, внутренний уровень), сформулировать достоинство трехуровневой архитектуры. Выделить основные этапы проектирования базы данных как процесса построения вышеуказанных моделей.  **4.1. Различные представления о данных в базах данных**  Создание базы данных предполагает интеграцию данных, предназначенных для решения нескольких прикладных задач разных пользователей. Соответственно, при интеграции данных должны учитываться требования к данным каждого пользователя, основанные на его представлении о данных и связях между ними. Далее эти требования должны обобщаться в единое представление, которое и будет служить основой для построения единой базы данных ([рис. 4.1](http://www.intuit.ru/department/database/databases/4/databases_4.html#image.4.1)).  Обобщение представлений всех пользователей о данных называется ***концептуальной моделью*** (схемой) БД. ***Концептуальная модель*** представляет информационное описание предметной области с учетом логических взаимосвязей, поэтому её еще называют инфологической (информационно-логической) моделью. В модели отсутствуют какие-либо понятия, связанные с ЭВМ, памятью ЭВМ, способами размещения данных в памяти ЭВМ, и, по сути, это модель только предметной области.  Обобщение представления пользователей о данных  **Рис. 4.1.**  Обобщение представления пользователей о данных  Как уже отмечалось, для создания базы данных и работы с ней используется система управления базами данных. Каждая конкретная СУБД поддерживает определенный вид данных (форматов записей и отношений), называемый *моделью данных СУБД* .  Следующий этап разработки базы данных предполагает выбор представления *концептуальной модели* с помощью модели данных конкретной СУБД. **Полученное таким образом представление** *концептуальной модели* **называется** *логической моделью* **БД. Или другими словами**, *логическая модель* **– это концептуальная схема, специфицированная в языке конкретной СУБД**. *Логическая модель* представляет данные и элементы данных вне зависимости от их содержания и среды хранения. Далее разработчик системы средствами СУБД отображает полученную логическую модель БД в память ЭВМ и определяет методы доступа. Полученное представление данных в памяти ЭВМ называется внутренним представлением или структурой хранения. Прикладные программы работают с логической моделью, причем каждому пользователю представляется подмножество этой логической модели (подсхема), отражающее его представление о предметной области. Каждая прикладная программа "видит" и обрабатывает только те данные, которые необходимы именно ей.  **Соответствующее "видение" данных прикладными программами (пользователями) представляет собой внешние представления**. Взаимосвязь вышеуказанных моделей изображена на [рис. 4.2](http://www.intuit.ru/department/database/databases/4/databases_4.html#image.4.2).  Различные представления о данных в БД  **Рис. 4.2.**  Различные представления о данных в БД  На данной схеме выделены три различных уровня описания данных (внешний, концептуальный, внутренний). **Эти уровни формируют так называемую** *трехуровневую архитектуру* ANSI/SPARC, предложенную в 1975 г. Комитетом планирования стандартов и норм SPARC (Standards Planning and Requirements Committee) Национального института стандартизации США (American National Standards Institute – ANSI). Основная цель этой архитектуры состоит в отделении пользовательского представления о данных в базе данных от их физического представления. Использование таких представлений о данных позволяет обеспечить выполнение основного требования к БД – независимости программ и данных. При изменении прикладных программ может измениться соответствующее *внешнее представление*, но *логическая модель* данных не изменяется и, соответственно, не будут изменяться другие прикладные программы. При изменении внутреннего представления (структур хранения) *логическая модель* не изменяется, соответственно, не изменяются прикладные программы.  Использование соответствующих представлений также позволяет четко разграничить полномочия различных лиц, работающих с базой данных.  Соответствующие представления позволяют описать "видение" базы данных разными лицами, работающими с ней:   * *внешнее представление* – представление специалиста предметной области (пользователя); * *внешнее представление* и *логическая модель* – представление прикладного программиста, разрабатывающего конкретное приложение для пользователя; * *логическая модель* и внутреннее представление – представление системного программиста, администрирующего базу данных.   **4.2. Основные этапы проектирования базы данных**  *Проектирование данных (базы данных)* представляет собой процесс последовательного отображения исследуемых явлений реального мира в виде данных в памяти ЭВМ ([рис. 4.3](http://www.intuit.ru/department/database/databases/4/databases_4.html#image.4.3)).  Общая схема проектирования  **Рис. 4.3.**  Общая схема проектирования  **Конкретные явления реального мира, представляющие интерес для проводимого исследования, будем называть предметной областью**.  *Проектирование (моделирование) базы данных* представляет собой многоэтапный процесс. *Основные этапы* этого процесса приведены на [рис. 4.4](http://www.intuit.ru/department/database/databases/4/databases_4.html#image.4.4).).  Этапы проектирования базы данных  **Рис. 4.4.**  Этапы проектирования базы данных  Подробно действия, отраженные на приведенном рисунке, будут рассмотрены в следующих лекциях. Здесь дадим лишь краткие комментарии к соответствующим блокам.  В блоках 1,2 необходимо особое внимание обратить на слово "абстрагирование". Имеется ввиду, что *проектирование базы данных* нужно вести не под конкретный документ, обрабатываемый пользователем, и не под конкретные действия пользователя с этим документом, а под обобщенный (абстрактный) образ документов и обобщенные (абстрактные) действия пользователей. Например, рассматривать документ не с конкретными числами строк и столбцов, а с абстрактными числами n и m; вместо требуемого пользователем поиска по конкретному полю (например, фамилии) рассматривать поиск по любому полю и т.д. Это очень важно, так как конкретные формы документов и действия пользователей при работе с ними достаточно часто изменяются. В этом случае при *проектировании базы данных* под конкретные формы документов и конкретные действия придется перепроектировать базу данных, что связано с существенными временными и стоимостными затратами.  Очень важным является выбор СУБД (блок 4), от которого в значительной степени зависит работоспособность построенной базы данных. Проблема выбора СУБД уже обсуждалась в лекции 3. Заметим здесь, что выбор СУБД зависит от количества форм документов, от сложности связей между данными, от объема обрабатываемых данных, от количества пользователей, работающих с БД и т.д.  Ранее отмечалось, что отображение логической модели базы данных в структуру хранения (представление данных в памяти компьютера) осуществляется системой управления базой данных. Тем не менее, во многих СУБД для повышения эффективности функционирования базы данных представляется возможность выбора ряда параметров, управляющих представлением данных в памяти компьютера. Выбор таких параметров и подразумевается в блоке 6.  Заметим, что очень важно при *проектировании базы данных* делать оценки ее возможной работоспособности. Так, по завершении проектирования обобщенного концептуального представления нужно попытаться оценить необходимое число производимых операций с элементами моделей при реализации возможных запросов пользователей. При невозможности в рамках построенной модели ответить на какой-то запрос пользователя или при значительном числе производимых при этом операций (что приведет к невозможности реализации соответствующего запроса в реальном масштабе времени) необходим возврат по схеме [рис. 4.4](http://www.intuit.ru/department/database/databases/4/databases_4.html#image.4.4). на шаг назад (построение более эффективного обобщенного концептуального представления). Аналогичные оценки необходимо делать и при завершении других этапов проектирования (блоки 5, 7). При этом возможен возврат назад на один или несколько шагов. Так, например, при проектировании логической модели (блок 5) не удается достичь адекватного представления*концептуальной модели* средствами *модели данных СУБД*. В этом случае необходимо либо вернуться на шаг назад и выбрать другую СУБД, либо вернуться к блоку 3 и изменить вид *концептуальной модели*. Так же, если полученные при реализации блока 7 оценки эксплуатационных характеристик не отвечают требованиям пользователя, возможны пересмотры всех ранее полученных решений (блоки 7, 6, 5, 4, 3). Кроме этого, необходим возврат на проектирование обобщенного концептуального представления при изменении внешних требований пользователей, а также при выявленных ошибках проектирования.  **Краткие итоги**: Рассмотрены различные представления о данных в базах данных - модели обрабатываемых данных (внешнее представление, концептуальная модель, структура хранения). Представлено отражение этих представлений в трехуровневой архитектуре базы данных (внешний уровень, концептуальный уровень, внутренний уровень), сформулировано достоинство трехуровней архитектуры. Описаны основные этапы проектирования базы данных как процесса построения вышеуказанных моделей и жизненный цикл проектирования базы данных (создание, апробация, исправление ошибок и улучшение характеристик, опытная эксплуатация). |

|  |
| --- |
| **5. Лекция: Первая стадия концептуального проектирования базы данных (концептуальное моделирование): версия для печати и PDA**  Лекция посвящена моделированию предметной области. Здесь рассматриваются понятия, с помощью которых описывается предметная область, средства графического представления концептуальной модели предметной области в виде *ER-диаграммы*, основные приемы, используемые при моделировании |
| http://www.intuit.ru/img/empty.gif |
| **http://www.intuit.ru/img/empty.gif** |
| http://www.intuit.ru/img/empty.gif |
| **Цель лекции**: показать, как описывается предметная область при концептуальном моделировании (с помощью каких понятий, средств представления и приемов построения) и как обеспечивается достоверность информации в базе данных за счет ограничений целостности концептуальной модели. 5.1. Описание информационного представления предметной области. ER-диаграмма Иллюстрацию вводимых понятий и этапов проектирования базы данных будем проводить на примере близкой для читателя конкретной предметной области: представление данных о студентах вуза. Дадим краткое описание рассматриваемой предметной области. В вузе имеется несколько факультетов, на каждом из которых ведется подготовка по нескольким специальностям или направлениям. Для каждой специальности на факультете есть свой учебный план, в котором приводится перечень изучаемых учебных курсов с указанием количества часов занятий. Студенты изучают соответствующие дисциплины, сдают экзамены и зачеты, получают оценки.  **Чаще всего** *концептуальная модель* **представляется в виде** *диаграммы сущностей – связей* (entity – relationship) или *ER-диаграммы*. Процесс построения *ER-диаграммы***называется ER-моделированием**.  Введем основные понятия, с помощью которых описывается предметная область.  *Сущность* (Entity) или объект – то, о чем будет накапливаться информация в информационной системе (нечто такое, за чем пользователь хотел бы наблюдать).  Если в системе обрабатывается информация о факультетах, *сущностью* будет являться факультет, если о студентах, сущность – студент и т.п.  Имя сущности при ER-моделировании, как правило, записывается заглавными буквами. Каждая *сущность* обладает определенным набором свойств (рассматриваем только свойства, представляющие интерес для пользователей в рамках проводимого исследования), которые запоминаются в информационной системе. Так, например, в качестве свойств сущности ФАКУЛЬТЕТ можно указать номер факультета, название факультета, в качестве свойств сущности СТУДЕНТ можно указать фамилию, дату рождения, место рождения, в качестве свойств сущности ЭКЗАМЕН – предмет, дату проведения экзамена, экзаменаторов.  Для информационного описания сущности вводится понятие атрибута.  *Атрибут* – поименованное свойство (характеристика) сущности. *Атрибут* представляет собой информационное отображение свойства сущности и принимает конкретное значение из множества допустимых значений. Так, например, для сущности ФАКУЛЬТЕТ атрибут "название" у конкретного экземпляра сущности принимает конкретное значение "вычислительной математики и кибернетики". Таким образом, *атрибут* представляет *информационное описание* количественных или качественных свойств сущности, описывает состояние сущности, позволяет идентифицировать *сущность*. Информация о сущности представляется совокупностью атрибутов. **Такую совокупность атрибутов часто называют записью об объекте**.  Совокупность сущностей, характеризующихся в информационной системе одним и тем же перечнем свойств, называется *классом сущностей* (набором объектов). Так, например, совокупность всех сущностей СТУДЕНТ составляет *класс сущностей* СТУДЕНТ, совокупность всех сущностей ФАКУЛЬТЕТ составляет *класс сущностей* ФАКУЛЬТЕТ.*Класс сущностей* описывается перечнем свойств сущностей, составляющих этот класс.  **Экземпляром сущности будем называть конкретную сущность (сущность с конкретными значениями соответствующих свойств)**. Выше мы определили сущность как то, о чем будет накапливаться информация в информационной системе. Это только одна сторона. Информация должна не просто храниться сама по себе, а использоваться для удовлетворения информационных потребностей пользователя. Для реализации подавляющего числа запросов пользователю прежде всего необходимо найти интересующий его экземпляр сущности (с целью обработки, корректировки, удаления). **Поэтому важнейшим свойством сущности является однозначная идентификация ее экземпляров по одному или группе атрибутов (уникальному идентификатору)**. У сущности ФАКУЛЬТЕТ это, например, номер факультета, у сущности СТУДЕНТ это может быть атрибут "фамилия", если у всех студентов разные фамилии, группа атрибутов "фамилия", "имя", "отчество", или специально введенный уникальный идентификатор, например дополнительно введенный атрибут "код студента".  Наиболее распространенным способом представления концептуальной модели является так называемая *ER-диаграмма*. В разных источниках используются разные системы обозначений в *ER-диаграммах*. На практике использование различных способов записи *ER-диаграмм* не представляет особой сложности – беглое ознакомление с соответствующим разделом документации позволяет быстро освоить используемую систему обозначений. В данном пособии в *ER-диаграмме* *класс сущностей* будем представлять в виде четырехугольника. В четырехугольнике записано уникальное имя класса сущности (прописными буквами) и имена атрибутов строчными буквами.  Пример класса сущностей СТУДЕНТ и конкретного экземпляра сущности показан на [рис. 5.1](http://www.intuit.ru/department/database/databases/5/databases_5.html#image.5.1)  Класс сущностей и экземпляр сущности  **Рис. 5.1.**  Класс сущностей и экземпляр сущности  Для реализации иформационных потребностей пользователя недостаточно найти интересующий его экземпляр сущности. Информационные потребности тесно связаны с функциональными взаимоотношениями, существующими в организации (например, необходимо определить, на каком факультете учится конкретный студент). Для реализации таких запросов (информационных потребностей пользователя) используются существующие в предметной области взаимоотношения между сущностями. **Соответствующие взаимоотношения сущностей выражаются связями (Relationships)**. Различают классы связей и экземпляры связей. **Классы связей – это взаимоотношения между классами сущностей, а экземпляры связи – взаимоотношения между экземплярами сущностей**.  Класс связей может затрагивать несколько *классов сущностей*. Число *классов сущностей*, участвующих в связи, называется *степенью связи* n = 2, 3, ... Так, например, *класс сущностей* СТУДЕНТ связан с *классом сущностей* ФАКУЛЬТЕТ *связью* "учится на факультете". Степень этой связи равна двум. При n =2 *связь* называется бинарной. Заметим, что *связь* нужно рассматривать как двустороннюю: "студент учится на факультете" и "на факультете учатся студенты". Рассмотрим классификацию *бинарных связей*. В зависимости от того, сколько *экземпляров сущности* одного класса связаны со сколькими *экземплярами сущности* другого класса, различают следующие *типы связей*:   * *Связь* 1:1. Одиночный экземпляр сущности одного класса связан с одиночным экземпляром сущности другого класса. Примером является *связь* между классами сущностей ФАКУЛЬТЕТ и УЧЕБНЫЙ ПЛАН ПО СПЕЦИАЛЬНОСТИ ДЛЯ ФАКУЛЬТЕТА (каждому факультету соответствует свой учебный план по специальности или направлению). * *Связь* 1:M. Единый экземпляр сущности одного класса связан со многими *экземплярами сущности* другого класса. Примером является *связь* между классами сущностей ФАКУЛЬТЕТ и СТУДЕНТ (на одном факультете учатся много студентов). * *Связь* M:N. Несколько *экземпляров сущности* одного класса связаны с несколькими *экземплярами сущности* другого класса. Примером является *связь* между классами сущностей ФАКУЛЬТЕТ и СПЕЦИАЛЬНОСТЬ (на факультете может быть несколько специальностей и одна и та же специальность может быть на нескольких факультетах).   Числа, описывающие типы *бинарных связей* ( 1:1, 1:M, M:N ), обозначают максимальное количество сущностей на каждой стороне связи. Эти числа называются максимальными кардинальными числами, а соответствующая пара чисел называется максимальной кардинальностью.  В данном пособии на *ER-диаграммах* связи между сущностями будем обозначать стрелками, рядом со стрелками указываем *имя связи*, а также тип связи. Пример *ER-диаграммы*, представляющей сущности СТУДЕНТ, ФАКУЛЬТЕТ, СПЕЦИАЛЬНОСТЬ и их взаимосвязи приводится на [рис. 5.2](http://www.intuit.ru/department/database/databases/5/databases_5.html#image.5.2).  Напомним, что каждый экземпляр сущности должен уникально идентифицироваться (иметь уникальный идентификатор). Так как могут быть несколько студентов с одинаковой фамилией, введем дополнительный атрибут "код студента". У сущностей ФАКУЛЬТЕТ и СПЕЦИАЛЬНОСТЬ атрибут "номер" является уникальным идентификатором.  Пример фрагмента ER-диаграммы  **Рис. 5.2.**  Пример фрагмента ER-диаграммы  Заметим, что по этой *ER-диаграмме* можно указать последовательность действий, производимых при реализации запроса пользователей. Например, для реализации запроса "на каком факультете учится студент Иванов" необходимо выполнить следующие действия: найти среди *экземпляров сущности* СТУДЕНТ экземпляр с фамилией Иванов, перейти по связи "Студент учится на факультете" к экземпляру сущности ФАКУЛЬТЕТ, значение атрибута "Название" этого экземпляра и есть искомое название факультета. Отметим также, что иногда на *ER-диаграммах* две связи между сущностями изображают одной двухсторонней стрелкой или просто линией. Заметим, что на приведенной *ER-диаграмме*не представлены какие-либо способы реализации этих связей (на логическом и, тем более, на физическом уровнях). Соответствующие способы реализации связей зависят от возможностей модели данных конкретной СУБД и будут рассмотрены в следующей лекции ([лекция 6](http://www.intuit.ru/department/database/databases/6/)) на второй стадии концептуального проектирования при *представлении концептуальной модели* средствами *модели данных СУБД*. 5.2. Построение концептуальной модели в виде ER-диаграммы5.2.1 Основные этапы построения Как уже отмечалось, *концептуальная модель* представляет собой обобщение представлений разных пользователей о данных. В связи с этим *построение концептуальной модели*, как правило, происходит в два этапа. На первом этапе производится сбор и анализ характеристик данных и строятся так называемые модели *локальных представлений* (локальные модели). Чаще всего локальная модель отражает представление отдельного пользователя (отдельной функциональной задачи). Иногда такая модель может описывать и некоторую независимую область данных нескольких функциональных задач (нескольких приложений). Здесь необходимо отметить, что моделирование представлений отдельных пользователей приводит к снижению уровня *интеграции данных*, а моделирование совместных представлений группы пользователей – к повышению сложности проектирования. В связи с этим при выборе области данных для локального моделирования приходится выбирать компромиссное решение между вышеуказанными вариантами.  При разработке концептуальной модели, прежде всего, следует определить сущности. С этой целью нужно сделать следующее:   * необходимо понять, какая информация должна храниться и обрабатываться и можно ли это определить как сущность; * присвоить этой сущности имя; * выявить *атрибуты сущности* и присвоить им имя; * определить *уникальный идентификатор сущности*.   Выявив сущности, необходимо определить, какие связи имеются между ними.  При определении связей (естественно, рассматриваем только те связи, которые имеют отношение к решаемым задачам обработки данных) необходимо учитывать следующее:   * то, как экземпляр одной сущности связан с экземпляром другой сущности; * то, как должны быть установлены связи, чтобы была возможность ответа на все запросы пользователей (исходя из их информационных потребностей).   Далее необходимо присвоить связям имена и определить тип связей.  На втором этапе построенные локальные модели объединяются в обобщенную концептуальную модель. 5.2.2. Моделирование локальных представлений Прежде всего, необходимо отметить, что построенная модель должна удовлетворять ряду требований:   * адекватно отражать представление пользователя о данных; * давать возможность ответа на возможные запросы пользователя, причем делать это с минимальными затратами по количеству просматриваемых сущностей; * представлять данные с минимальным дублированием.   Процесс построения модели, удовлетворяющей указанным требованиям, является творческим, и формализовать его, как правило, невозможно. Тем не менее можно указать некоторые способы порождения вариантов при моделировании. Выбор одного из таких вариантов на основе оценок объемов дублирования и числа просматриваемых объектов при ответах на запросы пользователей позволяет улучшить эксплуатационные характеристики проектируемой базы данных.  Вариативность моделирования обусловливается неоднозначностью выбора сущностей, атрибутов и связей. В одном варианте можно что-то взять за сущность, в другом варианте это же можно взять за атрибут (несколько атрибутов), в третьем варианте это можно определить как *связь*. Так, например, ранее мы определили сущность ФАКУЛЬТЕТ с атрибутами "номер факультета", "название факультета". Введем сущность КАФЕДРА с атрибутами "номер кафедры", "название кафедры". Между этими сущностями есть *связь* "факультет состоит из кафедр". Возможен другой вариант, в котором вышеуказанная *связь* представляется через *атрибуты сущности* (у сущности ФАКУЛЬТЕТ введем дополнительные атрибуты, представляющие номера и названия всех кафедр этого факультета).  После того как выбран рациональный вариант локальной модели, производится редактирование введенных наименований сущностей, атрибутов и связей. Здесь выполняются следующие действия:   * устраняются расплывчатые наименования (все наименования должны однозначно пониматься каждым пользователем); * устраняются синонимы (различные наименования одного и того же понятия); * устраняются омонимы (одно и то же наименование разных понятий).   Эти действия, вообще говоря, носят итерационный характер, т.к. после их выполнения вновь могут возникать и расплывчатые наименования, и синонимы, и омонимы. 5.2.3. Объединение локальных моделей На этом этапе ранее построенные модели *локальных представлений* отдельных пользователей (или групп пользователей) объединяются в единую концептуальную модель. Объединение локальных моделей производится следующими путями:   * слияние идентичных элементов; * установление связей между наборами сущностей разных моделей; * введение новых агрегированных элементов для представления связей между элементами разных моделей; * обобщение различных подобных *типов сущностей*, позволяющее трактовать эти сущности как одну обобщенную сущность.   Рассмотрим каждый из этих путей. Слияние идентичных элементов *Два или более элементов модели идентичны, если они имеют одинаковое смысловое значение*.  Объединение моделей с идентичными элементами осуществляется путем "слияния" этих элементов в один. Два набора сущностей СПЕЦИАЛЬНОСТЬ в модели 1 и 2 имеют одинаковое смысловое значение ([рис. 5.3](http://www.intuit.ru/department/database/databases/5/databases_5.html#image.5.3).), и могут быть заменены одним набором сущностей ([рис. 5.4](http://www.intuit.ru/department/database/databases/5/databases_5.html#image.5.4).).  Модели с идентичным элементом  **Рис. 5.3.**  Модели с идентичным элементом  Объединенная модель  **Рис. 5.4.**  Объединенная модель Установление связей между наборами сущностей разных моделей При рассмотрении наборов сущностей объединяемых моделей необходимо выявление связей между ними, т.к. именно эти связи и определяют в конечном итоге интегрированную базу данных. Введение агрегированных элементов *При объединении моделей* *связь* *между элементами разных моделей может рассматриваться как новый элемент*.  Рассмотрим в качестве примера моделирование информационного представления сдачи студентом экзаменов. Можно выделить ряд *локальных представлений* ([рис. 5.5](http://www.intuit.ru/department/database/databases/5/databases_5.html#image.5.5).).  Локальные представления  **Рис. 5.5.**  Локальные представления  Объединяя *локальные представления*, устанавливаем новые связи ([рис. 5.6](http://www.intuit.ru/department/database/databases/5/databases_5.html#image.5.6).).  Объединение локальных представлений  **Рис. 5.6.**  Объединение локальных представлений  Как уже отмечалось, одним из показателей "зрелости" модели является возможность ответа на запросы пользователей, и установление связей преследует именно эту цель. Нетрудно видеть, что какие бы связи в рассматриваемой модели ни вводились, невозможно ответить на запрос "какую оценку получил студент А по дисциплине В". В таком случае необходимо использовать принцип агрегации – необходимую *связь* между элементами модели ввести как некоторый новый элемент. В данном примере можно определить этот новый агрегированный элемент как ЭКЗАМЕН СТУДЕНТА ([рис. 5.7](http://www.intuit.ru/department/database/databases/5/databases_5.html#image.5.7).).  Агрегированный элемент  **Рис. 5.7.**  Агрегированный элемент  Далее процесс объединения локальных моделей продолжается обычным образом. Обобщение подобных типов сущностей Рассмотрим локальные модели разных факультетов, например – модель факультета вычислительной математики и кибернетики (ВМК), модель экономического факультета и так далее. В локальную модель факультета ВМК входят сущности СПЕЦИАЛЬНОСТИ ФАКУЛЬТЕТА ВМК и СТУДЕНТЫ ФАКУЛЬТЕТА ВМК, в локальную модель экономического факультета входят, соответственно, сущности СПЕЦИАЛЬНОСТИ ЭКОНОМИЧЕСКОГО ФАКУЛЬТЕТА и СТУДЕНТЫ ЭКОНОМИЧЕСКОГО ФАКУЛЬТЕТА ([рис. 5.8](http://www.intuit.ru/department/database/databases/5/databases_5.html#image.5.8).).  Модели с подобным элементом  **Рис. 5.8.**  Модели с подобным элементом  Два набора сущностей CПЕЦИАЛЬНОСТИ ФАКУЛЬТЕТА ВМК и СПЕЦИАЛЬНОСТИ ЭКОНОМИЧЕСКОГО ФАКУЛЬТЕТА в моделях 1 и 2 имеют одинаковое смысловое значение и могут быть заменены одним набором сущностей с добавлением нового атрибута – название факультета ([рис. 5.9](http://www.intuit.ru/department/database/databases/5/databases_5.html#image.5.9).).  Пример обобщенной сущности  **Рис. 5.9.**  Пример обобщенной сущности  Отметим, что в данном случае подобным образом можно слить и все остальные сущности локальных моделей факультетов, так как сущности СТУДЕНТЫ ЭКОНОМИЧЕСКОГО ФАКУЛЬТЕТА и СТУДЕНТЫ ВМК также имеют одинаковое смысловое значение и их также можно объединить. Однако в общем случае каждая локальная модель может содержать сущности и связи, которых нет в других локальных моделях.  Рассмотрим другой пример. Предположим, что мы храним данные о студентах (фамилия, имя, отчество, курс, группа) и о преподавателях (фамилия, имя, отчество, кафедра, должность). Соответственно, в предметной области выделяем две сущности – СТУДЕНТ и ПРЕПОДАВАТЕЛЬ.  Эти разные сущности можно в некоторых случаях трактовать как подобные. Для обобщения соответствующих сущностей необходимо, прежде всего, обобщить их атрибуты. Заметим, что атрибуты "Фамилия, Имя, Отчество" у обеих сущностей совпадают, атрибуты "Кафедра" и "Курс", "Группа" показывают место работы (учебы) и их можно заменить обобщенным атрибутом "Место работы (учебы)". Атрибут "Должность" можно использовать и у сущности СТУДЕНТ, если в качестве значения соответствующего атрибута использовать значение "студент". Тогда две сущности ПРЕПОДАВАТЕЛЬ и СТУДЕНТ можно трактовать как подобные и заменить их на обобщенную сущность. Дадим этой обобщенной сущности название КАДРОВАЯ ЕДИНИЦА ([рис. 5.10](http://www.intuit.ru/department/database/databases/5/databases_5.html#image.5.10).).  Пример обобщенной сущности  **Рис. 5.10.**  Пример обобщенной сущности  У студента атрибут "Место работы (учебы)" будет принимать значение соответствующее атрибутам "Курс. Группа", у преподавателя – название кафедры. Обобщенная модель представлена на [рис. 5.11](http://www.intuit.ru/department/database/databases/5/databases_5.html#image.5.11).  Обобщенная модель  **Рис. 5.11.**  Обобщенная модель  В этом случае почти в два раза упрощается структура концептуальной модели, и соответственно, структура базы данных. Для работы с данными о преподавателях и студентах достаточно одного набора программ. Таким образом, обобщение подобных типов объектов может существенно сократить последующие затраты на программирование.  В процессе объединения *локальных представлений*, как и при локальном моделировании, производится редактирование наименований (т.к. здесь появляются новые наименования). Процесс объединения также носит итерационный характер и продолжается до тех пор, пока не будут интегрированы все представления, согласованы и устранены все противоречия, отредактированы все наименования. Полученное в результате объединения *локальных представлений* обобщенное представление и является концептуальной моделью. 5.3. Ограничения целостности Под целостностью базы данных понимается то, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную область (правильная) информация.  Огромный объем данных, вводимых в базу данных (причем разные данные могут вводиться разными пользователями), обусловливает большое число ошибок ввода (занесения). Заметим, что при традиционной "бумажной" обработке информации также достаточно часто встречаются данные, записанные неверно. Но человек, работая с определенными данными, неявно использует для контроля имеющиеся у него представления об этих данных. Например, сотрудник отдела кадров, увидев в карточке работника год рождения 1693, сразу заметит эту ошибку и предположит, что просто переставлены две цифры и реальный год рождения 1963. То есть в представлениях сотрудника заключены некоторые логические ограничения на данные. Очевидно, что для контроля правильности вводимых данных при работе с базой данных целесообразно сформировать и использовать ограничения.  Соответствующие ограничения обычно разделяют на 3 группы: внешние, специально конструируемые и внутренние. К предметной области относятся первые две группы, которые мы кратко охарактеризуем в этом подразделе. Внутренние ограничения относятся уже к модели данных и будут рассматриваться в разделе, посвященном модели данных. Внешние ограничения **Эти ограничения связаны с адекватностью отражения предметной области**. Например, сотрудник организации не может быть моложе 17 и старше 90 лет. Соответствующее ограничение на год рождения (GR) можно записать следующим образом:  Текущий год – 17 > GR > Текущий год – 90.  Одним из способов задания таких ограничений является перечисление конечного множества допустимых значений какого-либо атрибута (так называемый "перечислимый" тип данных). Например, должность преподавателя в вузе может принимать одно из следующих значений: профессор, доцент, старший преподаватель, преподаватель, ассистент. Вводимое значение должности для конкретного экземпляра, не совпадающее с одним из перечисленных значений, является ошибкой. Ограничения, описанные с помощью специальных конструкций Например, в базу данных вуза вводятся данные о числе студентов и преподавателей. По нормативным документам задано конкретное *значение отношения* числа студентов к числу преподавателей. Проверку этого отношения можно использовать для контроля достоверности данных. Такие конструкции строятся исходя из специфики данных рассматриваемой предметной области. Можно, например, построить много конструкций следующего вида: сумма значений по заданному атрибуту по всем *экземплярам сущностей* должна совпадать со значением определенного атрибута в экземпляре другой сущности.  Таким образом, на стадии ER-моделирования для повышения достоверности данных необходимо сформулировать соответствующие ограничения на данные. В идеальном случае каждое значение атрибута должно каким-то образом контролироваться. Использование этих ограничений позволяет существенно повысить достоверность данных в базе данных.  **Краткие итоги**. Рассмотрен процесс моделирования предметной области. Определены используемые при этом основные понятия (сущность, атрибут, *идентификатор, связь*, типы связей, *ER-диаграмма*).  Рассмотрены основные этапы моделирования сущностей и связей (моделирование *локальных представлений*, объединение локальных моделей с ипользованием понятий идентичность, агрегация, обобщение).  Дано понятие *ограничений целостности*, имеющих непосредственное отношение к предметной области (*внешние ограничения*; ограничения, описанные с помощью специальных конструкций). |

|  |
| --- |
| **6. Лекция: Вторая стадия концептуального проектирования (Модели данных СУБД. Представление концептуальной модели средствами модели данных СУБД): версия для печати и PDA**  Лекция посвящена второй стадии концептуального проектирования – *представлению концептуальной модели* в терминах модели данных определенной СУБД. Здесь дается общее понятие *модели данных СУБД*, рассматриваются типовые классические модели данных, рассматриваются принципы *автоматизированного проектирования баз данных*. |
| http://www.intuit.ru/img/empty.gif |
| **http://www.intuit.ru/img/empty.gif** |
| http://www.intuit.ru/img/empty.gif |
| **Цель лекции**: дать общее представление о *модели данных СУБД* как средства для *представления концептуальной модели* при создании базы данных, рассмотреть типовые модели данных (*сетевая модель*, *иерархическая модель*, реляционная модель, *многомерная модель*), показать как представляется концептуальная модель в разных СУБД, рассмотреть основные принципы работы средств *автоматизированного проектирования баз данных*. 6.1. Представление концептуальной модели средствами модели данных СУБД **Общие представления о моделях данных СУБД**  В соответствии с *основными этапами проектирования базы данных* после построения концептуальной модели выбирается система управления базой данных, с помощью которой будет организована база данных и работа с ней. Каждая СУБД поддерживает определенные виды и типы данных, а также средства представления связей между данными, составляющими *модель данных* СУБД. Вторая стадия проектирования базы данных состоит в представлении построенной на предыдущей стадии концептуальной модели средствами *модели данных СУБД* или в отображении концептуальной модели в *модель данных* СУБД. Этот этап часто называют *логическим проектированием* базы данных. Полученная при этом модель часто также называется концептуальной моделью или схемой (но специфицированной к понятиям *модели данных СУБД*). В некоторых источниках полученную модель называют логической структурой данных или моделью данных базы данных.  Можно по-разному характеризовать понятие *модели данных* СУБД. С одной стороны, *модель данных* СУБД – это способ структурирования данных, которые рассматриваются как некоторая абстракция в отрыве от предметной области. С другой стороны, *модель данных* СУБД – это инструмент *представления концептуальной модели* предметной области и динамики ее изменения в виде базы данных.  Учитывая обе вышеуказанные стороны, определим основные структуры моделей данных СУБД, используемые для *представления концептуальной модели* предметной области (сущностей, атрибутов, связей).  *Элемент данных (поле) – наименьшая поименованная единица данных. Используется для представления значения атрибута*.  С элементом данных неразрывно связано понятие "тип данных", который может принимать соответствующее поле. В разных СУБД могут использоваться разные типы данных, наиболее распространенными из которых (используемые во многих СУБД) являются следующие: числовой (numeric), символьный (char), дата (date) и т.д.  *Запись – поименованная совокупность полей. Используется для представления совокупности атрибутов сущности (записи о сущности)*.  *Экземпляр записи – запись с конкретными значениями полей*.  *Первичный ключ – минимальный набор полей записи, однозначно идентифицирующий экземпляр записи файла*.  *Файл – поименованная совокупность экземпляров записей одного типа. Используется для представления однородного набора сущностей*.  *Набор файлов – поименованная совокупность файлов, обрабатываемых в системе. Используется для представления нескольких наборов сущностей*.  Введем понятие "группа", обобщающее понятия "файл" и "запись".  *Группа – это поименованная совокупность элементов данных и других групп*.  Важнейшим понятием концептуальной модели является понятие связи между сущностями (наборами сущностей). В *моделях данных СУБД* соответствующее понятие отражается понятием " **групповое отношение** ".  *Групповое отношение – поименованное бинарное отношение, заданное на двух множествах экземпляров рассматриваемых групп. По характеру бинарных связей различают групповые отношения вида 1:1, 1:M, M:1, M:N. Пары чисел называют коэффициентами группового отношения. В групповом отношении один член группы назначается владельцем отношения, другой – членом*.  *База данных – поименованная совокупность экземпляров групп и групповых отношений*.  Для представления группового отношения используется две формы:  а) **Графовая**. Группы изображаются вершинами графа, связи между группами – дугами, направленными от группы-владельца к группе-члену с указанием имени отношения и коэффициента.  *По типу графов различают*:   * *иерархическую модель (граф без циклов – дерево)* ; * *сетевую модель (ориентированный граф общего вида)*.   б) **Табличная**. Связь между группами изображается таблицей, столбцы которой представляют ключи соответствующих групп. Для формального описания таблицы используется математическое (теоретико-множественное) понятие отношения. Соответствующая *модель данных* называется реляционной моделью.  *Модель данных* **СУБД описывается следующим образом**:   * определены возможные типы и характеристики логических структур данных (полей, записей, файлов); * заданы правила составления структур более общего типа из структур более простых типов (например, записей из полей, файлов из записей и т.д.); * определен способ представления связей (отношений) между файлами и записями с помощью дополнительных полей ; * определены возможные действия над структурами и правила их выполнения, включающие:   + основные элементарные операции над данными;   + обобщенные операции (процедуры);   + средства контроля относительно простых условий корректности операций добавления, обновления или удаления данных (ограничения), реализуемые автоматически запускаемыми при выполнении вышеуказанных операций специальными процедурами (триггерами);   + средства контроля сколь угодно сложных условий корректности выполнения определенных действий (правила);   + специальный класс процедур (триггеры).   В качестве основных элементарных операций обычно рассматриваются следующие: поиск записи с заданным значением ключа, чтение нужной записи, добавление записи, корректировка, удаление. В *моделях данных СУБД* также предусматриваются специальные операции для установления групповых отношений.  Обобщенные операции или процедуры – последовательность операций, реализующая определенный алгоритм обработки данных. Процедуры могут инициироваться СУБД автоматически, а также могут запускаться пользователем. Примерами процедур являются процедуры копирования БД, восстановления БД, процедуры, вычисляющие значения определенных атрибутов в БД по значениям других атрибутов, и т.п.  Средства контроля используются для реализации ограничений целостности концептуальной модели. Простейшие средства контроля – ограничения – используются для реализации как *внешних ограничений* концептуальной модели, так и внутренних ограничений модели данных. В качестве последних ограничений, в частности, реализованы ограничения на ввод данных *несоответствующего типа*, несоответствующей характеристики (по числу битов, по числу полей, по количеству записей и т.п.). Более сложные средства контроля (правила) позволяют вызывать выполнение определенной последовательности операций (сколь угодно сложной) при изменении или добавлении данных в БД и тем самым реализовывать ограничения целостности, описанные с помощью специальных конструкций.  **Построение модели данных базы данных (отображение концептуальной модели в модель данных СУБД)**  После первой стадии концептуального проектирования у нас сформировано обобщенное представление пользователей о данных, как правило, представленное в виде *ER-диаграммы*. На следующей стадии (после того, как выбрана определенная СУБД с конкретной моделью данных) необходимо записать концептуальную схему в терминах и понятиях выбранной СУБД. На этой стадии каждая сущность концептуальной модели описывается как запись, состоящая из полей. Каждый атрибут описывается как поле с типом и характеристиками, возможными в выбранной СУБД. Описываются связи концептуальной модели в понятиях, соответствующих выбранной СУБД, определяется порядок реализации запросов пользователей к базе данных с помощью типовых операций СУБД и т. д.  Результатом этой стадии проектирования будет концептуальная модель, специфицированная к конкретной СУБД. 6.2 Типовые модели данных СУБД и представление концептуальной модели6.2.1. Сетевая модель данных Это одна из наиболее ранних моделей данных СУБД. Типовая *сетевая модель* данных была предложена рабочей группой по базам данных (Data Base Task Group – DBTG) системного комитета CODASYL (*Conference* of *Data System* Languages), основными функциями которого были анализ известных фирменных систем обработки управленческих данных с единых позиций и в единой терминологии, обобщение опыта организации таких систем и разработка рекомендаций по созданию соответствующих систем. Структура данных *сетевой модели* определяется в терминах раздела 6.1 (элемент, запись, группа, групповое отношение, файл, база данных).  Реализация групповых отношений в *сетевой модели* осуществляется с использованием специально вводимых дополнительных полей - указателей (адресов связи или ссылок), которые устанавливают связь между владельцем и членом группового отношения. Запись может состоять в отношениях разных типов (1:1, 1:N, M:N). Заметим, что если один из вариантов установления связи 1:1 очевиден (в запись – владелец отношения, поля которой соответствуют *атрибутам сущности*, включается дополнительное поле – указатель на запись – член отношения), то возможность представления связей 1:N и M:N таким же образом весьма проблематична. Поэтому наиболее распространенным способом организации связей в сетевых СУБД является введение дополнительного типа записей (и соответственно, дополнительного файла), полями которых являются указатели.  Рассмотрим для примера представление группового отношения M:N. В модель вводится *дополнительная группа* (дополнительный вид записей). Элементы этой записи представляют собой указатели на две исходные группы и указатели на экземпляры рассматриваемой дополнительной записи, связывающие их в список (цепь), соответствующий M и (или) N членам группового отношения ([рис. 6.1](http://www.intuit.ru/department/database/databases/6/databases_6.html#image.6.1).).  Представление связей типа M:N  **Рис. 6.1.**  Представление связей типа M:N  Представление связей 1:1, 1:M, N:1 является частным случаем связи типа M:N и осуществляется аналогично рассмотренному выше.  Заметим, что группа может быть членом более чем одного группового отношения. В этом случае вводится несколько *дополнительных групп*-указателей, а в группе – владельце отношений вводится несколько полей – указателей на *дополнительные группы*. Тогда множество записей (групп) и связей между ними образует некую сетевую структуру (ориентированный граф общего вида). Вершинами графа являются группы; дугами графа, направленными от владельца к члену группового отношения, – связи между группами.  *Сетевая модель* данных поддерживает все необходимые операции над данными, реализованные как действия со *списковыми структурами*. *Сетевая модель* данных является, вероятно, наиболее общей по возможностям *представления концептуальной модели*. По сути, любая ER-диаграмма без каких-либо изменений представляется средствами*сетевой модели*. К недостаткам *сетевой модели* обычно относят сложность получаемой на её основе концептуальной схемы и большую трудоемкость понимания соответствующей схемы внешним пользователем.  Рассмотрим пример записи части *ER-диаграммы* (СТУДЕНТ и ФАКУЛЬТЕТ) из предыдущей лекции в терминах сетевой СУБД. Для примера рассмотрим несколько *экземпляров сущности* СТУДЕНТ и сущности ФАКУЛЬТЕТ ([рис. 6.2](http://www.intuit.ru/department/database/databases/6/databases_6.html#image.6.2).).  Примеры экземпляров сущностей  **Рис. 6.2.**  Примеры экземпляров сущностей  Пусть студенты Иванов, Петров, Мишин учатся на факультете ВМК, Сидоров и Кашин на механико-математическом факультете. Тогда *сетевая модель* соответствующего фрагмента *ER-диаграммы* будет выглядеть следующим образом ([рис. 6.3](http://www.intuit.ru/department/database/databases/6/databases_6.html#image.6.3)).  Пример сетевой модели концептуального представления  **Рис. 6.3.**  Пример сетевой модели концептуального представления  Заметим, что в дополнительном файле один из указателей не потребовался, так как рассматриваемая связь имеет тип 1:N, а не M:N. Значок x обозначает отсутствие дальнейшей связи.  Наиболее существенным недостатком *сетевой модели* является "жесткость" получаемой концептуальной схемы. Связи закреплены в записях в виде указателей. При появлении новых аспектов использования этих же данных может возникнуть необходимость установления новых связей между ними. Это требует введения в записи новых указателей, т.е. изменения структуры БД, и, соответственно, переформирования всей базы данных.  СУБД, поддерживающие сетевую модель, широко использовались на вычислительных системах серии IBM 360/370 (ЕС ЭВМ). В качестве примеров таких систем можно указать*IDMS*, UNIBAD (БАНК), и их аналоги СЕДАН, СЕТОР. На персональных компьютерах сетевые СУБД не получили широкого распространения. Примером сетевой СУБД для персонального компьютера является db\_VISTA III. Отметим, что система db\_VISTA реализована на языке С и поэтому является переносимой. Система может эксплуатироваться на ПЭВМ типа IBM PC, SUN, Macintosh. 6.2.2. Иерархическая модель данных Это также одна из наиболее ранних моделей данных. Реализация групповых отношений в *иерархической модели*, как и в сетевой, может осуществляться с помощью указателей и представляется в виде графа. Однако, в отличие от *сетевой модели*, здесь существует ряд принципиальных особенностей.   1. Групповые отношения являются отношениями соподчиненности. Группа (запись) – владелец отношения имеет подчиненные группы – члены отношений. Исходная группа называется предком, подчиненная – потомком. 2. Групповые отношения образуют иерархическую структуру, которую можно описать как ориентированный граф следующего вида:    * имеется единственная особая вершина (соответствующая группе), называемая корнем, в которую не заходит ни одно ребро (группа не имеет предков);    * во все остальные вершины входит только одно ребро (все остальные группы имеют одного предка), а исходит произвольное количество ребер (группы имеют произвольное количество потомков);    * отсутствуют циклы. 3. *Иерархическая модель* данных может представлять совокупность нескольких деревьев. В терминологии *иерархической модели* деревья, описывающие структуру данных, называются деревьями описания данных, а сами структурированные данные (база данных) – деревьями данных.   Особенностью реализации операций поиска в *иерархической модели* является то, что операция всегда начинает поиск с корневой вершины и специфицирует иерархический путь (последовательность связанных вершин) от корня до вершины, экземпляры которой удовлетворяют условиям поиска.  Необходимо отметить, что программы, реализующие операции *иерархической модели*, существенно проще, чем аналогичные программы для *сетевой модели*, т.к. здесь много легче осуществлять навигацию по структуре. Целесообразность появления *иерархической модели* обусловлена, конечно, тем, что большинство организационных систем реального мира имеют иерархическую структуру (административное деление страны, организационная структура предприятия и т.п.). Соответствующее концептуальное представление также будет иметь иерархическую структуру и естественным образом может быть описано в терминах *иерархической модели*. В качестве недостатков*иерархической модели* можно назвать вышеуказанные недостатки сетевой.  СУБД, поддерживающие *иерархическую модель*, достаточно широко использовались на вычислительных системах IBM 360/370 (ЕС ЭВМ). В качестве примеров таких систем можно указать *IMS*, OKA и широко тиражируемую в СССР отечественную разработку ИНЕС. Примером иерархической СУБД для *персональных ЭВМ* является отечественная система НИКА (адаптация системы ИНЕС к IBM PC). 6.2.3. Реляционная модель данных Учитывая отмеченные в предыдущих разделах недостатки сетевых и иерархических моделей, можно сформулировать желательные требования к модели данных:   * модель должна быть понятна пользователю, не имеющему особых навыков в программировании; * появление новых аспектов использования данных и необходимость введения новых связей не должны приводить к реструктуризации всей модели данных и базы данных в целом.   Моделью данных, удовлетворяющей вышеуказанным требованиям, является *реляционная модель*, часто называемая также **табличной**.  Основным используемым понятием здесь является понятие отношения, представляемого в виде таблицы, столбцы которой соответствуют *атрибутам сущности* (структура строки таблицы аналогична структуре записи). Каждый атрибут может принимать определенное множество значений, называемое доменом. Строка таблицы с конкретными значениями полей здесь называется кортежем (соответствует понятию "*экземпляр записи*"). Поля таблицы предполагаются элементарными (неделимыми). Таким образом, понятие "таблица" здесь соответствует понятию "файл" модели данных. Первичный ключ здесь –минимальный набор атрибутов, однозначно идентифицирующий кортеж в отношении.  Групповое отношение может представляться двумя способами. При первом способе в таблицы, соответствующие группам – членам отношения, добавляются столбцы ключевых полей (атрибутов) другого члена отношения (связь описывается через ключевые атрибуты). При втором способе групповое отношение определяется как *дополнительная группа* (дополнительная таблица). Столбцами этой дополнительной таблицы являются ключи групп – членов отношения. Таким образом, при любом способе соответствующая модель данных представляет собой совокупность структур таблиц.  Рассмотрим пример записи *ER-диаграммы* (см. [рис. 5.2](http://www.intuit.ru/department/database/databases/5/1.html#image.5.2).) в терминах реляционных баз данных.  Сначала представим таблицы, соответствующие сущностям.  Таблица СТУДЕНТ   |  |  |  |  | | --- | --- | --- | --- | | Код | Фамилия | Дата рождения | Место рождения |   Таблица ФАКУЛЬТЕТ   |  |  | | --- | --- | | Номер | Название |   Таблица СПЕЦИАЛЬНОСТЬ   |  |  | | --- | --- | | Номер | Название |   Представим таблицы, описывающие связи.  Таблица "Студент учится на факультете"   |  |  | | --- | --- | | Код студента | Номер факультета |   Таблица "Студент учится по специальности"   |  |  | | --- | --- | | Код студента | Номер специальности |   Таблица "На факультете имеются специальности"   |  |  | | --- | --- | | Номер факультета | Номер специальности |   Заметим, что здесь реализован вышеописанный второй способ представления групповых отношений. Очевидно, что можно построить много вариантов таблиц, представляющих соответствующую ER-диаграмму.  Для приведенных таблиц не указаны домены атрибутов. Отсутствие указания доменов приводит к неоднозначной интерпретации содержания таблицы. Например, две таблицы с одинаковыми атрибутами   |  |  |  |  | | --- | --- | --- | --- | | Код студента | Фамилия | Дата рождения | Место рождения |   могут иметь разное смысловое значение и, соответственно, разное содержание (в одной таблице содержатся сведения о всех студентах факультета, в другой таблице сведения только о старостах факультета). Для устранения неоднозначной интерпретации в случае отсутствия указания доменов используют имя таблицы (СТУДЕНТ; СТАРОСТА)  *Для формального описания таблицы используется теоретико-множественное понятие отношения*.  Схемой отношения R называется перечень имен атрибутов отношения (соответствующих столбцам таблицы) с указанием доменов этих атрибутов и обозначается http://www.intuit.ru/img/tex/d0a2c52fccd17e4d80595532727d1a28.png, где {Ai} – множество значений, принимаемых атрибутом Ai(i=1,n).  *Совокупность схем отношений, используемых для представления концептуальной модели, называется схемой реляционной базы данных, а текущие значения соответствующих отношений – реляционной базой данных*.  В качестве основного недостатка реляционной модели можно указать дублирование информации при представлении связей.  Необходимо отметить, что большинство СУБД для *персональных ЭВМ* поддерживают именно реляционную модель данных. В качестве примеров таких наиболее распространенных СУБД можно указать все dBase-подобные системы, DB2, Paradox, Access, FoxPro, Oracle, MS SQL Server.  Более подробно *реляционная модель* данных будет рассмотрена в [следующей лекции](http://www.intuit.ru/department/database/databases/7/). 6.2.4. Многомерная модель данных Вернемся к понятию "сущность" концептуальной модели.  Сущность – это то, о чем накапливается информация в информационной системе. Часто оказывается, что информация об определенной сущности зависит еще от ряда параметров. Рассмотрим, например, сущность УСПЕВАЕМОСТЬ СТУДЕНТОВ со следующими атрибутами: число двоек, число троек, число четверок, число пятерок.  Значение атрибутов зависит от параметров "курс", "учебный год". Если использовать для описания соответствующей концептуальной схемы реляционную модель, то необходимо вводить множество таблиц УСПЕВАЕМОСТЬ СТУДЕНТОВ по каждому году для каждого курса. Так, при 5 курсах и необходимости анализировать данные за 10 лет число таблиц будет равно пятидесяти. Дублируются аналогичные структуры всех таблиц, достаточно сложна обработка данных, связанная с анализом однотипных данных при изменении значения одного из параметров и т.д.  Наиболее подходящей моделью данных для этого случая является так называемая *многомерная модель*, используемая в технологии OLAP (OnLine Analytical Processing – оперативная аналитическая обработка). Отметим, что многомерность модели данных означает здесь многомерное логическое представление структуры информации и, вообще говоря, не связана с многомерностью визуализации.  *Многомерные структуры представляются как гиперкубы данных. Каждая грань куба является размерностью. Основными понятиями, используемыми в многомерных моделях данных, являются "измерение" (dimension) и "ячейка" (cell)*.  *Измерение – упорядоченный набор значений, принимаемых конкретным параметром, соответствующий одной из граней гиперкуба*. Для нашего примера можно указать в качестве измерений: учебный год – 2006-2007, 2007-2008, 2008-2009; курсы – 1,2,3 и т.д.  *Ячейка или показатель – это поле, соответствующее атрибуту сущности, значение которого однозначно определяется фиксированным набором значений параметров (значениями "измерений"*, например, 2008-2009 учебный год, первый курс).  *В многомерной модели данных определяется ряд дополнительных операций, среди которых можно выделить операции "формирование среза" и "агрегация"*.  При формировании среза пользователю по его запросу предоставляется некоторое подмножество гиперкуба, полученное в результате фиксаций пользователем одного или нескольких значений параметров. Операция "агрегация" обеспечивает переход к более общему представлению информации из гиперкуба пользователю, например суммируя значения показателей по всем значениям одного из параметров, допустим, по всем курсам.  Такая модель позволяет легко сравнивать данные при разных значениях параметров, строить графики зависимости значений конкретных атрибутов от значений определенных параметров (например, изменение атрибута по годам) и т.п. Поэтому основное назначение технологии OLAP – обработка информации для проведения анализа и принятия решения.  Массовое использование СУБД, поддерживающих многомерную модель данных, только начинается. В качестве наиболее известных СУБД такого типа можно указать Oracle Express Server. 6.3. Средства автоматизированного проектирования концептуальной модели **Средства автоматизированного проектирования** концептуальной модели привлекают к себе в настоящее время большой интерес и используются в процессе создания структуры базы данных и интерфейса пользователя для доступа к данным.  Причина применения этих средств состоит в использовании в подавляющем большинстве реальных разработок баз данных спиральной модели жизненного цикла программного обеспечения, что предусматривает последовательное создание нескольких версий программного обеспечения. Каждая следующая версия включает в себя предыдущую (возможно, не полностью) и является ответом на замечания пользователя, полученные в результате тестирования предыдущей версии. При создании баз данных первая модель программного обеспечения, к сожалению, очень редко является удачной. Чаще всего заказчик отвергает первую версию, так как она недостаточно полно отвечает его требованиям. Причина такой ситуации заключается в том, что заказчик не может сразу, до создания начальной версии программы, четко и полно сформулировать свои требования. Обычно после получения первого варианта программного обеспечения заказчик выдвигает дополнительные требования, которые нельзя реализовать в рамках созданной базы данных. Это вынуждает разработчиков вносить изменения в структуру базы данных, а также, соответственно, в интерфейс пользователя для доступа к базе данных. Таких итераций может быть несколько до момента получения решения, адекватного запросам заказчика. Но даже после получения удовлетворительного решения процесс разработки базы данных не завершается. Жизнь не стоит на месте, и запросы заказчика меняются с течением времени. Часть этих изменений можно реализовать без изменения структуры базы данных, изменяя только интерфейс пользователя, другие же требуют изменения и интерфейса, и структуры базы данных. Надо заметить, что подобные изменения являются очень болезненными – работа по их внесению может оказаться трудоемкой и, что самое неприятное, потребовать замены большого количества отлаженного программного кода. Иными словами, замененный код был написан впустую, на самом деле его не нужно было писать.  Таким образом, создание работоспособной базы данных можно условно разделить на три этапа – проектирование базы данных, в процессе которого создаются рабочие прототипы, кодирование – создание структур баз данных и законченного интерфейса пользователя и сопровождение готовой базы данных.  Основная идея применения средств *автоматизированного проектирования баз данных* заключается в том, что процесс ручного кодирования начинается только после окончания процесса проектирования. На стадии проектирования схема базы данных и интерфейс пользователя для доступа к базе данных создаются автоматически, исходя из описания концептуальной модели, с помощью так называемых CASE-средств (Computer Aided Software/System Engineering). Конечно, созданный таким образом интерфейс не является законченным программным продуктом, однако он позволяет заказчику оценить возможности конечного продукта и внести свои коррективы. Только после одобрения заказчиком рабочего прототипа разработчики приступают к ручному кодированию – созданию законченного приложения.  При сопровождении все повторяется, за тем исключением, что генерируется не все приложение целиком, а только часть, которую надо изменять.  На практике чаще всего CASE-средства используются для создания схемы базы данных в виде ER-диаграмм и генерации структур баз данных для конкретной СУБД. После получения от заказчика изменений разработчики вносят соответствующие исправления в диаграмму "сущность – связь" и заново генерируют структуры баз данных. Средства автоматической генерации интерфейсов используются реже.  В настоящее время практически каждый производитель СУБД предлагает собственный программный продукт автоматизированного проектирования. Это Oracle Designer (Oracle), Power Desinger (Sybase) и другие. Демонстрационные версии данных программных продуктов можно загрузить с соответствующих сайтов ([www.oracle.com](http://www.intuit.ru/department/database/databases/6/www.oracle.com), [www.sybase.com](http://www.intuit.ru/department/database/databases/6/www.sybase.com)).  Кроме того, на рынке представлены решения третьих фирм, не производящих СУБД. Одними из самых распространенных являются программные продукты фирмы AllFusion – AllFusion ERwin Data Modeler и AllFusion Process Modeler (ранее – BPwin) и другие. На российском рынке данные программы предлагает фирма Interface Ltd. ([http://www.interface.ru](http://www.interface.ru/)). Создание диаграммы "сущность – связь" осуществляется с помощью AllFusion ERwin Data Modeler, дальнейшее моделирование, включая генерацию программного кода создания базы данных производится с помощью программы AllFusion Process Modeler.  Создав наглядную модель базы данных можно оптимизировать структуру БД и добиться её полного соответствия требованиям и задачам организации. Визуальное моделирование повышает качество создаваемой базы данных, продуктивность и скорость её разработки.  На сайте Interface Ltd. доступна для загрузки демонстрационная версия AllFusion ERwin Data Modeler, которая представляет собой полнофункциональную версию, ограниченную по времени.  **Краткие итоги**. В лекции рассмотрены вопросы, относящиеся ко второй стадии концептуального проектирования – представлениию концептуальной модели в терминах модели данных определенной СУБД. Дано описание общего представления о модели данных (основные используемые понятия - элемент, запись, файл; основные составляющие описания). Рассмотрены *модели данных СУБД* как инструмент *представления концептуальной модели* (*сетевая модель данных*, *иерархическая модель данных*, *реляционная модель данных*, *многомерная модель* данных и OLAP-технология). Приведены примеры записи концептуальной модели в терминах конкретной *модели данных СУБД*. Приводятся сведения о средствах автоматизированного проектирования концептуальной модели. |

|  |
| --- |
| **7. Лекция: Формализация реляционной модели: версия для печати и PDA**  В лекции рассматриваются вопросы, связанные с формализацией наиболее распространенной в настоящее время *модели данных СУБД* – реляционной модели. Здесь рассматривается формализованное описание отношений и средств манипулирования данными в реляционной модели. |
| http://www.intuit.ru/img/empty.gif |
| **http://www.intuit.ru/img/empty.gif** |
| http://www.intuit.ru/img/empty.gif |
| **Цель лекции**: рассмотреть формализованное описание реляционной модели и операций манипулирования данными как основу для использования математических методов проектирования баз данных и основу создания *языков запросов к базе данных*.  **7.1. Формализованное описание отношений и схемы отношений**  Как уже отмечалось в п. 6.2.3, реляционная модель описывает представление данных в виде двумерной таблицы, называемой *отношением*. Наименованиями столбцов этой таблицы **служат имена** *атрибутов*. **Рассмотрим формализованное описание соответствующих понятий**.  *Пусть* A1, A2, ..., An *имена* *атрибутов*. *Каждому имени* *атрибута* Ai *соответствует допустимое множество значений, которые может принимать* *атрибут* Ai. *Это множество значений* Di *называется* *доменом* *атрибута* Ai, i=1,n. *По определению*, *домены* *являются непустыми конечными или счетными множествами. Уточним, что в теории реляционных баз данных* *домен* *рассматривается как множество значений одного (причем простого) типа данных. Понятию* *домена* Di *соответствует множество значений, стоящих в столбце* Ai *рассматриваемой таблицы*.  *Схемой отношения* R {A1, A2, ..., An} *называется конечное множество имен атрибутов* {A1, A2, ..., An}, *причем атрибут* Ai *принимает значение из множества* Di (i=1, 2, ..., n), *где* n – *арность отношения*.  *Понятию* *"схема отношения"* *соответствует описание структуры двумерной таблицы (имена столбцов и допустимые множества значений)*.  Пусть http://www.intuit.ru/img/tex/0eb0b613e485db63fd298cea8017a0a5.png.  *Отношением* r *со схемой* R *называется конечное множество отображений* {t1, t2, ..., tp} *из множества* R: {A1, A2, ..., An} *в множество* http://www.intuit.ru/img/tex/6c797bb27cf04a49fdf0874ba6a5bbed.png,*таких, что* http://www.intuit.ru/img/tex/e24b861d8aa09de50d0723b368ba2861.png.  *Отображение* tk *называется* k *-м кортежем*, n – *размерность кортежа*.  Понятию k -го кортежа соответствует множество значений, стоящих в k -й строке рассматриваемой таблицы.  Понятию отношения r соответствует множество значений, стоящих во всех строках рассматриваемой таблицы.  *Ключом отношения* r *со схемой* R *называется минимальное подмножество* http://www.intuit.ru/img/tex/0d7f3082daaf91d7f253406b5c7bf560.png, *где* http://www.intuit.ru/img/tex/9e193dff5b259b63f6b0980dfd85c9c9.png, *такое, что любые два различных кортежа* http://www.intuit.ru/img/tex/bdafbf0d41d689e86c403e214facc043.png *не совпадают по значениям множества* K ={Ai1, Ai2, ..., Aim}.  Возможны случаи, когда *отношение* r имеет несколько ключей. Такие *ключи* называются потенциальными (возможными). **Выбранный из них ключ для идентификации кортежей называется первичным ключом**. Таким образом, достаточно знать значение кортежа на множестве K, чтобы однозначно его идентифицировать. Ключ используется для представления связей между отношениями. С этой целью первичный ключ одного отношения включается в структуру (набор атрибутов) связанного с ним отношения. Для второго отношения соответствующий ключ называется внешним ключом.  *Совокупность схем отношений, используемых для представления концептуальной модели, называется схемой реляционной базы данных (реляционной моделью данных). Текущие значения соответствующих отношений называются реляционной базой данных*.  Выпишем *реляционную модель данных* примера из [предыдущей лекции](http://www.intuit.ru/department/database/databases/6/) (см. [рис. 6.3](http://www.intuit.ru/department/database/databases/6/2.html#image.6.3).). Введем обозначения *атрибутов* всех соответствующих сущностей. Пусть A1 – код студента, A2 – фамилия, A3 – дата рождения, A4 – место рождения, A5 – номер факультета, A6 – название факультета, A7 – номер специальности, A8 – название специальности. Обозначим схему отношения СТУДЕНТ как R1, ФАКУЛЬТЕТ как R2, СПЕЦИАЛЬНОСТЬ как R3, СТУДЕНТ УЧИТСЯ НА ФАКУЛЬТЕТЕ как R4, СТУДЕНТ УЧИТСЯ ПО СПЕЦИАЛЬНОСТИ как R5, НА ФАКУЛЬТЕТЕ ИМЕЮТСЯ СПЕЦИАЛЬНОСТИ как R6.  Тогда реляционная модель соответствующего примера описывается следующей совокупностью схем отношений:  R1(A1, A2, A3, A4)  R2(A5, A6)  R3(A7, A8)  R4(A1, A5)  R5(A1, A7)  R6(A5, A7)  Напомним, что понятие *"схема отношения"* соответствует описанию структуры таблицы. Таблице с заполненными значениями (заполненными строками) соответствует понятие*"отношение"*. Для данного примера отношения, соответствующие вышеуказанным схемам отношений будем обозначать  r1, r2, r3, r4, r5, r6,  Отметим следующие свойства отношения:   1. *Отношение* имеет имя, которое отличается от имен всех других отношений. 2. Каждое значение элементов кортежей представляется простым (атомарным) типом данных. 3. Каждый *атрибут* имеет уникальное имя. 4. Значения всех атрибутов являются атомарными (неделимыми). Это следует из определения *домена* как множества значений простого типа данных, т.е. среди значений*домена* не могут содержаться множества. 5. Порядок рассмотрения атрибутов в схеме отношения (отношении) не имеет значения, т.к. для ссылки на значение атрибута в *кортеже отношения* всегда используется имя атрибута. 6. Порядок рассмотрения кортежей в отношении не имеет значения, т.к. *отношение* представляет собой множество кортежей, а элементы множества, по определению теории множеств, неупорядочены.   **7.2. Манипулирование данными в реляционной модели**  Для манипулирования данными в реляционной модели используются два формальных аппарата:   * *реляционная алгебра*, основанная на теории множеств; * *реляционное исчисление*, базирующееся на исчислении предикатов первого порядка.   Механизмы *реляционной алгебры* и *реляционного исчисления* эквивалентны, т.е. для любого допустимого выражения *реляционной алгебры* можно построить *эквивалентную формулу* *реляционного исчисления* и наоборот.  Отличаются два этих формальных аппарата уровнем процедурности. Выражения *реляционной алгебры* строятся на основе алгебраических операций (высокого уровня), и подобно тому, как интерпретируются арифметические и логические выражения, выражение *реляционной алгебры* также имеет процедурную интерпретацию. Другими словами, запрос, представленный на языке *реляционной алгебры*, может быть реализован как последовательность элементарных алгебраических операций с учетом их старшинства и возможного наличия скобок.  Для формулы *реляционного исчисления* однозначная интерпретация (соответствующая однозначная последовательность действий), вообще говоря, отсутствует. Формула только устанавливает условия, которым должны удовлетворять кортежи результирующего отношения. Поэтому языки *реляционного исчисления* являются более непроцедурными или декларативными.  Операции, реализуемые с помощью указанных аппаратов, обладают важным свойством: они замкнуты на множестве отношений. Это означает, что выражения *реляционной алгебры* и формулы *реляционного исчисления* определяются над отношениями реляционных БД и результатом вычисления также являются отношения. В результате любое выражение или формула могут интерпретироваться как *отношение*, что позволяет использовать их в других выражениях или формулах.  Как мы увидим, алгебра и исчисление обладают большой выразительной мощностью, очень сложные запросы к базе данных могут быть выражены с помощью одного выражения*реляционной алгебры* или одной формулы *реляционного исчисления*. Именно по этой причине такие механизмы включены в *реляционную модель данных*. *Конкретный язык манипулирования реляционными БД называется реляционно полным, если любой запрос, выражаемый с помощью одной* *операции реляционной алгебры* **или одной формулы реляционного исчисления, может быть выражен с помощью одного оператора этого языка**.  Заметим, что крайне редко алгебра или исчисление принимаются в качестве полной основы какого-либо языка БД. Обычно (как, например, в случае языка SQL) язык основывается на некоторой смеси алгебраических и логических конструкций. Тем не менее знание алгебраических и логических основ языков баз данных часто бывает полезно на практике.  **7.3. Операции реляционной алгебры**  *Операции реляционной алгебры* определены на множестве отношений и являются замкнутыми относительно этого множества (образуют алгебру). Оказывается, что любой произвольный запрос к БД можно представить в виде последовательности, составленной из пяти основных *операций реляционной алгебры*. Рассмотрим эти операции.  *Объединение* http://www.intuit.ru/img/tex/ad3696c4d52c1ec0da703bdcb8ab6a5c.png  *Объединением отношений* r и s *называется множество кортежей, которые принадлежат или* r, *или* s, *или им обоим. Для операции объединения требуется одинаковая арность отношений*.  Для примера, пусть   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **r** | | |  | **s** | | | | a | B | A |  | b | g | a | | d | A | F |  | d | a | f | | c | B | D |  |  |  |  |   Тогда   |  |  |  | | --- | --- | --- | | **http://www.intuit.ru/img/tex/ad3696c4d52c1ec0da703bdcb8ab6a5c.png** | | | | a | B | a | | d | A | f | | c | B | d | | b | G | a |   Заметим, что с помощью операции объединения может быть реализовано добавление нового кортежа к имеющемуся отношению. В этом случае r – исходное *отношение*, s –*отношение*, содержащее один добавляемый кортеж.  *Разность* r – s  *Разностью* *отношений* r *и* s *называется множество кортежей, принадлежащих* r, *но не принадлежащих* s. *Для этой операции также требуется одинаковая арность отношений*.   |  |  |  | | --- | --- | --- | | **r - s** | | | | a | B | a | | c | B | d |   Заметим, что с помощью *операции разности* может быть реализовано удаление кортежа из имеющегося отношения. В этом случае r – исходное *отношение*, s – *отношение*, содержащее один удаляемый кортеж.  *Декартово произведение* r x s  Пусть r и s – отношения *арности* k1 и k2 соответственно. *Декартовым произведением* r x s называется множество кортежей длины k1+k2, первые k1 компонентов которых образуют кортежи, принадлежащие r, а последние k2 – кортежи, принадлежащие s.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **r x s** | | | | | | | a | B | a | b | g | a | | a | B | a | d | a | f | | d | A | f | b | g | a | | d | A | f | d | a | f | | c | B | d | b | g | a | | c | B | d | d | a | f |   *Проекция* http://www.intuit.ru/img/tex/2073bb92aee76936a80a8515aaaf4527.png  *Проекция* http://www.intuit.ru/img/tex/2073bb92aee76936a80a8515aaaf4527.png есть множество кортежей, получаемых из *кортежей отношения* r выбором столбцов с именами Ai1, Ai2, ..., Aim.  Другими словами, это операция построения "вертикального" подмножества, получаемого путем выбора определенных атрибутов и исключения остальных. Повторяющиеся кортежи исключаются.   |  |  | | --- | --- | | **http://www.intuit.ru/img/tex/c499e7139a568c591a635846de26853d.png** | | | A | A | | D | F | | C | D |   **Выбор** ( *селекция* ) http://www.intuit.ru/img/tex/9a6b6580259d1d5e2ef0a1f0c3beebb1.png  Пусть F – формула, образованная: операндами, являющимися константами или именами атрибутов, *арифметическими операторами* сравнения, логическими операторами (и, или, не), тогда выбором (селекцией) http://www.intuit.ru/img/tex/d415f5d938b2364bded5ceae65aba1eb.png называется множество кортежей, компоненты которого удовлетворяют условию, заданному формулой F.   |  |  |  |  | | --- | --- | --- | --- | | http://www.intuit.ru/img/tex/33ee29bb5455d28fd11f57fa37e349ce.png | a | b | a |   Здесь F:(1)=(3) – содержимое первого столбца равно содержимому третьего столбца.  Приведем ряд примеров представления запросов с помощью формальных операций для реляционной модели (СТУДЕНТ, ФАКУЛЬТЕТ, СПЕЦИАЛЬНОСТЬ), рассмотренной выше.  *Пример 1*.  Сформировать список студентов (фамилия).  Рассмотрим схему отношения СТУДЕНТ.  *Атрибут* "Фамилия" обозначен здесь А2 Для ответа на запрос необходимо взять проекцию отношения r1 на столбец А2.  http://www.intuit.ru/img/tex/883ac5110a514c2a667509345c447716.png  *Пример 2*.  Выдать список фамилий и дат рождений студентов, которым на текущую дату (date) больше 35 лет.  Рассмотрим то же *отношение* r1. Сначала выбираем студентов, которым больше 35 лет:  http://www.intuit.ru/img/tex/f206281da338db5446f5eac5ea1d8940.png  Затем берем проекцию полученного отношения на столбцы  http://www.intuit.ru/img/tex/f46405d91e149d4a98f9a930580f5201.png  Заметим, что можно было бы выполнить эти две операции в другой последовательности – сначала *проекция*, а затем *селекция*. Предлагается оценить, какой из этих вариантов лучше по оценке числа выполняемых элементарных действий и объему требуемой памяти.  *Пример 3*.  Выдать список фамилий студентов, обучающихся по специальности "Информационные технологии". Название специальности является атрибутом отношения r3. Если бы в этом отношении присутствовал *атрибут* "фамилия", то задача решалась бы аналогично примеру 2. В отношении r5 присутствует *атрибут* "код студента", а "фамилия" присутствует в отношении r1. Для ответа на этот запрос необходимо связывать по "код студента" *отношение* r3 и *отношение* r1.  Сначала выберем из отношения r3 кортежи с названием специальности "Информационные технологии". Обозначим полученное *отношение* rp1. (Дальнейшие промежуточные отношения будем обозначать последовательно rp1, rp2, rp3 и т.д.).  http://www.intuit.ru/img/tex/761f068b3e6653b06fe368c7f973b09c.png  Далее нас будет интересовать только *атрибут* A1 – "код студента". Поэтому возьмем проекцию на эти столбцы.  http://www.intuit.ru/img/tex/8727b02f0268ad43413a6a27238eb400.png.  Далее необходимо связать отношения r1 и rp2 (склеить таблицы). Для склейки таблиц используется операция *"декартово произведение"*:  rp3=r1xrp2  В отношении r3 присутствуют два одинаковых столбца: A1 из отношения r1 и A1 из отношения rp2. Выбирая из отношения rp3 строки, в которых значения в соответствующих столбцах совпадают, получим сведения о студентах, обучающихся по специальности "Информационные технологии"  http://www.intuit.ru/img/tex/180c3c184348f40fb690d18ceee03880.png,  где A1 \* r1 и A1 \* rp2 обозначают соответственно столбец A1 соответствующей первой и второй составной части *декартова произведения*. Теперь осталось только выбрать фамилии соответствующих студентов  http://www.intuit.ru/img/tex/ecddb9c6a7f4dcc9bcfdb6698ce8ac50.png  Получаем требуемый результат. Заметим, что для экономии действий и памяти, перед тем как склеивать таблицы, целесообразно было сделать *операцию проекции отношения*r1 на столбцы A1, A2. (чтобы не включать в *декартово произведение* лишние столбцы).  Введенные пять основных *операций реляционной алгебры* позволяют реализовать любой запрос к реляционной базе данных. Однако наряду с основными операциями достаточно часто удобно использовать так называемые дополнительные *операции реляционной алгебры* (которые могут быть выражены через основные).  *Пересечение* http://www.intuit.ru/img/tex/c58def0ad74929b65dd53265f2994f41.png  *Пересечением* *отношений* r *и* s *называется множество кортежей, принадлежащих как* r, *так и* s. *Пересечение* *может быть выражено через операции разности*  http://www.intuit.ru/img/tex/2bf3c488083146acd6ea1185bd754866.png  http://www.intuit.ru/img/tex/72f66272f18e12f9ffaa9d88608a24fd.png **-соединение**http://www.intuit.ru/department/database/databases/7/07_001.gif  http://www.intuit.ru/img/tex/72f66272f18e12f9ffaa9d88608a24fd.png -соединение r и s по столбцам Ai и Aj представляет собой множество таких кортежей в *декартовом произведении* r и s, что i -й компонент r находится в отношении http://www.intuit.ru/img/tex/72f66272f18e12f9ffaa9d88608a24fd.png c j-м компонентом s, где http://www.intuit.ru/img/tex/72f66272f18e12f9ffaa9d88608a24fd.png – *арифметический оператор* сравнения. Если http://www.intuit.ru/img/tex/72f66272f18e12f9ffaa9d88608a24fd.png является оператором равенства, то эта операция называется *эквисоединением*  http://www.intuit.ru/department/database/databases/7/07_002.gif  *где* l *– арность отношения* r.  *Пример*.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | **r** | | |  | **S** | | | | | 1 | 2 | 3 |  | 3 | | 1 | | | 4 | 5 | 6 |  | 6 | | 2 | | | 7 | 8 | 9 |  |  | |  | | | **http://www.intuit.ru/department/database/databases/7/07_003.gif** | | | | | | | | 1 | 2 | 3 | 3 | | 1 | | | 1 | 2 | 3 | 6 | | 2 | | | 4 | 5 | 6 | 6 | | 2 | |   Заметим, что в примере 3 две последовательно идущие операции ( *декартово произведение* и *селекция* ) вместе как раз представляют операцию соединения. Причем использование *декартова произведения* для соединения таблиц обязательно обусловливает использование селекции как следующей операции для установления связи между таблицами. Поэтому целесообразно использовать такую объединенную операцию и программно реализовывать в СУБД именно операцию соединения.  *Естественное соединение*http://www.intuit.ru/department/database/databases/7/07_004.gif  Операция применима тогда и только тогда, когда столбцы имеют имена (являются атрибутами). Операция применима к отношениям, у которых есть одинаковые атрибуты.  Пусть  r = (A1, ..., Ak, B1,..., Bn), s = (A1, ..., Ak, C1,..., Cm),  имена A1, ..., Ak совпадают.  Тогдаhttp://www.intuit.ru/department/database/databases/7/07_004.gifопределяется следующим образом  http://www.intuit.ru/department/database/databases/7/07_005.gif  Для подчеркивания важности приведенных *операций реляционной алгебры*, а также для уточнения понятия реляционной СУБД приведем следующее определение одного из ведущих специалистов в области реляционных баз данных К.Дж. Дейта: " **Будем называть систему реляционной, если она поддерживает, по крайней мере, реляционные базы данных, т.е. базы данных, которые могут восприниматься пользователем как таблицы и только как таблицы, операции селекции, проекции и соединения реляционной алгебры, не требуя при этом, чтобы каким-то образом были предопределены физические пути доступа для поддержки этих операций** ".  **Краткие итоги**: В лекции рассматриваются вопросы, связанные с формализацией наиболее распространенной в настоящее время *модели данных СУБД* – реляционной модели. Формальное описание реляционной модели и полученные на этой основе математические методы и алгоритмы позволяют формализовать ряд шагов проектирования реляционной базы данных , получить оптимальную (по определенным критериям) структуру базы данных и эффективные алгоритмы обработки. Здесь рассматривается формализованное описание отношений, формальные средства манипулирования данными в реляционной модели (дано понятие *реляционного исчисления* и *реляционной алгебры*, приводятся основные *операции реляционной алгебры* ). Приводятся примеры представления запросов как последовательность формальных *операций реляционной алгебры*. |

|  |
| --- |
| **8. Лекция: Использование формального аппарата для оптимизации схем отношений: версия для печати и PDA**  Лекция посвящена вопросам оптимизации схем отношений на основе формальных методов *теории реляционных баз данных*. Разбирается пример приведения таблицы к третьей нормальной форме, оптимальной по ряду показателей. Рассматриваются вопросы целостности данных в реляционных СУБД. |
| http://www.intuit.ru/img/empty.gif |
| **http://www.intuit.ru/img/empty.gif** |
| http://www.intuit.ru/img/empty.gif |
| **Цель лекции**: показать возможность эффективного использования формальных методов построения оптимальной (по определенным показателям) структуры реляционной базы данных путем нормализации схем отношений.  **8.1. Проблема выбора рациональных схем отношений**  При представлении концептуальной схемы в виде реляционной модели возможны различные варианты выбора схем отношений. Одни варианты выбора рассматривались в предыдущих разделах (п. 6.2.3), другие получаются объединением (или разбиением) некоторых схем отношений. От правильного выбора схем отношений, представляющих концептуальную схему, в значительной степени будет зависеть эффективность функционирования базы данных.  Рассмотрим для примера конкретную схему отношений и проанализируем её недостатки. Предположим, что данные о студентах, факультетах, специальностях, включены в таблицу со следующей схемой отношения: СТУДЕНТ (Код студента, Фамилия, Название факультета, Название специальности).  Эта схема отношений обусловливает следующие недостатки соответствующей базы данных:   * Дублирование информации (избыточность). У студентов, обучающихся на одном факультете, будет повторяться название факультета. Для разных факультетов будут повторяться специальности. * Потенциальная противоречивость ( *аномалии обновления* ). Если, например, изменится название специальности, то изменяя её в одном кортеже (у одного студента), необходимо изменять и во всех других кортежах, где она присутствует. * Потенциальная возможность потери сведений ( *аномалии удаления* ). При удалении информации о всех студентах, поступающих на определенную специальность, мы теряем все сведения об этой специальности. * Потенциальная возможность невключения информации в базу данных ( *аномалии включения* ). В базе данных будут отсутствовать сведения о специальности, если на ней нет обучающихся студентов.   В *теории реляционных баз данных* существуют формальные методы построения реляционной модели базы данных, в которой отсутствует избыточность и *аномалии обновления*, удаления и включения.  **Нормализация. Первая нормальная форма**.  *Построение рационального варианта схем отношений (обладающего лучшими свойствами при операциях включения, модификации и удаления данных, чем все остальные наборы схем) осуществляется путем так называемой* *нормализации* *схем отношений*. *Нормализация* производится в несколько этапов. На начальном этапе схема отношений должна находиться в первой *нормальной форме* (*1НФ*).  *Отношение находится в первой* *нормальной форме*, *если все атрибуты отношения принимают простые значения (атомарные или неделимые), не являющиеся множеством или кортежем из более элементарных составляющих*.  Рассмотрим следующий пример.  Таблица представляет сущность ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Код студента** | **Фамилия** | **Код экзамена** | **Предмет и дата** | **Оценка** | | 1 | Сергеев | 1 | Математика 5.06.08 | 4 | | 2 | Иванов | 1 | Математика 5.06.08 | 5 | | 1 | Сергеев | 2 | Физика 9.06.08 | 5 | | 2 | Иванов | 2 | Физика 9.06.08 | 5 |   Теперь на пересечении любой строки и любого столбца находится одно значение и, следовательно, данная таблица находится в первой *нормальной форме*.  Далее отношение, представленное в первой *нормальной форме*, последовательно преобразуется во вторую и третью *нормальные формы*. Процесс построения второй и третьей *нормальных форм* будет описан в следующих подразделах. При некоторых предположениях о данных третья *нормальная форма* является искомым наилучшим вариантом.  Если эти предположения не выполняются, то процесс *нормализации* продолжается и отношение преобразуется в четвертую и пятую *нормальные формы*. Построение соответствующих форм описано в литературе и в данной книге не рассматривается.  Прежде чем перейти к построению второй *нормальной формы*, необходимо определить ряд формальных понятий.  **8.2. Функциональные зависимости (зависимости между атрибутами отношения)**  Пусть R(A1, A2, ..., An) – схема отношения, а X и Y – подмножества {A1, A2, ..., An}.  *Функциональная зависимость* **на отношении** R **– это утверждение вида "Если два кортежа** R **совпадают по атрибутам множества** http://www.intuit.ru/img/tex/a301024fbfa52c2ee9dd535b320ef520.png **(т.е. эти кортежи имеют в соответствующих друг другу компонентах одни и те же значения для каждого атрибута множества** X **), то они должны совпадать и по атрибутам множества** http://www.intuit.ru/img/tex/c965c88e0ff3624ab7fe7c483ecc301f.png. **Формально эта зависимость записывается выражением** X -> Y, **причем говорится, что** X **функционально определяет** Y. Часто используется другое утверждание: X *функционально определяет* Y *или* Y *функционально зависит от* X ( *обозначается* X -> Y ) *тогда и только тогда, когда каждое значение множества* X *отношения* R *связано с одним значением множества* Y *отношения* R. *Иначе говоря, если два кортежа* R *совпадают по значению* X, *они совпадают и по значению*Y.  Замечание. Вообще говоря, под термином "отношение" могут подразумеваться два понятия:   * отношение как переменная, которая может принимать разные значения (таблица, в строки и столбцы которой могут быть вписаны разные значения); * отношение, как набор конкретных значений (таблица с заполненными элементами).   Функциональные зависимости характеризуют все отношения, которые могут быть значениями схемы отношения R в принципе. Поэтому единственный способ определить функциональные зависимости – внимательно проанализировать семантику (смысл) атрибутов.  Функциональные зависимости являются, в частности, ограничениями целостности, поэтому целесообразно проверять их при каждом обновлении базы данных.  Пример функциональных зависимостей для отношения ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ  Код студента -> Фамилия  Код студента, Код экзамена -> Оценка  Пример функциональных зависимостей для отношения СТУДЕНТ, приведенного в начале настоящей лекции  Код студента -> Фамилия,  Код студента -> Факультет  Заметим, что последняя зависимость существует при условии, что один студент не может обучаться на нескольких факультетах.  **Полное множество функциональных зависимостей**  Для каждого отношения существует вполне определенное множество функциональных зависимостей между атрибутами данного отношения. Причем из одной или более функциональных зависимостей, присущих рассматриваемому отношению, можно вывести другие функциональные зависимости, также присущие этому отношению.  *Заданное множество функциональных зависимостей для отношения* R *обозначим* F, *полное множество функциональных зависимостей, которые логически можно получить из*F, *называется замыканием* F *и обозначается* F+.  *Если множество функциональных зависимостей совпадает с замыканием данного множества, то такое множество функциональных зависимостей называется полным*.  Введенные понятия позволяют формально определить понятие ключа.  *Пусть существует некоторая схема* R *с атрибутами* A1A2...An, F – *некоторое множество функциональных зависимостей и* X – *некоторое подмножество* R. *Тогда* X *называется ключом, если, во-первых, в* F+ *существует зависимость* X -> A1A2...An *и, во-вторых, ни для какого подмножества* Y, *входящего в* X, *зависимость* Y -> A1A2...An *не принадлежит* F+.  *Полной функциональной зависимостью называется зависимость неключевого атрибута от всего составного ключа*.  *Частичной функциональной зависимостью будем называть зависимость неключевого атрибута от части составного ключа*.  Для вычисления *замыкания множества функциональных зависимостей* используются следующие **правила вывода** ( *аксиомы Армстронга* ):  Пусть известна некоторая схема отношения R{A1, A2, ..., An} с множеством атрибутов U={A1, A2, ..., An} и множество функциональных зависимостей F, заданных на множестве U.  **Аксиома рефлективности**. Если Y входит в X, а X входит в http://www.intuit.ru/img/tex/76642a24c845907568a453e6c306d001.png, то X->Y логически следует из F. Это правило дает тривиальные зависимости, так как в них правая часть содержится в левой части.  **Аксиома пополнения**. Если X->Y и Z есть подмножество U, то XZ->YZ. В данном случае *функциональная зависимость* X->Y либо содержалась в исходном множестве F, либо может быть выведена из F с использованием описываемых аксиом.  **Аксиома транзитивности**. Если X->Y и Y->Z, то X->Z.  Справедлива следующая **теорема**. *Аксиомы Армстронга* *являются полными и надежными*.  Это значит, что используя их мы выведем все возможные функциональные зависимости, логически следующие из F, и не выведем никаких лишних зависимостей.  Существует несколько других правил вывода, которые следуют из *аксиом Армстронга*.  **Правило самоопределения**. X->Х.  **Правило объединения**. Если X->Y и X->Z, то http://www.intuit.ru/img/tex/ae48aa6661d5fd00b24fa6614511b88e.png.  **Правило псевдотранзитивности**. Если X->Y и http://www.intuit.ru/img/tex/10e93871611caff819af0536344488c9.png, то http://www.intuit.ru/img/tex/5daa7145bbe4e2cfcb740a2842d57482.png.  **Правило композиции**. Если X->Y и Z->W, то http://www.intuit.ru/img/tex/8029c2fc24342b5498c010fa0f566ce4.png.  **Правило декомпозиции**. Если X->Y и Z входит в Y, то X->Z.  Надо отметить, что вычисление *замыкания множества функциональных зависимостей* является трудоемкой задачей при достаточно большом количестве атрибутов (за счет выписывания большого количества тривиальных зависимостей).  **8.3. Декомпозиция схемы отношения**  Последовательный переход от одной *нормальной формы* к другой при *нормализации* схем отношений реализуется через декомпозицию. Основной операцией, с помощью которой осуществляется декомпозиция, является проекция.  *Декомпозицией схемы отношения* R = {А1, А2, ...,Аn} *называется замена ее совокупностью подмножеств* R, *таких, что их объединение дает* R. *При этом допускается, чтобы подмножества были пересекающимися*.  Алгоритм декомпозиции основан на следующей теореме.  **Теорема о декомпозиции**. *Пусть* R(A, B, C) – *отношение*, A, B, C – *атрибуты*.  *Если* R *удовлетворяет зависимости* A->B, *то* R *равно соединению его проекций* A, B *и* A, C  R(A, B, C) = R(A, B), R(A, C)  При *нормализации* необходимо выбирать такие декомпозиции, которые обладают свойством соединения без потерь. В этом случае, декомпозиция должна обеспечить то, что запросы (выборка данных по условию) к исходному отношению и отношениям, получаемым в результате декомпозиции, дадут одинаковый результат. Соответствующее условие будет выполняться, если каждый *кортеж отношения* R может быть представлен как *естественное соединение* его проекций на каждое из подмножеств. Для проверки, обладает ли декомпозиция данным свойством, используется специальные алгоритмы, описанные в литературе (в данной книге не рассматриваются).  Вторым важнейшим желательным свойством декомпозиции является свойство сохранения функциональных зависимостей. Стремление к тому, чтобы декомпозиция сохраняла зависимости, естественно. Функциональные зависимости являются некоторыми ограничениями на данные. Если декомпозиция не обладает этим свойством, то для того чтобы проверить, не нарушаются ли при вводе данных условия целостности (функциональные зависимости), нам приходится соединять все проекции.  Таким образом, для правильно построенного проекта базы данных необходимо, чтобы декомпозиции обладали свойством соединения без потерь, и желательно, чтобы они обладали свойством сохранения функциональных зависимостей.  **8.4 .Выбор рационального набора схем отношений путем нормализации**  **Вторая нормальная форма (2НФ)**  *Отношение находится в 2НФ, если оно находится в 1НФ и каждый неключевой атрибут зависит от всего первичного ключа (не зависит от части ключа)*.  Для перевода отношения в 2НФ необходимо, используя *операцию проекции*, разложить его на несколько отношений следующим образом:   1. построить проекцию без атрибутов, находящихся в *частичной функциональной зависимости* от первичного ключа; 2. построить проекции на части *составного ключа* и атрибуты, зависящие от этих частей.   **Третья нормальная форма (3НФ)**  *Отношение находится в 3НФ, если оно находится в 2НФ и каждый ключевой атрибут нетранзитивно зависит от первичного ключа*.  *Отношение находится в 3НФ в том и только том случае, если все неключевые атрибуты отношения взаимно независимы и полностью зависят от первичного ключа*.  *Оказывается, что любая схема отношений может быть приведена к 3НФ декомпозицией, обладающей свойствами соединения без потерь и сохраняющей зависимости*.  **Мотивировка третьей нормальной формы**  *Третья* *нормальная форма* *исключает избыточность и аномалии включения и удаления*.  К сожалению, 3НФ не предотвращает все возможные аномалии.  **Нормальная форма Бойса-Кодда (НФБК)**  *Если в* R *для каждой зависимости* X->A, *где* А *не принадлежит* X, X *включает в себя некоторый ключ, то говорят, что данное отношение находится в* *нормальной форме* *Бойса-Кодда*.  *Детерминантом функциональной зависимости называется минимальная группа атрибутов, от которой зависит некоторый другой атрибут или группа атрибутов, причем эта зависимость нетривиальная*.  *Отношение находится в НФБК тогда и только тогда, когда каждый его детерминант является потенциальным ключом*.  *НФБК является более строгой версией 3НФ. Иными словами, любое отношение, находящееся в НФБК, находится в 3НФ. Обратное неверно*.  **Мотивировка нормальной формы Бойса-Кодда**  *В* *нормальной форме* *Бойса-Кодда не существует избыточности и аномалий включения, удаления и модификации*. Оказывается, что любая схема отношения может быть приведена в *нормальную форму* Бойса-Кодда таким образом, чтобы декомпозиция обладала свойством соединения без потерь. Однако схема отношения может быть неприводимой в НФБК с сохранением зависимостей. В этом случае приходится довольствоваться третьей *нормальной формой*.  **8.5. Пример нормализации до 3НФ**  Для улучшения структуры реляционной базы данных (устранения возможных аномалий) необходимо привести все таблицы базы данных к третьей *нормальной форме* или в более высокой форме (если это возможно). Таким образом, задача сводится к проверке *нормализации* всех сущностей, отображающихся в таблицы базы данных. Если таблица, получающаяся из некоторой сущности, не является таблицей в третьей *нормальной форме*, то она должна быть заменена на несколько таблиц, находящихся в третьей *нормальной форме*.  Продолжим рассмотрение примера с отношением ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ  В начале этой лекции мы привели отношение к первой *нормальной форме*.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Код студента** | **Фамилия** | **Код экзамена** | **Предмет** | **Дата** | **Оценка** | | 1 | Сергеев | 1 | Математика | 5.08.03 | 4 | | 2 | Иванов | 1 | Математика | 5.08.03 | 5 | | 1 | Сергеев | 2 | Физика | 9.08.03 | 5 | | 2 | Иванов | 2 | Физика | 9.08.03 | 5 |   Ключом данного отношения будет совокупность атрибутов – Код студента и Код экзамена.  Для более краткой записи процесса нормализации введем следующие обозначения:  КС – код студента, КЭ – код экзамена, Ф – фамилия, П – предмет, Д – дата, О - оценка.  Выпишем функциональные зависимости  КС, КЭ -> Ф, П, Д, О  КС, КЭ -> Ф  КС, КЭ -> П  КС, КЭ -> Д  КС, КЭ -> О  КЭ -> П  КЭ -> Д  КС -> Ф  В соответствии с определением, отношение находится во второй *нормальной форме* (2НФ), если оно находясь в *1НФ* и каждый неключевой атрибут зависит от первичного ключа и не зависит от части ключа. Здесь атрибуты П, Д, Ф зависят от части ключа. Чтобы избавиться от этих зависимостей необходимо произвести декомпозицию отношения. Для этого используем теорему о декомпозиции.  Имеем отношение R(КС, Ф, КЭ, П, Д, О). Возьмем зависимость КС -> Ф в соответствии с формулировкой теоремы исходное отношение равно соединению его проекцийR1(КС, Ф) и R2(КС, КЭ, П, Д, О).  В отношении R1(КС, Ф) существует *функциональная зависимость* КС -> Ф, ключ КС – составной, не ключевой атрибут Ф не зависит от части ключа. Это отношение находится в 2НФ. Так как в этом отношении нет транзитивных зависимостей, отношение R(КС, Ф) находится в 3НФ, что и требовалось.  Рассмотрим отношение R2(КС, КЭ, П, Д, О) с составным ключом КС, КЭ. Здесь есть зависимость КЭ -> П, КЭ -> Д, КЭ -> П, Д. Атрибуты П,Д зависят от части ключа, следовательно отношение не находится в 2НФ. В соответствии с теоремой о декомпозиции исходное отношение (используем функциональную зависимость КЭ -> П, Д) равно соединению проекций R3(КЭ, П, Д), R4(КС, КЭ, О). В отношении R3( КЭ, П, Д) существуют функциональные зависимости КЭ -> П, КЭ -> Д, КЭ -> П, Д. Зависимости неключевых атрибутов от части ключа нет, следовательно отношение находится в 2НФ. Транзитивных зависимостей в этом отношении так же нет, следовательно отношение находится в 3НФ.  Таким образом, исходное отношение приведено в к трем отношениям, каждое из которых находится в третьей *нормальной форме* R1(КС, Ф), R3(КЭ, П, Д), R4(КС, КЭ, О).  Заметим, что в отношении R4 атрибуты КС, КЭ являются внешними ключами, используемыми для установления связей с другими отношениями. Представим полученную модель в виде диаграммы объектов-связей (ER-диаграммы). Для наглядности и возможности последующего программирования перейдем к английским названиям объектов (отношений) и атрибутов.  Отношение R1 представляет объект student с атрибутами id\_st (первичный ключ), surname.  Отношение R3 представляет объект exam\_st c атрибутами id\_ex (первичный ключ), subject, date.  Отношение R4 представляет объект mark\_st c атрибутами id\_st (внешний ключ), id\_ex (внешний ключ), mark. Первичный ключ здесь id\_st, id\_ex.  Соответствующая ER-диаграмма изображена на [рис. 8.1](http://www.intuit.ru/department/database/databases/8/databases_8.html#image.8.1).  ER-диаграмма, представляющая рассмотренный фрагмент предметной области  **Рис. 8.1.**  ER-диаграмма, представляющая рассмотренный фрагмент предметной области  **8.6. Целостная часть реляционной модели. Реализация условия целостности данных в современных СУБД**  Напомним, что под целостностью базы данных понимается то, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную часть (правильная) информация. Поддержка целостности в реляционных БД основана на выполнении следующих требований.  1. Первое требование называется **требованием целостности сущностей**. Объекту или сущности реального мира в реляционных БД соответствуют кортежи отношений. Конкретно требование состоит в том, что любой кортеж любого отношения отличим от любого другого кортежа этого отношения, т.е., другими словами, любое отношение должно обладать определенным первичным ключом. Это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.  2. Второе требование называется **требованием целостности по ссылкам**. Очевидно, что при соблюдении нормализованности отношений сложные сущности реального мира представляются в реляционной БД в виде нескольких кортежей нескольких отношений. Связь между отношениями осуществляется с помощью миграции ключа.  Пример внешнего ключа.  СТУДЕНТ (Код студента, Фамилия) сдает ЭКЗАМЕН (Код студента, Предмет, Оценка).  Атрибут Код студента сущности ЭКЗАМЕН называется *внешним ключом*, поскольку его значения однозначно характеризуют сущности, представленные кортежами некоторого другого отношения – отношения Студент (мы предполагаем, что поле Код студента является ключом отношения Студент).  Говорят, что отношение, в котором определен внешний ключ, ссылается на соответствующее отношение, в котором такой же атрибут является первичным ключом.  *Требование целостности по ссылкам или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа в ссылающемся отношении в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа либо значение внешнего ключа должно быть неопределенным* (т.е. ни на что не указывать).  Ограничения целостности сущности и по ссылкам должны поддерживаться СУБД. Для соблюдения целостности сущности достаточно гарантировать отсутствие в любом отношении кортежей с одним и тем же значением первичного ключа. (В Access для этого предназначена специальная реализация целочисленного поля – поле типа "Счетчик".) С целостностью по ссылкам дела обстоят несколько более сложно.  Понятно, что при обновлении ссылающегося отношения (вставке новых кортежей или модификации значения внешнего ключа в существующих кортежах) достаточно следить за тем, чтобы не появлялись некорректные значения внешнего ключа.  Но как быть при удалении кортежа из отношения, на которое ведет ссылка?  Здесь существуют три подхода, каждый из которых поддерживает целостность по ссылкам. Первый подход заключается в том, что запрещается производить удаление кортежа, на который существуют ссылки (т.е. сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа). При втором подходе при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным. Наконец, третий подход (каскадное удаление) состоит в том, что при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.  В развитых реляционных СУБД обычно можно выбрать способ поддержания целостности по ссылкам для каждой отдельной ситуации определения внешнего ключа. Конечно, для принятия такого решения необходимо анализировать требования конкретной прикладной области.  Заметим, что все современные СУБД поддерживают и целостность сущностей, и целостность по ссылкам, но позволяют пользователям выключать данные ограничения и, таким образом, строить базы данных, не соответствующие реляционной модели. Опыт показывает, что отход от основных положений реляционной модели приводит к краткосрочному выигрышу – алгоритмы становятся проще, но впоследствии серьезно усложняют задачу, особенно ее сопровождение.  **Краткие итоги**: Лекция посвящена вопросам оптимизации схем отношений (структуры реляционной базы данных) на основе формальных методов *теории реляционных баз данных*. Здесь рассматривается ряд необходимых для этого понятий ( *функциональная зависимость*, *нормальные формы*, *декомпозиция схем отношений* ). Разбирается пример приведения таблицы к третьей нормальной форме, оптимальной по ряду показателей (исключающей избыточность, *аномалии включения* и удаления). Рассматриваются вопросы реализации целостности данных в реляциионных СУБД.  В лекции рассматриваются вопросы использования формального аппарата для оптимизации схем отношений. Сформулирована проблема *выбора рациональных схем отношений*и пути реализации такого выбора путем *нормализации* (последовательного преобразования схемы отношения в ряд *нормальных форм* ). Для формального описания соответствующего процесса определены понятие функциональной зависимости (зависимости между атрибутами отношения), ключа, сформулированы правила вывода множества функциональных зависимостей, понятие *декомпозиции схемы отношения*. Определены первая, вторая, третья *нормальные формы* и *нормальная форма* Бойса-Кодда. Приведен пример *нормализации* до 3НФ. Рассмотрены вопросы реализации условий целостности данных в реляционных СУБД. |

|  |
| --- |
| **9. Лекция: Физические модели данных (внутренний уровень): версия для печати и PDA**  Лекция посвящена вопросам физической организации данных в памяти компьютера. Здесь описывается структура памяти компьютера и представлены *структуры хранения* данных в оперативной и внешней памяти. |
| http://www.intuit.ru/img/empty.gif |
| **http://www.intuit.ru/img/empty.gif** |
| http://www.intuit.ru/img/empty.gif |
| **Цель лекции**: дать представление об основных типовых способах организации данных в памяти ЭВМ в СУБД с оценкой соответствующих моделей по времени доступа к данным в базе данных и по объему занимаемой памяти.  Как уже отмечалось, концептуальная схема, специфицированная к СУБД, автоматически отображается в *структуру хранения* программами СУБД. Внешний пользователь может ничего не знать о том, как его представление о данных физически организовано в памяти вычислительной системы. Тем не менее от физического *размещения данных в памяти ЭВМ* существенно зависит время решения прикладных задач. В связи с этим, даже на одном из начальных этапов проектирования базы данных – этапе выбора СУБД, желательно знать возможности физических *структур хранения*, представляемых конкретными СУБД, и оценивать временные характеристики проектируемой базы данных с учетом этих возможностей.  Способы физической организации данных в различных СУБД, как правило, различны и определяются типом используемой ЭВМ, инструментальными средствами разработки СУБД, а также критериями, которыми руководствуются разработчики СУБД при выборе методов размещения данных и способов доступа к этим данным. Заметим, что наиболее распространенным критерием служит время доступа к данным, однако в качестве критерия может выбираться, например, трудоемкость реализации соответствующих методов.  В настоящей лекции будут рассмотрены типовые физические модели организации данных в конкретных СУБД.  *Физические модели данных* *служат для отображения моделей данных*. Основными понятиями модели данных являются поле, *логическая запись*, *логический файл*. Слово "логический" введено, чтобы отличать понятия, относящиеся к *логической модели данных*, от понятий, относящихся к *физической модели данных*. *Основными понятиями физической модели данных, используемыми для представления логической модели данных, являются поле*, *физическая запись*, *физический файл*. В частности, *логическая запись*, состоящая из полей, может быть представлена в виде физической записи (из тех же полей), *логический файл* – в виде физического файла. Прежде чем конкретизировать понятия, относящиеся к *физической модели данных*, рассмотрим структуру памяти ЭВМ. 9.1. Структура памяти ЭВМ Важнейшей особенностью памяти ЭВМ, в значительной степени определяющей методы организации данных и доступа к ним, является её неоднородность. Существуют два разных типа памяти – оперативная (ОП) и внешняя (ВП), причем процессор работает только с данными из оперативной памяти ([рис. 9.1](http://www.intuit.ru/department/database/databases/9/databases_9.html#image.9.1).).  Схема работы ЭВМ  **Рис. 9.1.**  Схема работы ЭВМ  Как уже многократно отмечалось, базы данных создаются для работы с большими объемами данных, что обусловливает необходимость использования внешней памяти. Поэтому организация данных и доступа к ним должна учитывать как специфику каждого типа памяти, так и способы их взаимодействия.  Отметим основные свойства оперативной памяти:   * единицей памяти является байт; * память прямоадресуема (каждый байт имеет адрес); * процессор выбирает для обработки нужные данные, непосредственно адресуясь к последовательности байтов, содержащих эти данные.   Отметим основные свойства внешней памяти:   * минимальной адресуемой единицей является *физическая запись* ; * для последующей обработки (например, работы с полями) запись должна быть считана в оперативную память; * время чтения записи в ОП на несколько порядков выше времени обработки процессором записи из ОП; * организация обмена осуществляется порциями, т.к. невозможно считать сразу всю базу данных.  9.2. Представление экземпляра логической записи *Логическая запись* представляется в оперативной памяти следующим образом:   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | ***Логическая запись*** | | | |  | **Последовательность байтов ОП** | | | | | Поле 1 | Поле 2 | ... | Поле N | http://www.intuit.ru/img/tex/cb03bcd01df3f76828a6a116d40fc984.png | B1 | B2 | ... | BN | | Тип поля | | | |  | Bi – последовательность байтов ОП, используемая для хранения поля i | | | | | Характеристика поля | | | | | Длина | | | |   Прямая адресация байтов позволяет процессору выбирать для обработки нужное поле.  Заметим, что указанное представление не делает различий для записей в сетевой, иерархической и реляционных моделях. В случае сетевой и *иерархической моделей*некоторые поля могут являться указателями, тогда последовательность байтов, используемая для хранения этих полей, содержит адрес начала последовательности байтов, соответствующей записи – члену отношения.  В большинстве современных СУБД используется формат записей фиксированной длины. В этом случае все записи имеют одинаковую длину, определяемую суммарной длиной полей, составляющих запись. В СУБД другие форматы записей (переменной длины, неопределенной длины) встречаются гораздо реже, поэтому в данной книге эти форматы не рассматриваются. Заметим, что поля записи, принимающие значения существенно разной длины в различных экземплярах записей, в предметной области встречаются достаточно часто. Примером может служить поле резюме в записи СОТРУДНИК. Резюме может составлять полстраницы текста, страницу и т.д. Возникает проблема – как эту информацию переменной длины представить в записи фиксированной длины. Возможным вариантом является установление размера соответствующего поля по максимальному значению. В этом случае у многих экземпляров записи указанное поле будет заполнено не полностью и, таким образом, память ЭВМ будет использоваться неэффективно. Более эффективный и часто используемый в СУБД прием организации таких записей состоит в следующем. Вместо поля (полей), принимающего значение существенно разной длины, в запись включается поле-указатель на область памяти, где будет размещаться значение исходного поля. Как правило, эта область является областью внешней памяти прямого доступа. В процессе ввода соответствующего значения в выделенной области занимается столько памяти, какова длина этого значения.  На [рис. 9.2](http://www.intuit.ru/department/database/databases/9/databases_9.html#image.9.2) представлен пример вышеуказанного представления *экземпляров записей* из N полей, причем поле N принимает значения соответственно разной длины у разных*экземпляров записей*.  Представление полей переменной длины  **Рис. 9.2.**  Представление полей переменной длины  Конкретной реализацией такой схемы является поле типа МЕМО в СУБД (dBase III+, FoxPro, Access и т.д.). 9.3. Организация обмена между оперативной и внешней памятью *Единицей обмена данными между оперативной и внешней памятью является* *физическая запись*. *Физическая запись* читается (записывается) за одно обращение к внешней памяти. В частности, *физическая запись* может соответствовать одному экземпляру логической записи. Число обращений к внешней памяти при работе с базой данных определяет время отклика системы. В связи с этим для уменьшения числа обращений к БД при работе с ней увеличивают длину физической записи (объединяют в одну*физическую запись* несколько экземпляров логических записей). В этом случае *физическую запись* называют также блоком, число k экземпляров логических записей, составляющих *физическую запись*, – коэффициентом блокировки.  Ввод исходных данных в БД осуществляется следующим образом:   * в ОП последовательно вводятся k экземпляров логических записей (кортежей); * введенные k экземпляров объединяются в *физическую запись* (блок); * *физическая запись* заносится во внешнюю память.   Ввод k *экземпляров записей* исходной таблицы, составляющих i-ю *физическую запись*, изображен на [рис. 9.3](http://www.intuit.ru/department/database/databases/9/databases_9.html#image.9.3).  Схема занесения записей во внешнюю память  **Рис. 9.3.**  Схема занесения записей во внешнюю память  Обработка данных, хранящихся во внешней памяти, осуществляется следующим образом:   * *физическая запись* (блок) считывается в оперативную память; * обрабатываются экземпляры логических записей внутри блока (выбираются нужные поля, производится сравнение ключевого поля с заданным значением, осуществляется корректировка полей, выполняются операции удаления и т.п.).   *В некоторых СУБД (например, MS SQL Server) единицей обмена между оперативной и внешней памятью является страница (вид физической записи, размер которой фиксирован и не зависит от длины логической записи)*. Организация обмена между оперативной и внешней памятью в этом случае аналогична описанной выше. Отличие здесь будет состоять в том, что экземпляры логических записей формируются в буфере, размером со страницу (если размер страницы не кратен длине логической записи, страница может быть заполнена неполностью, *физическая запись* на внешнем носителе, соответственно, будет заполнена не полностью). 9.4. Структуры хранения данных во внешней памяти ЭВМ В современных СУБД наибольшее распространение получили *табличные модели* данных. В связи с этим, а также для большей определенности в настоящем разделе мы будем говорить о *структурах хранения* для табличной модели. Однако отметим, что некоторые из рассматриваемых ниже *структур хранения* могут использоваться и для представления сетевых и *иерархических моделей*.  В качестве внешней памяти мы рассматриваем наиболее распространенную в современных ЭВМ память прямого доступа. Память прямого доступа дает возможность обращения к любой записи, если известен её адрес. Для упрощения изложения мы не будем конкретизировать ряд служебных полей, которые содержит *физическая запись*, и их рассмотрение опускаем. 9.4.1. Последовательное размещение физических записей В этой структуре хранения записи в памяти размещаются последовательно друг за другом. Как уже отмечалось, считаем, что все записи имеют равную длину. Физический адрес записи может быть легко вычислен по номеру записи (для вычисления необходимо знать формат соответствующей физической записи).  *Физическая запись* с номером I содержит логические записи с номерами  http://www.intuit.ru/img/tex/c77529625d0296b639a0e950eccdce04.png  знаком http://www.intuit.ru/img/tex/e0bd74adfaff9af80bec347e6f6659e9.png обозначим ближайшее целое, большее или равное N/k, – целое сверху.  Рассмотрим, как реализуются основные элементарные операции модели данных в этой структуре хранения, и оценим число этих операций. Напомним, что с точки зрения пользователя в табличной модели данных эти операции являются операциями над строками (столбцами) таблицы. Поиск записи с заданным значением ключа При последовательной структуре хранения поиск может осуществляться только перебором. Читается первая *физическая запись*, в ОП она разбивается на k логических записей (разблокируется), заданное значение ключа сравнивается со значением ключа каждой логической записи. При несовпадении читается следующая *физическая запись*и процесс повторяется. В лучшем случае нужная запись будет найдена за одно обращение, в худшем – необходимо считать все физические записи. Среднее число обращений к внешней памяти для поиска нужной записи ТР определяется следующей формулой  http://www.intuit.ru/img/tex/9ddc4d8d0302056f48838f9b64c7c6dd.png  где N – число логических записей, k – коэффициент блокировки, http://www.intuit.ru/img/tex/e0bd74adfaff9af80bec347e6f6659e9.png – число физических записей. Чтение записи с заданным значением ключа Сначала необходимо найти нужную запись (смотри операцию "поиск"). После окончания операции "поиск" нужная запись уже считана в ОП. Число обращений к ВП равно ТР. Корректировка записи Сначала необходимо найти нужную запись (смотри операцию "поиск"). После окончания операции "поиск" в ОП найденная *логическая запись* корректируется, формируется*физическая запись* (блок) и заносится во внешнюю память по тому адресу, откуда она была считана. Число обращений к ВП равно ТР+1. Удаление записи Аналогична операции корректировки. Служебное поле соответствующей логической записи помечается как "удаленная запись". Число обращений к ВП равно ТР+1. Добавление записи Рассмотрим два случая. В первом случае пользователь вводит новую *логическую запись* в конец таблицы. Тогда вводимая *логическая запись* добавляется в конец файла. Она заносится либо в последнюю *физическую запись* (если в ней меньше k логических записей – блок неполон), для чего эта запись должна быть считана в ОП, или формируется новая *физическая запись*, которая заносится в конец файла. Число обращений к ВП равно соответственно либо 2, либо 1.  Во втором случае пользователь вводит новую *логическую запись* в указываемую им i-ю строку таблицы ( i=1, 2, ..., n ). В этом случае читается *физическая запись* с номером http://www.intuit.ru/img/tex/04e9b8a5b6e2f62ced4c8d3387931203.png, содержащая i-ю *логическую запись*. Если соответствующая *физическая запись* содержит пустые логические записи, то добавляемая запись вставляется в этот блок, блок записывается на свое место в ВП. Число обращений к ВП равно 2. Если указанная *физическая запись* содержит k экземпляров логических записей исходной таблицы, читается *физическая запись* с номером http://www.intuit.ru/img/tex/547ca26d5a6a01e0f031e235c716fbff.png. Если эта *физическая запись* содержит пустые логические записи, добавляемая запись вставляется в этот блок, блок записывается на свое место в ВП. Суммарное число обращений в этом случае будет на единицу больше и равно 3.  Если физические записи с номерами http://www.intuit.ru/img/tex/04e9b8a5b6e2f62ced4c8d3387931203.png и http://www.intuit.ru/img/tex/547ca26d5a6a01e0f031e235c716fbff.png содержат по k экземпляров исходных логических записей, необходимо формировать дополнительную *физическую запись*. Соответствующий блок будет содержать добавляемую *логическую запись* и k-1 пустых логических записей. Блоки с номерами http://www.intuit.ru/img/tex/c18724420dbf1f2ac8a9deabc2803bf7.pngпереписываются на одну позицию ниже (сдвигаются). Сформированная *физическая запись* заносится на освободившееся место (место записи с номером http://www.intuit.ru/img/tex/547ca26d5a6a01e0f031e235c716fbff.png ).  В лучшем случае (i = N) ни один блок не сдвигается. В худшем случае (i = 1) сдвигаются все блоки. Среднее число обращений к ВП для перезаписи блоков (чтение + запись) составит http://www.intuit.ru/img/tex/221662840418f32eacd02e98f4d2dcf8.png. Тогда суммарное число обращений к ВП при добавлении записи в этом случае будет равно http://www.intuit.ru/img/tex/23b78dab8cc1a52a6d2dcaf23cde5b57.png.  Заметим, что если записи упорядочены по значениям ключа поиск может производиться дихотомическим методом и число обращений к внешней памяти будет пропорционально не http://www.intuit.ru/img/tex/20d3dc0b45ba405354f246b02f691c7d.png а http://www.intuit.ru/img/tex/d4a844dbd3e89f00572c536cc00bf175.png т.е. существенно меньше. Однако добавление записи потребует для сохранения упорядоченности, как правило, сдвига большого числа записей. Поэтому размещение физических записей с упорядочением их по значениям ключа в СУБД не используется. 9.4.2. Размещение физических записей в виде списковой структуры Основная проблема в использовании изложенного в п. 9.4.1 способа организации записей состоит в отображении добавления логической записи в произвольное место таблицы. При этом приходится переписывать в памяти (сдвигать на одну позицию) физические записи, соответствующие логическим записям таблицы, расположенным ниже места вставки добавляемой строки. Соответствующую проблему можно устранить, используя для представления физических записей связный список ([рис. 9.4](http://www.intuit.ru/department/database/databases/9/databases_9.html#image.9.4)).  Список физических записей  **Рис. 9.4.**  Список физических записей  Кроме этого списка в ВП формируется список свободных элементов ("пустых" физических записей), элементы которого используются при вводе новой записи с данными ([рис. 9.5](http://www.intuit.ru/department/database/databases/9/databases_9.html#image.9.5)).  Напомним, что каждая *физическая запись* состоит, как и ранее, из k логических записей.  Список свободных элементов  **Рис. 9.5.**  Список свободных элементов  Рассмотрим, как реализуются основные элементарные операции модели данных в этой структуре хранения. Поиск записи с заданным значением ключа Заметим, что упорядочение записей по значениям ключа не дает здесь ускорения процедуры поиска. Это связано с тем, что после ряда добавлений новых записей и удаления каких-то имеющихся записей физическая и логическая последовательность записей в списке будут существенно различаться. При этом будет невозможно по номеру записи определить ее адрес и обращаться к записи, соответствующей середине таблицы, для реализации дихотомического метода поиска. Поэтому поиск можно вести только с помощью перебора. В ОП читается первая запись списка, разблокируется, значения ключевых полей логических записей этой физической записи сравниваются с заданным значением. Если значения совпали, нужная запись найдена, если не совпали, из записи выбирается адрес следующей записи списка, читается эта запись. Далее процедура повторяется. Среднее число обращений к ВП будет равно, как и в 9.4.1, http://www.intuit.ru/img/tex/4a490ca50d7a78eb2121de2571a6fe1d.png. Чтение записи После завершения предыдущей операции запись считана в ОП. Оценка числа обращений к ВП та же. Корректировка записи Считанная запись корректируется и заносится в ВП на свое место (по своему адресу). Число обращений к ВП на единицу больше, чем при чтении. Удаление записи Заметим, что мы говорим об операциях над логическими записями. Операция удаления логической записи аналогична операции корректировки. Служебное поле соответствующей логической записи помечается как "удаленная запись". Сформированная *физическая запись* заносится в ВП. Число обращений к ВП равно ТР+1. Добавление записи Для определенности будем считать, что задан ключ логической записи, после которой должна быть добавлена новая запись. Осуществляется операция поиска и чтения физической записи, в которой расположена запись с ключом РК. Если в этом блоке есть *логическая запись*, помеченная как удаленная, добавляемая запись заносится на ее место. Блок записывается в ВП. Число обращений к ВП равно ТР+1. Если в этом блоке нет логических записей, помеченных как удаленные, необходимо добавлять новую*физическую запись*, выбираемую из списка свободных элементов. С этой целью адрес связи найденной ранее физической записи заменяется на адрес начала списка свободных элементов.  Читается первая *физическая запись* списка свободных элементов. Адрес связи этой записи заменяет адрес начала пустого списка. В ОП формируется новая *физическая запись*, содержащая добавляемую *логическую запись*. В качестве ее адреса связи заносится адрес связи из физической записи, предшествующей добавляемой. Каждая из этих записей заносится в ВП. Число обращений к ВП при добавлении записи будет примерно равно ТР+3.  Рассмотренный метод организации *структуры хранения* достаточно эффективно решает проблемы добавления и удаления записей, но не уходит от перебора при поиске нужной записи. 9.4.3. Использование индексов (индексирование) Как уже отмечалось, упорядочение записей позволяет использовать дихотомический метод поиска нужной записи и тем самым существенно сократить одну из основных составляющих времени поиска – число обращений к ВП. Однако при этом возникают проблемы с добавлением записей, связанные с необходимостью перезаписи части физических записей (сдвига).  *Для того чтобы использовать дихотомический поиск и не перемещать физические записи при добавлении новых записей, используется так называемое логическое упорядочение физических записей* ( *индексирование* ). Основная *структура хранения* содержит записи исходной таблицы и представлена в виде неупорядоченной последовательности физических записей (см. п. 5.4.1). *Для возможной реализации дихотомического поиска по определенному ключу создается дополнительная структура хранения (так называемый индекс)*. Число записей в индексе равно числу записей исходной таблицы (числу физических записей в основной структуре хранения). Каждая запись индекса имеет два поля: ключевое поле записи основной структуры и указатель – адрес записи основной структуры с соответствующим значением ключа.  Записи индекса (индексного файла) упорядочены по значению ключа. Адреса связи этих записей определяют логическое упорядочение записей основной *структуры хранения*. Пример соответствующей *структуры хранения* приводится в предположении k=1 на [рис. 9.6](http://www.intuit.ru/department/database/databases/9/databases_9.html#image.9.6).  Рассматриваемую *структуру хранения* называют еще инвертированным списком. Смысл этого термина состоит в следующем. Можно было бы упорядочить записи основной*структуры хранения*, не переставляя их, а объединив в соответствующий упорядоченный список. В нашем случае адреса связи как бы удаляются из списка и включаются в состав файла-индекса (инвертируются). Поэтому полученная структура интерпретируется как инвертированный список.  Индексирование  **Рис. 9.6.**  Индексирование  Поиск нужной записи по заданному значению ключа осуществляется в индексном файле *методом половинного деления*. Заметим, что так как записи индекса содержат всего два поля, суммарный объем записей индекса невелик, поэтому индекс, как правило, целиком считывается для обработки в ОП за одно обращение к ВП. После того как в индексном файле обнаружена искомая запись, по адресу связи читается полная соответствующая запись основной *структуры хранения*. Если необходим поиск по другому ключу, строится еще один индекс по соответствующему ключу. Таким образом, по любому ключу поиск можно осуществлять дихотомическим методом.  Оценим число обращений к ВП при реализации элементарных операций. Соответствующие оценки сделаны для случая, когда *физическая запись* состоит из одной логической записи (коэффициент блокировки k равен 1). Расчет оценок для произвольного k производится по аналогии с расчетами пп. 9.4.1–9.4.2. Поиск записи с заданным значением ключа Из ВП читается индексный файл (число обращений к ВП для этого зависит от объема индексного файла, как правило, невелико и много меньше числа записей N ). После нахождения нужной записи в индексном файле читается соответствующая запись основного файла (одно обращение к ВП). Чтение записи В ходе операции поиска искомая запись считана в ОП. Корректировка записи Считанная запись корректируется и заносится на свое место (еще одно обращение к ВП). Удаление записи Найденная запись помечается как удаленная в основном файле, соответствующая запись в индексном файле удаляется, измененный индекс записывается в ВП. Число обращений к ВП в этом случае по сравнению с числом обращений к ВП при поиске увеличивается на два. Добавление записи Добавляемая запись заносится в конец основного файла. Формируется новая запись индекса, соответствующая добавляемой записи. Записи индекса переупорядочиваются по значениям ключа, и индекс заносится в ВП. Число обращений к ВП в этом случае, в основном, определяется чтением-записью индекса.  Таким образом, использование индексов позволяет ценой некоторого увеличения объема используемой памяти (за счет индекса) существенно сократить время реализации основных операций. В связи с этим *индексирование* используется во многих современных СУБД. 9.4.4. В-дерево *Структура* *В-дерева* *(сбалансированное дерево) является следствием дальнейшего расширения концепции использования индексов (строится индекс над индексом) и представляет собой многоуровневые индексы*.  *В-дерево* строится следующим образом. Последовательность записей, соответствующая записям исходной таблицы, упорядочивается по значениям первичного ключа. Логические записи объединяются в блоки (по k записей в блоках).  Значением ключа блока является минимальное значение ключа у записей, входящих в блок. Последовательность блоков представляет собой последний уровень *В-дерева*. Строится индекс предыдущего уровня. Записи этого уровня содержат значение ключа блока следующего уровня и указатель-адрес связи соответствующего блока; записи этого уровня также объединяются в блоки (по k записей). Затем аналогично строится индекс более высокого уровня и т.д., пока количество записей индекса на определенном уровне будет не более k.  Рассмотрим процедуру работы с B-деревом на примере. Пусть имеется файл экземпляров логических записей, ключи которых принимают значения 2, 7, 8, 12, 15, 27, 28, 40, 43, 50. Для определенности возьмем k=2 (в блок объединяем по 2 *экземпляра записей*). Построенное для этого примера *В-дерево* изображено на [рис. 9.7](http://www.intuit.ru/department/database/databases/9/databases_9.html#image.9.7) (для упрощения рисунка на уровне 4 представлены только ключи логических записей и не представлены значения других полей этих записей).  В-дерево  **Рис. 9.7.**  В-дерево  В блоках указано значение ключа соответствующего блока. Значение k принято равным 2.  По построению *В-дерева* все исходные записи находятся на одном расстоянии от верхнего индекса (дерево является сбалансированным).  Рассмотрим реализацию основных операций. Поиск и чтение записи с заданным значением ключа Читается верхний индекс. Сравниваем заданное значение ключа со значением ключа последней записи индекса. Если заданное значение ключа больше, чем значение ключа очередной записи индекса (если такая запись имеется), или равно ему, то по адресу связи, указанному в текущей записи, читается блок записей индекса следующего уровня. Далее процесс повторяется.  Считаем, что все блоки расположены в ВП. Тогда число обращений к ВП при поиске информации будет равно числу уровней дерева. Число уровней дерева равно минимальному значению l, при котором выполняется условие kl >= N ( N – число логических записей). Модификация (корректировка) записи После поиска и чтения записи изменяются корректируемые поля. Если корректируется не ключ записи, то измененная запись заносится на свое место. Если изменено значение ключа, то старая запись удаляется (в соответствующем блоке появляется "пустая" запись), а измененная запись заносится так же, как вновь добавляемая. Удаление записи После поиска найденная запись удаляется (в соответствующий блок на место этой записи заносится "пустая" запись). Добавление записи Прежде всего определяется, где должна быть расположена добавляемая запись с заданным значением ключа. Процедура поиска блока, где должна быть расположена эта запись, аналогична вышеописанной процедуре поиска записей с заданным значением ключа. Если в найденном блоке низшего уровня есть "пустая" запись, добавляемая запись заносится в этот блок (с необходимым переупорядочением записей внутри блока).  Если в соответствующем блоке низшего уровня нет пустого места, блок делится на два блока. В первый из них заносится [k/2] записей, во второй заносятся остальные. Значением ключа каждого из указанных блоков будет являться, как и описано ранее, минимальное значение ключей у записей, входящих в блок. Добавляемая запись заносится в тот блок, значение ключа которого меньше значения ключа добавляемой записи. Появление нового блока с новым значением ключа обусловливает необходимость формирования соответствующей новой записи в индексе на предыдущем уровне. Эта запись содержит новое значение ключа нового блока и указатель на его месторасположение. Процедура добавления такой записи аналогична описанной выше. Находится блок предыдущего уровня, куда должна быть помещена эта запись. Если в блоке есть пустое место, запись добавляется в блок, если блок полон, он делится на два блока, запись заносится в один из блоков, формируется запись индекса предыдущего уровня и т.д.  Возможен вариант, когда придется делить блок самого верхнего уровня и формировать еще один уровень дерева.  Рассмотрим для примера, изображенного на [рис. 9.7](http://www.intuit.ru/department/database/databases/9/databases_9.html#image.9.7), добавление записи с ключом 10.  1. Сравнение на первом уровне.  2<10<43  Движение по левой ветви.  2. Сравнение на втором уровне.  2<10<15  Движение по левой ветви.  3. Сравнение на третьем уровне.  2<8<10  Движение по правой ветви.  Искомый блокhttp://www.intuit.ru/department/database/databases/9/09_001.gif  4. Блок заполнен.  Он делится на 2 блока  http://www.intuit.ru/department/database/databases/9/09_002.gifhttp://www.intuit.ru/department/database/databases/9/09_003.gif  Сравнение 8<10<12.  Запись с ключом 10 заносится в блок 1  http://www.intuit.ru/department/database/databases/9/09_004.gifhttp://www.intuit.ru/department/database/databases/9/09_003.gif  На низшем уровне появилась новая запись с значением ключа 12. Необходимо добавление новой записи с ключом 12 и указателем на запись низшего уровня к индексу предыдущего уровня.  5. Запись с ключом 12 уровня 3 должна добавляться в блокhttp://www.intuit.ru/department/database/databases/9/09_005.gif. Блок полон, он делится на два блока  http://www.intuit.ru/department/database/databases/9/09_006.gifhttp://www.intuit.ru/department/database/databases/9/09_002.gif  Сравнение 8<12.  Запись добавляется во второй блокhttp://www.intuit.ru/department/database/databases/9/09_001.gif  6. На уровне 3 появился блок с новым ключом 8. Необходимо добавление новой записи с ключом 8 и указателем на соответствующий блок уровня 3 на уровне 2.  7. Запись с ключом 8 уровня 2 должна добавиться в блокhttp://www.intuit.ru/department/database/databases/9/09_007.gif. Блок полон, он делится на два блока.  http://www.intuit.ru/department/database/databases/9/09_006.gifhttp://www.intuit.ru/department/database/databases/9/09_008.gif  2<8<15  Запись добавляется в блок 1http://www.intuit.ru/department/database/databases/9/09_005.gif.  8. На уровне 2 появился блок с новым ключом 15, необходимо добавление новой записи с ключом 15 и указателем на соответствующий блок уровня 2 на уровне 1.  9. Запись с ключом 15 уровня 1 должна добавляться в блокhttp://www.intuit.ru/department/database/databases/9/09_005.gif. Блок полон, он делится на два блока.  http://www.intuit.ru/department/database/databases/9/09_006.gifhttp://www.intuit.ru/department/database/databases/9/09_010.gif  2<15<43  Запись с ключом 15 добавляется в первый блок  http://www.intuit.ru/department/database/databases/9/09_007.gifhttp://www.intuit.ru/department/database/databases/9/09_010.gif  10. Необходимо сформировать еще один уровень дереваhttp://www.intuit.ru/department/database/databases/9/09_009.gif.  Полученная структура будет иметь вид, представленный на [рис. 9.8](http://www.intuit.ru/department/database/databases/9/databases_9.html#image.9.8).  В-дерево после добавления элемента  **Рис. 9.8.**  В-дерево после добавления элемента  Необходимо заметить, что используемый прием деления пополам полностью заполненного блока при добавлении в него записи приведет к тому, что блоки будут заполнены, в среднем, наполовину. Тогда процедура добавления записи будет существенно менее трудоемкой (если в нужном блоке есть место, запись добавляется в этот блок и вышестоящие уровни не перестраиваются).  *Структура хранения* в виде B-дерева позволяет эффективно проводить операции поиска, чтения, удаления, модификации с оценкой числа обращений к внешней памяти числом уровней дерева l http://www.intuit.ru/img/tex/3339cc7143498249b66f3751054d4583.png, что существенно меньше числа обращений при переборе http://www.intuit.ru/img/tex/e0bd74adfaff9af80bec347e6f6659e9.png.  Процедура добавления записи тоже достаточно эффективна. Соответствующая *структура хранения*, в частности, используется в отечественной СУБД НИКА (ранее использовалась в системе ИНЕС) и на реальных задачах показала высокую эффективность. 9.4.5. Размещение записей с использованием хэширования Как в любом другом способе организации *структур хранения*, логические записи группируются в физические записи (блоки) по k штук. Однако в отличие от всех других способов организации *структур хранения* здесь выбран особенный способ группировки. *Определенным образом выбирается так называемая* *хэш-функция* f. *Аргументом этой функции является значение x первичного ключа логической записи. Тогда* f(x) *указывает адрес расположения блока, в котором должна находиться логическая запись со значением ключа* x.  Функция f должна, по возможности, равномерно распределять значения x по физическим блокам. Обсуждению возможных *хэш-функций* посвящено достаточно много литературы, поэтому здесь мы не будем касаться этого вопроса. Можно лишь добавить, что иногда, исходя из специфики множества значений x первичного ключа, можно построить функцию f, удовлетворяющую всем необходимым условиям. Таким образом, *логическая запись* таблицы со значением x первичного ключа размещается в блоке внешней памяти по адресу f(x). В этом блоке может находиться не более k записей. Может оказаться, что выбранная функция отображает в один адрес памяти (один блок) более k записей. Возникает так называемая коллизия. Возможным способом разрешения коллизий является использование дополнительной области переполнения следующим образом. Если очередная запись распределяется с помощью функции хэширования в блок, а он полностью заполнен, то в области переполнения формируется список записей, соответствующих этому блоку, с включением в него указанной записи, а в сам блок заносится указатель – адрес связи на первую запись этого списка. Возможны и другие способы разрешения коллизий.  Рассмотрим реализацию основных операций и дадим оценку числа обращений к ВП при их выполнении. Поиск записи с заданным значением ключа и чтение По заданному значению ключа x подсчитывается значение функции f(x). Далее из ВП считывается блок, находящийся по адресу f(x). В ОП внутри этого блока перебором ищется нужная запись. Если записей в блоке нет, то по указателю в блоке (адресу связи) читается первая запись списка переполнения, относящаяся к этому блоку. Далее необходимая запись ищется по этому списку. Число обращений к ВП при этом равно:   * единице, если запись находится в блоке; * единице плюс число записей в соответствующем этому блоку списке области переполнения (как правило, небольшое число).  Модификации записи Осуществляется поиск и чтение записи, затем в ОП модифицируются поля записи (не являющиеся первичным ключом), запись заносится на свое место. Число обращений к ВП в этом случае на единицу больше, чем при чтении записи. Если модифицируется значение ключа, то занесение записи осуществляется как ввод новой записи (добавление). Удаление записи Осуществляется поиск и чтение записи. Если удаляемая запись находилась в блоке основной памяти, на ее место заносится "пустая" запись (или признак "пустой" записи). Если удаляемая запись находилась в списке области переполнения, удаление ее производится по правилам удаления элемента списка. Число обращений к ВП при удалении находится примерно в тех же пределах, что и для предыдущих операций. Добавление записи При добавлении записи со значением ключа x подсчитывается адрес соответствующего блока f(x). Блок считывается в ОП. Если в нем есть место, запись заносится в блок, блок записывается в ВП по своему адресу. Если блок заполнен, из него выбирается адрес начала списка записей, переполняющих блок. Далее добавление записи в список производится по правилам добавления элемента в список. Число обращений к ВП при добавлении записей находится примерно в тех же пределах, что и для предыдущих операций.  Таким образом, описанная *структура хранения* с использованием хэширования является наиболее эффективной (из рассмотренных выше) по критерию минимизации числа обращений к ВП при реализации основных операций. 9.4.6. Комбинированные структуры хранения Необходимо заметить, что в СУБД могут использоваться как каждая из вышерассмотренных структур в отдельности, так и их комбинация. Так, например, в ряде промышленных систем UNIBAD, БАНК для ЭВМ типа IBM 360/370 (ЕС ЭВМ), PARADOX для персональных ЭВМ *используются следующие комбинации методов*:   * *размещение записей по первичному ключу организовано с использованием хэширования;* * *последовательность записей по вторичному ключу задается с помощью* *списковой структуры*.   **Краткие итоги**: Лекция посвящена вопросам физической организации данных в памяти компьютера (организации *структур хранения*). Физические модели представления данных жестко заложены в структуру конкретной СУБД и различны в различных системах управления базами данных. Заметим, что в данной лекции рассматриваются не*структуры хранения* конкретной СУБД, а некоторые типовые *структуры хранения*, на основе которых и реализуются физические модели организации данных в конкретных СУБД. Здесь описывается двухуровневая структура памяти компьютера как среда размещения данных; организация обмена между внешней и оперативной памятью, определяющая специфику обработки данных. Представлены типовые физические модели ( *структуры хранения* данных) во внешней памяти ЭВМ (последовательное размещение физических записей, размещение физических записей в виде *списковой структуры*, использование индексов, организация данных в виде *В-дерева*, размещение записей с использованием хэширования, а также комбинированные *структуры хранения* ). Для основных *структур хранения* сделана оценка числа действий при выполнении операций поиска данных, чтения, занесения данных, модификации (корректировки), удаления. |

|  |
| --- |
| **11. Лекция: Программное обеспечение работы с современными базами данных: версия для печати и PDA**  В лекции рассматриваются общие принципы организации программного обеспечения работы с реляционными базами данных, включающего: - создание и ведение базы данных; - создание пользовательских приложений, включающих разработку пользовательского интерфейса по работе с базой данных. |
| http://www.intuit.ru/img/empty.gif |
| **http://www.intuit.ru/img/empty.gif** |
| http://www.intuit.ru/img/empty.gif |
| **Цель лекции**: дать общее представление об основных задачах *программного обеспечения баз данных*, существующих подходов к решению этих задач, в том числе и о*структурированном языке* запросов SQL. 11.1. Основные задачи программного обеспечения баз данных При работе с реляционными базами данных можно условно выделить две основные задачи:   * собственно работа с базой данных , включающая создание и ведение базы данных (создание структур таблиц, добавление записи в таблицу, удаление записи, обновление, выборка нужной записи ); * создание пользовательских приложений, включающих разработку пользовательского интерфейса по работе с базой данных.   Для решения указанных задач современные СУБД в своем составе могут содержать следующие программные средства: языки процедурного пошагового программирования, средства визуального программирования (графический интерфейс, диспетчер проекта, мастера и построители), средства создания объектно-ориентированных приложений. Кроме этого, при разработке пользовательских программ во многих СУБД допускается использование других языков программирования, а также использование библиотек разного рода. Так, например, при работе с СУБД ACCESS можно использовать язык программирования ACCESS , мастер ACCESS и язык программирования VISUAL BASIC.  При работе с клиент-серверными системами ситуация немного сложнее. Здесь в работе участвуют два типа компьютеров (сервер и клиент) и, соответственно, различают клиентское и серверное программное обеспечение. Серверное программное обеспечение включает язык программирования, поддерживающий создание и ведение базы данных, также реализацию поступающих от клиентов запросов пользователей к базе данных. Пользовательские приложения создаются и работают на компьютерах-клиентах. Именно эти компьютеры должны иметь, наряду со средствами формирования запросов к базе данных, *средства разработки интерфейса*. В связи с этим, для клиент-серверных СУБД программное обеспечение разделяется на две части: программное обеспечение – клиент и программное обеспечение – сервер. Заметим, что наряду с программным обеспечением – клиент, при разработке пользовательских программ в конкретной СУБД могут использоваться другие языки программирования, специальные библиотеки, другие системы программирования (определенные для этой СУБД). В качестве примера в таблице приводятся возможные варианты использования программного обеспечения для организации клиент-серверного взаимодействия в СУБД Microsoft SQL Server.   |  |  | | --- | --- | | **Таблица 11.1. Возможные варианты использования программного обеспечения в СУБД MS SQL Server** | | | **Средства ведения баз данных на сервере MS SQL** | **Средства разработки клиентских приложений** | | Службы SQL-сервер (MS SQL server и др.) | * Программное обеспечение клиента SQL-сервер (Transact SQL, SQL Server Query Analyzer и др.) * MS Access (ODBC) * MS Visual Basic * MS Visual Studio * MS *Visual FoxPro* * Java (JDBC) * Borland Delphi * Borland C++Builder и др. * Библиотеки функций (API, ODBC и др.) |   Полное рассмотрение всего спектра программного обеспечения работы СУБД очень обширно и выходит за рамки данного пособия. Поэтому в данной работе будет рассмотрены только средства создания и ведения базы данных. 11.2. Проблемы создания и ведения реляционных баз данных При создании базы данных и организации работы с ней возникают три основные проблемы:   * собственно создание базы данных (создание таблиц, индексов, ограничений целостности); * обеспечение безопасности и разграничения доступа; * организация доступа к элементам таблицы (выборка, редактирование, удаление, добавление).   Первая проблема может быть решена посредством создания в каждой конкретной СУБД некоторой утилиты, позволяющей пользователю в определенный момент осуществлять все необходимые действия по созданию базы данных. Однако это не полностью решает проблему. Такая утилита не позволяет создать таблицу динамически во время работы прикладной программы, не позволяет, например, добавить в таблицу сформированнный во время работы пользовательской программы столбец. Необходимы средства, дающие возможность формирования во время работы прикладной программы запроса на изменение структуры и содержания базы данных. То же самое можно сказать и о решении второй проблемы.  Более подробно рассмотрим возможный путь решения третьей проблемы. Организация доступа к базе данных является важнейшей функцией информационной системы. Пользователи постоянно работают с данными. Рассмотрим простой пример. Пусть у нас есть таблица СТУДЕНТ, хранящая информацию следующего рода:  СТУДЕНТ (Код студента, Фамилия, Имя, Отчество, Дата поступления).  Теперь мы хотим выполнить некоторый запрос к базе данных, результатом которого должны стать те строки таблицы СТУДЕНТ, для которых дата поступления окажется больше 01.06.2006. Рассмотрим последовательность действий для реализации данного запроса.   1. Получаем доступ к таблице СТУДЕНТ и устанавливаем указатель текущей строки на первую строку таблицы. 2. Анализируем поле "Дата поступления " в текущей строке. 3. Если значение "Дата поступления" > "01.06.2006", распечатываем на экране данные об абитуриенте. 4. Если таблица не кончилась, перемещаем указатель текущей строки на следующую строку и переходим к шагу 2, иначе заканчиваем работу.   Любой человек, знакомый с программированием, легко представит себе реализацию подобного алгоритма на любом языке программирования высокого уровня. Вот, в частности, пример реализации на *Object Pascal*:  Table.First;  while (not Table.Eof) do  begin  if FieldByName("Дата поступления").Value >  "01.06.2006"  then List.Add(FieldByName("Фамилия").AsString);  Table.Next;  end;  В этом случае разработчик приложения сам организует работу по выборке данных, программируя каждое движение по таблице (осуществляет навигацию по таблице). *Такой подход к обработке данных, ориентированный на последовательную работу с отдельными записями, называется навигационным*. Здесь на конкретном языке программирования мы описываем процедуру - последовательность действий, необходимых для получения результата. Языки, в которых используется такой подход, называются процедурными. Очевидно, что с возрастанием сложности запроса существенно возрастает сложность процедуры и, соответственно, объем текста программы.  При работе с информационной системой пользователь реализует свои запросы к базе данных с помощью разработанных программистами прикладных программ. При навигационном подходе должны быть запрограммированы все возможные запросы. Очевидно, что заранее предугадать все запросы, потребность в которых может возникнуть и запрограммировать их, невозможно. С учетом того, что подавляющее большинство пользователей не владеет навыками программирования, это означает, что объем их действий будет ограничен рамками написанных программ, а именно, теми запросами, реализация которых предусмотрена заранее.  Кроме того, должен учитываться механизм взаимодействия в рамках архитектуры "клиент – сервер". Пользовательская программа выполняется на компьютере–клиенте. Запрос к базе данных реализуется компьютером – сервером. Необходим механизм формирования в пользовательской программе клиента запроса к базе данных сервера. В этом случае навигационный подход неприемлем. *В связи с этим, для работы с базами данных разработан и используется другой подход, основанный на использовании так называемых языков запросов, которые задают не последовательность необходимых действий, а условия, которым должен удовлетворять результат* (при добавлении столбца, выборке записи, добавлении записи и т. п.). Такой подход решает все три вышеперечисленные проблемы.  *С этой целью разработан и активно используется во всех базах данных – специальный язык запросов SQL*. Особо отметим, что основой языка являются *операции реляционной алгебры*.  *Язык SQL* *(Structured Query Language – структурированный язык запросов) применяется для общения пользователя с реляционной базой данных* **и состоит из трех частей**[[[23]](http://www.intuit.ru/department/database/databases/popup.lit.html#23)]:   * DDL (*Data Definition Language*) – язык определения данных. Предназначен для создания базы данных (таблиц, индексов и т.д.) и редактирования ее схемы. * *DCL* (Data *Control Language*) – язык управления данными. Содержит операторы для разграничения доступа пользователей к объектам базы данных. * DML (*Data Manipulation Language*) – *язык обработки данных*. Содержит операторы для внесения изменений в содержимое таблиц базы данных.   Как видно из написанного выше, SQL решает все рассмотренные ранее вопросы, предоставляя пользователю достаточно простой и понятный *механизм доступа к данным*, не связанный с конструированием алгоритма и его описанием на языке программирования высокого уровня. Так, вместо указания того, как необходимо действовать, пользователь при помощи операторов SQL объясняет СУБД, что ему нужно сделать. Далее СУБД сама анализирует текст запроса и определяет, как именно его выполнять.  В архитектуре "клиент – сервер" *язык SQL* занимает очень важное место. Именно он используется как язык общения клиентского программного обеспечения с серверной СУБД, расположенной на удаленном компьютере. Так, клиент посылает серверу запрос на языке SQL, а сервер разбирает его, интерпретирует, выбирает план выполнения, выполняет запрос и отсылает клиенту результат.  Посмотрим, как выглядит запрос на языке SQL, решающий задачу о выборке студентов по дате поступления.  SELECT Фамилия  FROM Студент  WHERE Дата поступления > "01.06.2006"  Может сложиться ложное впечатление, что появление языка SQL является альтернативой языков программирования высокого уровня. Это не соответствует действительности. Выполнение запроса средствами SQL все равно сводится к работе с отдельными записями, и от этого никуда не уйти. Важно понимать, что появление языка SQL дало, по крайней мере, две новые возможности.   1. Появился новый уровень абстракции между пользователем и СУБД. Этот уровень находится ближе к пользователю, чем уровень программирования на языке высокого уровня, что снижает требования к квалификации пользователей. 2. Многие типовые задачи, возникающие при работе с базами данных, и ранее решаемые каждым программистом по-своему (зачастую дублируя действия другого программиста) решены реализацией языка SQL. Таким образом, отпала необходимость самостоятельного решения многих проблем, решенных в СУБД соответствующим образом. Язык SQL предоставляет средства для доступа к этим типовым возможностям СУБД.  11.3. Понятие языка SQL и его основные части11.3.1. История возникновения и стандарты языка SQL История возникновения языка SQL восходит к 1970 году [[[5]](http://www.intuit.ru/department/database/databases/popup.lit.html#5)], когда доктор Е.Ф. Кодд предложил реляционную модель в качестве новой модели базы данных. Для доказательства жизнеспособности новой модели данных внутри компании IBM был создан мощный исследовательский проект, получивший название System/R. Проект включал разработку собственно реляционной СУБД и специального языка запросов к базе данных. Так в начале 70-х годов появился первый исследовательский прототип реляционной СУБД. Для этого прототипа разрабатывались и опробовались разные языки запросов, один из которых получил название *SEQUEL* (Structured English Query Language). С момента создания и до наших дней этот язык претерпел массу изменений, но идеология осталась неизменной.  Период с 1979 года (окончание проекта System/R) до настоящего времени характеризуется развитием и совершенствованием языка SQL и его постоянно увеличивающейся ролью в индустрии, связанной с созданием и эксплуатацией баз данных. Совершенно очевидно, что язык никогда не получил бы мирового признания, если бы на него не было никаких стандартов. Стандартизация – важная часть технологических процессов конца XX века. Именно наличие разработанных и официально признанных стандартов позволило утвердиться многим современным технологиям (не только в индустрии разработки программного обеспечения, но и в многих других сферах человеческой деятельности). Как обстоит дело со стандартами языка SQL и их поддержкой в распространенных СУБД?  Когда ведут речь о стандартах в области, связанной с разработкой программного обеспечения, обычно подразумевают две организации:   * ANSI (American National Standards Institute) – Американский национальный институт стандартов; * ISO (*International Standards* Organization) – Международную организацию по стандартизации.   Работа над официальным стандартом языка SQL началась в 1982 году [8] в рамках комитета ANSI. В 1986 году (обратите внимание, сколько времени ушло на разработку стандарта и согласование деталей!) был утвержден первый вариант стандарта ANSI, а в 1987 году этот стандарт был утвержден и ISO. В 1989 году стандарт претерпел незначительные изменения, но именно этот вариант получил название SQL-1 или *SQL-89*. В чем особенность *SQL-89*? За время разработки стандарта (1982–1989 гг.) были созданы, представлены на рынке и активно использовались несколько различных СУБД, в которых в том или ином виде был реализован некоторый диалект языка SQL. С учетом того, что разработкой стандартов занимались те же люди, кто внедрял SQL в СУБД, *стандарт SQL-89* представлял собой плод множества компромиссов, приведших к наличию в нем большого количества "белых пятен", т.е. мест, которые не были описаны, а отданы на усмотрение разработчиков диалекта. В результате чуть ли не все имеющиеся диалекты стали совместимыми со стандартом, но особой пользы это не принесло.  Следующая реализация стандарта была призвана решить эту проблему. В результате длительных обсуждений и согласований в 1992 году был принят новый стандарт ANSI SQL-2 или SQL-92. SQL-92, который заполнил многие "белые пятна", впервые добавив в стандарт возможности, еще не реализованные в существующих коммерческих СУБД.  Работа над стандартизацией продолжается и далее. Появились *стандарты SQL-1999*, SQL-2003. Тем не менее, все эти стандарты не решили всех проблем, связанных с наличием нескольких диалектов языка. Как правило, разработчики как игнорировали, так и игнорируют некоторые положения стандарта, с одной стороны, отказываясь реализовывать некоторые его части и, с другой стороны, реализуя то, что отсутствует в стандарте. Несмотря на имеющиеся отличия, все коммерческие СУБД поддерживают некоторое ядро языка, описанное в стандарте, одинаково. Отличий не очень много, они не носят слишком принципиального характера. Хотя каждая СУБД по-прежнему поддерживает свой диалект языка.  В систему управления базами данных Microsoft SQL Server входит язык Transact-SQL, разработанный на основе одного из стандартов SQL. 11.3.2. Достоинства языка SQL Для ознакомления с достоинствами языка обратимся к соответствующей литературе [[[5]](http://www.intuit.ru/department/database/databases/popup.lit.html#5)]. Вот некоторые из них:   * межплатформенная переносимость; * наличие стандартов; * одобрение и поддержка компанией IBM (СУБД DB2); * поддержка со стороны компании Microsoft (СУБД SQL Server, протокол ODBC и технология ADO); * реляционная основа; * высокоуровневая структура; * возможность выполнения специальных *интерактивных запросов*; * обеспечение программного доступа к базам данных; * возможность различного представления данных; * полноценность как языка, предназначенного для работы с базами данных; * возможность динамического определения данных; * поддержка архитектуры клиент/сервер; * поддержка корпоративных приложений; * расширяемость и поддержка объектно-ориентированных технологий; * возможность доступа к данным в Интернете; * интеграция с языком Java (протокол JDBC); * промышленная инфраструктура.  11.3.2. Общая характеристика SQL *Язык запросов SQL* основан на операциях реляционной алгебры и, таким образом ориентирован на работу с множествами (отношениями), а не с отдельными записями. Как и в*реляционной алгебре*, операндами языка являются отношения (таблицы), результатами выполнения операции также являются отношения (таблицы). Таким образом, *язык SQL*предназначен для выполнения операций над таблицами, причем как над таблицами в целом (создание, удаление, изменение структуры), так и над данными таблиц (выборка, изменение, добавление и удаление). Отметим, что в явном виде *язык SQL* не является универсальным языком программирования в обычном понимании. В нем отсутствуют операторы условного перехода, организации циклов, позволяющие управлять ходом выполнения программы. Поэтому *язык SQL* относится к классу непроцедурных языков программирования. Это именно *язык запросов к базе данных*, который служит исключительно для организации базы данных и работы с ней. Как уже отмечалось выше, для разработки прикладных программ необходимо использовать другие базовые средства программирования, в который операторы языка SQL будут встраиваться. Языку SQL посвящено большое количество литературы, в том числе и учебников. Подробное изучение языка SQL не входит в задачи настоящего курса, это может занимать отдельный курс. Заметим, что этому языку посвящено большое количество литературы, в том числе и учебников. В связи с этим, здесь будут изложены только общие сведения о языке, как фундаментальном инструменте работы с базами данных. Терминология *Под запросом, реализуемым с помощью языка SQL- запросов к базе данных, понимается команда, предназначенная для выполнения (и выполняемая) системой управления базами данных определяемого этой командой действия с базой данных*.  *Запрос реализуется с помощью операторов языка SQL. Операторы состоят из отдельных логических частей, называемых предложениями. Стандарты языка SQL регламентируют синтаксис операторов*. Несмотря на то, что *язык SQL* работает с реляционной базой данных, вместо термина "отношение" здесь используется термин "таблица", вместо терминов "кортеж" и "атрибут" используются соответственно термины "строка" и "столбец". Разновидности SQL Как отмечалось выше, в отличие от "обычных" языков программирования в SQL отсутствует возможность объявления переменных, нет инструкции IF, нет цикла FOR и т.д. Собственно программирование (разработка прикладных программ) на подобном языке практически невозможно. Поэтому к настоящему моменту используются следующие технологии (режимы) работы с базой данных на языке SQL (в некоторых источниках эти технологии называют разновидностями языка SQL):   * формирование непосредственно пользователем запроса на языке SQL в интерактивном режиме ( **интерактивный SQL** ); * формирование запроса на языке SQL в прикладной программме (программный или встроенный SQL):   + статическое формирование запроса ( **статический SQL** );   + динамическое формирование запроса ( **динамический SQL** );   + формирование запроса с помощью библиотек ( **API – интерфейсы вызова подпрограмм** ).   **В интерактивном режиме** работы с базой данных: пользователь работает с базой данных в прямом диалоге: вводит запрос на языке SQL – получает результат, вводит другой запрос – получает другой результат и т.д.  **Встроенный SQL** представляется операторами языка SQL, встроенные в прикладные программы, написанные на других языках программирования (в других программных средах). Это дает возможность работы с базой данных с помощью прикладных программ, написанных на других алгоритмических языках, но требует включения дополнительных средств, обеспечивающих интерфейс между операторами языка SQL и соответствующим языком программирования.  При статическом использовании языка ( **статический SQL** ) в текст прикладной программы включаются конкретные операторы SQL, и после компиляции исходной программы в выполняемый модуль жестко включаются соответствующие этим операторам функции SQL. Изменения в вызываемых функциях могут здесь определяться только изменениями параметров операторов SQL, инициируемых с помощью переменных языка программирования.  При динамическом использовании языка ( **динамический SQL** ) формирование SQL-запросов, соответствующие вызовы SQL-функций для обращения к базе данных осуществляется динамически в ходе выполнения программы.  Еще одним способом динамического формирования SQL-запросов в прикладной программе является обращение к соответствующим SQL-функциям с помощью специальных**интерфейсов программирования приложений** (библиотек функций, разработанных для связи прикладной программы и СУБД посредством SQL-запросов).  В настоящем пособии для всех указанных технологий (разновидностей SQL) будут приведены основные идеи и рассмотрены ключевые концепции. Интерактивный SQL будет рассмотрен более подробно, чем программный. Детальное рассмотрение статического, динамического SQL и различных API-интерфейсов (ODBC, JDBC, DB Library и др.) выходит за рамки нашего курса.  **Краткие итоги**: В лекции рассматриваются общие принципы организации программного обеспечения работы с реляционными базами данных, включающего   * создание и ведение базы данных; * создание пользовательских приложений, включающих разработку пользовательского интерфейса по работе с базой данных.   Рассматриваются подходы к организации доступа к данным (навигационный подход и подход, основанный на использовании интерпретируемых языков запросов). Дается общее представление о языке SQL (история возникновения и стандарты языка SQL, достоинства языка SQL, основная терминология, технологии работы). |

|  |
| --- |
| **12. Лекция: Основные операторы языка SQL. Интерактивный SQL: версия для печати и PDA**  В лекции дается общая характеристика операторов языка SQL, используемых, в частности, для работы с базой данных в интерактивном режиме (создание таблиц, выбор информации из таблиц, добавление, удаление и модификация элементов). Приводятся примеры запросов к базе данных на языке SQL |
| http://www.intuit.ru/img/empty.gif |
| **http://www.intuit.ru/img/empty.gif** |
| http://www.intuit.ru/img/empty.gif |
| **Цель лекции**: дать общую характеристику операторов языка SQL и показать, как записываются основные запросы к базе данных на языке SQL (в интерактивном режиме). 12.1. Общее представление об основных операторах языка SQL Как уже отмечалось в [лекции 11](http://www.intuit.ru/department/database/databases/11/), все операторы языка SQL разделяются на три составные части: DDL – язык определения данных, *DCL* – язык управления данными, DML – *язык обработки данных*.  Приведем примеры основных операторов из вышеуказанных частей (без описания синтаксиса). Описание синтаксиса операторов SQL можно посмотреть в многочисленных книгах по языку SQL, в меню "Справка" конкретных СУБД.  **Операторы разграничения доступа пользователей к объектам базы данных (DCL)**.  GRANT – создание в системе безопасности записи, разрешающей пользователю работать с данными или выполнять определенные операции SQL.  DENY - создание в системе безопасности записи, запрещающей доступ для определенной учетной записи.  **Операторы определения данных (язык DDL)**.  Соответствующие операторы предназначены для создания, удаления, изменения основных объектов модели данных реляционных СУБД: таблиц, представлений, индексов.  CREATE TABLE <имя> - создание новой таблицы в базе данных.  DROP TABLE <имя> - *удаление таблицы* из базы данных.  ALTER TABLE <имя> - изменение структуры существующей таблицы или ограничений целостности, задаваемых для данной таблицы.  При выполнении аналогичных операций с представлениями или индексами в указанных операторах вместо служебного слова TABLE записывается слово VIEW (представление) или слово INDEX (индекс)  **Операторы манипулирования данными (язык DML)**.  Операторы DML работают с базой данных и используются для изменения данных и получения необходимых сведений.  SELECT – выборка строк, удовлетворяющих заданным условиям. Оператор реализует, в частности, такие *операции реляционной алгебры* как "селекция" и "проекция".  UPDATE – изменение значений определенных полей в строках таблицы, удовлетворяющих заданным условиям.  INSERT – вставка новых строк в таблицу.  DELETE – удаление строк таблицы, удовлетворяющих заданным условиям. Применение этого оператора учитывает принципы поддержки целостности, поэтому он не всегда может быть выполнен корректно. 12.2 Интерактивный режим работы с SQL (интерактивный SQL) Соответствующий режим предусматривает непосредственную работу пользователя с базой данных по следующему алгоритму: используя прикладную программу (клиентское приложение) или стандартную утилиту, входящую в СУБД, пользователь:   * устанавливает соединение с БД (подтверждая наличие прав доступа); * вводит соответствующий оператор SQL, при необходимости в режиме диалога вводит дополнительную информацию; * инициирует выполнение команды.   Текст запроса поступает в СУБД, которая:   * осуществляет синтаксический анализ запроса (проверяет, является ли запрос корректным); * проверяет, имеет ли пользователь право выполнять подобный запрос (например, пользователь, у которого определены права только на чтение, пытается что-то удалить); * выбирает, каким образом осуществлять выполнение запроса – *план выполнения* запроса; * выполняет запрос; * результат выполнения отсылает пользователю.   Схема взаимодействия пользователя и СУБД с использованием интерактивного SQL приводится на [рис. 12.1](http://www.intuit.ru/department/database/databases/12/databases_12.html#image.12.1).  Схема работы интерактивного SQL  **Рис. 12.1.**  Схема работы интерактивного SQL 12.3. Использование языка SQL для выбора информации из таблицы Выборка данных осуществляется с помощью оператора SELECT, который является самым часто используемым оператором языка SQL. Синтаксис оператора SELECT имеет следующий вид:  SELECT [ALL/DISTINCT] <список атрибутов>/\*  FROM <список таблиц>  [WHERE <условие выборки>]  [ORDER BY <список атрибутов>]  [GROUP BY <список атрибутов>]  [HAVING <условие>]  [UNION<выражение с оператором SELECT>]  В квадратных скобках указываются элементы, которые могут в запросе отсутствовать.  Ключевое слово ALL означает, что результатом будут все строки, удовлетворяющие условию запроса, в том числе и одинаковые строки. DISTINCT означает, что в результирующий набор не включаются одинаковые строки. Далее идет список атрибутов исходной таблицы, которые будут включены в таблицу-результат. Символ \* означает, что в таблицу-результат включаются все атрибуты исходной таблицы.  Обязательным ключевым словом является слово FROM, за ним следуют имена таблиц, к которым осуществляется запрос.  В предложении с ключевым словом WHERE задаются условия выборки строк таблицы. В таблицу-результат включаются только те строки, для которых условие, указанное в предложении WHERE, принимает значение истина.  Ключевое слово ORDER BY задает операцию упорядочения строк таблицы-результата по указанному списку атрибутов.  В предложении с ключевым словом GROUP BY задается список атрибутов группировки (разъяснение этого и последующего ключевого слова будет представлено немного позднее).  В предложении HAVING задаются условия, накладываемые на каждую группу.  Отдельно отметим, что ключевые слова FROM, WHERE, ORDER BY используются аналогичным образом и в других операторах манипулирования данными языка SQL.  Рассмотрим реализацию запросов для конкретного примера, представленного в [лекции 8](http://www.intuit.ru/department/database/databases/8/) (см. [рис. 8.1](http://www.intuit.ru/department/database/databases/8/2.html#image.8.1))  **Выдать список всех студентов**.  SELECT \*  FROM student  или  SELECT id\_st, surname  FROM student  Заметим, что если добавить к данному запросу предложение ORDER BY surname, то список будет упорядочен по фамилии. По умолчанию подразумевается, что сортировка производится по возрастанию. Если необходимо упорядочение по убыванию, после имени атрибута добавляется слово DESC.  **Выдать список оценок, которые получил студент с кодом "1"**.  SELECT id\_st, mark  FROM mark\_st  Where id\_st = 1  **Выдать список кодов студентов, которые получили на экзаменах хотя бы одну двойку или тройку**.  В предложении WHERE можно записывать выражение с использованием *арифметических операторов* сравнения (<, >, и т.д.) и логических операторов ( AND, OR, NOT ) как и в обычных языках программирования.  SELECT id\_st, mark  FROM mark\_st  WHERE ( MARK >= 2 ) AND ( MARK <= 3 )  Наряду с операторами сравнения и логическими операторами для составления условий в языке SQL (из-за специфики области применения) существуют ряд специальных операторов, которые, как правило, не имеют аналогов в других языках. Вот эти операторы:   * IN – вхождение в некоторое множество значений; * BETWEEN – вхождение в некоторый диапазон значений; * LIKE – проверка на совпадение с образцом; * IS NULL – проверка на неопределенное значение.   Оператор IN используется для проверки вхождения в некоторое множество значений. Так, запрос  SELECT id\_st, mark  FROM mark\_st  WHERE mark IN (2,3)  дает тот же результат, что и вышеуказанный запрос (выведет идентификаторы всех абитуриентов, получивших хотя бы одну двойку или тройку на экзаменах).  Того же результата можно добиться, используя оператор BETWEEN:  SELECT id\_st, mark  FROM mark\_st  WHERE mark BETWEEN 2 AND 3  **Выдать список всех студентов, фамилии которых начинаются с буквы А**.  В этом случае удобно использовать оператор LIKE.  Оператор LIKE применим исключительно к символьным полям и позволяет устанавливать, соответствует ли значение поля образцу. Образец может содержать специальные символы:  \_ (символ подчеркивания) – замещает любой одиночный символ;  % (знак процента) – замещает последовательность любого числа символов.  SELECT id\_st, surname  FROM student  WHERE surname LIKE 'А%'  Очень часто возникает необходимость произвести вычисление минимальных, максимальных или средних значений в столбцах. Так, например, может понадобиться вычислить средний балл. Для осуществления подобных вычислений SQL предоставляет специальные *агрегатные функции*:   * MIN – минимальное значение в столбце; * MAX – максимальное значение в столбце; * SUM – сумма значений в столбце; * AVG – среднее значение в столбце; * COUNT – количество значений в столбце, отличных от NULL.   Следующий запрос считает среднее среди всех баллов, полученных студентами на экзаменах.  SELECT AVG(mark)  FROM mark\_st  Естественно, можно использовать *агрегатные функции* совместно с предложением WHERE:  SELECT AVG(mark)  FROM mark\_st  WHERE id\_st = 100  Данный запрос вычислит средний балл студента с кодом 100 по результатам всех сданных им экзаменов.  SELECT AVG(mark)  FROM mark\_st  WHERE id\_ex = 10  Данный запрос вычислит средний балл студентов по результатам сдачи экзамена с кодом 10.В дополнение к рассмотренным механизмам *язык SQL* предоставляет мощный аппарат для вычисления агрегатных функций не для всей таблицы результатов запроса, а для разных значений по группам. Для этого в SQL существует специальная конструкция GROUP BY, предназначенная для указания того столбца, по значениям которого будет производиться группировка. Так, например, мы можем вычислить средний балл по всем экзаменам для каждого студента. Для этого достаточно выполнить следующий запрос:  SELECT id\_st, AVG(mark)  FROM mark\_st  GROUP BY id\_st  Все это, как обычно, может быть совмещено с предложением WHERE. При этом, не вдаваясь в тонкости выполнения запроса внутри СУБД, можно считать, что сначала выполняется выборка тех строк таблицы, которые удовлетворяют условиям из предложения WHERE, а потом производится группировка и агрегирование.  Приведем запрос, который вычисляет средний балл по оценкам, полученным на экзамене с кодом 100, для каждого студента.  SELECT id\_st, AVG(mark)  FROM mark\_st  WHERE id\_ex = 100  GROUP BY id\_st  Заметим, что группировка может производиться более чем по одному полю.  Для запросов, содержащих секцию GROUP BY существует важное ограничение: такие запросы могут включать в качестве результата столбцы, по которым производится группировка, и столбцы, которые содержат собственно результаты агрегирования.  Для того чтобы форматировать вывод, существуют различные возможности SQL. Так, например, допустимым является включение текста в запрос. Рассмотрим пример того, как это делается:  SELECT 'Средний балл=', AVG(mark)  FROM mark\_st  WHERE id\_ex = 10  В результате данного запроса пользователь увидит не просто некоторое число, а число, сопровожденное поясняющим текстом. 12.4. Использование SQL для выбора информации из нескольких таблиц До сих пор мы рассматривали выбор информации из единственной таблицы. Можно запрашивать информацию из нескольких таблиц, реализуя описанные в соответствующем разделе учебника *реляционные операции*. Стоит упомянуть, что полное рассмотрение темы выходит за рамки данного учебника. Подробно этот вопрос можно изучить при помощи, например, [[[5]](http://www.intuit.ru/department/database/databases/popup.lit.html#5), [[23]](http://www.intuit.ru/department/database/databases/popup.lit.html#23)]. Рассмотрим некоторые примеры того, как это делается.  Как правило, в тех случаях когда возникает необходимость выбирать информацию из разных таблиц, они тем или иным образом связаны друг с другом, например отношениями один к многим или один к одному по некоторому полю.  Еще раз вернемся к примеру из [лекции 8](http://www.intuit.ru/department/database/databases/9/). Рассмотрим соответствующую *ER-диаграмму* ([рис. 12.2](http://www.intuit.ru/department/database/databases/12/databases_12.html#image.12.2).).  Пример связанных таблиц  **Рис. 12.2.**  Пример связанных таблиц  В этом примере тоже присутствуют связанные таблицы. Рассмотрим таблицы student, mark\_st и exam\_st.  Таблица mark\_st связана с таблицей exam\_st по полю id\_ex.  Таблица mark\_st связана с таблицей student по полю id\_st.  Допустим, требуется распечатать список студентов с оценками, которые они получили на экзаменах. Для этого необходимо выполнить следующий запрос:  SELECT student.surname, mark\_st.id\_ex, mark\_st.mark  FROM student, mark\_st  WHERE student.id\_st = mark\_st.id\_st  Отметим следующие изменения по сравнению с запросами к одной таблице.   1. В секции FROM указаны две таблицы. 2. Так как таблиц стало больше одной, появилась некоторая неоднозначность при упоминании полей. Так, во многих случаях неизвестно, из какой таблицы из списка FROM брать поле. Для устранения неоднозначности имена полей указываются с префиксом – именем таблицы. Имя таблицы от имени поля отделяется точкой. 3. В предложении WHERE указано условие соединения таблиц.   Нетрудно заметить, что использование префиксов-имен таблиц сильно загромождает запрос. Для того чтобы избежать подобного загромождения, используются псевдонимы. Так, можно переписать предыдущий запрос следующим образом:  SELECT E.surname, M.id\_ex, M.mark  FROM student E, mark\_st M  WHERE E.id\_st = M. id\_st 12.5. Использование SQL для вставки, редактирования и удаления данных в таблицах Для добавления данных в таблицу в стандарте SQL предусмотрена команда INSERT. Рассмотрим ряд примеров запросов.  INSERT INTO mark\_st  VALUES (1, 2, 5)  Данный запрос вставляет в таблицу mark\_st строку, содержащую значения, перечисленные в списке VALUES. Если не нужно указывать значение какого-то поля, можно присвоить ему NULL:  INSERT INTO mark  VALUES (1, 2, NULL)  В случае если необходимо использование для некоторых полей значений по умолчанию, SQL позволяет явно указать, какие поля необходимо заполнить конкретными данными, а какие – значениями по умолчанию:  INSERT INTO mark\_st (id\_st, id\_ex)  VALUES (1, 2)  Для удаления данных из таблицы существует команда DELETE:  DELETE  FROM student  Этот запрос удаляет все данные из таблицы student.  Можно ограничить диапазон удаляемой информации следующим образом:  DELETE  FROM student  WHERE surname > 'И'  Для обновления данных используется команда UPDATE.  UPDATE mark\_st  SET mark = '5'  WHERE id\_st = 100 AND id\_ex = 10  При помощи этого запроса изменится на "5" оценка у студента с кодом 100 по экзамену с кодом 10. 12.5. Язык SQL и операции реляционной алгебры *Язык SQL* является средством выражения мощного математического аппарата теории множеств и *реляционной алгебры*. В данном разделе рассматривается связь операторов языка SQL с операциями *реляционной алгебры* и теории множеств. Операция объединения Средствами языка SQL операция объединения представляется следующим образом:  SELECT \*  FROM A  UNION  SELECT \*  FROM B Операция разности Средствами языка SQL операция разности представляется следующим образом:  SELECT \*  FROM A  EXCEPT  SELECT \*  FROM B Операция проекции SELECT Fieldi1, ..., Fieldin  FROM A Операция выборки (селекции) SELECT \*  FROM A  WHERE (<condition>) Операция пересечения SELECT \*  FROM A  INTERSECT  SELECT \*  FROM B Операция соединения, эквисоединения http://www.intuit.ru/img/tex/d97485e1839824ce9a8f645aee45fb4d.png  Если http://www.intuit.ru/img/tex/5771268cf9b2865a96b627e0d41a9101.png – операция "=", то это эквисоединение. Операция естественного соединения Пусть есть отношения A(X1, ..., Xn, A1, ..., Am) и B(X1, ..., Xn, B1, ..., Br).  SELECT A.X1, ..., A.Xn, A.A1, ..., A.Am, B.B1, ..., B.Br  FROM A, B  WHERE (A.X1 = B.X1) AND ... AND (A.Xn = B.Xn)  **Краткие итоги**: В лекции дается общая характеристика операторов языка SQL, используемых, в частности, для работы с базой данных в интерактивном режиме (создание таблиц, выбор информации из таблиц, добавление, удаление и модификация элементов). Дается понятие интерактивного режима работы с SQL. Рассматриваются основные*операторы SQL*, используемые для манипулирования данными (выбор информации из таблиц, добавление, удаление и модификация элементов). Приводятся примеры записи запросов к базе данных на языке SQL с использованием операторов select, insert, update, delete. Рассматривается связь между операциями *реляционной алгебры* и операторами языка SQL. |

|  |
| --- |
| **14. Лекция: Направления развития баз данных: версия для печати и PDA**  В лекции рассматриваются перспективные направления в теории и практике создания баз данных – объектно-ориентированные и распределенные базы данных, а также новое направление в аналитической обработке данных - хранилища данных |
| http://www.intuit.ru/img/empty.gif |
| **http://www.intuit.ru/img/empty.gif** |
| http://www.intuit.ru/img/empty.gif |
| **Цель лекции**: выделить основные черты в новых направлениях развития теории и практики создания баз данных (новые свойства, присущие объектно-ориентированным и распределенным базам данных) и хранилищ данных. 14.1. Объектно-ориентированный подход к организации баз данных В начале 90-х годов XX века начались активные попытки по внедрению объектно-ориентированных технологий в отрасль проектирования и разработки баз данных. Бытовала точка зрения о том, что соответствующие технологии быстро вытеснят все остальные, так же как и во многих других программистских отраслях, но ничего подобного не произошло. Объектно-ориентированное программирование Рассмотрим термин "объектно-ориентированное программирование". Заметим, что это термин, принятый преимущественно в российской литературе. В западной литературе [[[26]](http://www.intuit.ru/department/database/databases/popup.lit.html#26)] под этим понимается сразу три аспекта:   * Объектно-ориентированный анализ – *OOA*, *object-oriented analysis*. **Объектно-ориентированный анализ – это методология, при которой требования к системе воспринимаются с точки зрения** *классов* **и** *объектов* **, выявленных в предметной области**. * *Объектно-ориентированное проектирование* – *OOD*, *object-oriented design*. **Объектно-ориентированное проектирование – это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы**. * Объектно-ориентированное программирование – OOP, *object-oriented programming*. **Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности** *объектов* **, каждый из которых является экземпляром определенного** *класса***, а классы образуют иерархию наследования**.   Здесь и далее по тексту условимся не отступать от традиций и понимать под объектно-ориентированным программированием (ООП) сразу три указанных выше аспекта.  *Основой объектно-ориентированной технологии является так называемая объектная модель, которая возникает как результат объектно-ориентированной декомпозиции*. Она выделяет основные абстракции предметной области, определяет *классы* абстракций и выясняет, какими данными (атрибутами) описывается каждая абстракция, какую функциональность эти абстракции должны обеспечивать. В отличие от традиционных технологий программирования объектно-ориентированная технология представляет программу как совокупность *классов* и *объектов*, взаимодействующих друг с другом.  *Объект* – *конкретная материализация абстракции; сущность с хорошо определенными границами, в которой инкапсулированы состояние и поведение*.  *Объект ООП – инкапсулированная структура, имеющая атрибуты и методы*.  *Термин "инкапсулированная структура" означает, что объект является самодостаточным, программы, внешние по отношению к объекту, ничего "не знают" о его структуре и такое "знание" им не требуется*. "Внешний" вид *объекта* называется его интерфейсом.  В таком понимании *объект* – это черный ящик, нам неизвестно, чтo у него внутри, мы лишь можем вызвать его методы и только через них взаимодействовать с ним. Кроме этого, объекты могут принадлежать иерархии "от общего к частному", которая реализуется путем наследования. Инкапсулированные состояния объекта могут быть как простыми типами данных, так и другими объектами, или даже массивами объектов. Каждый *объект* содержит определенную совокупность методов, *классы* взаимодействуют друг с другом посредством механизма сообщений. *Объекты* идентифицируются с помощью специальных указателей – дескрипторов. Методы объектов ООП представляют собой последовательности инструкций, выполняемых объектом. Например, у объекта может быть метод, отображающий данный объект, создающий данный объект и изменяющий его.  *Предметная область моделируется как множество классов взаимодействующих объектов. Объект характеризуется набором свойств, которые являются как бы его пассивными характеристиками, и набором методов работы с этим объектом*.Работать с объектом можно только с использованием его методов. Атрибуты объекта могут принимать множество допустимых значений, набор конкретных значений атрибутов определяет состояние объекта. Используя методы работы с объектом можно изменять значение его атрибутов и тем самым как бы изменить состояние самого объекта. Множество объектов с одним и тем же набором атрибутов и методов образует *класс* объектов. *Класс*, объекты которого могут служить значениями атрибута объектов другого класса, называется доменом этого атрибута.  К числу основных идей объектно-ориентированной технологии, как правило, относят [[[25]](http://www.intuit.ru/department/database/databases/popup.lit.html#25)]: **абстрагирование, инкапсуляцию, модульность, иерархичность, типизацию, полиморфизм, наследование**.  *Инкапсуляция ограничивает область видимости имени атрибута пределами того объекта, в котором оно определено*. Смысл этого атрибута будет определяться тем объектом, в котором оно инкапсулировано.  *Полиморфизм – способность одного и того же программного кода работать с разнообразными данными*. Другими словами, он допускает возможность в объектах разных типов иметь методы (процедуры или функции) с одинаковыми именами. Во время выполнения объектной программы одни и те же методы оперируют с разными объектами в зависимости от типа аргумента.  *Наследование*. *Допускается порождение нового класса на основе уже существующего класса, и этот процесс называется* *наследованием*. *В этом случае новый класс, называемый подклассом существующего класса, наследует все атрибуты и* *методы класса*. В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы. Различают случаи простого и множественного наследования. В первом случае подкласс может определяться только на основе одного класса, во втором случае – на основе нескольких классов. Набор классов образует иерархическую структуру. Объектно-ориентированные базы данных К настоящему моменту терминология еще не устоялась, существует много разных определений и трактовок. *Представляется, что объектно-ориентированная база данных (ООБД) – база данных, основанная на принципах объектно-ориентированной технологии*. К основным описательным моментам, связанным с ООБД, в литературе [[[26]](http://www.intuit.ru/department/database/databases/popup.lit.html#26)] относят:   * *объекты* (в ООБД любая сущность – объект и обрабатывается как объект); отметим, что здесь используется понятие "объект" объектно-ориентированного программирования, которое отличается от понятия "объект", рассматриваемого ранее в данном учебном пособии. * *классы* (понятие "тип данных" реляционной модели заменяется понятиями "класс" и "подкласс"); * *наследование* (классы образуют *иерархию наследования*, заимствуя свойства друг друга); * атрибуты (характеристики объекта моделируются его атрибутами); * сообщения и методы (каждый класс имеет определенную совокупность методов, классы взаимодействуют друг с другом посредством механизма сообщений); * инкапсуляция (внутренняя структура объектов скрыта); * *идентификаторы объектов* – дескрипторы.   Схема представления объекта приводится на [рис. 14.1](http://www.intuit.ru/department/database/databases/14/databases_14.html#image.14.1)  Схема представления объекта  **Рис. 14.1.**  Схема представления объекта  *Система управления объектно-ориентированной базой данных называется объектно-ориентированной СУБД (ООСУБД)*. Цель ООСУБД – обеспечение постоянного хранения объектов, причем в отличие от традиционной СУБД ООСУБД должна хранить в составе объекта данные и программы.  Поскольку каждый объект данного класса имеет один и тот же набор методов, методы сохраняются только один раз – как *методы класса* (данные каждого экземпляра объекта хранятся отдельно).  Схема представления класса объектов приводится на [рис. 14.2](http://www.intuit.ru/department/database/databases/14/databases_14.html#image.14.2)  Схема представления класса объектов  **Рис. 14.2.**  Схема представления класса объектов  Используя *наследование*, всем объектам ПОДРАЗДЕЛЕНИЕ можно приписать свойство объекта-родителя (ФАКУЛЬТЕТ) – название факультета, номер факультета. Схема представления объектов ФАКУЛЬТЕТ и ПОДРАЗДЕЛЕНИЕ приводится на [рис. 14.3](http://www.intuit.ru/department/database/databases/14/databases_14.html#image.14.3).  Фрагменты представления конкретных объектов  **Рис. 14.3.**  Фрагменты представления конкретных объектов  Сравнивая объектно-ориентированный и реляционный подходы к БД, можно отметить следующие особенности. В реляционных БД (РБД) реальные объекты представляются как структуры, состоящие из набора *элементарных типов* данных. Такое представление имеет понятную интерпретацию – строка в плоской таблице. В том случае, когда специфика предметной области позволяет работать с такого рода приближением реальных объектов, РБД отлично справляются со своей задачей. Довольно часто реляционная модель и ее способ описания предметной области в виде набора плоских таблиц не отражают внутренней структуры для многих предметных областей, являются искусственными и становятся совершенно непонятными при увеличении количества таблиц. Основная причина несостоятельности реляционного подхода заключается в слишком сильной абстракции реального объекта, что ведет к потере семантики.  В отличие от реляционных баз данных *объектно-ориентированные базы данных* обладают простой и естественной связью с предметной областью, представляя ее структуру и состав, что облегчает проектирование и положительно сказывается на понимании принципов функционирования программ. Так, в сложных неоднородных предметных областях использование ООБД (в частности, там, где разные объекты имеют разные методы) должно действительно упростить процесс проектирования и разработки.  К сожалению, в ООБД существуют свои проблемы. В ООБД отсутствует универсальная модель данных, и соответственно, отсутствует мощная математическая база, как, например, в реляционной модели. В связи с этим у ООБД нет языка запросов высокого уровня, аналогичного SQL, и при доступе к данным используется мало эффективный навигационный подход. ООСУБД отличаются от реляционных СУБД тем, что программный интерфейс создания приложения либо очень слаб, либо вообще отсутствует. Это означает, для написании приложения, работающего с ООБД, не существует мастеров и конструкторов (не считая, например, конструктора создания списка полей в объекте, который поставляется вместе с ООСУБД ObjectStore). Поэтому разработчик создает приложения на одном из алгоритмических языков.  По нашему мнению, существенным ограничением развития объектно-ориентированного подхода к созданию баз данных является то, что методы объекта содержатся внутри объекта и неразрывно связаны с ним. Это делает, по сути, невозможным создание для объектно-ориентированной базы данных соответствующей системы управления базой данных в традиционном понимании СУБД, функциями которой, в частности, является реализация операций обработки данных. Поэтому ООСУБД часто является не системой управления базами данных, а библиотекой программ, с помощью которой можно построить объектно-ориентированную базу данных. Примером такой библиотеки является ООСУБД ObjectStore. В связи с этим, возникает проблема реализации непредвиденных запросов.  Для перехода к *объектно-ориентированным БД* стандарт объектного программирования был дополнен стандартизованными средствами доступа к базам данных (стандарт *ODMG*93; Object *Database Management* Group – группа управления *объектно-ориентированными базами данных* ). К настоящему времени этот стандарт не реализован. Состояние проблемы подробно описано также в работах [[[26]](http://www.intuit.ru/department/database/databases/popup.lit.html#26), [[4]](http://www.intuit.ru/department/database/databases/popup.lit.html#4), [[2]](http://www.intuit.ru/department/database/databases/popup.lit.html#2), [[18]](http://www.intuit.ru/department/database/databases/popup.lit.html#18), [[3]](http://www.intuit.ru/department/database/databases/popup.lit.html#3) и др.]. Отметим только, что ООБД используются, но пока не стали реальной альтернативой реляционным базам данных.  Объектно-ориентированные возможности появляются в ведущих современных СУБД, таких, как, например, Oracle. Предпринимаются попытки внесения изменений в стандарты языка SQL с целью его частичной адаптации к ООБД. Так, новый стандарт SQL-3 включает большой раздел, посвященный этому вопросу. Объектно-реляционные СУБД В настоящее время реляционные СУБД доминируют среди систем управления данными. Преимущества объектно-ориентированного подхода для создания сложных специализированных приложений с одной стороны, и стремление разработчиков систем управления базами данных с другой стороны расширить границы применения соответствующих СУБД обусловили *включение объектно-ориентированных компонент (расширяемая пользователем система типов, инкапсуляция, наследование, полиморфизм и т. п.) в модель данных реляционной СУБД. Соответствующие СУБД, называемые объектно-реляционными*, соединяют в себе лучшие качества реляционных и *объектно-ориентированных баз данных*. Отметим, что в разных СУБД реализован разный набор из перечисленных объектно-ориентированных компонент. Таким образом, не существует общепринятой объектно-реляционной модели, а скорее имеется несколько таких моделей, поддерживающих определенный набор объектно-ориентированных компонент. Однако, основой всех таких моделей являются реляционные таблицы, используется язык запросов, включено понятие объекта, а в некоторых дополнительно реализована возможность сохранения методов в базе данных.  Соответствующие изменения реляционной модели обусловили необходимость расширения стандарта языка запросов SQL. Первый вариант такого стандарта получил название*SQL3*. Работа над стандартом продолжается и в настоящее время.  В качестве примера в максимальной степени объектно-ориентированной СУБД можно указать исследовательскую *СУБД Postgres* [[[4]](http://www.intuit.ru/department/database/databases/popup.lit.html#4)].  Отметим считающиеся объектными расширениями элементы СУБД Microsoft Server 2008.   * **Пользовательские расширения**. Пользователи имеют возможность вмешиваться в изначально предоставляемый СУБД инструментарий, создавая, в частности, новые*пользовательские типы данных*. * **Хранение больших объемов данных**. Наряду с теми данными, которые хранились в БД традиционно, *Microsoft SQL Server 2008* позволяет хранить в столбцах таблицы данные больших размеров (поддерживаются соответствующие типы данных). * **Новые, ориентированные на определенные классы объектов, типы данных**. В системе определены новые типы данных (*geometry*, geography), характерные для тех направлений, в которых объектно-ориентированный подход весьма эффективен и часто используется (картография и соответствующие приложения, геометрическое представление объектов самой разной природы). * **Хранимые процедуры**. В определенном смысле хранимые процедуры также являются объектным расширением, осуществляя необходимые пользователю воздействия на данные (стандартный для ООП процедурный подход).  14.2. Распределенные базы данных База данных – интегрированная совокупность данных, с которой работают много пользователей. Изложение всех предыдущих разделов предполагало единую базу данных, размещаемую на одном компьютере. Напомним основные принципы, положенные в основу теории баз данных:   * централизованное хранение данных; * централизованное обслуживание данных (ввод, корректировка, чтение, контроль целостности).   Заметим, что базы данных появились в период господства больших ЭВМ. База данных велась на одной ЭВМ, все пользователи работали именно на ЭВМ (возможные режимы работы описаны в [лекции 3](http://www.intuit.ru/department/database/databases/3/)). Других вариантов использования вычислительной техники в то время просто не существовало. Если проанализировать работу пользователей с данными в компаниях, организациях, предприятиях в "докомпьютерное" время, то нетрудно заметить, что на отдельных участках пользователи работали со "своими" данными (осуществляли сбор определенных данных, их хранение, обработку, передачу обработанных данных на другие участки или уровни управления).  У такой технологии были существенные недостатки, которые уже отмечались в предыдущих разделах: дублирование некоторых данных, отсутствие возможности сравнительного анализа данных всех участков. Однако у этой технологии были и существенные достоинства: данные вводились и хранились в местах их порождения; с этими данными работал пользователь, являющийся специалистом именно по этим данным, что позволяло ему вести эффективный контроль правильности данных на всех стадиях обработки; данные находились непосредственно у пользователя, что давало возможность их оперативной обработки. Централизация данных на одной ЭВМ, несомненно, дающая эффективные возможности хранения и обработки данных, не позволяла реализовывать вышеназванные достоинства.  Развитие вычислительных компьютерных сетей обусловило новые возможности в организации и ведении баз данных, позволяющие каждому пользователю иметь на своем компьютере свои данные и работать с ними и в то же время позволяющие работать всем пользователям со всей совокупностью данных как с единой централизованной базой данных. Соответствующая совокупность данных называется распределенной базой данных.  Термин " **распределенная база данных** " достаточно часто встречается в литературе [[[26]](http://www.intuit.ru/department/database/databases/popup.lit.html#26), [[4]](http://www.intuit.ru/department/database/databases/popup.lit.html#4), [[2]](http://www.intuit.ru/department/database/databases/popup.lit.html#2), [[18]](http://www.intuit.ru/department/database/databases/popup.lit.html#18), [[3]](http://www.intuit.ru/department/database/databases/popup.lit.html#3)]. Однако в разных источниках под этим термином понимаются совершенно разные вещи. Часть авторов понимают под распределенной базой данных то, что имеется удаленный сервер, на котором расположены данные, а также клиентские компьютеры, расположенные территориально в другом месте. Такая трактовка нам представляется неправильной. Настоящая **распределенная база данных** располагается на нескольких компьютерах. При этом часть файлов расположена на одном компьютере, часть на другом и т.д. Более того, возможна и даже часто встречается ситуация, когда информация на этих компьютерах пересекается, дублируется.  *Распределенная база данных* – совокупность логически взаимосвязанных *разделяемых данных* (и описаний их структур), физически распределенных в компьютерной сети.  *Система управления распределенной базой данных* – программная система, обеспечивающая работу с распределенной базой данных и позволяющая пользователю работать как с его локальными данными, так и со всей базой данных в целом.  *Система управления распределенной базой данных* (РаСУБД) является распределенной системой. Каждый фрагмент базы данных работает под управлением отдельной СУБД, которая осуществляет доступ к данным этого фрагмента. Пользователи взаимодействуют с распределенной базой данных через локальные и глобальные приложения. Локальные приложения дают пользователю возможность работать со своими локальными данными и не требуют доступа к другим фрагментам. Глобальные приложения дают пользователю возможность работать с другими фрагментами базы данных, расположенными на других компьютерах сети. Общая схема распределенной базы данных представлена на [рис. 14.4](http://www.intuit.ru/department/database/databases/14/databases_14.html#image.14.4).  Объединение данных организуется виртуально. Соответствующий подход, по сути, отражает организационную структуру предприятия (и даже общества в целом), состоящего из отдельных подразделений. Причем, хотя каждое подразделение обрабатывает свой набор данных (эти наборы, как правило, пересекаются), существует необходимость доступа к этим данным как к единому целому (в частности, для управления всем предприятием).  Одним из примеров реализации такой модели может служить сеть Интернет: данные вводятся и хранятся на разных компьютерах по всему миру, любой пользователь может получить доступ к этим данным, не задумываясь о том, где они физически расположены.  Распределенная база данных  **Рис. 14.4.**  Распределенная база данных  К.Дж. Дейт провозглашает следующий фундаментальный принцип распределенной базы данных [[[4]](http://www.intuit.ru/department/database/databases/popup.lit.html#4)]. Для пользователя распределенная система должна выглядеть точно так же, как нераспределенная. Из этого принципа следует ряд правил:   1. Локальная автономия. 2. Независимость от центрального узла. 3. Непрерывное функционирование. 4. Независимость от расположения. 5. Независимость от фрагментации. 6. Независимость от репликации. 7. Обработка распределенных запросов. 8. Управление *распределенными транзакциями*. 9. Независимость от аппаратного обеспечения. 10. Независимость от операционной системы. 11. Независимость от сети. 12. Независимость от СУБД.   *Заметим, что понятие распределенной базы данных можно интерпретировать как следующий шаг в развитии понятий о данных* (см. [лекцию 1](http://www.intuit.ru/department/database/databases/1/)), *обусловленный распределенностью данных в реальных предметных областях, а также новым этапом развития средств вычислительной техники – широким использованием вычислительных сетей*.  *В этой интерпретации распределенную базу данных можно понимать как совокупность логически взаимосвязанных распределенных по разным компьютерам баз данных*.  Перечислим основные проблемы создания распределенной базы данных.   1. Фрагментация данных и распределение по компьютерам. 2. Составление глобального каталога, содержащего информацию о каждом фрагменте БД и его местоположении в сети. (Каталог может храниться на одном узле или быть распределенным) 3. Организация обработки запросов (синхронизация нескольких запросов к одним и тем же данным, исключение *аномалий удаления* и обновления одних и тех же данных, расположенных на различных узлах, оптимизация последовательности шагов при обработке запроса и т.д.).   Значительным достоинством этой модели является приближение данных к месту их порождения, что позволяет существенно повысить их достоверность, недостатком – достаточно высокая сложность управления данными как единым целым.  К сожалению, процесс создания и обслуживания распределенных баз данных связан и с техническими трудностями, среди которых можно выделить жесткие требования к пропускной способности каналов связи, а также низкую производительность, обусловленную значительными затратами коммуникационных и вычислительных ресурсов при их синхронизации во время выполнения транзакций (особенно при интенсивных обращениях из разных узлов к одному фрагменту).  В задачу данного учебника не входит подробное изучение принципов построения распределенных баз данных. Интересующимся рекомендуем обратиться к соответствующей литературе, например [[[4]](http://www.intuit.ru/department/database/databases/popup.lit.html#4), [[2]](http://www.intuit.ru/department/database/databases/popup.lit.html#2), [[18]](http://www.intuit.ru/department/database/databases/popup.lit.html#18), [[3]](http://www.intuit.ru/department/database/databases/popup.lit.html#3)] Здесь мы хотим лишь обрисовать проблему и сделать некоторые выводы по перспективам ее решения. Технология, связанная с использованием распределенных баз данных, в наибольшей степени соответствует организационной человеческой деятельности (информация распределена по месту деятельности людей, и они обмениваются ей в процессе работы) и позволяет наиболее успешно решать важнейшие проблемы ведения баз данных:   * повысить достоверность информации (информация вводится в месте ее порождения лицом, которое лучше всех понимает ее смысловое значение); * повысить оперативность локальной обработки информации (соответствующие вопросы решаются на локальном компьютере с фрагментом базы данных).   Поэтому очевидно, что задача проектирования, создания и функционирования распределенных баз данных является весьма существенной, активно изучается в настоящее время и будет решаться и далее. 14.3. Хранилища данных Как уже неоднократно отмечалось, технологии баз данных предназначены, как правило, для решения текущих задач обработки данных организации. В базу данных постоянно вносятся изменения, то есть база данных отражает моментальный снимок определенной области деятельности предприятия. Для эффективного принятия решений руководством при управлении организацией важно не только знать текущее положение дел, но и иметь возможность анализировать динамику (изменение во времени) основных показателей, причем, зачастую из разных баз данных. Такую возможность дает технология так называемых хранилищ данных.  Приведем определение *хранилища данных* (Bill Inmon).  *Хранилище данных* **– предметно-ориентированный, интегрированный, привязанный ко времени и неизменяемый набор данных, предназначенный для поддержки принятия решений**.  Под **предметной ориентированностью** здесь понимается ориентированность на предметы (определенные группы данных), а не на конкретные приложения. Например, ориентация на данные о сотрудниках, а не только о расчете их заработной платы.  Под **интегрированностью** здесь понимается возможное объединение данных из разных источников (баз данных), имеющих разный формат и несогласованных.  **Привязка ко времени** предполагает, что для всех данных указан момент или промежуток времени, в который они корректны.  *Данные в хранилище не изменяются*, они лишь регулярно пополняются из оперативных баз данных.  Общая схема взаимодействия информационного хранилища и баз данных приводится на [рис. 14.5](http://www.intuit.ru/department/database/databases/14/databases_14.html#image.14.5).  Схема организации работы хранилища данных  **Рис. 14.5.**  Схема организации работы хранилища данных  Еще раз подчеркнем, что основной целью хранилищ данных является бизнес-анализ или информационная поддержка принятия управленческих решений.  Для реализации всей необходимой обработки информации в соответствии с этой схемой необходимы следующие программные средства:   * средства извлечения данных из баз данных; * средства управления данными хранилища (система управления базой данных хранилища); * средства анализа данных хранилища (используется OLAP-технология): * средства доставки данных; * средства визуализации результатов обработки для конечных пользователей.   Для работы соответствующих программных средств необходимо описание структуры содержимого информационного хранилища (метаописание).  Для самого общего случая, если данные берутся из баз данных, управляемых разными СУБД, из файлов разных типов, а данные разнородны, средства управления данными хранилища пока не созданы. Однако, если данные в информационное хранилище выбираются только из реляционных баз данных, то в качестве средств управления данными хранилища может быть взята мощная реляционная СУБД. Поэтому разработчики современных СУБД включают в состав программного обеспечения СУБД средства организации работы с хранилищами данных.  Рассмотрим в качестве **примера** *возможности СУБД Microsoft SQL Server 2008 для организации хранилищ данных*.  *Microsoft SQL Server 2008* содержит в своем составе средства извлечения, преобразования и загрузки данных (SQL Server 2008 Integration Services), способные интегрировать данные из различных источников, проверять данные на допустимость и преобразовывать перед загрузкой в хранилище. Эти средства также способствуют перемещению данных, поддерживают текстовый анализ и нечеткий поиск. Нужно отметить также среду визуальной разработки (Business Intelligence Development Studio) для создания многомерных кубов, отчетов, пакетов извлечения, преобразования и загрузки данных.  Существенной особенностью хранилищ данных является их очень большой объем. *Microsoft SQL Server 2008* как средство управления данными хранилища позволяет работать с большими объемами данных, причем для сокращения времени обработки предусмотрена поддержка параллельных вычислений (путем разделения таблиц и индексов на секции и обеспечение параллельной обработки секций). В системе предусмотрена возможность сжатия данных (таблиц), что позволяет уменьшить физический размер таблиц и существенно сокращает время обмена между оперативной и внешней памятью.  В качестве средств анализа данных хранилища используется SQL Server 2008 Analysis Services, применяемый для построения многомерных кубов (многомерных моделей данных). Это средство содержит семь эффективных алгоритмов анализа данных с целью поддержки принятия управленческих решений, в том числе анализ тенденций и статистический анализ данных.  В качестве средств представления аналитических данных пользователям предлагается использовать средство генерации отчетов SQL Server 2008 Reporting Services.  Таким образом, *Microsoft SQL Server 2008* является эффективным средством реализации хранилищ данных на основе реляционных баз данных.  **Краткие итоги**: В лекции рассмотрены перспективные направления в теории и практике создания баз данных – объектно-ориентированные и распределенные базы данных. Здесь описываются основные идеи объектно-ориентированного программирования ( *объект*, *класс*, *методы класса*, *наследование* ) и их приложение к теории баз данных. Отмечены основные достоинства и недостатки *объектно-ориентированных баз данных*.  Рассматривается понятие распределенных баз данных как следующий шаг в развитии понятий о данных. Отмечены основные достоинства распределенных баз данных и проблемы, возникающие при их разработке.  Рассматривается понятие *хранилища данных*, в качестве примера системы управления данными хранилища приводится СУБД *Microsoft SQL Server 2008*. |