**Лекция 1: Введение**

### **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**

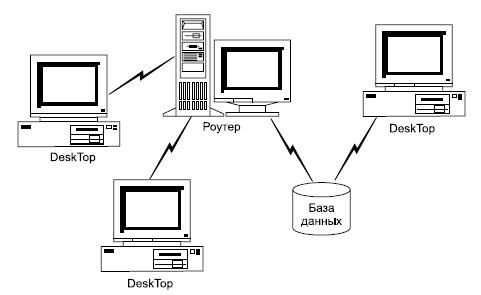
Хорошо известно, что история развивается по спирали, поэтому после процесса "персонализации" начался *обратный* процесс — *интеграция*. Множится количество локальных сетей, все больше информации передается между компьютерами, остро встает задача согласованности данных, хранящихся и обрабатывающихся в 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:
  + структурной целостности — допустимыми являются только данные, представленные в виде отношений реляционной модели;
  + языковой целостности, то есть языков манипулирования данными высокого уровня (в основном SQL);
  + ссылочной целостности, контроля за соблюдением ссылочной целостности в течение всего времени функционирования системы, и гарантий невозможности со стороны СУБД нарушить эти ограничения.
* Большинство современных СУБД рассчитаны на многоплатформенную архитектуру, то есть они могут работать на компьютерах с разной архитектурой и под разными операционными системами, при этом для пользователей доступ к данным, управляемым СУБД на разных платформах, практически неразличим.
* Необходимость поддержки многопользовательской работы с базой данных и возможность 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](https://intuit.ru/studies/courses/1001/297/lecture/7399?page=2#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.



**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?

**Лекция 2: Основные понятия и определения**

Современные авторы часто употребляют термины "*банк данных*" и "*база данных*" как синонимы, однако в общеотраслевых руководящих материалах *по* созданию банков данных Государственного комитета *по* науке и технике (ГКНТ), изданных в 1982 г., эти понятия различаются. Там приводятся следующие определения банка данных, *базы данных* и *СУБД*:

*Банк данных* (БнД) — это система специальным образом организованных данных — баз данных, программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.

*База данных* (*БД*) — именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой *предметной области*.

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

Программы, с помощью которых пользователи работают с базой данных, называются *приложениями*. В общем случае с одной базой данных могут работать множество различных приложений. Например, если *база данных* моделирует некоторое предприятие, то для работы с ней может быть создано *приложение*, которое обслуживает подсистему учета кадров, другое *приложение* может быть посвящено работе подсистемы расчета заработной платы сотрудников, третье *приложение* работает как подсистемы складского учета, четвертое *приложение* посвящено планированию производственного процесса. При рассмотрении приложений, работающих с одной базой данных, предполагается, что они могут работать параллельно и независимо друг от друга, и именно *СУБД* призвана обеспечить работу *множества* приложений с единой базой данных таким образом, чтобы каждое из них выполнялось корректно, но учитывало все изменения в базе данных, вносимые другими приложениями.

### **Пользователи банков данных**

Как любой программно-организационно-техничеcкий комплекс, *банк данных* существует во времени и в пространстве. Он имеет определенные стадии своего развития:

1. Проектирование.
2. Реализация.
3. Эксплуатация.
4. Модернизация и развитие.
5. Полная реорганизация.

На каждом этапе своего существования с банком данных связаны разные категории пользователей.

Определим основные категории пользователей и их роль в функционировании банка данных:

* **Конечные пользователи**. Это основная категория пользователей, в интересах которых и создается *банк данных*. В зависимости от особенностей создаваемого банка данных круг его конечных пользователей может существенно различаться. Это могут быть случайные пользователи, обращающиеся к БД время от времени за получением некоторой информации, а могут быть регулярные пользователи. В качестве случайных пользователей могут рассматриваться, например, возможные клиенты вашей фирмы, просматривающие каталог вашей продукции или услуг с обобщенным или подробным описанием того и другого. Регулярными пользователями могут быть ваши сотрудники, работающие со специально разработанными для них программами, которые обеспечивают автоматизацию их деятельности при выполнении своих должностных обязанностей. Например, менеджер, планирующий работу сервисного отдела компьютерной фирмы, имеет в своем распоряжении программу, которая помогает ему планировать и распределять текущие заказы, контролировать ход их выполнения, заказывать на складе необходимые комплектующие для новых заказов. Главный принцип состоит в том, что от конечных пользователей не должно требоваться каких-либо специальных знаний в области вычислительной техники и языковых средств.
* **Администраторы банка данных**. Это группа пользователей, которая на начальной стадии разработки банка данных отвечает за его оптимальную организацию с точки зрения одновременной работы множества конечных пользователей, на стадии эксплуатации отвечает за корректность работы данного банка информации в многопользовательском режиме. На стадии развития и реорганизации эта группа пользователей отвечает за возможность корректной реорганизации банка без изменения или прекращения его текущей эксплуатации.
* **Разработчики и администраторы приложений**. Это группа пользователей, которая функционирует во время проектирования, создания и реорганизации банка данных. Администраторы приложений координируют работу разработчиков при разработке конкретного приложения или группы приложений, объединенных в функциональную подсистему. Разработчики конкретных приложений работают с той частью информации из базы данных, которая требуется для конкретного приложения.

Не в каждом банке данных могут быть выделены все типы пользователей. Мы уже знаем, что при разработке информационных систем с использованием настольных *СУБД* *администратор* банка данных, *администратор* приложений и разработчик часто существовали в одном лице. Однако при построении современных сложных корпоративных баз данных, которые используются для автоматизации всех или большей части бизнес-процессов в крупной фирме или корпорации, могут существовать и группы администраторов приложений, и отделы разработчиков. Наиболее сложные обязанности возложены на группу *администратора БД*.

Рассмотрим их более подробно.

В составе группы *администратора БД* должны быть:

* системные аналитики;
* проектировщики структур данных и внешнего по отношению к банку данных информационного обеспечения;
* проектировщики технологических процессов обработки данных;
* системные и прикладные программисты;
* операторы и специалисты по техническому обслуживанию.

Если речь идет о коммерческом банке данных, то важную роль здесь играют специалисты *по* маркетингу.

### **Основные функции группы администратора БД**

1. **Анализ предметной области**: описание предметной области, выявление ограничений целостности, определение статуса (доступности, секретности) информации, определение потребностей пользователей, определение соответствия "данные—пользователь", определение объемно-временных характеристик обработки данных.
2. **Проектирование структуры БД**: определение состава и структуры файлов БД и связей между ними, выбор методов упорядочения данных и методов доступа к информации, описание БД на языке описания данных (ЯОД).
3. **Задание ограничений целостности при описании структуры БД и процедур обработки БД**:
   * задание декларативных ограничений целостности, присущих предметной области;
   * определение динамических ограничений целостности, присущих предметной области в процессе изменения информации, хранящейся в БД;
   * определение ограничений целостности, вызванных структурой БД;
   * разработка процедур обеспечения целостности БД при вводе и корректировке данных;
   * определение ограничений целостности при параллельной работе пользователей в многопользовательском режиме.
4. **Первоначальная загрузка и ведение БД**:
   * разработка технологии первоначальной загрузки БД, которая будет отличаться от процедуры модификации и дополнения данными при штатном использовании базы данных;
   * разработка технологии проверки соответствия введенных данных реальному *состоянию предметной области*. База данных моделирует реальные объекты некоторой предметной области и взаимосвязи между ними, и на момент начала штатной эксплуатации эта модель должна полностью соответствовать состоянию объектов предметной области на данный момент времени;
   * в соответствии с разработанной технологией первоначальной загрузки может понадобиться проектирование системы первоначального ввода данных.
5. **Защита данных**:
   * определение системы паролей, принципов регистрации пользователей, создание групп пользователей, обладающих одинаковыми правами доступа к данным;
   * разработка принципов защиты конкретных данных и объектов проектирования; разработка специализированных методов кодирования информации при ее циркуляции в локальной и глобальной информационных сетях;
   * разработка средств фиксации доступа к данным и попыток нарушения системы защиты;
   * тестирование системы защиты;
   * исследование случаев нарушения системы защиты и развитие *динамических методов* защиты информации в БД.
6. **Обеспечение восстановления БД**:
   * разработка организационных средств архивирования и принципов восстановления БД;
   * разработка дополнительных программных средств и технологических процессов восстановления БД после сбоев.
7. **Анализ обращений пользователей БД**: сбор статистики по характеру запросов, по времени их выполнения, по требуемым выходным документам
8. **Анализ эффективности функционирования БД**:
   * анализ показателей функционирования БД;
   * планирование реструктуризации (изменение структуры) БД и реорганизации БнД.
9. **Работа с конечными пользователями**:
   * сбор информации об изменении предметной области;
   * сбор информации об оценке работы БД;
   * обучение пользователей, консультирование пользователей;
   * разработка необходимой методической и учебной документации по работе конечных пользователей.
10. **Подготовка и поддержание системных средств**:
    * анализ существующих на рынке программных средств и анализ возможности и необходимости их использования в рамках БД;
    * разработка требуемых организационных и программно-технических мероприятий по развитию БД;
    * проверка работоспособности закупаемых программных средств перед подключением их к БД;
    * курирование подключения новых программных средств к БД.
11. **Организационно-методическая работа по проектированию БД**:
    * выбор или создание методики проектирования БД;
    * определение целей и направления развития системы в целом;
    * планирование этапов развития БД;
    * разработка общих словарей-справочников проекта БД и концептуальной модели;
    * стыковка внешних моделей разрабатываемых приложений;
    * курирование подключения нового приложения к действующей БД;
    * обеспечение возможности комплексной отладки множества приложений, взаимодействующих с одной БД.

**Лекция 3. Основы информационной безопасности баз данных**

**Общая характеристика составляющих информационной безопасности**

Для баз данных и систем, основанных на хранении данных, важны три *основных аспекта информационной безопасности* (иначе *базовые свойства защищаемой информации*) – *конфиденциальность* (требуется защита от несанкционированного доступа), *целостность* (требуется защита от потери согласованности и достоверности данных) и *доступность* (требуется защита от несанкционированного удержания информации и ресурсов, поддержка работоспособности и защита от разрушения), т. е. под защитой данных понимают систему мер, предохраняющих данные от несанкционированного использования, разрушения или искажения.

Необходимо полагать, что абсолютная защита данных практически не реализуема, поэтому обычно довольствуются *относительной защитой информации* - гарантированно защищают ее на тот период времени, пока несанкционированный доступ к ней, потеря ее целостности или доступности влечет какие-либо последствия.

Перечислим типовые ситуации, в которых хранимые данные могут быть похищены, искажены или потеряны.

**Несанкционированный доступ (НСД)**. Если любой желающий может получить доступ к любым хранимым данным и внести в них любые изменения, то данные могут быть разрушены некомпетентным или злонамеренным пользователем либо использованы с ущемлением прав владельца или во вред ему.

**Неуправляемый параллелизм**. Как правило, в многопользовательской системе с базой данных одновременно работает несколько пользователей. Некоторые из них могут пытаться одновременно изменять состояние БД. Если они будут делать это независимо, то результирующее состояние БД может оказаться несогласованным.

**Локальный сбой**. В процессе выполнения прикладной программы может возникнуть аварийная ситуация (например, деление на ноль), в результате которой выполнение программы будет прекращено. Если прикладная программа выполняла обновление данных, то БД может оказаться в несогласованном состоянии.

**Потеря оперативной памяти (мягкий сбой)**. Несмотря на высокую надежность персональных компьютеров (ПК), в любой момент может произойти системный сбой (например, отключение питания), в результате которого будет утрачено содержимое системных буферов и буферов приложений, размещенных в оперативной памяти. Подобные ситуации называются мягкими сбоями системы. Поскольку состояние БД в момент мягкого сбоя непредсказуемо, оно может оказаться несогласованным после перезагрузки системы. В отличие от локального сбоя при мягком сбое могут пострадать данные всех пользователей.

**Физическое разрушение данных (жесткий сбой)**. В процессе эксплуатации устройства внешней памяти может быть испорчена поверхность диска, блок головок дисковода и т.п. Подобные ситуации называются жесткими сбоями системы. Вероятность жесткого сбоя очень мала, тем не менее, ею нельзя пренебречь.

Требования защиты данных от разрушения в обобщенном виде сводятся к следующему: никакие случайные или преднамеренные разрушения данных не могут быть необратимыми.

Вышеперечисленные ситуации могут быть, в свою очередь, следствием:

1. Некомпетентных, безответственных или преступных действий пользователя;

2. Несогласованных обновлений, производимых одновременно и независимо несколькими пользователями;

3. Возникновения необработанной должным образом ошибки в прикладной программе;

4. Внедрением вредоносного кода в программное обеспечение;

5. Аварийного отключения энергии, сбоя или отказа различных устройств системы, нарушения линий связи;

6. Физическим уничтожением компьютерной системы или ее частей.

Следовательно, для защиты данных от потери целостности, разрушения и несанкционированного использования система должна обеспечивать:

1. Поддержку определяемых информационной политикой предприятия ограничений прав и полномочий доступа пользователей к данным;

2. Защиту от вредоносных действий пользователей, направленных на повреждение данных или получения конфиденциальных данных на основе анализа доступной информации;

3. Дисциплину параллельной работы многих пользователей, исключающую возможность внесения несогласованных изменений в состояние БД;

4. Восстановление целостного состояния БД после локальных сбоев и мягких сбоев системы;

5. Восстановление БД после жестких сбоев.

Для обеспечения безопасности данных применяются компьютерные и некомпьютерные средства защиты. Вначале рассмотрим компьютерные средства защиты. Традиционными являются следующие средства контроля и защиты;

- Авторизация пользователей и разграничение доступа. Авторизация пользователей, означающая предоставление определенных прав пользователю, позволяющих их владельцу иметь законный доступ ко всей системе или к ее отдельным объектам может быть осуществлена как средствами операционной системы, так и средствами самой СУБД.

- Доступ к данным через представления. Представление - это результат одной или нескольких реляционных операций с базовыми отношениями с целью создания некоторого иного отношения. Представление является виртуальным отношением, которого реально в БД не существует, но которое создается по требованию отдельного пользователя в момент поступления этого требования.

Механизм представления является мощным и гибким инструментом организации защиты данных, позволяющим скрыть от определенных пользователей некоторые части БД. В результате пользователи не будут иметь никаких сведений о существовании любых атрибутов или строк данных, которые недоступны через представления, находящиеся в их распоряжении. Представление может быть определено на базе нескольких отношений, после чего пользователю будут предоставлены необходимые привилегии доступа к этому представлению, но не к базовым отношениям. В данном случае использование представления являет собой более жесткий механизм контроля доступа, чем обычное предоставление пользователю тех или иных прав доступа к базовым отношениям.

- Обновление данных и реализация правил «бизнес логики» с помощью хранимых процедур, функций и триггеров. Хранимые процедуры, функции и триггеры представляют собой фрагменты кода на высокоуровневом языке программирования, являющем собой расширение языка SQL, хранящееся и выполняющиеся на сервере. Реализация всех правил «бизнес логики» с помощью хранимых процедур, функций и триггеров предоставляет значительно более высокий уровень защиты данных, нежели их реализация в клиентских программах.

- Резервное копирование и восстановление. Резервное копирование - периодически выполняемая процедура получения копии БД и ее файла журнала (а также, возможно программ СУБД) во внешней постоянной памяти, чаще всего на отдельном носителе. В случае

отказа, в результате которого БД становится непригодной для дальнейшей эксплуатации, резервная копия и зафиксированная в файле журнала оперативная информация используются для восстановления БД до последнего согласованного состояния.

Ведение журнала - это процедура создания и обслуживания файла журнала, содержащего сведения обо всех изменениях, внесенных в БД с момента создания последней резервной копии, и предназначенного для обеспечения эффективного восстановления системы в случае ее отказа. Если в отказавшей системе функция ведения системного журнала не использовалась, базу данных можно будет восстановить только до того состояния, которое было зафиксировано в последней созданной резервной копии. Все изменения, которые были внесены в БД после создания последней резервной копии, окажутся потерянными.

- Обеспечение целостности. Обеспечение целостности включает проверку целостности данных и их восстановление в случае обнаружения противоречий в БД. Целостное состояние БД описывается с помощью ограничителей целостности в виде условий, которым должны удовлетворять хранимые в базе данные.

- Управление параллелизмом выполнения транзакций. Управление транзакциями является одной из наиболее важных задач СУБД. Развитые многопользовательские СУБД имеют достаточно совершенные средства управления параллелизмом выполнения транзакций, позволяющим множеству пользователей иметь одновременный доступ к данным.

- Аудит (регистрация всех обращений к защищаемой информации). Для определения активности использования БД анализируются ее файлы журнала. Эти же источники могут быть использованы для выявления любых необычных действий в системе. Регулярное проведение аудиторских проверок, дополненное постоянным контролем содержимого файлов журнала с целью выявления аномальной активности в системе, очень часто позволяет своевременно обнаружить и пресечь любые попытки нарушения защиты.

- Шифрование. Надежная (на определенное время) защита конфиденциальных данных возможна лишь с помощью шифрования данных. Шифрование, это кодирование данных с помощью специального алгоритма, в результате чего данные становятся недоступными для чтения. Для успешного дешифрования данных требуется знание ключа дешифрования. Некоторые СУБД предлагают встроенные средства шифрования хранимых данных, производящие дешифрование «на лету» в момент обращения пользователя к данным.

- Вспомогательные процедуры, такие как защита от SQL инъекций, агрегирования и инференции.

Некомпьютерные средства защиты включают принятие административных мер, выработку ограничений и соглашений. К ним относятся:

- Меры обеспечения безопасности и планирование защиты от непредвиденных обстоятельств;

- Контроль над персоналом и в целом контроль за физическим доступом;

- Защита помещений и хранилищ с помощью различного рода охранных систем;

- Заключение гарантийных соглашений и договоров о сопровождении аппаратных средств и программного обеспечения систем безопасности, производимых третьей стороной, что позволяет на определенный срок продлить уверенность в том, что эти средства не станут причиной реализации угроз безопасности.

В документах по мерам обеспечения безопасности должно быть определено следующее:

- область деловых процессов организации, для которой они устанавливаются;

- ответственность и обязанности отдельных работников;

- дисциплинарные меры, принимаемые в случае обнаружения нарушения установленных ограничений;

- перечень обязательно выполняемых процедур.

**Основные понятия и определения**

Понятие защиты применимо не только к сохраняемым данным. Бреши в системе защиты могут возникать и в других частях системы, например, линий связи, что в свою очередь, подвергает опасности и собственно БД. Следовательно, защита БД должна охватывать используемое оборудование, программное обеспечение, персонал и собственно данные.

***Информационная безопасность (***information security) – состояние рассматриваемой БД, при которой она, с одной стороны, способна противостоять дестабилизирующему воздействию внешних и внутренних информационных угроз, а с другой – функционирование системы и сам факт ее наличия не создают угроз для ее пользователей, для внешней среды и для элементов самой БД. Иначе, ***безопасность информации*** – состояние защищенности информации, при котором обеспечены ее конфиденциальность, целостность и доступность.

***Защита информации*** - комплекс мероприятий, направленных на обеспечение информационной безопасности (целостности, доступности и, если нужно, конфиденциальности информации и ресурсов, используемых для ввода, хранения, обработки и передачи данных).

***Защищаемая информация*** (sensitive information) – информация, являющаяся предметом собственности и подлежащая защите в соответствии с требованиями правовых документов или требованиями, устанавливаемыми собственником информации. Очень часто наряду с термином защищаемая информация употребляется и термин ***конфиденциальная информация*** – информация, доступ к которой ограничивается в соответствии с законодательством или требованиями владельца. ***Конфиденциальность*** (confidentiality или secrecy) - это свойство информации быть известной только допущенным и прошедшим проверку субъектам системы.

***Защищаемая информационная система*** – система, предназначенная для обработки защищаемой информации с требуемым уровнем ее защищенности.

***Система называется безопасной***, если она, используя соответствующие аппаратные и программные средства, управляет доступом к информации так, что только должным образом авторизованные лица или же действующие от их имени процессы получают право читать, писать, создавать и удалять информацию. Очевидно, что абсолютно безопасных систем нет, и здесь речь идет о надежной системе в смысле «система, которой можно доверять». ***Система считается надежной (иначе доверенной)***, если она с использованием достаточных аппаратных и программных средств обеспечивает одновременную обработку информации разной степени секретности группой пользователей без нарушения прав доступа.

Основными критериями оценки надежности являются политика безопасности и гарантированность.

***Политикой безопасности (***security policy) называют качественное описание комплекса организационно-технологических и программно-технических мер по обеспечению защиты данных в БД, включает в себя анализ возможных угроз и выбор соответствующих мер противодействия. Политика безопасности, являясь активным компонентом защиты, отображает тот набор законов, правил и норм поведения, которым пользуется конкретная организация при обработке, защите и распространении информации. Выбор конкретных механизмов обеспечения безопасности системы производится в соответствии со сформулированной политикой безопасности.

***Гарантированность***, являясь пассивным элементом защиты, отображает меру доверия, которое может быть оказано архитектуре и реализации системы (другими словами, показывает, насколько корректно выбраны механизмы, обеспечивающие безопасность системы). Гарантированность можно определить тестированием системы в целом и отдельных ее компонентов. При оценке степени гарантированности, с которой систему можно считать надежной, центральное место занимает доверенная (надежная) вычислительная база (ДВБ) (в английском – trusted computing base, TCB). ДВБ представляет собой полную совокупность защитных механизмов БД, которая используется для претворения в жизнь соответствующей политики безопасности. Надежность зависит исключительно от ее реализации и корректности введенных данных. Граница ДВБ образует периметр безопасности. Компоненты ДВБ, находящиеся внутри этой границы, должны быть надежными (следовательно, для оценки надежности БД достаточно рассмотреть только ее ДВБ). От компонентов, находящихся вне периметра безопасности, вообще говоря, не требуется надежности. Контроль допустимости выполнения субъектами определенных операций над объектами в рамках ДВБ выполняется монитором безопасности. При каждом обращении пользователя к программам или данным монитор проверяет допустимость данного обращения. Реализация монитора обращений, а иногда и всей ДВБ, называется ядром безопасности, на базе которого и строятся все защитные механизмы системы. Ядро безопасности должно гарантировать собственную неизменность.

Формальное (математическое, алгоритмическое, схемотехническое) выражение и формулирование политики безопасности называют ***моделью безопасности*** (security model). В основе большинства моделей безопасности лежит ***субъектно-объектная* модель** компьютерных систем*,* в том числе и баз данных как ядра автоматизированных информационных систем. Выделяются ***субъекты*** (subjects) базы данных (активные сущности), ***объекты*** (objects) базы данных (пассивные сущности) и порождаемые действиями субъектов ***процессы*** над объектами***.***

Таким образом, ***субъектом доступа*** (access subject) называется активный компонент системы, который может стать причиной инициализации потока информации или изменения состояния системы. Субъект представляет собой лицо или процесс, действия которого регламентируются правилами разграничения доступа.

***Объект доступа*** (access object) - пассивный компонент системы, хранящий, принимающий или передающий информацию, доступ к которой регламентируется правилами разграничения доступа. Объектами доступа в БД является практически все, что содержит конечную информацию: таблицы (базовые или виртуальные), представления, а также более мелкие элементы данных: столбцы и строки таблиц и даже отдельные поля.

***Правила разграничения доступа*** (security policy) - совокупность правил, регламентирующих права субъектов доступа к объектам доступа. Доступ к информации разделяют на санкционированный и несанкционированный: ***санкционированный доступ*** (authorized access to information) - доступ к информации, который не нарушает установленных правил разграничения доступа, а ***несанкционированный доступ*** (unauthorized access to information) - доступ к информации, который нарушает установленные правила разграничения. Контроль доступа обязательно включает идентификацию (персонализацию) и аутентификацию всех субъектов и их процессов и разграничение полномочий субъектов по отношению к объектам с последующей обязательной проверкой введенных полномочий.

***Идентификация*** (identification) - присвоение объектам и субъектам доступа идентификатора и (или) сравнение предъявляемого идентификатора с перечнем присвоенных идентификаторов. ***Идентификатор доступа*** (access identifier) - уникальный признак объекта или субъекта доступа.

***Аутентификация*** (authentication) - проверка принадлежности субъекту доступа предъявленного им идентификатора, подтверждение подлинности. Тем самым, задача идентификации – ответить на вопрос «кто это?», а задача аутентификации – «а он ли это на самом деле».

После идентификации и проверки подлинности устанавливается сфера действий субъекта (доступные ему ресурсы БД). Эту процедуру называют предоставлением полномочий (авторизацией).

***Авторизация (***authorization) - это предоставление прав (или привилегий), позволяющих их владельцу иметь законный доступ к системе или ее объектам. Иногда в процесс авторизации включает и идентификацию, и аутентификацию субъектов, требующих получения доступа к объектам. ***Привилегия доступа*** (уровень полномочий субъекта доступа) (subject privilege) - совокупность прав доступа субъекта доступа.

Вышеприведенные понятия в большей степени связаны с организацией защиты конфиденциальных данных. Наряду с защитой конфиденциальных данных в понятие безопасности данных входит также обеспечение целостности и доступности данных.

***Целостность данных*** (data integrity) - свойство данных сохранять точность и непротиворечивость независимо от внесенных изменений. Зависит от способности системы обеспечить неизменность данных в условиях случайного и (или) преднамеренного искажения (разрушения). Субъектов интересует обеспечение более широкого свойства – достоверности данных (информации), которая складывается из адекватности (полноты и точности) отображения состояния предметной области и непосредственно целостности данных, т. е. ее не искаженности.

***Доступность данных*** (data availability) - возможность реализации беспрепятственного доступа к информации субъектов, имеющих на это надлежащие полномочия. Обеспечивается способностью системы обеспечивать своевременный беспрепятственный доступ субъектов к запрашиваемым данным.

**Лекция 4. Классификация угроз информационной безопасности**

Под ***угрозой безопасности информации*** понимается осуществляемое или потенциально осуществимое воздействие на компьютерную систему, которое прямо или косвенно может нанести ущерб безопасности информации. Угрозы реализуют или пытаются реализовать ***нарушители*** информационной безопасности.

***Нарушитель информационной безопасности*** (security policy violator) - субъект доступа, случайно или преднамеренно совершивший действие, следствием которого является нарушение информационной безопасности организации. Формализованное описание или представление комплекса возможностей нарушителя по реализации тех или иных угроз безопасности информации называют ***моделью нарушителя*** (злоумышленника).

***Уязвимость*** – свойство информационной системы, обуславливающее возможность возникновения на каком-либо этапе жизненного цикла системы такого ее состояния, при котором создаются условия для реализации угроз безопасности информации.

***Атакой на систему*** называют действие, предпринимаемое нарушителем, которое заключается в поиске и использовании той или иной уязвимости. Иначе говоря, атака на систему является реализацией угрозы безопасности информации в ней.

Классификация угроз может быть проведена по множеству признаков. На самом верхнем уровне угрозы информационной безопасности могут быть классифицированы по природе возникновения. Все угрозы информационной безопасности можно разделить на угрозы, не зависящие от деятельности человека (*естественные угрозы*), например стихийные природные явления, и угрозы, вызванные человеческой деятельностью (*искусственные угрозы*). Последние являются наиболее опасными угрозами и им уделяется повышенное внимание.

Искусственные угрозы, исходя из их мотивов, разделяются на *непреднамеренные* (случайные) и *преднамеренные* (умышленные) (классификация по *степени преднамеренности*). К непреднамеренным угрозам относятся:

- ошибки в проектировании системы;

- ошибки в разработке программных средств системы;

- случайные сбои в работе аппаратных средств системы, линий связи, энергоснабжения;

- ошибки пользователей системы (некомпетентное использование средств защиты, а также любых других программных и аппаратных средств системы);

- воздействие на аппаратные средства системы физических полей других электронных устройств и др.

Преднамеренные угрозы связаны с целенаправленными действиями нарушителя. Действия нарушителя могут быть обусловлены самыми разнообразными мотивами: недовольство служащего своей карьерой, материальным интересом, любопытством, конкурентной борьбой, стремлением самоутвердиться и т.д. В зависимости от целей преднамеренных угроз безопасности информации в системе угрозы могут быть разделены на четыре основные группы:

*Похищение и фальсификация данных.* Может происходить не только в среде БД, вся организация подвержена этому риску. Однако действия по похищению и фальсификации данных всегда производятся людьми, поэтому основное внимание должно быть сосредоточено на сокращении общего количества удобных ситуаций для выполнения подобных действий;

*Утрата конфиденциальности* (нарушение тайны) и *нарушение неприкосновенности личных данных*, т.е. утечка информации ограниченного доступа, хранящейся в системе или передаваемой от одной системы к другой. Следствием разглашения конфиденциальных данных может быть утрата позиций в конкурентной борьбе, тогда как следствием нарушения неприкосновенности личных данных могут быть юридические меры, принятые в отношении этих лиц или организации;

*Утрата целостности данных*, т. е. преднамеренное воздействие на информацию, хранящуюся в системе или передаваемую между системами, приводящее к потере согласованности данных (достоверности) данных. Целостность информации может быть также нарушена, если к несанкционированному изменению или уничтожению информации приводит случайная ошибка в работе программных или аппаратных средств системы. Напротив, санкционированным является изменение или уничтожение информации, сделанное уполномоченным лицом с обоснованной целью;

*Потеря доступности данных*, т. е. отказ в обслуживании, вызванный преднамеренными действиями одного из пользователей системы (нарушителя), при котором блокируется доступ к некоторому ресурсу системы со стороны других пользователей (постоянно или на большой период времени). Потеря доступности может быть также вызвана умышленным или неумышленным разрушением данных.

Опосредованной угрозой безопасности информации в системе является угроза раскрытия параметров подсистемы защиты информации, входящей в состав компьютерной системы. Реализация этой угрозы дает возможность реализации перечисленных ранее непосредственных угроз безопасности информации. Выделяют следующие преднамеренные угрозы информационной безопасности БД:

- несанкционированный доступ посторонних лиц и ознакомление с хранимой конфиденциальной информацией;

-ознакомление легальных лиц с информацией, к которой они не должны иметь доступ;

- несанкционированное копирование программ и данных;

- кража носителей, содержащих конфиденциальную информацию;

- кража распечатанных документов;

- кража оборудования;

- умышленное уничтожение данных, в том числе разрушение информации, вызванное вирусными воздействиями;

- несанкционированная модификация данных;

- фальсификация сообщений, передаваемых по каналам связи;

- отказ от авторства сообщения, переданного по каналам связи или отказ от факта получения информации;

- навязывание ранее переданного сообщения.

Несанкционированный доступ (НСД) является наиболее распространенным и многообразным видом компьютерных нарушений. Суть НСД состоит в получении пользователем (нарушителем) доступа к объекту в нарушение правил разграничения доступа, установленных в соответствии с принятой в организации политикой безопасности. НСД использует любую ошибку в системе защиты и возможен при нерациональном выборе средств защиты, их некорректной установке и настройке. НСД может быть осуществлен как штатными средствами системы, так и специально созданными аппаратными и программными средствами. Из всего разнообразия способов и приемов несанкционированного доступа остановимся на следующих распространенных и связанных между собой нарушениях:

- Перехват паролей или обход защиты *(Bypassing Controls*). Перехват паролей обычно осуществляется специально разработанными программами. Например, при попытке законного пользователя войти в систему программа-перехватчик имитирует на экране дисплея ввод имени и пароля пользователя, которые сразу пересылаются владельцу программы-перехватчика, после чего на экран выводится сообщение об ошибке и управление возвращается операционной системе. Пользователь предполагает, что допустил ошибку при вводе пароля. Он повторяет ввод и получает доступ в систему. Владелец программы-перехватчика, получивший имя и пароль законного пользователя, может теперь использовать их в своих целях. Разумеется, существуют и другие способы перехвата паролей;

-Маскарад *(Masquerade*). Маскарад – это выполнение каких-либо действий одним пользователем от имени другого пользователя, обладающего соответствующими полномочиями. Целью маскарада является приписывание каких-либо действий другому пользователю либо присвоение полномочий и привилегий другого пользователя. Примерами реализации маскарада являются: вход в систему под именем и паролем другого пользователя (этому маскараду предшествует перехват пароля) и передача сообщений в сети от имени другого пользователя;

- Незаконное использование привилегий *(Misuses of authority*). Большинство систем защиты устанавливают определенные наборы привилегий для выполнения заданных функций. Каждый пользователь получает свой набор привилегий. Несанкционированный захват привилегий, например, посредством маскарада, приводит к возможности выполнения нарушителем определенных действий в обход системы защиты. Следует отметить, что незаконный захват привилегий возможен либо при наличии ошибок в системе защиты, либо из-за халатности администратора при управлении системой и назначении привилегий.

Так как доступ к БД обычно производится из удаленного компьютера и фрагменты БД могут быть распределены по нескольким узлам сети, то следует остановиться на угрозах, которым могут подвергаться и компьютерные сети. Связь между узлами сети осуществляется физически с помощью сетевых линий связи и программно с помощью механизма сообщений. При этом управляющие сообщения и данные, пересылаемые между объектами сети, передаются в виде пакетов. При вторжении в компьютерную сеть злоумышленник может использовать как пассивные, так и активные методы вторжения.

При пассивном вторжении (перехвате информации) нарушитель только наблюдает за прохождением информации по каналу связи, не вторгаясь ни в информационный поток, ни в содержание передаваемой информации. Как правило, злоумышленник может определить пункты назначения и идентификаторы либо только факт прохождения сообщения, его длину и частоту обмена, если содержимое сообщения не распознаваемо, т.е. выполнить анализ трафика (потока сообщений) в данном канале. При активном вторжении нарушитель стремится подменить информацию, передаваемую в сообщении. Он может выборочно модифицировать, изменить или добавить правильное или ложное сообщение, удалить, задержать или изменить порядок следования сообщений. Злоумышленник может также аннулировать и задержать все сообщения, передаваемые по каналу. Подобные действия можно квалифицировать как отказ в передаче сообщений. Отказ от авторства сообщения, переданного по каналам связи, отказ от факта получения информации, навязывание ранее переданного сообщения также можно отнести к угрозам сетевой безопасности.

Компьютерные сети характерны еще тем, что кроме обычных локальных атак, осуществляемых в пределах одной системы, против объектов сетей предпринимают так называемые удаленные атаки. Под ***удаленной атакой*** понимают информационное разрушающее воздействие на распределенную компьютерную сеть, программно осуществленную по каналам связи.

Результатом реализации угроз безопасности информации в компьютерной системе может быть утечка (копирование) информации, ее утрата (разрушение) или искажение (подделка), блокирование информации. Поскольку сложно заранее определить возможную совокупность угроз безопасности информации и результатов их реализации, модель потенциальных угроз безопасности информации в компьютерной системе должна создаваться совместно собственником (владельцем) системы и специалистами по защите информации на этапе проектирования системы. Созданная модель должна затем уточняться в ходе эксплуатации системы.

**Лекция 5. Основные принципы обеспечения информационной безопасности**

Рассмотренные подходы обеспечения безопасности могут быть реализованы при обеспечении следующих основных принципов:

- системности;

- комплексности;

- непрерывности защиты;

- разумной достаточности;

- гибкости управления и применения;

- открытости алгоритмов и механизмов защиты;

- простоты применения защитных мер и средств.

***Принцип системности***. Системный подход к защите компьютерных систем предполагает необходимость учета всех взаимосвязанных, взаимодействующих и изменяющихся во времени элементов, условий и факторов на всех этапах жизненного цикла с учетом взаимодействия объекта защиты с внешней средой. Система защиты должна строиться не только с учетом всех известных каналов проникновения, но и с учетом возможности появления принципиально новых путей реализации угроз безопасности.

***Принцип комплексности***. В распоряжении специалистов по компьютерной безопасности имеется широкий спектр мер, методов и средств защиты компьютерных систем. В частности, современные средства вычислительной техники, операционные системы и прикладное программное обеспечение обладают теми или иными встроенными элементами защиты. Комплексное их использование предполагает согласование разнородных средств при построении целостной системы защиты, перекрывающей все существенные каналы реализации угроз и не содержащей слабых мест на стыках отдельных ее компонентов.

***Принцип непрерывности защиты.*** Защита информации - это не разовое мероприятие и даже не конкретная совокупность мероприятий по разворачиванию и настройке средств защиты, а непрерывный целенаправленный процесс, предполагающий принятие соответствующих мер на всех этапах жизненного цикла компьютерной системы (начиная с самых ранних стадий проектирования, а не только на этапе ее эксплуатации). Разработка системы защиты должна вестись параллельно с разработкой самой защищаемой системы. Это позволит учесть требования безопасности при проектировании архитектуры и, в конечном счете, позволит создать более эффективные (как по затратам ресурсов, так и по стойкости) защищенные системы. Большинству физических и технических средств защиты для эффективного выполнения своих функций необходима постоянная организационная (административная) поддержка (своевременная смена и обеспечение правильного хранения и применения имен, паролей, ключей шифрования, переопределение полномочий и т.п.). Перерывы в работе средств защиты могут быть использованы злоумышленниками для анализа применяемых методов и средств защиты, внедрения специальных программных и аппаратных «закладок» и других средств преодоления системы защиты после восстановления ее функционирования.

***Разумная достаточность.*** Создать абсолютно непреодолимую систему защиты принципиально невозможно: при достаточных времени и средствах можно преодолеть любую защиту. Например, средства криптографической защиты в большинстве случаев не гарантируют абсолютную стойкость, а обеспечивают конфиденциальность информации при использовании для дешифрования современных вычислительных средств в течение приемлемого для защищающейся стороны времени. Поэтому имеет смысл вести речь только о некотором приемлемом уровне безопасности. Высокоэффективная система защиты стоит дорого, использует при работе существенную часть мощности и ресурсов компьютерной системы и может создавать ощутимые дополнительные неудобства пользователям. Важно правильно выбрать тот достаточный уровень зашиты, при котором затраты, риск и размер возможного ущерба были бы приемлемыми (задача анализа риска).

***Гибкость системы защиты***. Часто приходится создавать систему защиты в условиях большой неопределенности. Поэтому принятые меры и установленные средства защиты, особенно в начальный период их эксплуатации, могут обеспечивать как чрезмерный, так и недостаточный уровень защиты. Естественно, что для обеспечения возможности варьирования уровнем защищенности средства защиты должны обладать определенной гибкостью. Особенно важно это свойство в тех случаях, когда средства защиты необходимо устанавливать на работающую систему, не нарушая процесс ее нормального функционирования. Кроме того, внешние условия и требования с течением времени меняются. В таких ситуациях свойство гибкости спасает владельцев компьютерной системы от необходимости принятия кардинальных мер по полной замене средств защиты.

***Открытость алгоритмов и механизмов защиты***. Суть принципа открытости алгоритмов и механизмов защиты состоит в том, что защита не должна обеспечиваться только за счет секретности структурной организации и алгоритмов функционирования ее подсистем. Знание алгоритмов работы системы защиты не должно давать возможности ее преодоления (даже автору). Однако это вовсе не означает, что информация о конкретной системе защиты должна быть общедоступна - необходимо обеспечивать защиту от угрозы раскрытия параметров системы.

***Принцип простоты применения средств защиты***. Механизмы защиты должны быть интуитивно понятны и просты в использовании. Применение средств защиты не должно быть связано со знанием специальных языков или с выполнением действий, требующих значительных дополнительных трудозатрат при обычной работе законных пользователей, а также не должно требовать от пользователя выполнения рутинных малопонятных ему операций (ввод нескольких паролей и имен и т.д.). Это неизбежно приведет к вывешиванию списка паролей на самом видном месте, например, на монитор.

**Лекция 6: Модели транзакций**

Транзакцией называется последовательность операций, производимых над базой данных и переводящих базу данных из одного непротиворечивого (согласованного) состояния в другое непротиворечивое (согласованное) состояние.

*Транзакция* рассматривается как некоторое неделимое действие над базой данных, осмысленное с точки зрения пользователя. В то же время это логическая *единица* работы системы. Рассмотрим несколько примеров. Что может быть названо транзакцией? Кем определяется, какая последовательность операций над базой данных составляет транзакцию? Конечно, однозначно именно разработчик определяет, какая последовательность операций составляет единое целое, то есть транзакцию. Разработчик приложений или хранимых процедур определяет это исходя из смысла обработки данных, именно *семантика* совокупности операций над базой данных, которая моделирует с точки зрения разработчика некоторую одну неразрывную работу, и составляет транзакцию. Допустим, выделим работу *по* вводу данных о поступивших книгах, новых книгах, которых не было раньше в библиотеке. Тогда эту операцию можно разбить на две последовательные: сначала ввод данных о книге — это новая строка в таблице BOOKS, а потом ввод данных обо всех экземплярах новой книги — это ввод набора новых строк в таблицу *EXEMPLAR* в количестве, равном количеству поступивших экземпляров книги. Если эта последовательность *работ* будет прервана, то наша *база данных* не будет соответствовать реальному объекту, поэтому желательно выполнять ее как единую работу над базой данных.

Следующий пример, который связан с принятием заказа в фирме на изготовление компьютера. *Компьютер* состоит из комплектующих, которые сразу резервируются за данным заказом в момент его формирования. Тогда транзакцией будет вся последовательность операций, включающая следующие *операции*:

* ввод нового заказа со всеми реквизитами заказчика;
* изменения состояния для всех выбранных комплектующих на складе на "занято" с привязкой их к определенному заказу;
* подсчет стоимости заказа с формированием платежного документа типа выставляемого счета к оплате;
* включение нового заказа в производство.

С точки зрения работника, это единая последовательность операций; если она будет прервана, то *база данных* потеряет свое *целостное состояние*.

Еще один пример, который весьма характерен для учебных заведений. При длительной болезни преподавателя или при его увольнении перед администрацией кафедры встает задача перераспределения всей нагрузки, которую ведет преподаватель, *по* другим преподавателям кафедры. Видов нагрузки может быть несколько: чтение лекций и проведение занятий *по* текущему расписанию, руководство квалификационными работами бакалавров, руководство дипломными проектами специалистов, руководство магистерскими диссертациями, индивидуальная научно-исследовательская работа со студентами. И для каждого вида нагрузки необходимо найти исполнителей и назначить им дополнительную нагрузку.

### **Свойства транзакций. Способы завершения транзакций**

Существуют различные модели транзакций, которые могут быть классифицированы на основании различных свойств, включающих структуру транзакции, параллельность внутри транзакции, продолжительность и т. д.

В настоящий момент выделяют следующие типы транзакций: плоские или классические транзакции, цепочечные транзакции и *вложенные транзакции*.

Плоские, или традиционные, транзакции, характеризуются четырьмя классическими свойствами: атомарности, согласованности, изолированности, долговечности (прочности) — *ACID* (Atomicity, Consistency, *Isolation*, *Durability*). Иногда традиционные транзакции называют *ACID-транзакциями*. Упомянутые выше свойства означают следующее:

* *Свойство атомарности*(Atomicity) выражается в том, что транзакция должна быть выполнена в целом или не выполнена вовсе.
* *Свойство согласованности*(Consistency) гарантирует, что по мере выполнения транзакций данные переходят из одного согласованного состояния в другое — транзакция не разрушает взаимной согласованности данных.
* *Свойство изолированности*(*Isolation*) означает, что конкурирующие за доступ к базе данных транзакции физически обрабатываются последовательно, изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно.
* *Свойство долговечности*(*Durability*) трактуется следующим образом: если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах (даже в случае последующих ошибок).

Возможны два варианта завершения транзакции. Если все *операторы* выполнены успешно и в процессе выполнения транзакции не произошло никаких сбоев программного или аппаратного обеспечения, *транзакция* фиксируется.

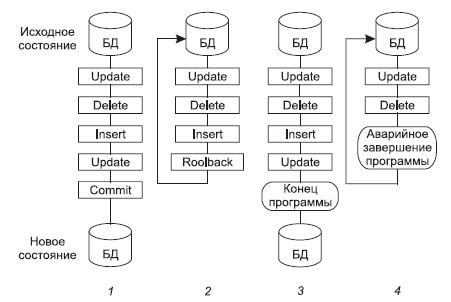
*Фиксация транзакции* — это действие, обеспечивающее *запись* на *диск* изменений в базе данных, которые были сделаны в процессе выполнения транзакции.

До тех пор пока *транзакция* не зафиксирована, допустимо *аннулирование* этих изменений, восстановление *базы данных* в то состояние, в котором она была на момент начала транзакции. *Фиксация транзакции* означает, что все результаты выполнения транзакции становятся постоянными. Они станут видимыми другим транзакциям только после того, как текущая *транзакция* будет зафиксирована. До этого момента все данные, затрагиваемые транзакцией, будут "видны" пользователю в состоянии на начало текущей транзакции.

Если в процессе выполнения транзакции случилось нечто такое, что делает невозможным ее нормальное завершение, *база данных* должна быть возвращена в исходное состояние. *Откат* транзакции — это действие, обеспечивающее *аннулирование* всех изменений данных, которые были сделаны операторами *SQL* в теле текущей незавершенной транзакции.

Каждый оператор в транзакции выполняет свою часть работы, но для успешного завершения всей работы в целом требуется безусловное завершение всех операторов. Группирование операторов в транзакции сообщает *СУБД*, что вся эта *группа* должна быть выполнена как единое целое, причем такое выполнение должно поддерживаться автоматически.

В стандарте ANSI/ISO *SQL* определены модель транзакций и функции операторов COMMIT и ROLLBACK. Стандарт определяет, что *транзакция* начинается с первого *SQL*-оператора, инициируемого пользователем или содержащегося в программе, изменяющего текущее состояние *базы данных*. Все последующие *SQL*-*операторы* составляют тело транзакции. *Транзакция* завершается одним из четырех возможных путей ([рис. 11.1](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=1#image.11.1)):



**Рис. 11.1.**Модель транзакций ANSI/ISO

1. оператор COMMIT означает успешное завершение транзакции; его использование делает постоянными изменения, внесенные в базу данных в рамках текущей транзакции;
2. оператор ROLLBACK прерывает транзакцию, отменяя изменения, сделанные в базе данных в рамках этой транзакции; новая транзакция начинается непосредственно после использования ROLLBACK ;
3. успешное завершение программы, в которой была инициирована текущая транзакция, означает успешное завершение транзакции (как будто был использован оператор COMMIT );
4. ошибочное завершение программы прерывает транзакцию (как будто был использован оператор ROLLBACK ).

В этой модели каждый оператор, который изменяет состояние *БД*, рассматривается как *транзакция*, поэтому при успешном завершении этого оператора *БД* переходит в новое устойчивое состояние.

### **Журнал транзакций**

Реализация в *СУБД* принципа подтверждения или отката транзакции обеспечивается специальным механизмом, для поддержки которого создается некоторая системная структура, называемая *Журналом транзакций*.

Однако назначение журнала транзакций гораздо шире. Он предназначен для обеспечения надежного хранения данных в *БД*.

А это требование предполагает, в частности, возможность восстановления согласованного состояния *базы данных* после любого рода аппаратных и программных сбоев. Очевидно, что для выполнения восстановлений необходима некоторая дополнительная *информация*. В подавляющем большинстве современных реляционных *СУБД* такая избыточная дополнительная *информация* поддерживается в виде *журнала изменений* *базы данных*, чаще всего называемого *Журналом транзакций*.

Итак, общей целью журнализации изменений баз данных является обеспечение возможности восстановления согласованного состояния *базы данных* после любого сбоя. Поскольку основой поддержания *целостного состояния* *базы данных* является механизм транзакций, *журнализация* и восстановление тесно связаны с понятием транзакции. Общими принципами восстановления являются следующие:

* результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных;
* результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных.

Это, собственно, и означает, что восстанавливается последнее *по* времени согласованное состояние *базы данных*.

Возможны следующие ситуации, при которых требуется производить восстановление состояния *базы данных*.

* Индивидуальный откат транзакции. Этот откат должен быть применен в следующих случаях:
* стандартной ситуацией отката транзакции является ее явное завершение оператором ROLLBACK ;
* аварийное завершение работы прикладной программы, которое логически эквивалентно выполнению оператора ROLLBACK, но физически имеет иной механизм выполнения;
* принудительный откат транзакции в случае взаимной блокировки при параллельном выполнении транзакций. В подобном случае для выхода из тупика данная транзакция может быть выбрана в качестве "жертвы" и принудительно прекращено ее выполнение ядром СУБД.
* Восстановление после внезапной потери содержимого оперативной памяти (мягкий сбой). Такая ситуация может возникнуть в следующих случаях:
  + при аварийном выключении электрического питания;
  + при возникновении неустранимого сбоя процессора (например, срабатывании контроля оперативной памяти) и т. д. Ситуация характеризуется потерей той части базы данных, которая к моменту сбоя содержалась в буферах оперативной памяти.
* Восстановление после поломки основного внешнего носителя базы данных (жесткий сбой). Эта ситуация при достаточно высокой надежности современных *устройств внешней памяти* может возникать сравнительно редко, но тем не менее СУБД должна быть в состоянии восстановить базу данных даже и в этом случае. Основой восстановления является архивная копия и *журнал изменений* базы данных.

Для восстановления согласованного состояния *базы данных* при индивидуальном откате транзакции нужно устранить последствия операторов модификации *базы данных*, которые выполнялись в этой транзакции. Для восстановления непротиворечивого состояния *БД* при мягком сбое необходимо восстановить содержимое *БД* *по* содержимому журналов транзакций, хранящихся на дисках. Для восстановления согласованного состояния *БД* при жестком сбое надо восстановить содержимое *БД* *по* архивным копиям и журналам транзакций, которые хранятся на неповрежденных внешних носителях.

Во всех трех случаях основой восстановления является избыточное *хранение данных*. Эти избыточные данные хранятся в журнале, содержащем последовательность записей об изменении *базы данных*.

Возможны два основных варианта ведения журнальной информации. В первом варианте для каждой транзакции поддерживается отдельный локальный *журнал изменений* *базы данных* этой транзакцией. Такие журналы называются локальными журналами. Они используются для индивидуальных откатов транзакций и могут поддерживаться в оперативной (правильнее сказать, в виртуальной) памяти. Кроме того, поддерживается общий *журнал изменений* *базы данных*, используемый для восстановления состояния *базы данных* после мягких и жестких сбоев.

Этот подход позволяет быстро выполнять индивидуальные откаты транзакций, но приводит к дублированию информации в локальных и общем журналах. Поэтому чаще используется второй вариант — поддержание только общего *журнала изменений* *базы данных*, который используется и при выполнении индивидуальных откатов. Далее мы рассматриваем именно этот вариант.

Общая структура журнала условно может быть представлена в виде некоторого *последовательного файла*, в котором фиксируется каждое изменение *БД*, которое происходит в ходе выполнения транзакции. Все транзакции имеют свои внутренние номера, поэтому в едином журнале транзакций фиксируются все изменения, проводимые всеми транзакциями.

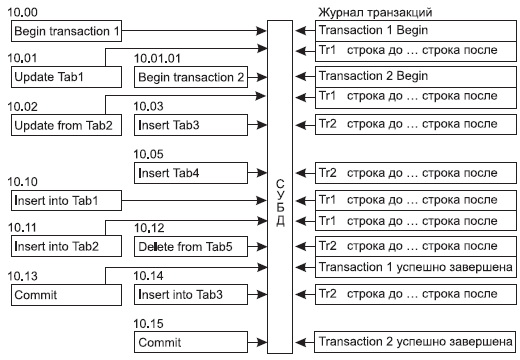
Каждая *запись* в журнале транзакций помечается номером транзакции, к которой она относится, и значениями атрибутов, которые она меняет. Кроме того, для каждой транзакции в журнале фиксируется *команда* начала и завершения транзакции (см. [рис. 11.3](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=2#image.11.3)).

Для большей надежности *журнал транзакций* часто дублируется системными средствами коммерческих *СУБД*, именно поэтому объем внешней памяти во много раз превышает реальный объем данных, которые хранятся в хранилище.

Имеются два альтернативных варианта ведения журнала транзакций: протокол с отложенными обновлениями и протокол с немедленными обновлениями.

Ведение журнала *по* принципу отложенных изменений предполагает следующий механизм выполнения транзакций:

1. Когда транзакция Т1 начинается, в протокол заносится запись <Т1 Begin transaction>
2. На протяжении выполнения транзакции в протоколе для каждой изменяемой записи записывается новое значение: <T1,ID\_RECORD, атрибут, новое значение … >. Здесь ID\_RECORD — уникальный номер записи.
3. Если все действия, из которых состоит транзакция Т1, успешно выполнены, то транзакция частично фиксируется и в протокол заносится <Т1 COMMIT>.
4. После того как транзакция фиксирована, записи протокола, относящиеся к T1, используются для внесения соответствующих изменений в БД.
5. Если происходит сбой, то СУБД просматривает протокол и выясняет, какие транзакции необходимо переделать. Транзакцию Т1 необходимо переделать, если протокол содержит обе записи <T1 BEGIN TRANSACTION> и <T1 COMMIT>. БД может находиться в несогласованном состоянии, однако все новые значения измененных элементов данных содержатся в протоколе, и это требует повторного выполнения транзакции. Для этого используется некоторая системная процедура REDO(), которая заменяет все значения элементов данных на новые, просматривая протокол в прямом порядке.
6. Если в протоколе не содержится команда *фиксации транзакции* COMMIT, то никаких действий проводить не требуется, а транзакция запускается заново.



**Рис. 11.3.**Журнал транзакций

Альтернативный механизм с немедленным выполнением предусматривает внесение изменений сразу в *БД*, а в протокол заносятся не только новые, но и все старые значения изменяемых атрибутов, поэтому каждая *запись* выглядит <Т1, IDRECORD, атрибут новое значение старое значение ...>. При этом *запись* в журнал предшествует непосредственному выполнению *операции* над *БД*. Когда *транзакция* фиксируется, то есть встречается *команда* <T1 COMMIT> и она выполняется, то все изменения оказываются уже внесенными в *БД* и не требуется никаких дальнейших действий *по* отношению к этой транзакции.

При откате транзакции выполняется системная процедура UNDO(), которая возвращает все старые значения в отмененной транзакции, последовательно проходя *по* протоколу начиная с команды BEGIN TRANSACTION.

Для восстановления при сбое используется следующий механизм:

* Если транзакция содержит команду начала транзакции, но не содержит команды фиксации с подтверждением ее выполнения, то выполняется последовательность действий как при откате транзакции, то есть восстанавливаются старые значения.
* Если сбой произошел после выполнения последней команды изменения БД, но до выполнения команды фиксации, то команда фиксации выполняется, а с БД никаких изменений не происходит. Работа происходит только на уровне протокола.
* Однако следует отметить, что проблемы восстановления выглядят гораздо сложнее приведенных ранее алгоритмов, с учетом того, что изменения как в журнал, так и в БД заносятся не сразу, а буферируются. Этому посвящен следующий раздел.

### **Журнализация и буферизация**

*Журнализация* изменений тесно связана не только с управлением транзакциями, но и с буферизацией страниц *базы данных* в оперативной памяти.

Если бы *запись* об изменении *базы данных*, которая должна поступить в журнал при выполнении любой *операции* модификации *базы данных*, реально немедленно записывалась бы во внешнюю *память*, это привело бы к существенному замедлению работы системы. Поэтому записи в журнале тоже буферизуются: при нормальной работе очередная страница выталкивается во внешнюю *память* журнала только при полном заполнении записями.

Проблема состоит в выработке некоторой общей политики выталкивания, которая обеспечивала бы возможность восстановления состояния *базы данных* после сбоев.

Проблема не возникает при индивидуальных откатах транзакций, поскольку в э-тих случаях содержимое оперативной памяти не утрачено и можно пользоваться содержимым как буфера журнала, так и буферов страниц *базы данных*. Но если произошел мягкий сбой и содержимое буферов утрачено, для проведения восстановления *базы данных* необходимо иметь некоторое согласованное состояние журнала и *базы данных* во внешней памяти.

Основным принципом согласованной политики выталкивания буфера журнала и буферов страниц *базы данных* является то, что *запись* об изменении объекта *базы данных* должна попадать во внешнюю *память* журнала раньше, чем измененный *объект* оказывается во внешней памяти *базы данных*. Соответствующий протокол журнализации (и управления буферизацией) называется *Write* Ahead *Log* (WAL) — "пиши сначала в журнал" и состоит в том, что если требуется записать во внешнюю *память* измененный *объект* *базы данных*, то перед этим нужно гарантировать *запись* во внешнюю *память* журнала транзакций записи о его изменении.

Другими словами, если во внешней памяти *базы данных* находится некоторый *объект* *базы данных*, *по* отношению к которому выполнена операция модификации, то во внешней памяти журнала обязательно находится *запись*, соответствующая этой *операции*. Обратное неверно, то есть если во внешней памяти журнале содержится *запись* о некоторой *операции* изменения объекта *базы данных*, то сам измененный *объект* может отсутствовать во внешней памяти *базы данных*.

Дополнительное условие на выталкивание буферов накладывается тем требованием, что каждая успешно завершившаяся *транзакция* должна быть реально зафиксирована во внешней памяти. Какой бы сбой не произошел, система должна быть в состоянии восстановить состояние *базы данных*, содержащее результаты всех зафиксированных к моменту сбоя транзакций.

Простым решением было бы выталкивание буфера журнала, за которым следует массовое выталкивание буферов страниц *базы данных*, изменявшихся данной транзакцией. Довольно часто так и делают, но это вызывает существенные накладные *расходы* при выполнении *операции* *фиксации транзакции*.

Оказывается, что минимальным требованием, гарантирующим возможность восстановления последнего согласованного состояния *базы данных*, является выталкивание при *фиксации транзакции* во внешнюю *память* журнала всех записей об изменении *базы данных* этой транзакцией. При этом последней записью в журнал, производимой от имени данной транзакции, является специальная *запись* о конце транзакции.

Рассмотрим теперь, как можно выполнять *операции* восстановления *базы данных* в различных ситуациях, если в системе поддерживается общий для всех транзакций журнал с общей буферизацией записей, поддерживаемый в соответствии с протоколом WAL.

#### **Индивидуальный откат транзакции**

Для того чтобы можно было выполнить по общему журналу индивидуальный откат транзакции, все записи в журнале по данной транзакции связываются в обратный список. Началом списка для незакончившихся транзакций является запись о последнем изменении базы данных, произведенном данной транзакцией. Для закончившихся транзакций (индивидуальные откаты которых уже невозможны) началом списка является запись о конце транзакции, которая обязательно вытолкнута во внешнюю память журнала. Концом списка всегда служит первая запись об изменении базы данных, произведенном данной транзакцией. Обычно в каждой записи проставляется уникальный идентификатор транзакции, чтобы можно было восстановить прямой список записей об изменениях базы данных данной транзакцией.

Итак, индивидуальный откат транзакции (еще раз подчеркнем, что это возможно только для незакончившихся транзакций) выполняется следующим образом:

* Выбирается очередная запись из списка данной транзакции.
* Выполняется противоположная по смыслу операция: вместо операции INSERT выполняется соответствующая операция DELETE, вместо операции DELETE выполняется INSERT и вместо прямой операции UPDATE обратная операция UPDATE, восстанавливающая предыдущее состояние объекта базы данных.
* Любая из этих обратных операций также заносится в журнал. Собственно, для индивидуального отката это не нужно, но при выполнении индивидуального отката транзакции может произойти мягкий сбой, при восстановлении после которого потребуется откатить такую транзакцию, для которой не полностью выполнен индивидуальный откат.
* При успешном завершении отката в журнал заносится запись о конце транзакции. С точки зрения журнала такая транзакция является зафиксированной.

#### **Восстановление после мягкого сбоя**

К числу основных проблем восстановления после мягкого сбоя относится то, что одна логическая операция изменения базы данных может изменять несколько физических блоков базы данных, например, страницу данных и несколько страниц индексов. Страницы базы данных буферизуются в оперативной памяти и выталкиваются независимо. Несмотря на применение протокола WAL, после мягкого сбоя набор страниц внешней памяти базы данных может оказаться несогласованным, то есть часть страниц внешней памяти соответствует объекту до изменения, часть — после изменения. К такому состоянию объекта неприменимы операции логического уровня.

Состояние внешней памяти базы данных называется физически согласованным, если наборы страниц всех объектов согласованы, то есть соответствуют состоянию объекта либо до его изменения, либо после изменения.

Будем считать, что в журнале отмечаются точки физической согласованности базы данных — моменты времени, в которые во внешней памяти содержатся согласованные результаты операций, завершившихся до соответствующего момента времени, и *отсутствуют результаты* операций, которые не завершились, а буфер журнала вытолкнут во внешнюю память. Немного позже мы рассмотрим, как можно достичь физической согласованности. Назовем такие точки *tpc* (time of physical consistency) — точками физического согласования.

Тогда к моменту мягкого сбоя возможны следующие состояния транзакций:

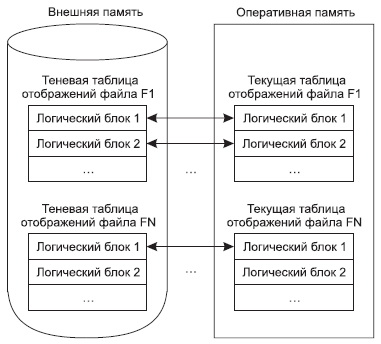
* транзакция успешно завершена, то есть выполнена операция подтверждения транзакции COMMIT и для всех операций транзакции получено подтверждение ее выполнения во внешней памяти;
* транзакция успешно завершена, но для некоторых операций не получено подтверждение их выполнения во внешней памяти;
* транзакция получила и выполнила команду отката **ROLLBACK** ;
* транзакция не завершена.

#### **Физическая согласованность базы данных**

Каким же образом можно обеспечить наличие точек физической согласованности базы данных, то есть как восстановить состояние базы данных в момент *tpc*? Для этого используются два основных подхода: подход, основанный на использовании теневого механизма, и подход, в котором применяется *журнализация* постраничных изменений базы данных.

При открытии файла таблица отображения номеров его логических блоков в адреса физических блоков внешней памяти считывается в оперативную память. При модификации любого блока файла во внешней памяти выделяется новый блок. При этом текущая таблица отображения (в оперативной памяти) изменяется, а теневая — сохраняется неизменной. Если во время работы с открытым файлом происходит сбой, во внешней памяти автоматически сохраняется состояние файла до его открытия. Для явного восстановления файла достаточно повторно считать в оперативную память теневую таблицу отображения.

Общая идея теневого механизма показана на [рис. 11.4](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=3#image.11.4).



**Рис. 11.4.**Использование теневых таблиц отображения информации

В контексте базы данных теневой механизм используется следующим образом. Периодически выполняются операции установления точки физической согласованности базы данных (*checkpoints*). Для этого все логические операции завершаются, все буферы оперативной памяти, содержимое которых не соответствует содержимому соответствующих страниц внешней памяти, выталкиваются. Теневая таблица отображения файлов базы данных заменяется на текущую (правильнее сказать, текущая таблица отображения записывается на место теневой).

Восстановление к *tpc* происходит мгновенно: текущая таблица отображения заменяется на теневую (при восстановлении просто считывается теневая таблица отображения). Все проблемы восстановления решаются, но за счет слишком большого перерасхода внешней памяти. В пределе может потребоваться вдвое больше внешней памяти, чем реально нужно для хранения базы данных. Теневой механизм — это надежное, но слишком грубое средство. Обеспечивается согласованное состояние внешней памяти в один общий для всех объектов момент времени. На самом деле достаточно иметь совокупность согласованных наборов страниц, каждому из которых может соответствовать свои временные отсчеты.

Для выполнения такого более слабого требования наряду с логической журнали-зацией операций изменения базы данных производится *журнализация* постраничных изменений. Первый этап восстановления после мягкого сбоя состоит в постраничном откате незакончившихся логических операций. Подобно тому как это делается с логическими записями по отношению к транзакциям, последней записью о постраничных изменениях от одной логической операции является запись о конце операции.

В этом подходе имеются два метода решения проблемы. При использовании первого метода поддерживается общий журнал логических и страничных операций. Естественно, наличие двух видов записей, интерпретируемых абсолютно по-разному, усложняет структуру журнала. Кроме того, записи о постраничных изменениях, актуальность которых носит локальный характер, существенно (и не очень осмысленно) увеличивают журнал.

Поэтому все более популярным становится поддержание отдельного (короткого) журнала постраничных изменений. Такая техника применяется, например, в известном продукте Informix Online.

Предположим, что некоторым способом удалось восстановить внешнюю память базы данных к состоянию на момент времени *tpc* (как это можно сделать — немного позже). Тогда:

* Для транзакции T1 никаких действий производить не требуется. Она закончилась до момента *tpc*, и все ее результаты отражены во внешней памяти базы данных.
* Для транзакции T2 нужно повторно выполнить оставшуюся часть операций ( redo ). Действительно, во внешней памяти полностью отсутствуют следы операций, которые выполнялись в транзакции T2 после момента *tpc*. Следовательно, повторная прямая интерпретация операций T2 корректна и приведет к логически согласованному состоянию базы данных (поскольку транзакция T2 успешно завершилась до момента мягкого сбоя, в журнале содержатся записи обо всех изменениях, произведенных этой транзакцией).
* Для транзакции T3 нужно выполнить в обратном направлении первую часть операций ( undo ). Действительно, во внешней памяти базы данных полностью *отсутствуют результаты* операций T3, которые были выполнены после момента *tpc*. С другой стороны, во внешней памяти гарантированно присутствуют результаты операций T3, которые были выполнены до момента *tpc*. Следовательно, обратная интерпретация операций T3 корректна и приведет к согласованному состоянию базы данных (поскольку транзакция T3 не завершилась к моменту мягкого сбоя, при восстановлении необходимо устранить все последствия ее выполнения).
* Для транзакции T4, которая успела начаться после момента *tpc* и закончиться до момента мягкого сбоя, нужно выполнить полную повторную прямую интерпретацию операций ( redo ).
* Наконец, для начавшейся после момента *tpc* и не успевшей завершиться к моменту мягкого сбоя транзакции T5 никаких действий предпринимать не требуется. Результаты операций этой транзакции полностью отсутствуют во внешней памяти базы данных.

#### **Восстановление после жесткого сбоя**

Понятно, что для восстановления последнего согласованного состояния базы данных после жесткого сбоя *журнала изменений* базы данных явно недостаточно. Основой восстановления в этом случае являются журнал и архивная копия базы данных.

Восстановление начинается с обратного копирования базы данных из архивной копии. Затем для всех закончившихся транзакций выполняется redo, то есть операции повторно выполняются в прямом порядке.

Более точно, происходит следующее:

* по журналу в прямом направлении выполняются все операции;
* для транзакций, которые не закончились к моменту сбоя, выполняется откат.

На самом деле, поскольку жесткий сбой не сопровождается утратой буферов оперативной памяти, можно восстановить базу данных до такого уровня, чтобы можно было продолжить даже выполнение незакончившихся транзакций. Но обычно это не делается, потому что восстановление после жесткого сбоя — это достаточно длительный процесс.

Хотя к ведению журнала предъявляются особые требования по части надежности, в принципе возможна и его утрата. Тогда единственным способом восстановления базы данных является возврат к архивной копии. Конечно, в этом случае не удастся получить последнее согласованное состояние базы данных, но это лучше, чем ничего.

Последний вопрос, который мы коротко рассмотрим, относится к производству архивных копий базы данных. Самый простой способ — архивировать базу данных при переполнении журнала. В журнале вводится так называемая "желтая зона", при достижении которой образование новых транзакций временно блокируется. Когда все транзакции закончатся и, следовательно, база данных придет в согласованное состояние, можно производить ее архивацию, после чего начинать заполнять журнал заново.

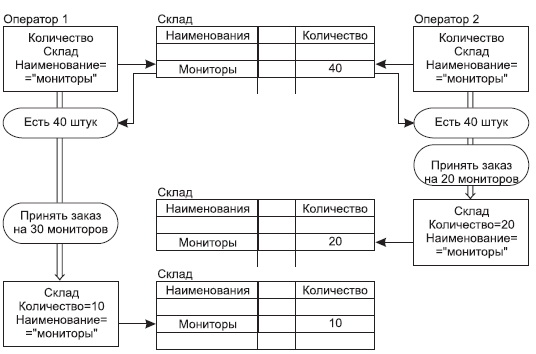
Можно выполнять архивацию базы данных реже, чем переполняется журнал. При переполнении журнала и окончании всех начатых транзакций можно архивировать сам журнал. Поскольку такой архивированный журнал, по сути дела, требуется только для воссоздания архивной копии базы данных, журнальная информация при архивации может быть существенно сжата.

### **Параллельное выполнение транзакций**

Если с *БД* работают одновременно несколько пользователей, то обработка транзакций должна рассматриваться с новой точки зрения. В этом случае *СУБД* должна не только корректно выполнять индивидуальные транзакции и восстанавливать согласованное состояние *БД* после сбоев, но она призвана обеспечить корректную параллельную работу всех пользователей над одними и теми же данными. *По* теории каждый *пользователь* и каждая *транзакция* должны обладать свойством изолированности, то есть они должны выполняться так, как если бы только один *пользователь* работал с *БД*. И средства современных *СУБД* позволяют изолировать пользователей друг от друга именно таким образом. Однако в этом случае возникают проблемы замедления работы пользователей. Рассмотрим более подробно проблемы, которые возникают при параллельной обработке транзакций.

Основные проблемы, которые возникают при параллельном выполнении транзакций, делятся условно на 4 типа:

* Пропавшие изменения. Эта ситуация может возникать, если две транзакции одновременно изменяют одну и ту же запись в БД. Например, работают два оператора на приеме заказов, первый оператор принял заказ на 30 мониторов. Когда он запрашивал склад, то там числилось 40 мониторов, и он, получив подтверждение от клиента, выставил счет и оформил продажу 30 мониторов из 40. Параллельно с ним работает второй оператор, который принимает заказ на 20 таких же мониторов (ну уж очень хорошая модель и дешево) и, в свою *очередь запросив* состояние склада и получив исходно ту же цифру 40, он успешно оформляет заказ для своего клиента. Заканчивая работу с данным заказом, он выполняет команду **Обновить**( UPDATE ), которая заносит 20 как остаток любимых мониторов на складе. Но после этого, наконец, любезно попрощавшись со своим клиентом и заверив его в скорейшей доставке заказанных мониторов, заканчивает работу со своим заказом первый оператор и также выполняет команду **Обновить**и заносит 10 как остаток тех же мониторов на складе. Каждый из них доволен своей работой, но мы-то знаем, что произошло. Прежде всего, они продали 50 мониторов из наличествующих 40 штук, и далее на складе еще числится 10 подобных мониторов. БД теперь находится в несогласованном состоянии, а у фирмы возникли серьезные проблемы. Изменения, сделанные вторым оператором, были проигнорированы программой выполнения заказа, с которой работал первый оператор. Подобная ситуация представлена на [рис. 11.5](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=4#image.11.5).
* Проблемы промежуточных данных. Рассмотрим ту же проблему одновременной работы двух операторов. Допустим, первый оператор, ведя переговоры со своим заказчиком, ввел заказанные 30 мониторов, но перед окончательным оформлением заказа клиент захотел выяснить еще некоторые характеристики товара. Приложение, с которым работает первый оператор, уже изменило остаток мониторов на складе, и там сейчас находится информация о 10 оставшихся мониторах. В это время второй оператор пытается принять заказ от своего клиента на 20 мониторов, но его приложение показывает, что на складе осталось всего 10 мониторов, и оператор вынужден отказать выгодному клиенту, который идет в другую фирму, весьма неудовлетворенный работой нашей компании. А в этот момент клиент оператора 1 заканчивает обсуждение дополнительных характеристик наших мониторов и принимает весьма невыгодное решение не покупать у нас мониторы, и приложение оператора 1 выполняет откат транзакции, и на складе снова оказывается 40 мониторов. Мы потеряли выгодного заказчика, но еще хуже было бы, если бы клиент второго оператора согласился на 10 оставшихся мониторов, и приложение, с которым работает оператор два, отработав свой алгоритм, занесло 0 (ноль) оставшихся мониторов на складе, а после этого приложение оператора один снова бы записало исходные 40 мониторов на складе, хотя 10 из них уже проданы. Такая ситуация оказалась возможной потому, что приложение второго оператора имело доступ к промежуточным данным, которые сформировало первое приложение.



**Рис. 11.5.**Проблема пропавших обновлений

* Проблемы несогласованных данных. Рассмотрим ту же самую ситуацию с заказом мониторов. Предположим, что ситуация несколько изменилась. И оба оператора начинают работать практически одновременно. Они оба получают начальное состояние склада 40 мониторов, а далее первый оператор успешно завершает переговоры со своим клиентом и продает ему 30 мониторов. Он завершает работу своего приложения, и оно выполняет команду *фиксации транзакции* COMMIT. Состояние базы данных непротиворечивое. В этот момент, выяснив все тонкости и характеристики наших мониторов, клиент второго оператора также решает сделать заказ, и второй оператор, повторно получая состояние склада, видит, что оно изменилось. База данных находится в непротиворечивом состоянии, но второй оператор считает, что нарушена целостность его транзакции, в течение выполнения одной работы он получил два различных состояния склада. Эта ситуация возникла потому, что приложение первого оператора смогло изменить кортеж с данными, который уже прочитало приложение второго оператора.
* Проблемы строк-призраков (строк-фантомов). Предположим, что администратор нашей фирмы поручил секретарю напечатать итоговый отчет по результатам работы за текущий месяц. И допустим, что приложение печатает отчет в двух видах: в подробном и в укрупненном. В момент, когда приложение печати начало формировать свой первый вид отчета, один из операторов принимает еще один заказ, поэтому к моменту формирования укрупненного отчета в БД появились новые сведения о продажах, которые и были внесены в укрупненный отчет. Мы получили два отчета в одном приложении, которые содержат разные цифры и не совпадают друг с другом. Такое стало возможно потому, что приложение печати выполнило два одинаковых запроса и получило два разных результата. БД находится в согласованном состоянии, но приложение печати работает некорректно.

Для того чтобы избежать подобных проблем, требуется выработать некоторую процедуру согласованного выполнения параллельных транзакций. Эта процедура должна удовлетворять следующим правилам:

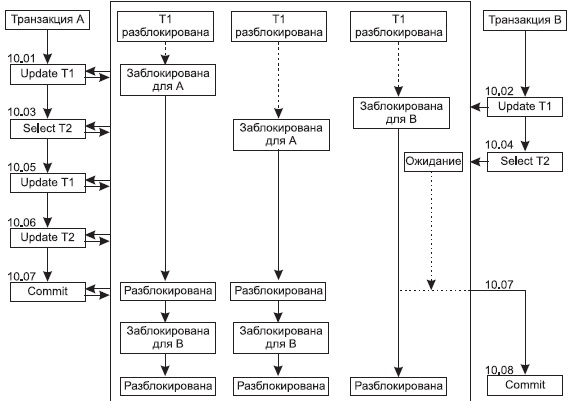
* В ходе выполнения транзакции пользователь видит только согласованные данные. Пользователь не должен видеть несогласованных промежуточных данных.
* Когда в БД две транзакции выполняются параллельно, то СУБД гарантированно поддерживает принцип независимого выполнения транзакций, который гласит, что результаты выполнения транзакций будут такими же, как если бы вначале выполнялась транзакция 1, а потом транзакция 2, или наоборот, сначала транзакция 2, а потом транзакция 1.

Такая процедура называется сериализацией транзакций. Фактически она гарантирует, что каждый *пользователь* (*программа*), обращающаяся к базе данных, работает с ней так, как будто не существует других пользователей (программ), одновременно с ним обращающихся к тем же данным.

Для поддержки параллельной работы транзакций строится специальный план.

План (способ) выполнения набора транзакций называется сериальным, если результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций.

Самым простым было бы последовательное выполнение транзакций, но такой план не оптимален *по* времени, существуют более гибкие методы управления параллельным доступом к *БД*. Наиболее распространенным механизмом, который используется коммерческими *СУБД* для реализации на практике сериализации транзакций является механизм блокировок. Самый простой вариант — это *блокировка* объекта на все время действия транзакции. Подобный пример рассмотрен на [рис. 11.6](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=4#image.11.6). Здесь две транзакции, названные условно А и В, работают с тремя таблицами: Т1, Т2 и Т3. В момент начала работы с любым объектом этот *объект* блокируется транзакцией, которая с ним начала работу, и он становится недоступным всем другим транзакциям до окончания транзакции, заблокировавшей ("захватившей") данный *объект*. После окончания транзакции все заблокированные ею объекты разблокируются и становятся доступными другим транзакциям. Если *транзакция* обращается к заблокированному объекту, то она остается в состоянии ожидания до момента разблокировки этого объекта, после чего она может продолжать обработку данного объекта. Поэтому *транзакция* В ожидает разблокировки таблицы Т2 транзакцией А. Над прямоугольниками стоит условное *время выполнения* операций.



**Рис. 11.6.**Блокировки при одновременном выполнении двух транзакций

В общем случае на момент выполнения *транзакция* получает как бы монопольный *доступ* к объектам *БД*, с которыми она работает. В этом случае другие транзакции не получают доступа к объектам *БД* до момента окончания транзакции. Такой механизм действительно ликвидирует все перечисленные ранее проблемы: пропавшие изменения, неподтвержденные данные, несогласованные данные, строки-фантомы. Однако такая *блокировка* создает новые проблемы — задержку выполнения транзакций из-за блокировок.

Рассмотрим существующие типы конфликтов между двумя параллельными транзакциями. Можно выделить следующие типы:

* W-W — транзакция 2 пытается изменять объект, измененный незакончив-шейся транзакцией 1;
* R-W — транзакция 2 пытается изменять объект, прочитанный незакончив-шейся транзакцией 1;
* W-R — транзакция 2 пытается читать объект, измененный незакончившейся транзакцией 1.

Практические методы сериализации транзакций основываются на учете этих конфликтов.

Блокировки, называемые также синхронизационными захватами объектов, могут быть применены к разному типу объектов. Наибольшим объектом блокировки может быть вся *БД*, однако этот вид блокировки сделает *БД* недоступной для всех приложений, которые работают с данной *БД*. Следующий *тип объекта* блокировки — это таблицы. *Транзакция*, которая работает с таблицей, блокирует ее на все *время выполнения* транзакции. Именно такой вид блокировки рассмотрен в примере 11.7. Этот вид блокировки предпочтительнее предыдущего, потому что позволяет параллельно выполнять транзакции, которые работают с другими таблицами.

В ряде *СУБД* реализована *блокировка* на уровне страниц. В этом случае *СУБД* блокирует только отдельные страницы на диске, когда *транзакция* обращается к ним. Этот вид блокировки еще более мягок и позволяет разным транзакциям работать даже с одной и той же таблицей, если они обращаются к разным страницам данных.

В некоторых *СУБД* возможна *блокировка на уровне строк*, однако такой механизм блокировки требует дополнительных затрат на поддержку этого вида блокировки.

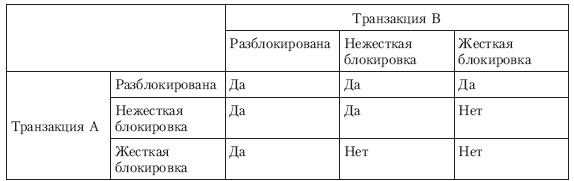
В настоящее время проблема блокировок является предметом большого числа исследований.

Для повышения параллельности выполнения транзакций используется комбинирование разных типов синхронизационных захватов.

Рассматривают два типа блокировок (синхронизационных захватов):

* совместный режим блокировки — нежесткая, или *разделяемая, блокировка*, обозначаемая как **S**(Shared). Этот режим обозначает разделяемый захват объекта и требуется для выполнения операции чтения объекта. Объекты, заблокированные таким образом, не изменяются в ходе выполнения транзакции и доступны другим транзакциям также, но только в режиме чтения;
* монопольный режим блокировки — жесткая, или эксклюзивная, блокировка, обозначаемая как **X**(eXclusive). Данный режим блокировки предполагает монопольный захват объекта и требуется для выполнения операций занесения, удаления и модификации. Объекты, заблокированные данным типом блокировки, фактически остаются в монопольном режиме обработки и недоступны для других транзакций до момента окончания работы данной транзакции.

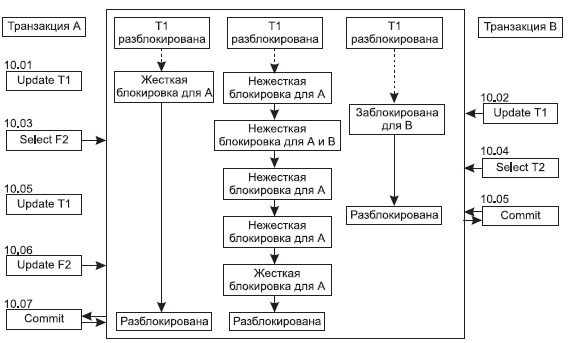
Захваты объектов несколькими транзакциями *по* чтению совместимы, то есть нескольким транзакциям допускается читать один и тот же *объект*, *захват* объекта одной транзакцией *по* чтению не совместим с захватом другой транзакцией того же объекта *по* записи, и захваты одного объекта разными транзакциями *по* записи не совместимы. Правила совместимости захватов одного объекта разными транзакциями изображены на [рис. 11.7](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=5#image.11.7):



**Рис. 11.7.**Правила применения жесткой и нежесткой блокировок транзакций

В примере, представленном на [рис. 11.7](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=5#image.11.7) считается, что первой блокирует *объект* *транзакция* А, а потом пытается получить к нему *доступ* *транзакция* В.

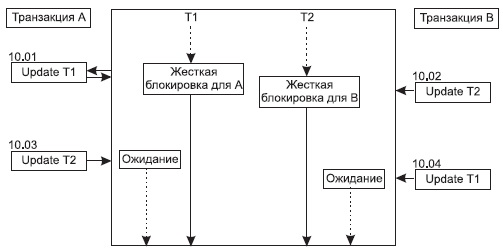
На [рис. 11.8](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=5#image.11.8) приведен ранее рассмотренный пример с выполнением транзакций 1 и 2, но с учетом разных типов блокировки. На рисунке видно, что, применив нежесткую блокировку к таблице 2 со стороны транзакции 1, мы обеспечили существенное уменьшение времени выполнения транзакции 2. Теперь *транзакция* 2 не ждет окончания транзакции 1, и поэтому завершает свою работу намного раньше.



**Рис. 11.8.**Использование жесткой и нежесткой блокировки

К сожалению, применения разных типов блокировок приводит к проблеме тупиков. Эта проблема не нова. Проблема тупиков возникла при рассмотрении выполнения параллельных процессов в операционных средах и также была связана с управлением разделяемыми (совместно используемыми) ресурсами.

Действительно, рассмотрим пример. Пусть *транзакция* А сначала жестко блокирует таблицу 1, а потом жестко блокирует таблицу 2. *Транзакция* B, наоборот, сначала жестко блокирует таблицу 2, а потом жестко блокирует таблицу 1. Если обе эти транзакции начали работу одновременно, то после выполнения операций модификации первыми объектами каждой транзакции они обе окажутся в бесконечном ожидании: *транзакция* А будет ждать завершения работы транзакции B и разблокировки таблицы 2, а *транзакция* В также безрезультатно будет ждать окончания работы транзакции А и разблокировки таблицы 1 (см. [рис. 11.9](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=5#image.11.9)).



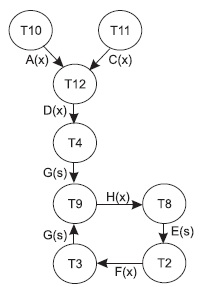
**Рис. 11.9.**Взаимная блокировка транзакций

Ситуации могут быть гораздо более сложными. Количество взаимно заблокированных транзакций может оказаться гораздо больше. Эту ситуацию каждая из транзакций обнаружить самостоятельно не может. Ее должна разрешить *СУБД*. И действительно, в большинстве коммерческих *СУБД* существует механизм обнаружения таких *тупиковых ситуаций*.

Основой обнаружения *тупиковых ситуаций* является построение (или постоянное поддержание) графа ожидания транзакций. *Граф* ожидания транзакций может строиться двумя способами. *Граф* ожидания — это *направленный граф*, в вершинах которого расположены имена транзакций. Если *транзакция* А ждет окончания транзакции В, то из вершины А в вершину В идет стрелка. Дополнительно стрелки могут быть помечены именами заблокированных объектов и типом блокировки. Пример такого графа ожиданий приведен на [рис. 11.10](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=5#image.11.10).

Этот *граф* ожиданий построен для транзакций Т1, Т2,\_\_\_,Т12, которые работают с объектами *БД* A, В, ..., Н.

Перечень действий, которые совершают транзакции над объектами, приведен в [табл. 11.1](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=5#table.11.1).



**Рис. 11.10.**Пример графа ожиданий транзакций

|  |  |  |
| --- | --- | --- |
| Таблица 11.1. Перечень действий множества транзакций | | |
| **Время** | **Транзакция** | **Действие** |
| 0 | T1 | Select A |
| 1 | T2 | Select B |
| 2 | T1 | Select C |
| 3 | T4 | Select D |
| 4 | T5 | Select A |
| 5 | T2 | Select E |
| 6 | T2 | Update E |
| 7 | T3 | Select F |
| 8 | T2 | Select F |
| 9 | T5 | Update A |
| 10 | T1 | Commit |
| 11 | T6 | Select A |
| 12 | T5 | Commit |
| 13 | T6 | Select C |
| 14 | T6 | Update C |
| 15 | T7 | Select G |
| 16 | T8 | Select H |
| 17 | T9 | Select G |
| 18 | T9 | Update G |
| 19 | T8 | Select E |
| 20 | T7 | Commit |
| 21 | T9 | Select H |
| 22 | T3 | Select G |
| 23 | T10 | Select A |
| 24 | T9 | Update H |
| 25 | T6 | Commit |
| 26 | T11 | Select C |
| 27 | T12 | Select D |
| 28 | T12 | Select C |
| 29 | T2 | Update F |
| 30 | T11 | Update C |
| 31 | T12 | Select A |
| 32 | T10 | Update A |
| 33 | T12 | Update D |
| 34 | T2 | Select G |
| 35 | - | - |

На графе объекты блокировки помечены типами блокировок, S — нежесткая (*разделяемая) блокировка*, X — жесткая (эксклюзивная) *блокировка*.

На диаграмме состояний ожидания видно, что транзакции Т9, Т8, Т2 и Т3 образуют цикл. Именно наличие *цикла* и является признаком возникновения тупиковой ситуации. Поэтому в момент 3 перечисленные транзакции будут заблокированы.

Разрушение тупика начинается с выбора в цикле транзакций так называемой транзакции-жертвы, то есть транзакции, которой решено пожертвовать, чтобы обеспечить возможность продолжения работы других транзакций.

Критерием выбора является *стоимость* транзакции; жертвой выбирается самая дешевая *транзакция*. *Стоимость* транзакции определяется на основе многофакторной оценки, в которую с разными весами входят *время выполнения*, число накопленных захватов, приоритет.

После выбора транзакции-жертвы выполняется *откат* этой транзакци. При этом, естественно, освобождаются захваты и может быть продолжено выполнение других транзакций.

Естественно, такое насильственное устранение *тупиковых ситуаций* является нарушением принципа изолированности пользователей.

Заметим, что в централизованных системах *стоимость* построения графа ожидания сравнительно невелика, но она становится слишком большой в *по*-настоящему распределенных *СУБД*, в которых транзакции могут выполняться в разных узлах сети. Поэтому в таких системах обычно используются другие методы сериализации транзакций.

Для обеспечения сериализации транзакций синхронизационные захваты объектов, произведенные *по* инициативе транзакции, можно снимать только при ее завершении. Это требование порождает двухфазный протокол синхронизационных захватов — 2PL(two *phase* lock) или *2PC* (two *phase commit*). В соответствии с этим протоколом выполнение транзакции разбивается на две фазы:

* первая фаза транзакции — накопление захватов;
* вторая фаза (фиксация или откат) — освобождение захватов.

В языке *SQL* введен оператор явной блокировки таблицы, который позволяет точно задать тип блокировки для всей таблицы. *Синтаксис* *операции* блокировки имеет вид:

LOCK TABLE имя\_таблицы IN {SHARED | EXCLUSIVE} MODE

Имеет смысл блокировать таблицу полностью, когда выполняется операция множественной модификации одной таблицы, то есть когда в ней изменяется большое количество строк. Эта операция иногда называется пакетным обновлением.

Конечно, у блокировки таблицы есть тот недостаток, что все остальные транзакции должны ждать окончания обновления таблицы. Но режим пакетного обновления одной таблицы работает достаточно быстро, и общая *производительность* выполнения *множества* транзакций может даже повыситься в этом случае.

#### **Уровни изолированности пользователей**

Достаточно легко убедиться, что при соблюдении двухфазного протокола синхронизационных захватов действительно обеспечивается полная сериализация транзакций. Однако иногда приложению, которое выполняет транзакцию, не столько важны точные данные, сколько скорость выполнения запросов. Например, в системах поддержки принятия решений по электронным торгам важно просто иметь представление об общей картине торгов, на основании которого принимается решение об повышении или снижении ставок и т. д. Для смягчения требований сериализации транзакций вводится понятие уровня изолированности пользователя.

Уровни изолированности пользователей связаны с проблемами, которые возникают при параллельном выполнении транзакций и которые были рассмотрены нами ранее.

Всего введено 4 уровня изолированности пользователей. Самый высокий уровень изолированности соответствует протоколу сериализации транзакций, это уровень *SERIALIZABLE*. Этот уровень обеспечивает полную изоляцию транзакций и *полную корректную* обработку параллельных транзакций.

Следующий уровень изолированности называется уровнем подтвержденного чтения — REPEATABLE READ. На этом уровне транзакция не имеет доступа к промежуточным или окончательным результатам других транзакций, поэтому такие проблемы, как пропавшие обновления, промежуточные или несогласованные данные, возникнуть не могут. Однако во время выполнения своей транзакции вы можете увидеть строку, добавленную в БД другой транзакцией. Поэтому один и тот же запрос, выполненный в течение одной транзакции, может дать разные результаты, то есть проблема строк-призраков остается. Однако если такая проблема критична, лучше ее разрешать алгоритмически, изменяя алгоритм обработки, исключая повторное выполнение запроса в одной транзакции.

Второй уровень изолированности связан с подтвержденным чтением, он называется *READ COMMITED*. На этом уровне изолированности транзакция не имеет доступа к промежуточным результатам других транзакций, поэтому проблемы пропавших обновлений и промежуточных данных возникнуть не могут. Однако окончательные данные, полученные в ходе выполнения других транзакций, могут быть доступны нашей транзакции. При этом уровне изолированности транзакция не может обновлять строку, уже обновленную другой транзакцией. При попытке выполнить подобное обновление транзакция будет отменена автоматически, во избежание возникновения проблемы пропавшего обновления.

И наконец, самый низкий уровень изолированности называется уровнем неподтвержденного, или *грязного, чтения*. Он обозначается как READ UNCOMMITED. При этом уровне изолированности текущая транзакция видит промежуточные и несогласованные данные, и также ей доступны строки-призраки. Однако даже-при этом уровне изолированности СУБД предотвращает пропавшие обновления.

В стандарте *SQL2* существует оператор задания уровня изолированности выполнения транзакции. Он имеет следующий синтаксис:

SET TRANSACTION IZOLATION LEVEL

[{SERIALIZABLE | REPEATABLE READ | READ COMMITED | READ UNCOMMITED}]

[{READ WRITE | READ ONLY }]

Дополнительно в этом операторе может быть указано, операции какого типа выполняются в транзакции. По умолчанию предполагается уровень *SERIALIZABLE*. Если задан уровень READ UNCOMMITED, то допустимы только операции чтения в транзакции, поэтому в этом случае нельзя установить операции READ WRITE. В [табл. 11.11](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=6#table.11.11) приведено соответствие уровней изолированности транзакций и проблем, возникающих при параллельном выполнении транзакций.

|  |
| --- |
| Таблица 11.11. Уровни изолированности транзакций и проблемы многопользовательской работы |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Уровень изоляции** | **Появление пропавших обновлений** | **Появление промежуточных данных** | **Появление несогласованных данных** | **Появление строк-призраков** | | *SERIALIZABLE* | СУБД предотвращает | СУБД предотвращает | СУБД предотвращает | СУБД предотвращает | | REPEATABLE READ | СУБД предотвращает | СУБД предотвращает | СУБД предотвращает | Может произойти | | READ COMMITED | СУБД предотвращает | СУБД предотвращает | Может произойти | Может произойти | | READ UNCOMMITED | СУБД предотвращает | Может произойти | Может произойти | Может произойти |   В разных коммерческих СУБД могут быть реализованы не все уровни изолированности, это необходимо выяснить в технической документации. **Гранулированные синхронизационные захваты** Мы уже говорили, что объектами блокирования могут быть объекты разного уровня, начиная с целой БД и заканчивая кортежем.  Понятно, что чем крупнее объект синхронизационного захвата (неважно, какой природы этот объект — логический или физический), тем меньше синхронизационных захватов будет поддерживаться в системе, и при этом, соответственно, будут меньшие накладные расходы. Более того, если выбрать в качестве уровня объектов для захватов файл или отношение, то будет решена даже проблема фантомов (если это не ясно сразу, посмотрите еще раз на формулировку проблемы фантомов и определение двухфазного протокола захватов).  Но вся беда в том, что при использовании для захватов крупных объектов возрастает вероятность конфликтов транзакций и тем самым уменьшается допускаемая степень их параллельного выполнения. Фактически при укрупнении объекта синхронизационного захвата мы умышленно огрубляем ситуацию и видим конфликты в тех ситуациях, когда на самом деле конфликтов нет. Действительно, если транзакция Т1 обрабатывает первую, пятую и двадцатую строку в таблице R1, но блокирует всю таблицу, то транзакция T2, которая обрабатывает шестую и восьмую строки той же таблицы не сможет получить к ним доступ, хотя на уровне строк никаких конфликтов нет.  В большинстве современных систем используются покортежные, то есть построковые синхронизационные захваты.  Однако нелепо было бы применять покортежную блокировку в случае выполнения, например, операции удаления всего отношения или удаления всех строк в отношении.  Подобные рассуждения привели к понятию *гранулированных* синхронизационных захватов и разработке соответствующего механизма.  При применении этого подхода синхронизационные захваты могут запрашиваться по отношению к объектам разного уровня: файлам, отношениям и кортежам. Требуемый *уровень объекта* определяется тем, какая операция выполняется (например, для выполнения операции уничтожения отношения объектом синхронизационного захвата должно быть все отношение, а для выполнения операции удаления кортежа — этот кортеж). Объект любого уровня может быть захвачен в режиме **S**(разделяемом) или **X**(монопольном). Вводится специальный протокол *гранулированных* захватов и определены новые типы захватов: перед захватом объекта в режиме **S**или **X**соответствующий объект более высокого уровня должен быть захвачен в режиме **IS, IX**или **SIX**.  **IS**(Intented for Shared lock, предваряющий разделяемую блокировку) по отношению к некоторому *составному объекту* **O**означает намерение захватить некоторый входящий в **O**объект в совместном режиме. Например, при намерении читать кортежи из отношения **R**это отношение должно быть захвачено в режиме **IS**(а до этого в таком же режиме должен быть захвачен файл).  **IX**(Intented for *eXclusive lock*, предваряющий жесткую блокировку) по отношению к некоторому *составному объекту* **O**означает намерение захватить некоторый входящий в **O**объект в монопольном режиме. Например, при намерении удалять кортежи из отношения **R**это отношение должно быть захвачено в режиме **IX**(а до этого в таком же режиме должен быть захвачен файл).  **SIX**(Shared, Intented for *eXclusive lock*, *разделяемая блокировка* объекта, предваряющая дальнейшие жесткие блокировки его составляющих) по отношению к некоторому *составному объекту* O означает совместный захват всего этого объекта с намерением впоследствии захватывать какие-либо входящие в него объекты в монопольном режиме. Например, если выполняется длинная операция просмотра отношения с возможностью удаления некоторых просматриваемых кортежей, то экономичнее всего захватить это отношение в режиме **SIX**(а до этого захватить файл в режиме **IS** ).  Весьма трудно описать словами все возможные ситуации. Приведем полную таблицу совместимости захватов, анализируя которую можно выявить все случаи (см. [табл. 11.2](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=6#table.11.2)).   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Таблица 11.2. Матрица совместимости блокировок. | | | | | | | **L1\L2** | **X** | **S** | **IX** | **IS** | **SIX** | | Нет блокировки | Да | Да | Да | Да | Да | | **X** | Нет | Нет | Нет | Нет | Нет | | **S** | Нет | Да | Нет | Да | Нет | | **IX** | Нет | Нет | Да | Да | Нет | | **IS** | Нет | Да | Да | Да | Да | | **SIX** | Нет | Нет | Нет | Да | Нет |   Протокол *гранулированных* захватов требует соблюдения следующих правил:   1. Прежде чем транзакция установит **S** -блокировку на данный кортеж, она должна установить блокировку **IS**или другую, более сильную блокировку на отношение, в котором содержится данный кортеж. 2. Прежде чем транзакция установит **Х** -блокировку на данный кортеж, она должна установить **IX** -блокировку или другую более сильную блокировку на отношение, в которое входит кортеж.   Блокировка **L1**называется более сильной по отношению к блокировке **L2**тогда и только тогда, когда для любой конфликтной ситуации (Нет — недопустимо) в столбце блокировки **L2**в некоторой строке матрицы совместимости блокировок (см. [табл. 11.2](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=6#table.11.2)) существует также конфликт в столбце блокировки **L1**в той же строке.  Диаграмма приоритетов блокировок приведена на [рис. 11.12](https://intuit.ru/studies/courses/1001/297/lecture/7419?page=6#image.11.12).  Диаграмма приоритета блокировок различных типов  **Рис. 11.12.**Диаграмма приоритета блокировок различных типов **Метод временных меток** Альтернативный метод сериализации транзакций, хорошо работающий в условиях редких конфликтов транзакций и не требующий построения графа ожидания транзакций, основан на использовании временных меток.  Основная идея метода (у которого существует множество разновидностей) состоит в следующем: если транзакция T1 началась раньше транзакции T2, то система обеспечивает такой режим выполнения, как если бы T1 была целиком выполнена до начала T2.  Для этого каждой транзакции T предписывается временная метка t, соответствующая времени начала T. При выполнении операции над объектом r транзакция T помечает его своей временной меткой и типом операции (чтение или изменение).  Перед выполнением операции над объектом r транзакция T1 выполняет следующие действия:   * Проверяет, не закончилась ли транзакция T, пометившая этот объект. Если T закончилась, T1 помечает объект r и выполняет свою операцию. * Если транзакция T не завершилась, то T1 проверяет конфликтность операций. Если операции неконфликтны, при объекте r остается или проставляется временная метка с меньшим значением, и транзакция T1 выполняет свою операцию. * Если операции T1 и T конфликтуют, то если t(T) > t(T1) (то есть транзакция T является более "молодой", чем T1 ), производится откат T и T1 продолжает работу. * Если же t(T) < t(T1) ( T "старше" T1 ), то T1 получает новую временную метку и начинается заново.   К недостаткам метода временных меток относятся потенциально более частые откаты транзакций, чем в случае использования синхронизационных захватов. Это связано с тем, что конфликтность транзакций определяется более грубо. Кроме того, в распределенных системах не очень просто вырабатывать глобальные временные метки с отношением полного порядка (это отдельная большая наука).  Но в распределенных системах эти недостатки окупаются тем, что не нужно распознавать тупики, а как мы уже отмечали, построение графа ожидания в распределенных системах стоит очень дорого.  **Лекция 7. Специфика защиты от НСД в базах данных**  При построении защиты баз данных необходимо учитывать ряд специфических угроз безопасности информации, связанных с концентрацией в базах данных большого количества разнообразной информации, а также с возможностью использования сложных запросов обработки данных. К таким угрозам относятся:  - инференция и агрегирование (Logical Inference and Aggregation);  - комбинация разрешенных запросов для получения закрытых данных (*Browsing*);  - организация скрытых каналов передачи информации (*Covert Channel*s);  - SQL инъекции (SQL Injection);  - программные закладки, отладочный код (Backdoors, Trapdoors)  - троянские кони (Trojan Horses).  **Инференция и агрегирование**  Под инференцией понимается получение конфиденциальной информации из сведений с меньшей степенью конфиденциальности путем умозаключений. Если учитывать, что в базах данных хранится информация, полученная из различных источников в разное время, отличающаяся степенью обобщенности, то аналитик может получить конфиденциальные сведения путем сравнения, дополнения и фильтрации данных, к которым он допущен. Кроме того, он обрабатывает информацию, полученную из открытых баз данных, средств массовой информации, а также использует просчеты лиц, определяющих степень важности и конфиденциальности отдельных явлений, процессов, фактов, полученных результатов. Такой способ получения конфиденциальных сведений, например, по материалам средств массовой информации, используется давно, и показал свою эффективность.  Близким к инференции является другой способ добывания конфиденциальных сведений – агрегирование. Под агрегированием понимается способ получения более важных сведений по сравнению с важностью тех отдельно взятых данных, на основе которых и получаются эти сведения. Так сведения о деятельности одного отделения или филиала корпорации обладают определенным весом. Данные же обо всей корпорации имеют куда большую значимость.  Если инференция и агрегирование являются способами добывания информации, которые применяются не только в отношении баз данных, то способ специального комбинирования запросов используется только при работе с базами данных. Использование сложных, а также последовательности простых логически связанных запросов позволяет получать данные, к которым доступ пользователю закрыт. Такая возможность имеется, прежде всего, в базах данных, позволяющих получать статистические данные. При этом отдельные записи, поля, (индивидуальные данные) являются закрытыми. В результате запроса, в котором могут использоваться итоговые операции, пользователь может получить такие величины как количество записей, сумма, максимальное или минимальное значение. Используя сложные перекрестные запросы и имеющуюся в его распоряжении дополнительную информацию об особенностях интересующей записи (поля), злоумышленник путем последовательной фильтрации записей может получить доступ к нужной записи (полю).  Рассмотрим примеры инференции и комбинирования запросов. Допустим, мы имеем базу данных, которая содержит информацию о сотрудниках некоторого университета и наша цель состоит в определении величины заработной платы определенных сотрудников. Прямой доступ к этой информации запрещен. Но мы можем сформулировать запрос на среднюю зарплату определенной категории сотрудников, к примеру, средней зарплаты женщин профессоров компьютерных наук. Ответ составил 100 000$. Затем мы можем сформулировать еще один запрос, но уже на количество женщин профессоров компьютерных наук. Ответ - один. Остается выполнить разрешенный запрос на поиск женщины профессора компьютерных наук.  К другим каналам утечки информации можно отнести анализ ошибок и сообщений об ошибках времени выполнения и анализ времени выполнения запроса. Например, запрос:  select \* from employee where 1/(salary-100000) = 0.23  позволит легко определить наличие сотрудника, получающего 100000. Генерация запроса или подзапроса, выполняющего сложные (длительные) вычисления при наличии определенного (искомого) значения некоторого поля в БД также позволит выявить наличие такой попытки.  Противодействие подобным угрозам осуществляется следующими методами:  - блокировка ответа при неправильном числе запросов;  - искажение ответа путем округления и другой преднамеренной коррекции данных;  - разделение баз данных;  - случайный выбор записи для обработки  -контекстно-ориентированная защита;  - контроль поступающих запросов.  Метод блокировки ответа при неправильном числе запросов предполагает отказ в выполнении запроса, если в нем содержится больше определенного числа совпадающих записей из предыдущих запросов. Таким образом, данный метод обеспечивает выполнение принципа минимальной взаимосвязи запросов. Этот метод сложен в реализации, так как необходимо запоминать и сравнивать все предыдущие запросы.  Метод коррекции заключается в незначительном изменении точного ответа на запрос пользователя. Для того, чтобы сохранить приемлемую точность статистической информации, применяется так называемый свопинг данных. Сущность его заключается во взаимном обмене значений полей записи, в результате чего все статистики i-го порядка, включающие i атрибутов, оказываются защищенными для всех i, меньших или равных некоторому числу. При свопинге результат выполнения статистической операции не меняется, но если злоумышленник и сможет выявить некоторые обобщенные данные, то он не сможет определить, к какой конкретно записи они относятся.  Применяется также метод разделения баз данных на группы. В каждую группу может быть включено не более определенного числа записей. Запросы разрешены к любому множеству групп, но запрещаются к подмножеству записей из одной группы. Применение этого метода ограничивает возможности выделения данных злоумышленником на уровне не ниже группы записей. Метод разделения баз данных не нашел широкого применения из-за сложности получения статистических данных, обновления и реструктуризации данных.  Эффективным методом противодействия исследованию баз данных является метод случайного выбора записей для статистической обработки. Такая организация выбора записей не позволяет злоумышленнику проследить множество запросов.  Сущность контекстно-ориентированной защиты заключается в назначении атрибутов доступа (чтение, вставка, удаление, обновление, управление и т. д.) элементам базы данных (записям, полям, группам полей) в зависимости от предыдущих запросов пользователя. Например, пусть пользователю доступны в отдельных запросах поля: «идентификационные номера» и «фамилии сотрудников», а также «идентификационные номера» и «размер заработной платы». Сопоставив ответы по этим запросам, пользователь может получить закрытую информацию о заработной плате конкретных работников. Для исключения такой возможности пользователю следует запретить доступ к полю «идентификатор сотрудника» во втором запросе, если он уже выполнил первый запрос.  Одним из наиболее эффективных методов защиты информации в базах данных является контроль поступающих запросов на наличие «подозрительных» запросов или комбинации запросов. Анализ подобных попыток позволяет выявить возможные каналы получения несанкционированного доступа к закрытым данным.  **Скрытые каналы передачи информации**  Неформально под скрытым каналом (Covert channel) передачи информации понимают любой канал связи, изначально для передачи информации не предназначенный. В более общем виде под скрытым, иначе косвенным каналом, нарушения конфиденциальности подразумевается механизм, посредством которого субъект, имеющий высокий уровень допуска, может предоставить определенные аспекты конфиденциальной информации субъектам, степень допуска которых ниже уровня конфиденциальности этой информации.  Для организации скрытого канала необходимы три вещи. Во-первых, отправитель и получатель должны иметь доступ к общему ресурсу. Во-вторых, отправитель должен иметь возможность изменять некоторые свойства общего ресурса, а получатель – иметь доступ на просмотр общего ресурса. Наконец, отправитель и получатель должны иметь возможность синхронизировать свои действия.  Скрытые каналы практически невозможно устранить, и наши усилия должны быть направлены на минимизацию пропускной способности этих каналов. Наверное, единственный способ устранения скрытых каналов заключается в полной ликвидации всех общих ресурсов и всех коммуникаций.  Выделяют следующие типы скрытых каналов:  1. Скрытые каналы по памяти, в которых информация передаётся через доступ отправителя на запись и получателя на чтение к одним и тем же ресурсам или объектам;  2. Скрытые каналы по времени, которые характеризуются доступом отправителя и получателя к одному и тому же процессу или изменяемому во времени атрибуту.  Приведём примеры скрытых каналов передачи информации. Рассмотрим систему, в которой имеются два уровня секретности: Высокий и Низкий. Передача информации с уровня Низкий на уровень Высокий разрешена, а в обратном направлении – запрещена. Цель нарушителя состоит в том, чтобы организовать скрытый канал для передачи информации от программно-аппаратного агента, функционирующего в среде Высокий, к другому программно-аппаратному агенту, функционирующему в среде Низкий. Субъект, функционирующий в среде Высокий, может совершать совершенно безобидные действия, например, менять настройки параметров безопасности элементов файловой системы, доступные для наблюдения в среде Низкий. В этом случае злоумышленник может закодировать передаваемую информацию в значениях параметров безопасности тех или иных элементов файловой системы.  Пример скрытого канала по времени. В данном случае между уровнями Высокий и Низкий нет общих ресурсов, за исключением некоторой системной библиотеки или приложения, доступ к которой возможен только на чтение. Для организации скрытого канала передачи информации субъект в среде Высокий может модулировать определённым образом интервалы занятости библиотеки, а субъект в среде Низкий – сканировать время занятости библиотеки, осуществляя запросы к ней с заданной периодичностью.  Еще один пример: сотрудник Alice с уровнем допуска Top Secret хочет передать некоторое количество секретной информации сотруднику той же организации, но уже с минимальным уровнем допуска. У обоих сотрудников есть доступ на чтение некоторого каталога в общем ресурсе. Дополнительно Alice может создавать и удалять файлы в этом каталоге. Тогда для передачи информации Alice может создавать и удалять файл с определенным именем в этом каталоге с заранее условленной периодичностью. Если пользователь Bob, просматривая каталог, видит в нем файл, то передана 1, если нет – 0. Конечно, для передачи, например, файла размером 100Мб с периодичностью 1 сек (1 бит в секунду) Alice потребуется 25 лет, но для передачи 256-bit AES key уже требуется меньше 5 минут.  Подходы к решению задачи выявления скрытых каналов передачи информации в настоящее время активно изучаются и совершенствуются. На сегодняшний день наиболее распространены следующие методы:  1. Метод разделяемых ресурсов Кемерера, который состоит в следующем: для каждого разделяемого ресурса в системе строится матрица, строки которой соответствуют всевозможным атрибутам разделяемого ресурса, а столбцы – операциям, выполняемым в системе. Значения в ячейках матрицы соответствуют воздействиям, осуществляемым при выполнении тех или иных операций в отношении атрибутов разделяемых ресурсов. Получившаяся в результате матрица позволяет отследить информационные потоки, существующие в системе.  2. Сигнатурный анализ исходных текстов программного обеспечения. Данный метод предполагает проведение анализа исходных текстов программ с целью выявления конструкций, характерных для скрытых каналов передачи информации. Необходимость проведения анализа автоматизированных систем в ходе проведения сертификационных испытаний регламентируется соответствующими оценочными стандартами и обычно является необходимым для высоко доверенных систем.  **SQL инъекции**  Инъекцией SQL (SQL Injection) называют технологию взлома, которая добавляет в параметр динамического SQL-запроса не требуемое значение, а некоторый код SQL. Эта технология особо опасна тем, что любой, кто имеет доступ к БД или Web-сайту организации, обращающейся к БД, и способен вводить данные в текстовые поля, потенциально может стать источником атак с помощью инъекций SQL. SQL-инъекция может возникнуть в следующих случаях:  - передача вредоносного кода в параметры динамического запроса (прикрепляемый код и измененные предложения WHERE);  - неправильная обработка типов;  - поиск уязвимости в СУБД;  - условные ошибки.  Существует множество вредоносных приемов, использующих прикрепляемый код и измененные предложения WHERE. Рассмотрим следующие из них:  **Прикрепление вредоносного кода.** Добавление терминатора инструкции, другой инструкции SQL и содержимого ввода позволяет взломщику передать программный код в строку выполнения. Например, приложение считывает параметр customer из поля ввода, размещенного на форме, и динамически формирует строку SQL вида  query = " SELECT \* FROM Customers WHERE CustomerName ='" + customer +"'"  Если пользователь взамен ввода имени клиента вводит следующую строку:  Сидоров'; Delete FROM Orders--  то в динамическую строку SQL будет добавлена инструкция DELETE, которая и будет выполнена в пакете с основной:  SELECT \* FROM Customers WHERE CustomerID ='Сидоров'; Delete FROM Orders--'  Терминатор инструкции (;) завершает заложенную программистом операцию, после чего СУБД рассматривает продолжающийся текст как следующую инструкцию в пакете. Завершающая кавычка могла бы привести к ошибке выполнения, однако эта проблема просто решается путем добавления маркера комментария (-- для MS SQL Server). В результате получим пустую таблицу заказов (Orders).  Среди других популярных прикрепляемых кодов можно отметить запуск команды xp\_commandshell (в MS SQL Server) и установку пароля для пользователя sa.  **Прикрепление Or 1=1.** Еще одним методом "инъекций" SQL является модификация предложения WHERE для выборки большего количества строк, чем было изначально предусмотрено. Если пользователь вводит в текстовое поле строку  123' Or 1=1 --  то условие 1=1 (которое всегда истинно) внедряется в предложение WHERE:  SELECT \* FROM Customers WHERE CustomerlD='123' or 1=1 --'  Такая инструкция отбирает все строки таблицы, а не одну строку, удовлетворяющую условию CustomerID='123'. Или введем в поле ввода пароля следующее  secreta net' Or 1=1 -- Получим:  SELECT \* FROM users WHERE username = 'james' AND password = 'secret' OR 1=1--'  Выражение 1=1 всегда истинно, таким образом, пароль уже не будет иметь значения.  **Комментирование кода.** Еще одной распространенной "инъекцией" кода SQL является комментирование остальной части кода, подлежащего выполнению. Если пользователь вводит в Web-форму регистрации на сайте:  UserName: Joe' --  Password : qwerty  то результирующую инструкцию SQL можно будет прочитать следующим образом:  SELECT UserID FROM Users WHERE UserName = 'Joe'--' AND Password = 'qwerty'  Включение маркера комментария в поле имени пользователя приводит к тому, что дальнейшая часть предложения WHERE, включая условие для пароля, игнорируется.  **Генерация ошибок.** Проверка уязвимости для SQL инъекций путем ввода '\*\*\* в поле ввода. Одиночные кавычки вставляют также в строку URL запроса. Возникающая необработанная ошибка может предоставить много информации для нарушителя. Если взломщик получит ошибку вида  Microsoft OLE DB Provider for ODBC Drivers error '80040e14'  [Microsoft][ODBC SQL Server Driver][SQL Server]Incorrect syntax near the  keyword 'or'.  /wasc.asp, line 69  то вероятность осуществления вторжения путем SQL инъекций будет довольно высокой.  Можно также ввести кавычку при запросе определенного файла (например музыкального файла). В ответ получим что-то подобное следующему: Warning: file(./data/music/Sunny.mp3): failed to open stream: No such file or directory in C:\Website\index.php on line 27. Полученное сообщение об ошибке позволяет легко узнать, где хранятся файлы.  Определенным образом сформулированные запросы могут дать доступ к структуре данных (именам полей и таблиц). Например, для поиска уязвимости на сайте, написанном на языке PHP и использующим СУБД MySQL, можно попробовать следующее:  http://.../php?id = 12 + union + select + null,null,null + from + users/\*  (здесь /\* это символ комментария в MySQL). Правильные выполненные запросы будут соответствовать существующим именам таблиц. Следует проверить на существование таблиц users, passwords, regusers и т.д. Затем выполняем:  http://.../php?id=9999+union+select+'test',null,null/\*  http://.../php?id=9999+union+select+null,'test',null/\*  Ввод подобных команд необходимо продолжать до тех пор, пока название поля (слово test) окажется в нужном месте. «Полезными» являются также следующие команды:  http://.../php?id=9999+union+select+null,DATABASE(),null/\*  http://.../php?id=9999+union+select+null,USER(),null/\*  http://.../php?id=9999+union+select+null,VERSION(),null/\*  Можно также попробовать и уязвимость, позволяющую загружать файлы с сервера:  http://.../php?id=9999+union+select+null,LOAD\_FILE('/etc/passwd'),null/\*  Если нет возможности применения union в запросе (например, MySQL имеет версию 3.\*), то инъекцию можно использовать для того, чтобы заставить сервер базы данных исчерпать все свои ресурсы. Для этого можно использовать функцию BENCHMARK, которая повторяет выполнение выражения expr заданное количество раз, указанное в аргументе count. Составленный запрос будет выглядеть следующим образом:  http://.../php?id=BENCHMARK(10000000,BENCHMARK(10000000,md5(current\_date)))  1000000 запросов md5 выполняются (в зависимости от мощности сервера), примерно 5 секунд. Вложенный benchmark будет выполняться очень долго на любом сервере.  **Защита от SQL инъекций.** Предотвратить проникновение "инъекций" кода можно несколькими путями.  1. Взамен динамического формирования инструкций использовать подготовленные инструкции, например:  PreparedStatement pstmt= conn.prepareStatement("select balance from account where account\_number =?“);  pstmt.setString(1,accnt\_number);  pstmt.execute();  2. Производить проверку типа вводимых данных и ограничивать число вводимых символов.  3. Использовать сохраненные процедуры и функции.  4. Применять функции, автоматически устраняющие потенциально опасные символы или последовательности символов, содержащие терминаторы инструкций, комментарии, одиночные кавычки и символы хр\_ (для SQL Server) из вводимых данных. Например: функция блокировки mysql\_real\_escape\_string() в MySQL блокирует ввод одиночных кавычек, терминаторов инструкций и прочее.  5. Тщательно определять права доступа, чтобы инструкции не имели разрешений на запуск DDL инструкций.  6. Отключать выдачу необработанных ошибок.  Проблеме инъекций кода SQL следует уделять повышенное внимание. Если ваше приложение предполагает ввод данных из Интернета, и вы не предусмотрите действия, направленные против потенциальных инъекций, разрушение базы данных станет только вопросом времени. |

**Лекция 8. Управление доступом к данным**

В любой организации действуют определенные правила накопления и использования сведений, ограничивающие доступ к информационным ресурсам. Конкретные правила определяются информационной политикой руководства предприятия, однако в любом случае разумные ограничения доступа основаны на следующих принципах:

- предприятие является собственником всей служебной информации, полученной его подразделениями или служащими;

- подразделение или служащий является владельцем полученной им информации и может использовать её в интересах предприятия без ограничений;

- служащий имеет право доступа к тем и только тем сведениям, которые необходимы для исполнения его служебных обязанностей;

- служащий имеет право выполнять те, и только те манипуляции доступными сведениями, которые обусловлены его служебными обязанностями.

Эти принципы реализуются в виде системы правил, ограничивающих права доступа служащих к информационным ресурсам предприятия. Современные СУБД имеют хорошо развитые средства поддержки подобных правил - подсистемы администрирования данных. Целью подсистемы администрирования является обеспечение санкционированного доступа служащих предприятия к хранимым данным. С концептуальной точки зрения она должна обеспечивать наделение пользователей СУБД привилегиями по отношению к данным и контролировать предоставление конкретному пользователю только определённых для него привилегий.

За предоставление пользователям доступа к компьютерной системе обычно отвечает системный администратор, в обязанности которого входит создание учетных записей пользователей. Каждому пользователю присваивается уникальный идентификатор, который используется операционной системой для того, чтобы определить, кто есть кто. С каждым идентификатором связывается пароль, выбираемый пользователем и известный операционной системе. При регистрации пользователь должен предоставлять системе свой пароль для аутентификации. Подобная процедура позволяет организовать контролируемый доступ к компьютерной системе, но не обязательно предоставляет право доступа к СУБД или иной прикладной программе. Для получения пользователем права доступа к СУБД может использоваться отдельная подобная процедура. Ответственность за предоставление прав доступа к СУБД обычно несет администратор базы данных, в обязанности которого входит создание индивидуальных идентификаторов пользователей, на этот раз уже в среде самой СУБД. Каждый из идентификаторов пользователей СУБД также связывается с паролем, который должен быть известен только данному пользователю.

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

Исходя из вышесказанного, сформулируем термин «управление доступом» более детально. ***Управление доступом*** есть метод защиты информации путем регулирования использования ресурсов системы (элементов БД, программных и технических средств). Включает следующие функции защиты:

* + идентификация пользователей и ресурсов системы;
  + установление подлинности объекта или субъекта по предъявленному им идентификатору (аутентификация);
  + разграничение и проверка полномочий (авторизация). Создание условий работы в пределах установленного регламента;
  + регистрация обращений к защищаемым ресурсам (протоколирование и аудит);
  + реагирование при попытках несанкционированного доступа.

**1 Идентификация и аутентификация пользователей**

Для того чтобы получить доступ к БД, пользователь должен указать свой идентификатор (ID) и подтвердить право на его использование. Опознав пользователя, система готова выполнять операции над данными от его имени. В зависимости от требуемой степени защищённости системы могут использоваться различные процедуры установления личности пользователя. Приведем простейшие и наиболее часто используемые программные процедуры аутентификации, не связанные с анализом «почерка» пользователя (например, по скорости нажатия клавиш клавиатуры и длительности пауз между отдельными нажатиями):

***Парольная защита***.В простейшем варианте сIDсвязывается пароль, известный только владельцу ID и системе набор символов (секрет). Минимальный набор полномочий владельца ID в этом варианте включает права входа в систему и изменения своего пароля. Ни один пользователь (включая администратора системы) не имеет права просмотра паролей. Пароли хранятся в виде хэш-кода или в зашифрованном виде.

Реализуя парольную защиту, администратор должен определить алфавит паролей и их длину, а также установить максимальное число попыток ввода пароля и/или время реакции пользователя на запрос пароля. Следует иметь в виду, что чем богаче алфавит и длиннее пароль, тем труднее взломать защиту путём его подбора. Однако, с другой стороны, длинный и изощрённый пароль труднее ввести без ошибок. Для повышения надёжности защиты может быть установлен срок действия пароля или максимальное число подключений с этим паролем. Дополнительно вводят задержку при вводе неправильного пароля, отбраковку паролей по словарю и журналу истории паролей. В ряде случаев вводят запрет на выбор пароля пользователем и производят автоматическую генерацию пароля.

***Защита "вопрос-ответ"***.Более сложный вариант защиты входаможет быть таким. Владелец ID вводит при регистрации серию вопросов вместе с ответами. Вопросы имеют личный характер, поэтому правильные ответы на них невозможно угадать. Например: "На какую школьную кличку отзывается Ваш племянник?", "Где родилась Ваша бабушка?" и т.п. Для установления личности пользователя при попытке входа система предъявляет ему случайно выбранный вопрос(ы) из этой серии.

***Предопределённый алгоритм***.Если потенциальный злоумышленникможет подключиться к коммуникационной линии, связывающей компьютер-клиент с сервером, то для защиты входа можно использовать предопределенный алгоритм, известный владельцу ID и системе. При попытке входа система предъявляет пользователю случайное число. Пользователь должен выполнить нужные преобразования и ввести результат. Посторонний наблюдатель на линии может увидеть только исходное и конечное числа. Такого вида защита получила широкое распространение в Интернет системах доступа к банковскому счету. Владельцу счета выдается специальное портативное устройство, оснащенное считывателем информации из чипа банковской смарт-карты. При установлении соединения владелец должен вначале вставить карту в считыватель, ввести пин-код, а уже затем ввести случайным образом сгенерированное на странице доступа число. Ответ данного устройства посылается обратно на сервер.

При соединении по сети аутентификацию должны пройти оба соединяющихся объекта. После установления соединения необходимо выполнить требование защиты при обмене сообщениями:

- получатель должен быть уверен в подлинности источника данных;

- получатель должен быть уверен в подлинности передаваемых данных;

- отправитель должен быть уверен в доставке данных получателю;

- отправитель должен быть уверен в подлинности доставленных данных.

Выполнить это требование защиты можно с помощью так называемой цифровой подписи. Если все эти четыре требования реализованы в системе, то обеспечивается функция подтверждения (неоспоримости передачи). Отправитель не может отрицать ни факта посылки сообщения, ни его содержания, а получатель не может отрицать ни факта получения сообщения, ни подлинности его содержания.

Информация о зарегистрированных пользователях БД хранится в ее системном каталоге. Современные СУБД не имеют общего синтаксиса SQL-предложения соединения с базой данных, так как их собственный синтаксис сложился раньше, чем стандарт ISO. Наиболее часто употребляется предложение CONNECT. Например, для IBM DB2:

CONNECT TO <БД> USER <пользователь> USING <пароль>

Соединение с системой не идентифицированных пользователей и пользователей, у которых проверка подлинности предъявленного идентификатора при аутентификации не подтвердилась, исключается. В процессе сеанса работы пользователя (от удачного прохождения идентификации и аутентификации до отсоединения от системы) все его действия непосредственно связываются с результатом идентификации. Отсоединение пользователя может быть как нормальным, так и насильственным (исходящим от пользователя-администратора, например в случае удаления пользователя или при аварийном обрыве канала связи клиента и сервера). Во втором случае пользователь должен быть проинформирован об этом, и все его действия аннулируются до последней фиксации изменений, произведенных им в таблицах базы данных. В любом случае на время сеанса работы идентифицированный пользователь будет субъектом доступа для средств защиты информации от несанкционированного доступа СУБД.

**2 Авторизация пользователей**

Авторизация пользователей (субъектов доступа) заключается в предоставлении определенных прав (или привилегий), позволяющих их владельцу иметь законный доступ как к самой системе, так и к ее отдельным объектам. Все субъекты доступа могут быть разделены для системы на ряд категорий, например: CONNECT, RESOURCE и DBA. Набор таких категорий определяется производителем СУБД. Нарастание возможностей (полномочий) для каждого отдельного вида подключения происходит в указанном порядке:

CONNECT - конечные пользователи. По умолчанию им разрешено только соединение с базой данных и выполнение запросов к данным, все их действия регламентированы выданными им привилегиями;

RESOURCE - привилегированные пользователи, обладающие правом создания собственных объектов в базе данных (таблиц, представлений, хранимых процедур и триггеров).

DBA - категория администраторов базы данных. Включает возможности обеих предыдущих категорий, а также возможность регистрировать субъекты защиты или изменять их категорию.

В некоторых СУБД административные действия также разделены, что обусловливает наличие дополнительных категорий. Так, в Oracle пользователь с именем DBA является администратором сервера баз данных, а не одной базы данных. В IBM DB2 существует ряд категорий администраторов: SYSADM (наивысший уровень; системный администратор, обладающий всеми привилегиями); DBADM (администратор базы данных, обладающий всем набором привилегий в рамках конкретной базы данных). Привилегии управления сервером баз данных имеются у пользователей с именами SYSCTRL (наивысший уровень полномочий управления системой, который применяется только к операциям, влияющим на системные ресурсы; непосредственный доступ к данным запрещен, разрешены операции создания, модификации, удаления базы данных, создание и удаление табличных пространств). Для каждой административной операции в IBM DB2 определен необходимый набор административных категорий, к которым должен принадлежать пользователь, выполняющий тот или иной запрос администрирования. Так, выполнять операции назначения привилегий пользователям может SYSADM или DBADM, а для того чтобы создать объект данных, пользователь должен обладать привилегией CREATETAB.

Администратор каждой БД занимается созданием списка возможных пользователей БД и разграничением полномочий этих пользователей.

Данные о разграничениях располагаются в системном каталоге БД. Очевидно, что данная информация может быть использована для несанкционированного доступа и поэтому также подлежит защите. Защита этих данных осуществляется средствами самой СУБД.

В настоящее время наиболее часто используется три основных подхода к разграничению доступа: избирательный, обязательный и ролевой. Избирательный подход описывается дискреционной моделью разграничения доступа, Discretionary Access Control (DAC), обязательный или полномочный – мандатной моделью разграничения доступа, Mandatory Access Control (MAC). Оба обеспечивают создание системы безопасности как для БД в целом, так и для отдельных её объектов - таблиц, представлений, кортежей и т.д. вплоть до конкретного значения некоторого атрибута в определенном кортеже определенного отношения. Ролевая модель разграничения доступа, Role Based Access Control (RBAC) является в некотором смысле комбинацией вышеуказанных моделей. Следует упомянуть также и модель Китайской стены, заключающаяся в возведении физического барьера между БД и группой авторизованных лиц, и остальным миром.

**3 Дискреционное разграничение доступа**

Избирательный подход, описываемый дискреционной моделью разграничения доступа, является наиболее простым одноуровневым подходом к обеспечению безопасности. Основными его понятиями являются:

- субъект - системный идентификатор, от имени которого СУБД выполняет определенные действия над определенными объектами. Понятие субъекта отличается от понятия пользователь компьютерной системы, поскольку инициировать изменение информации могут также и системные процессы;

- объект защиты - часть БД, на которую распространяется действие конкретного правила безопасности; это может быть группа отношений, отдельное отношение, подмножества атрибутов и т.д.;

- привилегия - действие над объектом защиты, которое может быть совершено от имени конкретного идентификатора.

Информация о привилегиях сохраняется в системном каталоге. Она используется системой для принятия решения о выполнении запрошенных субъектом операций над данными. При этом действует принцип: запрещено всё, что не разрешено явно. Выделяют два типа привилегий:

1. Системные привилегии - права на создание и модификацию объектов БД (пользователей, именованных отношений, правил и т.п.);
2. Объектные привилегии - права на использование объектов в операциях манипулирования данными.

Дискреционная модель разграничения доступа основывается на следующих положениях:

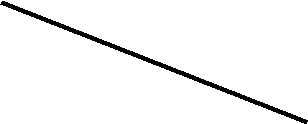
1. Все субъекты и объекты должны быть однозначно идентифицированы;
2. Для любого объекта должен быть определен пользователь-владелец;
3. Владелец объекта обладает правом определения прав доступа к объекту со стороны любых субъектов;
4. В системе существует привилегированный пользователь, обладающий правом полного доступа к любому объекту (или правом становиться владельцем любого объекта).

Последнее свойство определяет невозможность существования в системе потенциально недоступных объектов, владелец которых отсутствует. Но реализация этого положения не означает, что привилегированный пользователь может использовать свои полномочия незаметно для реального владельца объекта.

Чаще всего владельцем базы данных является Администратор БД. Ему предоставлены все системные и объектные привилегии. В частности, он имеет право регистрации новых пользователей и предоставления им привилегий, как системных, так и объектных. Пользователь, имеющий системные привилегии, является владельцем всех созданных им объектов, имеет по отношению к ним все привилегии и может предоставлять их полностью или частично другим пользователям. Минимальной системной привилегией является право подключения к СУБД - привилегия входа. Она должна быть предоставлена каждому служащему, который в силу своих служебных обязанностей имеет такое право.

Дискреционное разграничение доступа реализуется на основе множества разрешенных отношений доступа в виде троек – «субъект доступа – тип доступа – объект доступа». Наглядным и распространенным способом формализованного представления дискреционного доступа является матрица доступа, устанавливающая перечень пользователей (субъектов) и перечень разрешенных операций (процессов) по отношению к каждому объекту базы данных (таблицы, запросы, формы, отчеты). На рис. 1 приведен пример, иллюстрирующий матрицу доступа. Данное представление является схематичным, поскольку в реляционных БД вследствие требования первой нормальной формы для каждого типа доступа некоторого субъекта к некоторому объекту отводится отдельная запись.





В рамках дискреционной модели существует два подхода управления доступом:

 добровольное управление доступом;

 принудительное управление доступом.

При добровольном управлении доступом вводится так называемое владение объектами. Как правило, владельцами объектов являются те субъекты базы данных, процессы которых создали соответствующие объекты. Добровольное управление доступом заключается в том, что права на доступ к объектам определяют их владельцы. Иначе говоря, соответствующие ячейки матрицы доступа заполняются теми субъектами (пользователями), которые являются владельцами соответствующих объектов базы данных. Владение некоторым объектом предоставляет его владельцу весь возможный набор привилегий в отношении этого объекта. Это правило применяется ко всем авторизированным пользователям, получающим права владения определенными объектами. Любой вновь созданный объект автоматически передается во владение его создателю, который и получает весь возможный набор привилегий для данного объекта. Принадлежащие владельцу привилегии могут быть переданы им другим авторизированным пользователям. При предоставлении пользователю некоторой привилегии дополнительно можно указывать, передается ли ему право предоставлять эту привилегию другим пользователям (уже от имени этого пользователя). Естественно, что в этом случае СУБД должна контролировать всю цепочку предоставления привилегий пользователям с указанием того, кто именно ее предоставил, что позволит поддерживать корректность всего набора установленных в системе привилегий. В частности, эта информация будет необходима в случае отмены предоставленных ранее привилегий для организации каскадного распространения вносимых изменений среди цепочки пользователей. В результате при добровольном управлении доступом реализуется полностью децентрализованный принцип организации и управления процессом разграничения доступа. Такой подход обеспечивает гибкость настраивания системы разграничения доступа в базе данных на конкретную совокупность пользователей и ресурсов, но затрудняет общий контроль и аудит состояния безопасности данных в системе.

Принудительный подход к управлению доступом предусматривает введение единого централизованного администрирования доступом. В базе данных выделяется специальный доверенный субъект (администратор), который (и только он), собственно, и определяет разрешения на доступ всех остальных субъектов к объектам базы данных. Иначе говоря, заполнять и изменять ячейки матрицы доступа может только администратор системы. Принудительный способ обеспечивает более жесткое централизованное управление доступом. Вместе с тем он является менее гибким и менее точным в плане настройки системы разграничения доступа на потребности и полномочия пользователей, так как наиболее полное представление о содержимом и конфиденциальности объектов (ресурсов) имеют, соответственно, их владельцы.

На практике может применяться комбинированный способ управления доступом, когда определенная часть полномочий на доступ к объектам устанавливается администратором, а другая часть владельцами объектов.

Некоторыми объектами в среде СУБД владеет сама СУБД. Обычно это владение организуется посредством использования специального идентификатора особого суперпользователя - например, с именем system administrator (sa).

Пользователи могут быть объединены в специальные группы пользователей. Один пользователь может входить в несколько групп. Для пользователей с минимальным стандартным набором прав вводится понятие группы PUBLIC. По умолчанию предполагается, что каждый вновь создаваемый пользователь, если специально не указано иное, относится к группе PUBLIC. Если СУБД поддерживает использование идентификаторов как отдельных пользователей, так и их групп, то, как правило, идентификатор пользователя имеет более высокий приоритет, чем идентификатор группы.

Привилегии конкретному пользователю могут быть назначены администратором явно и неявно, например, через роль. Роль - это еще один возможный именованный носитель привилегий. Существует ряд стандартных ролей, которые проектируются при разработке СУБД. Также имеется возможность создавать новые роли, группируя в них произвольные полномочия. Введение ролей позволяет упростить управление привилегиями пользователей, структурировать этот процесс. Кроме того, введение ролей не связано с конкретными пользователями, поэтому роли могут быть определены и сконфигурированы до того, как определены пользователи системы. Роли удобно использовать, когда тот или иной набор привилегий необходимо предоставить или отнять сразу для группы пользователей. С одной стороны, это облегчает администратору управление привилегиями, с другой - вносит определенный порядок в случае необходимости изменить набор привилегий для группы пользователей сразу. Более подробно с ролевой моделью разграничения доступа познакомимся позже.

**Модель Харрисона-Руззо-Ульмана.** Математической формализацией дискреционной модели является разработанная в 1971 г. модель Харрисона-Руззо-Ульмана. В данной модели дополнительно к субъектам S, объектам O (причем для того, чтобы включить в область действия модели и отношения между субъектами, принято считать, что все субъекты одновременно являются и объектами, *S*  *O* ) и правами доступа R, определяющим матрицу доступа M, вводится также пространство состояний системы Q. Пространство состояний системы образуется декартовым произведением множеств составляющих ее объектов, субъектов и прав *S*  *O*  *R .* Любая ячейка матрицыMсодержит набор прав субъектаSкобъекту O, принадлежащих множеству прав доступа R. Поведение системы во времени моделируется переходами между различными её состояниями. Для заданной системы начальное состояние Q0 = {S0, O0, M0}, где M0 – текущее состояние матрицы доступа, называется безопасным относительно права *r*, если не существует применимой к Q0 последовательности команд, в результате выполнения которых право *r* будет занесено в ячейку матрицы M, в которой оно отсутствовало в состоянии Q0. Другими словами это означает, что субъект никогда не получит право доступа *r* к объекту, если он не имел его изначально. Если же право *r* оказалось в ячейке матрицы M, в которой оно изначально отсутствовало, то говорят, что произошла утечка права *r*.

Очевидно, что для каждого субъекта S, активизированного в момент времени *t*, существует единственный субъект S', который активировал (создал) некоторое право доступа *r* для субъекта S. Поэтому посредством анализа графа отношений между субъектами и объектами можно выявить единственного пользователя, от имени которого активизировано исследуемое право субъекта S. Соответственно, для любого объекта существует единственный пользователь, создавший данный объект и сформировавший для него список прав доступа. В этом случае схематичное изображение канала утечки в виде разрешенного доступа от имени разных пользователей к одному и тому же объекту будет выглядеть следующим образом: Si → (Запись в момент времени *t*1) → O и Sj → (Чтение момент времени *t*2) → O, где i≠j и t1 < t2. Фактически это означает, что ничто не мешает легальному пользователю (например, вследствие программной ошибки) перебросить секретную информацию во вновь созданный объект, доступ к которому открыт всем желающим, или организовать скрытый канал утечки при помощи «троянского» коня. Главным недостатком модели является то, что в ней контролируются только операции доступа субъектов к объектам, а не потоки информации между ними. Поэтому, когда «троянская» программа переносит информацию из доступного некоторому пользователю объекта в объект, доступный нарушителю, то формально никакое правило дискреционной политики безопасности не нарушается, но утечка информации происходит.

Рассмотрим еще один пример утечки права. Пусть в начальном состоянии в системе имеются объект O и два субъекта: S и T; S не обладает никаким правом по отношению к O, а T обладает некоторым правом *a* по отношению к O. Сформулируем последовательность шагов, показывающих, как субъект S может получить право доступа *a* по отношению к субъекту O:

* 1. Начальное состояние;
  2. Субъект S создаёт новый объект X, по отношению к которому автоматически получает права чтения и записи (потому что при создании объекта он становится владельцем объекта);
  3. Субъект S передаёт субъекту T права чтения и записи по отношению к X;
  4. Субъект T копирует часть или все записи объекта O в объект X;
  5. Субъект S, имея полные права доступа к X, получает право доступа к скопированным данным, что означает фактический перенос права *a* по отношению к O от субъекта T к субъекту S.

Классическая модель Харрисона-Руззо-Ульмана широко используется при проведении формальной верификации корректности построения систем разграничения доступа в высоко защищённых автоматизированных системах. Проблему безопасности можно сформулировать в следующем общем виде: существует ли какое-либо достижимое состояние Q, при котором некоторый субъект будет обладать правами доступа к определенному объекту. В рамках решения данной проблемы сформулированы и доказаны две теоремы:

*Теорема 1.* Проблема безопасности разрешима для моно-условныхсистем, т.е. запросы которых содержат одну примитивную операцию во фразе WHERE.

*Теорема 2.* Проблема безопасности не разрешима в общем случае.

В 1976 году Харрисон, Руззо и Ульман доказали, что в общем случае вопрос определения безопасности компьютерной системы неразрешим. Иными словами, не существует алгоритма, позволяющего определить, будет ли компьютерная система безопасна или небезопасна в общем случае (т.е. задача определения того, является ли исходное состояние системы безопасным для данного права *r*, является вычислительно неразрешимой). Однако в частных случаях проблема безопасности решается, а именно, авторы показали, что безопасными являются монотонные системы (не содержащие операции DROP и DELETE), системы, не содержащие операций CREATE, и моно-условные системы (запрос к которым содержит только одно условие).

Для оценки безопасности системы также широко используется модель Take-Grant. В качестве основных элементов модели используются граф доступа и правила его преобразования. В модели доминируют два правила: "давать" и "брать". Они играют в ней особую роль, переписывая правила, описывающие допустимые пути изменения графа. В общем существует 4 правила преобразования: правило «брать», правило «давать», правило «создать» и правило «удалить». Используя эти правила, можно воспроизвести состояния, в которых будет находиться система в зависимости от распределения и изменения прав доступа. Следовательно, можно проанализировать возможные угрозы для данной системы.

**Операторы SQL предоставления и отмены привилегий**

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

Оператор предоставления привилегий имеет следующий формат:

GRANT {<список действий>|ALL PRIVILEGES} ON <имя объекта>

TO {<список пользователей>|PUBLIC} [WITH GRANT OPTION]

где <список действий> определяет набор действий из доступного списка действий над объектом данного типа (параметр ALL PRIVILEGES указывает, что разрешены все действия, допустимые для объектов данного типа), <имя объекта> определяет имя объекта защиты: таблицы, представления, хранимой процедуры или триггера, <список пользователей> определяет список идентификаторов пользователей, кому предоставляются данные привилегии. Вместо списка идентификаторов можно воспользоваться параметром PUBLIC. Параметр WITH GRANT OPTION является необязательным и определяет режим, при котором передаются не только права на указанные действия, но и право передавать эти права другим пользователям. Передавать права в этом случае пользователь может только в рамках разрешенных ему действий. В общем случае набор привилегий зависит от реализации СУБД (определяется производителем). Выделяются следующие привилегии манипулирования данными:

SELECT – просматривать данные;

INSERT [(<список полей>)] – добавлять данные;

UPDATE [(<список полей>)] – обновлять данные;

DELETE – удалять данные;

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

USAGE – использовать домены и ограничители целостности;

EXECUTE – выполнять сохраненные процедуры и функции.

Среди привилегий создания/изменения объектов БД выделим наиболее часто используемые:

CREATE <тип объекта> - создание объекта некоторого типа;

ALTER <тип объекта> - изменение структуры объекта;

DROP <тип объекта> - удаление объекта;

ALL - все возможные действия над объектом.

В реализациях могут присутствовать и другие разновидности привилегий, например:

CONTROL (IBM DB2) - комплексная привилегия управления структурой таблицы;

RUNSTAT — выполнение сбора статистической информации по таблице и другие.

Рассмотрим пример: пусть у нас существуют три пользователя с уникальными именами User1, User2 и User3. Все они являются пользователями одной БД. User1 создал объект Tab1, он является владельцем этого объекта и может передать права на работу с этим объектом другим пользователям. Допустим, что пользователь User2 является оператором, который должен вводить данные в Tab1 (например, таблицу новых заказов), а пользователь User3 является менеджером отдела, который должен регулярно просматривать введенные данные. Тогда логично будет выполнить следующие назначения:

GRANT INSERT ON Tab1 TO User2

(или GRANT INSERT, DELETE, UPDATE ON Tab1 TO User2)

GRANT SELECT ON Tab1 TO User3

Эти назначения означают, что пользователь User2 имеет право только вводить новые строки в отношение Tab1, а пользователь User3 имеет право просматривать все строки в таблице Tab1. При назначении прав доступа на операцию модификации можно уточнить, значение каких полей может изменять пользователь. Допустим, что менеджер отдела имеет право изменять цену на предоставляемые услуги, задаваемую в поле COST отношения Tab1. Тогда операция назначения привилегий пользователю User3 может измениться и выглядеть следующим образом:

GRANT SELECT, UPDATE(COST) ON Tab1 TO User3

Если пользователь User1 предполагает, что пользователь User4 может его замещать в случае его отсутствия, то он может предоставить этому пользователю все права по работе с созданной таблицей Tab1.

GRANT ALL PRIVILEGES ON Tab1 TO User4 WITH GRANT OPTION

В этом случае пользователь User4 может сам назначать привилегии по работе с таблицей Tab1 в отсутствие владельца объекта пользователя User1. Поэтому в случае появления нового оператора пользователя User5 он может назначить ему права на ввод новых строк в таблицу командой

GRANT INSERT ON Tab1 TO User5

Если при передаче полномочий набор операций над объектом ограничен, то пользователь, которому переданы эти полномочия, может передать другому пользователю только те полномочия, которые есть у него, или часть этих полномочий. Поэтому если пользователю User4 были делегированы следующие полномочия:

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

то пользователь User4 не сможет передать полномочия на ввод данных пользователю User5, потому что эта операция не входит в список разрешенных для него самого.

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

REVOKE {<список действий>|ALL PRIVILEGES} ON <имя объекта>

FROM {<список пользователей>|PUBLIC} {CASCADE|RESTRICT}

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

REVOKE ALL PRIVILEGES ON Tab1 FROM User4 CASCADE

будут отменены привилегии и пользователя User5, которому пользователь User4 успел присвоить привилегии. Параметр RESTRICT ограничивает отмену привилегий только пользователю, непосредственно упомянутому в операторе REVOKE. Но при наличии делегированных привилегий при выполнении инструкции REVOKE возникнет ошибка. Так, например, операция:

REVOKE ALL PRIVILEGES ON Tab1 FROM User4 RESTRICT

не будет выполнена, потому что пользователь User4 передал часть своих полномочий пользователю User5.

Посредством оператора REVOKE можно отобрать все или только некоторые из ранее присвоенных привилегий по работе с конкретным объектом. При этом из описания синтаксиса оператора отмены привилегий видно, что можно отобрать привилегии одним оператором сразу у нескольких пользователей или у целой группы (параметр PUBLIC). Поэтому корректным будет следующее использование оператора REVOKE:

REVOKE INSERT ON Tab1 FROM User2, User4 CASCADE

При работе с другими объектами (например, с хранимыми процедурами) изменяется список операций, которые используются в операторах GRANT и REVOKE. По умолчанию, действие, соответствующее запуску (исполнению) хранимой процедуры, назначается всем членам группы PUBLIC. Если требуется изменить это условие, то после создания хранимой процедуры необходимо записать оператор REVOKE

REVOKE EXECUTE ON COUNT\_EX FROM PUBLIC CASCADE

а затем назначить новые права определенным пользователям

GRANT EXECUTE ON COUNT\_EX FROM User4

Объектные привилегии назначает администратор или владелец БД. Например, (в SQL Server):

GRANT CREATE DATABASE ON SERVER\_0 TO main\_user

По принципу иерархии пользователь main\_user, создав свою БД, теперь может предоставить права на создание или изменение любых объектов в этой БД другим пользователям, например:

GRANT CREATE TABLE, ALTER TABLE, DROP TABLE ON MyDB TO User1

В этом случае пользователь User1 может создавать, изменять или удалять таблицы в БД MyDB, однако он не может разрешить создавать или изменять таблицы в этой БД другим пользователям, потому что ему дано разрешение без права делегирования своих возможностей.

**Достоинства и недостатки дискреционного разграничения доступа**

К достоинствам дискреционного разграничения доступа относятся относительно простая реализация (проверка прав доступа субъекта к объекту производится в момент открытия этого объекта в процессе субъекта), хорошая изученность, универсальность, наглядность и гибкость. Однако дискреционная защита является довольно слабой, так как привилегии существуют отдельно от данных и доступ ограничивается только к именованным объектам, а не собственно к хранящимся данным. В случае реляционной БД объектом будет, например, именованное отношение (таблица). В этом случае нельзя в полном объеме ограничить доступ только к части информации, хранящейся в таблице. Это связано с тем, что даже если ввести отдельный атрибут, который будет хранить информацию о метке конфиденциальности документа, то средствами SQL можно будет получить выборку данных без учета атрибута данной метки. Фактически это означает, что либо сам сервер баз данных должен предоставить более высокий уровень защиты информации, либо придется реализовать данный уровень защиты информации с помощью жесткого ограничения операций, которые пользователь может выполнить посредством SQL. На некотором уровне такое разграничение можно реализовать с помощью хранимых процедур, но не полностью - в том смысле, что само ядро СУБД позволяет разорвать связь «защищаемый объект - метка конфиденциальности». Дискреционное разграничение доступа имеет ряд и других недостатков. Перечислим все недостатки дискреционной модели в виде списка:

1. Хранение привилегий доступа отдельно от данных;

2. Ограничение доступа производится на уровне именованных объектов, а не самих хранящихся данных;

3. Уязвимость по отношению к вредоносным программам вида Троянских коней. Дискреционная модель позволяет пользователям без ограничений передавать свои права другим пользователям (что и используется Троянскими конями). Нет различия между пользователем и субъектом, т.е. между человеком, кому, в конечном счете, были назначены определенные права доступа к объектам и процессам, порожденным данным пользователем. Это также позволяет Троянским коням, запущенным от имени авторизованных пользователей, получать свободный доступ к данным.

4. Статичность разграничения доступа — права доступа к уже открытому объекту в дальнейшем не изменяются независимо от изменения состояния компьютерной системы;

5. Отсутствие средств защиты от утечки конфиденциальной информации. Иначе говоря, дискреционное разграничение доступа не обеспечивает возможность проверки, не приведет ли разрешение доступа к объекту для некоторого субъекта к нарушению безопасности информации в компьютерной системе;

6. Средства защиты не позволяют отследить передачу секретных материалов;

7. Возможность множественного назначения и отзыва привилегий доступа к одному и тому же объекту может привести к неконтролируемому доступу к данным. Предположим, субъект s1 предоставил определенные права доступа к объекту o1 субъекту s2. Затем субъект s3 предоставил те же привилегии к o1 все тому же субъекту s2, будучи не поставленным в известность, что это уже было сделано субъектом s1. Позднее субъект s3 изменил свое мнение и отозвал предоставленные им привилегии. Но его действие не вызвало желаемый эффект, поскольку отозванные им привилегии по-прежнему остаются в матрице доступа, поскольку они были ранее назначены субъектом s1;

8. При большом количестве пользователей трудно отследить все пути доступа.

Отмеченных недостатков во многом лишено мандатное разграничение доступа.

Дискреционная модель является очень популярной у разработчиков СУБД. Она реализована в практически всех SQL-совместимых СУБД.

Операторы SQL GRANT, REVOKE, DENY, реализующие дискреционную модель разграничения доступа, определены в стандарте языка SQL.

**4** **Мандатное разграничение доступа**

* основным характеристикам мандатного (обязательного) подхода разграничения доступа относятся следующие положения:
* все субъекты и объекты должны быть однозначно идентифицированы;
* имеется линейно упорядоченный набор меток конфиденциальности (секретности) и соответствующих им уровней (степеней) допуска (нулевая метка или степень соответствуют общедоступному объекту и степени допуска к работе только с общедоступными объектами), например, *U - Unclassified, SU* *–* *Sensitive but unclassified, S – Secret, TS – Top secret*;
* каждому объекту присвоена метка конфиденциальности;
* каждому субъекту присваивается степень допуска;
* право на чтение информации из объекта получает только тот субъект, чья степень допуска не меньше метки конфиденциальности данного объекта;
* право на запись информации в объект получает только тот субъект, чей уровень конфиденциальности не больше метки конфиденциальности данного объекта. Это означает также, что всякая информация, записанная некоторым субъектом, автоматически получает уровень конфиденциальности, равный уровню допуска этого субъекта;
* в процессе своего существования каждый субъект имеет свой уровень конфиденциальности, равный максимуму из меток конфиденциальности объектов, к которым данный субъект получил доступ.

Мандатный подход используется специальными (trusted) системами, предназначенными для государственных, военных и других организаций с жёсткой структурой. Основной целью мандатного разграничения доступа к объектам является предотвращение утечки информации из объектов с высокой меткой конфиденциальности в объекты с низкой меткой конфиденциальности (противодействие созданию каналов передачи информации «сверху вниз»).

Для мандатного разграничения доступа к объектам компьютерной системы формально доказано следующее важное утверждение (принципиально отличающее MAC от DAC): если начальное состояние компьютерной системы безопасно и все переходы из одного состояния системы в другое не нарушают правил разграничения доступа, то любое последующее состояние компьютерной системы также безопасно. К другим достоинствам мандатного разграничения доступа относятся:

* более высокая надежность работы системы, так как при разграничении доступа к объектам контролируется и состояние самой системы, а не только соблюдение установленных правил;
* большая простота определения правил разграничения доступа по сравнению с дискреционным разграничением.

Главное отличие MAC от DAC состоит в том, что в МАС метки конфиденциальности неизменны на всем протяжении существования объекта защиты (они создаются и уничтожаются только вместе с ним) и располагаются вместе с защищаемыми данными, а не в системном каталоге, как это происходит в DAC. Другим важным отличием является то, что в мандатной модели контролируются не операции, осуществляемые субъектом над объектом, а потоки информации, которые могут быть только двух видов: либо от субъекта к объекту (запись), либо от объекта к субъекту (чтение).

Мандатный принцип построения системы разграничения доступа в СУБД реализует ***многоуровневую модель безопасности данных,*** называемую еще моделью Белл - ЛаПадула (по имени ее авторов - Д. Белла и Л. ЛаПадула), введенную в 1975 г. В модели Белл - ЛаПадула устанавливаются и поддерживаются два основных *ограничения* политики безопасности:

1. Простое правило безопасности (Simple Security), реализующее запрет чтения вверх (No Read Up — NRU);
   1. \*-свойство (star-property), реализующее запрет записи вниз (No Write Down - NWD).

Ограничение NRU является логическим следствием мандатного принципа разграничения доступа, запрещая субъектам читать данные из объектов более высокой степени секретности, чем позволяет их допуск. Ограничение NWD предотвращает перенос (утечку) конфиденциальной информации путем ее копирования из объектов с высоким уровнем конфиденциальности в неконфиденциальные объекты или в объекты с меньшим уровнем конфиденциальности.

Ограничения NRU и NWD приводят к тому, что по разным типам доступа (чтение, запись, создание, удаление) в модели Белл - ЛаПадула устанавливается разный порядок доступа конкретного субъекта к объектам. В частности, по типу доступа «создание» субъект с низшим уровнем допуска имеет возможность создавать объекты (записи) в объектах более высокого уровня конфиденциальности. Такой подход, тем не менее, отражает реальные ситуации, когда служащий отдела кадров может порождать первичные документы личных дел новых сотрудников, но при этом не имеет собственно самого доступа к этим документам по другим типам операций (чтение, удаление, изменение).

Ключевым понятием в модели Белла и ЛаПадулла является понятие решетки безопасности (security lattice). Математически, решеткой безопасности называется алгебраическая система, состоящая из оператора, определяющего отношение порядка (dominance) для уровней секретности и операторов наименьшей верхней и наибольшей нижней границ. Отношение порядка обладает свойствами рефлективности (разрешены потоки информации между субъектами и объектами одного уровня секретности) и транзитивности (если информация может передаваться от субъектов и объектов уровня А к субъектам и объектам уровня В и от субъектов и объектов уровня В к субъектам и объектам уровня С, то она может передаваться от субъектов и объектов уровня А к субъектам и объектам уровня С). Операторы наименьшей и наибольшей границ определяются таким образом, чтобы для каждой пары уровней секретности существовал единственный элемент наименьшей верхней границы и единственный элемент наибольшей нижней границы.

Математическая формализация модели позволяет сформулировать основные положения безопасности системы и по возможности строго доказать их. Состояние системы называется безопасным по чтению (или simple-безопасным), если для каждого субъекта, осуществляющего в этом состоянии доступ по чтению к объекту, уровень доступа субъекта доминирует над уровнем секретности объекта. Состояние системы называется безопасным по записи (или \*-безопасным), если для каждого субъекта, осуществляющего в этом состоянии доступ по записи к объекту, уровень секретности объекта доминирует над уровнем доступа субъекта. Состояние системы называется безопасным, если оно безопасно по чтению и по записи и наконец, система называется безопасной, если начальное и все последующие состояния безопасны. Как уже упоминалось, в рамках данной модели доказано важное утверждение, если начальное состояние системы безопасно и все переходы из одного состояния системы в другое не нарушают правил разграничения доступа, то любое последующее состояние системы также безопасно, что позволяет применять мандатную модель в системах с высоким уровнем секретности.

Отметим и недостатки мандатного разграничения доступа:

* невозможность автоматизации назначения уровней секретности и определения границ защищаемых данных, что в больших системах может приводить к практически бесконечному ручному процессу конфигурации системы;
* снижение эффективности работы компьютерной системы, так как проверка прав доступа субъекта к объекту выполняется не только при открытии объекта в процессе субъекта, но и перед выполнением любой операции чтения из объекта или записи в объект;
* создание дополнительных неудобств в работе пользователей компьютерной системы, связанных с невозможностью изменения информации в неконфиденциальном объекте, если тот же самый процесс использует информацию из конфиденциального объекта (его уровень конфиденциальности больше нуля). Это зачастую решается путем разрешения пользователю выступать от имени субъекта с меньшим уровнем доступа, что в свою очередь приводит к деградации системы защиты;
* пользователь нижнего уровня имеет право записи в объекты всех уровней, т.о. этот пользователь может переписать существующий объект, что равносильно его удалению. Для устранения этого недостатка второе правило изменяется так, что пользователь имеет доступ на запись только на своем уровне.

Из-за отмеченных недостатков мандатного разграничения доступа в реальных СУБД множество объектов, к которым применяется мандатное разграничение, является подмножеством множества объектов, доступ к которым осуществляется на основе дискреционного разграничения. В целях уменьшения негативных последствий ограничения NWD в систему вводят привилегированного пользователя, имеющего специальные полномочия на удаление любого объекта системы и понижения метки конфиденциальности. Имеются также расширения мандатной модели, adapted mandatory access model и др., некоторым образом снимающие недостатки MAC.

Примером реализации мандатного подхода разграничения доступа можно считать компонент Oracle Label Security (OLS), реализованный в СУБД Oracle, начиная с 9i версии. Примером Российской СУБД, реализующей стандарт SQL-92 является СУБД ЛИНТЕР.

**5 Ролевое разграничение доступа**

Ролевая модель безопасности представляет собой существенно усовершенствованную модель Харрисона-Руззо-Ульмана, однако ее нельзя отнести ни к дискреционному, ни к мандатному подходу, потому что управление доступом в ней осуществляется как на основе матрицы прав доступа для ролей, так и с помощью правил, регламентирующих назначение ролей пользователям и их активацию во время сеансов. Поэтому ролевое разграничение доступа представляет собой совершенно особый тип контроля доступа, основанного на компромиссе между гибкостью управления доступом, характерной для дискреционных моделей, и жесткостью правил контроля доступа, присущей мандатным моделям.

Ролевое разграничение доступа основано на том соображении, что в реальной жизни организации ее сотрудники выполняют определенные функциональные обязанности не от своего имени, а в рамках некоторой занимаемой ими должности (или роли). Количество ролей в системе может не соответствовать количеству реальных пользователей – один пользователь, если на нем лежат различные обязанности, требующие различных полномочий, может выполнять (одновременно или последовательно) несколько ролей, а несколько пользователей могут пользоваться одной и той же ролью, если они выполняют одинаковую работу. Реализация ролевого разграничения доступа к объектам СУБД или компьютерной системы в целом требует разработки набора (библиотеки) ролей, определяемых как набор прав доступа к объектам информационной системы (прав на выполнение над ними определенного набора действий). Этот набор прав должен соответствовать выполняемым работником обязанностям.

Ролевое разграничение доступа оперирует следующими понятиями:

* субъекты и объекты доступа;
* привилегии - минимально возможные действия субъекта, требующие разрешения или запрещения этого действия;
* правила - объединение привилегии, подмножества объектов, для которых может быть определена данная привилегия, и признака разрешения или запрещения этой привилегии;
* роль - набор правил, соответственно определяющих, какими привилегиями и по отношению к каким объектам будет обладать субъект, если ему будет назначена эта роль;
* сессия - подмножество ролей, которые активировал субъект после входа в систему в течение определенного интервала времени.

При использовании ролевой политики управление доступом осуществляется в три стадии: во-первых, для каждой роли указывается набор полномочий, представляющий набор прав доступа к объектам, во-вторых, каждому пользователю назначается список доступных ему ролей, и, в-третьих, после авторизации пользователя в системе для него создается сессия. Полномочия назначаются ролям в соответствии с принципом наименьших привилегий, из которого следует, что каждый пользователь должен обладать только минимально необходимым для выполнения своей работы набором полномочий. При реализации ролевой модели может быть введен ряд ограничений: например, назначение роли главного администратора (суперпользователя) может быть предоставлено только одному пользователю, вводится запрет совмещения одним пользователем определенных ролей, ограничивается количество пользователей, одновременно выполняющих определенную роль и т.п.

Критерий безопасности системы при использовании ролевой модели можно сформулировать следующим образом: система считается безопасной, если любой пользователь в системе может осуществлять только те действия, которые требуют полномочий, входящих в совокупность полномочий всех ролей, доступных для него в данной сессии. В отличие от других моделей, ролевая модель управления доступом практически не гарантирует безопасность с помощью формальных доказательств, а только определяет характер ограничений, соблюдение которых и служит критерием безопасности системы.

Ролевая модель разграничения доступа сочетает элементы мандатного разграничения (объединение объектов и привилегий доступа в одном правиле) и дискреционного разграничения (назначение ролей отдельным субъектам). Этим обеспечивается жесткость правил разграничения доступа и гибкость настройки механизма разграничения в конкретных условиях применения.

Преимущества ролевого разграничения доступа к объектам проявляются при организации коллективного доступа к ресурсам сложных информационных систем с большим количеством пользователей и объектов. К достоинству ролевого разграничения доступа следует также отнести наличие иерархии ролей, разделение обязанностей, возможность регистрации в системе с наименьшей ролью (принцип наименьших привилегий) и возможность запуска приложений, требующих фиксированного набора прав доступа, не зависящих от прав доступа пользователя, запустившего данное приложение, что возможно при назначении пользователю определенной роли.

К недостаткам ролевого разграничения доступа относится отсутствие формальных доказательств безопасности компьютерной системы, возможность внесения дублирования и избыточности при предоставлении пользователям прав доступа и сложность конструирования ролей.

Наряду с дискреционной, ролевая модель разграничения доступа, реализована во множестве развитых коммерческих СУБД, например, Microsoft SQL Server, Oracle, IBM DB2.

**Лекция 9. Архитектура системы безопасности СУБД SQL Server**

Система безопасности СУБД MS SQL Server реализует дискреционный и ролевой подходы к управлению доступом и имеет два уровня: уровень сервера и уровень базы данных (рис. 3). На уровне сервера разрешается или отклоняется доступ пользователей к самому серверу. На уровне базы данных пользователи, имеющие доступ на уровне сервера, получают доступ к объектам базы данных. Такой подход позволяет более гибко управлять доступом пользователей к базам данных.



На уровне сервера система безопасности оперирует следующими понятиями:

* аутентификация (authentication);
* учетная запись (login);
* встроенные роли сервера (fixed server roles).

На уровне базы данных используются понятия:

* пользователь базы данных (database user);
* фиксированная роль базы данных (fixed database role);
* пользовательская роль базы данных (users database role);
* роль приложения (application role).

Следовательно, можно выделить две группы ролей:

* роли сервера (server role);
* роли базы данных (database role).

**1 Система безопасности уровня сервера**

SQL Server использует двухэтапную схему аутентификации. Вначале пользователь аутентифицируется сервером. Только после успешной аутентификации на сервере пользователю может быть предоставлен доступ к одной или нескольким БД. SQL Server хранит всю информацию о регистрационных записях в базе данных master.

SQL Server предлагает два режима аутентификации пользователей:

* Режим аутентификации средствами OS Windows. В этом режиме SQL Server полностью доверяет операционной системе при аутентификации пользователей;
* Смешанный режим аутентификации (Windows Authentication and SQL Server Authentication). В этом режиме системы аутентификации Windows и SQL Server существуют независимо друг от друга.

При установке SQL Server одним из первых решений, которые следует принять, является выбор используемого метода аутентификации. Установленный при инсталляции режим аутентификации можно изменить на странице Security диалогового окна SQL Server Properties в утилите Management Studio. В программном коде установленный режим аутентификации можно проверить с помощью системной хранимой процедуры xp\_loginconfig:

EXEC xp\_loginconfig 'login mode'

Результат выполнения этой процедуры выглядит следующим образом:

name config\_value

login mode Mixed

Соответственно, пользователь должен быть идентифицирован сервером одним из следующих методов:

* с помощью учетной записи пользователя Windows (в общем случае принадлежащей некоторому домену или рабочей группе);
* на основе членства в одной из групп пользователей Windows;
* с помощью отдельной регистрационной записи SQL Server (если на сервере используется смешанная модель безопасности).

Набор ролей сервера строго ограничен. Никто, включая администратора сервера, не может создать новую или удалить существующую роль сервера. Поэтому они называются *фиксированными ролями* (fixed server roles). Ниже приведен список некоторых фиксированных ролей сервера с кратким описанием каждой из них:

Sysadmin

(System Adminis-

trators)

Setupadmin

(Setup Adminis-

trators)

Serveradmin

(Server Adminis-

trators)

Securityadmin

(Security Admin-

Члены этой роли имеют абсолютные права в SQL Server.

Никто не имеет больших прав доступа, чем члены данной

роли.

Этой роли предоставлены права управления связанными

серверами, конфигурирования хранимых процедур, запуска-

емых автоматически при старте SQL Server, а также право

добавлять учетные записи в роль setupadmin.

Обычно в эту роль включаются пользователи, которые

должны выполнять администрирование сервера. Имеют пра-

во на останов сервера (SHUTDOWN), изменять параметры

работы служб (sp\_conf igure), применять изменения

(RECONFIGURE), управлять полнотекстовым поиском

(sp\_fulltext\_service).

Члены данной роли имеют возможность создавать новые

учетные записи, которым они могут предоставлять права на

создание баз данных и ее объектов, а также управлять свя-

Dbcreator (Database Crea-tors)

занными серверами, включать учетные записи в роль

securityadmin и читать журнал ошибок SQL Server.

Члены этой роли могут создавать новые базы данных, уда-

лять и переименовывать имеющиеся, восстанавливать ре-

зервные копии базы данных и журнала транзакций.

Для добавления учетной записи пользователя в ту или иную фиксированную роль сервера можно воспользоваться хранимой процедурой sp\_addsrvrolemember.

Например, рассмотрим добавление учетной записи Windows с именем Admin

EXEC sp\_addsrvrolemember 'STORAGE\Admin', 'sysadmin'

Для получения информации о том, какая учетная запись в какую фиксированную роль сервера включена, используется системная хранимая процедура sp\_helpsrvrolemember.

Если процедура вызывается без параметров, то выводится полный список учетных записей, включенных в любую из ролей сервера. Когда же необходимо получить список учетных записей, включенных в конкретную роль сервера, требуется указать имя роли. В приведенном ниже примере выводится информация о членах роли sysadmin:

EXEC sp\_helpsrvrolemember 'sysadmin'

Для удаления учетной записи из фиксированной роли сервера предназначена системная хранимая процедура sp\_dropsrvrolemember.

Из приведенного списка ролей уровня сервера видно, что включение пользователей в эти роли позволяет определить только администраторов сервера, а не рядовых пользователей БД. Путь добавления обычных пользователей БД зависит от выбранного режима аутентификации пользователей.

* 1. **Аутентификация Windows**

Использование аутентификации Windows означает, что пользователю для доступа к SQL Server достаточно иметь учетную запись Windows. Идентификатор безопасности Windows передается из операционной системы на сервер баз данных. SQL Server не выполняет никаких дополнительных проверок. Пользователь автоматически получает соответствующие права доступа к данным SQL Server сразу же после регистрации в домене. Такой метод предоставления доступа называется *установлением доверительного соединения.* Операционная система работает с *учетными записями* (logins), которые содержат все данные о пользователе, включая его имя, пароль, членство в группах, каталог по умолчанию и т. д. Каждая учетная запись имеет *уникальный идентификатор* (Login ID)или,как его называют по-другому, *идентификатор безопасности* (SID, Security Identification),с помощью которого пользователь регистрируется в сети.

Идентификатор представляет собой длинное (85-битовое) шестнадцатеричное число, которое генерируется случайным образом операционной системой во время создания учетной записи. Такой подход позволяет избежать подделки учетных записей пользователей. Если пользователь был удален из домена, то даже повторное создание пользователя с аналогичными характеристиками (имя учетной записи, пароль, членство в группе и т. д.) не даст возможности получить доступ к объектам, к которым имел доступ оригинальный пользователь. Применительно к SQL Server можно сказать, что если пользователь домена имел определенные права доступа, но был удален, то никто не сможет присвоить его права доступа.

Аутентификация Windows предусматривает сохранение в системной базе данных Master только идентификационного номера учетной записи пользователя в домене (SID). Информация об имени пользователя, его пароле и т. д. хранится в базе данных домена. Изменение имени пользователя или его пароля никак не отразится на правах доступа к SQL Server. Информация об учетной записи пользователя и его членстве в группах Windows считывается SQL Server из базы данных системы безопасности домена во время подключения пользователя. Если администратор внес какие-то изменения в учетную запись пользователя, например, исключил его из некоторой группы, то изменения отразятся только во время очередной регистрации пользователя в домене или в SQL Server в зависимости от категории сделанных изменений.

Аутентификация Windows дает определенные преимущества. На пользователях автоматически отражаются все правила политики безопасности, установленные в домене. Пользователю не приходится запоминать еще один пароль. Это повышает уровень общей защищенности данных. Например, автоматически контролируется минимальная длина пароля и срок его действия. Операционная система требует от пользователя периодической смены пароля. Дополнительно можно запретить пользователям установку паролей, уже указывавшихся ранее. Кроме того, Windows имеет встроенные средства защиты от подбора паролей. Аутентификация Windows работает также с группами пользователей. Когда имя группы Windows передается в SQL Server в качестве регистрационной записи, любой член этой группы может быть аутентифицирован сервером баз данных. SQL Server также известно истинное имя пользователя Windows, входящего в группу, вследствие чего приложение может выполнять аудит на уровне пользователей, а также на уровне групп пользователей.

Для создания учетной записи пользователя или группы Windows в SQL Server в программном коде используется системная хранимая процедура sp\_grantlogin. В качестве аргумента ей необходимо передавать полное имя пользователя, включающее имя домена, как в следующем примере:

EXEC sp\_grantlogin 'RFE\Sidorov'

Для удаления учетной записи или группы Windows из SQL Server программным путем используется системная хранимая процедура sp\_revokelogin:

EXEC sp\_revokelogin 'RFE\Sidorov'

Разумеется, эта операция не удаляет данную учетную запись из операционной системы – произойдет всего лишь исключение пользователя из списка пользователей сервера баз данных.

Запрет учетной записи Windows можно произвести с помощью системной хранимой процедуры sp\_denylogin. Следовательно, доступ любого пользователя в SQL Server может быть принудительно закрыт. Это полностью запрещает доступ пользователя к SQL Server, даже если он был открыт с помощью другого метода. Например:

EXEC sp\_denylogin 'RFE\Sidorov'

Если пользователь Windows был добавлен в SQL Server, а затем был удален из домена Windows, он продолжает существовать как пользователь в сервере баз данных, но считается уже «осиротевшим». Это значит, что, несмотря на то, что пользователь формально имеет доступ к серверу, он не имеет доступа к ресурсам сети и, следовательно, ко всему комплексу средств SQL Server. Системная хранимая процедура sp\_validatelogins позволяет найти "осиротевших" пользователей и возвращает их идентификаторы системы безопасности Windows NT и имена учетных записей. В следующем примере пользователю RFE\Joe был открыт доступ к SQL Server, после чего его учетная запись была удалена из Windows:

EXEC sp\_validatelogins

SID

Ox010500000000000515000000FCE31531A931

NT Login

RFE\Joe

Это нельзя считать брешью в системе безопасности. Не имея учетной записи Windows с таким идентификатором (невозможно повторно создать учетную запись и заданным SID), пользователь не сможет подключиться к SQL Server. Для разрешения проблемы "осиротевших" пользователей можно использовать следующий протокол:

1. Отзовите права доступа пользователя ко всем базам данных с помощью хранимой процедуры sp\_revokedbaccess;
2. Отзовите право доступа данного пользователя к серверу с помощью sp\_revokelogin;
3. Создайте для пользователя новую учетную запись;
4. Зарегистрируйте учетную запись на сервере.

**1.2 Аутентификация SQL Server**

Этот тип аутентификации реализуется на самом сервере SQL Server. В этом случае в системной базе Master хранится полная информация о пользователях, включая имя пользователя и его пароль. Для каждого пользователя указывается имя учетной записи, уникальный идентификатор SQL Server, пароль и другая информация. При попытке пользователя подключиться к серверу система безопасности потребует ввести имя учетной записи и ее пароль. Затем система сравнивает введенные данные с информацией, хранящейся в системных таблицах. Если данные совпадают, то доступ предоставляется. В противном случае пользователь получает сообщение об ошибке, и соединение не устанавливается.

Наличие дополнительных регистрационных записей SQL Server уместно тогда, когда аутентификация Windows не доступна или не подходит для создаваемой системы. Эта регистрация имеет обратную совместимость с предыдущими версиями сервера, а также с приложениями, в которых регистрационная запись встроена в программный код. Аутентификация SQL Server в основном применяется клиентами, для которых недоступна регистрация в домене Windows. Например, пользователями Novell NetWare, Unix и т. д. При подключении к SQL Server через Internet регистрация в домене не выполняется, поэтому в данном случае также необходимо использовать аутентификацию SQL Server.

В процессе инсталляции SQL Server и выборе смешанного режима аутентификации мастер установки создает специальную учетную запись sa (system administrator) с пустым паролем, являющийся членом фиксированной серверной роли sysadmin и имеющий все права доступа к серверу. Зачастую ему забывают присвоить пароль, и это открывает путь к серверу любому взломщику. Как правило, наличие такой бреши хакеры проверяют в первую очередь. Исходя из этого, прежде всего нужно задать пароль или просто отключить пользователя sa и назначить административной роли sysadmin какого-то другого пользователя (или создать дополнительные роли с административными привилегиями). Эта учетная запись оставлена для сохранения обратной совместимости с предыдущими версиями SQL Server. Ранее учетная запись была обязательной, имела абсолютные права по управлению сервером и не могла быть удалена. Пользователю sa в процессе установки всегда присваивается одинаковый на всех компьютерах идентификатор безопасности 0x01.

Идентификатор безопасности хранится в столбце sid таблицы sysxlogins системной базы данных Master. В ней хранится информация об учетных записях как SQL Server, так и Windows. Для учетных записей SQL Server максимальный размер идентификатора безопасности составляет 16 байт, для учетных записей Windows - 28 байт. Каждая строка этой таблицы соответствует одной учетной записи. Таким образом, в таблице sysxlogins может содержаться довольно много строк с информацией об учетных записях. Для более удобной работы с локальными учетными записями можно использовать представление syslogins, включающее только те строки таблицы sysxlogins, которые имеют в столбце srvid (идентификационный номер сервера) значение NULL.

Для регистрации пользователя используется системная хранимая процедура sp\_addlogin:

sp\_addlogin 'имя', 'пароль', 'база\_по\_умолчанию',

'язык\_по\_умолчанию', 'идентификатор\_пользователя\_сервера', 'параметр\_шифрования'

Среди передаваемых процедуре аргументов обязательным является только имя регистрационной записи. Так как в данном случае требуется настройка пользователя, а не просто выбор его из списка, данная задача более сложная, чем выполнение процедуры sp\_grantlogin. Например, следующий программный код создает пользователя SQL Server с именем Den и назначает ему в качестве базы данных по умолчанию учебную базу test:

EXEC sp\_addlogin 'Den', 'myoldpassword', 'test'

Если некоторый пользователь создается на двух серверах как один и тот же, то для второго сервера следует явно задать идентификатор SID, присвоенный первым сервером. Для получения идентификатора SID используется представление sysserver\_principals:

SELECT Name, SID FROM sysserver\_principals WHERE Name = 'Den'

Name

Den

SID

0X1EFDC478DEB52 045B52D241B33B2CD7E

Пароль может быть изменен с помощью системной хранимой процедуры sp\_password, например:

EXEC sp\_password 'myoldpassword', 'mynewpassword', 'Den'

Если пароль пуст, то в хранимую процедуру вместо пустой строки (' ') передается NULL. Если параметр @sid опущен или для него указано значение NULL (также являющееся значением по умолчанию для параметра @sid), то хранимая процедура sp\_addlogin самостоятельно сгенерирует идентификатор безопасности. Для примера создадим учетную запись с конкретным идентификатором безопасности и паролем:

USE pubs

EXEC sp\_addlogin 'Andrey' ,'analitik', @sid = 0x0123456789ABCDEF0123456789ABCDEF

После создания учетной записи можно изменить идентификатор безопасности с помощью команды UPDATE, напрямую обратившись к системной таблице. При выборе значения для параметра @sid следует соблюдать требование уникальности идентификаторов безопасности. То есть на текущем сервере к моменту регистрации не должно быть учетных записей с идентификатором безопасности, равным выбранному значению. Список используемых идентификаторов безопасности локального сервера можно просмотреть с помощью следующего запроса:

SELECT sid FROM syslogins

Для удаления регистрационной записи SQL Server используется системная хранимая процедура sp\_droplogin, например:

EXEC sp\_droplogin 'Den'

Удаление учетной записи приводит к удалению и всех ее настроек безопасности.

**2 Система безопасности уровня базы данных**

Для того чтобы пользователь имел возможность выполнять те или иные действия с объектами базы данных, он должен предварительно получить необходимые права доступа. Владелец базы данных или конкретного объекта должен предоставить пользователям базы данных возможность обращения к объектам базы данных. Помимо прав доступа, выдаваемых пользователю явно, существует класс *неявных прав доступа.* Неявные права доступа предоставляются пользователю через членство в фиксированной роли сервера или базы данных. Например, нет никакой другой возможности предоставить пользователю право на создание баз данных, кроме как включить его в роль сервера dbcreator. Другим примером работы неявных прав доступа является получение владельцем объекта абсолютных прав на управление объектом владения. Если пользователь создал объект в базе данных, то ему автоматически предоставляются максимальные права по управлению этим объектом.

Таким образом, на уровне базы данных пользователю могут быть предоставлены привилегии с помощью прикрепления к фиксированной роли базы данных или путем явного назначения привилегий с помощью оператора SQL GRANT. Исключением являются пользователи, учетные записи которых включены в фиксированную роль сервера sysadmin. Члены данной роли имеют неограниченные права в пределах сервера, и, как следствие, полный доступ к любой базе данных, имеющейся на сервере.

Перечислим все фиксированные роли базы данных:

Пользовательские роли - это дополнительные роли, служащие в качестве групп. Роли может быть разрешен доступ к объекту базы данных, а пользователю может быть назначена пользовательская роль базы данных. Все пользователи автоматически становятся членами стандартной роли базы данных public.

Включение пользователей в роль реализует неявное назначение привилегий. Явные разрешения к объектам назначаются с помощью инструкций SQL GRANT, REVOKE, DENY и они достаточно детализированы. Существуют отдельные разрешения для каждого из возможных действий (SELECT, INSERT, UPDATE, RUN и т.д.) над любым объектом. Запрет привилегии замещает собой ее предоставление, а предоставление привилегии замещает собой ее отзыв. Пользователю может быть предоставлено множество разрешений к объекту (индивидуальных, наследованных от некоторой роли или обеспеченных принадлежностью к роли public).

Некоторые фиксированные роли базы данных также влияют на доступ к объекту, например, управляя возможностью считывать информацию и записывать ее в базу данных. Вполне возможна ситуация, когда пользователь был распознан в SQL Server, но у него нет доступа ни к одной из его баз данных. Также возможно и обратное: пользователю открыт доступ к базам данных, но он не был распознан сервером. Перемещение базы данных и ее разрешений на другой сервер без параллельного перемещения регистрационных записей сервера может привести к возникновению таких "осиротевших" пользователей.

В любой базе данных автоматически создаются два пользователя:

1. dbo (database owner). Это специальный пользователь базы данных, являющийся ее *владельцем.* Владелец базы данных имеет абсолютные права по управлению ею. Пользователя dbo нельзя удалить. По умолчанию в пользователя dbo отображается учетная запись sa, которой тем самым предоставляются максимальные права в базе данных. Кроме того, все члены роли базы данных dbowner также считаются владельцами базы данных. Пользователь dbo включен в роль db\_owner и не может быть удален из нее;
2. guest. Если учетной записи явно не предоставлен доступ к базе данных, то она автоматически отображается сервером в пользователя guest. С помощью этого пользователя можно предоставлять разрешения на доступ к объектам базы данных, необходимые любому пользователю. Разрешив доступ пользователю guest, вы, тем самым, даете аналогичные права доступа всем учетным записям, сконфигурированным на SQL Server. Для повышения безопасности хранящейся информации рекомендуется удалять пользователя guest из базы данных.

Информация о пользователях, созданных в базе данных, хранится в системной таблице sysusers. Информацию о пользователях текущей базы данных можно получить с помощью системной хранимой процедуры sp\_helpuser:

USE pubs

EXEC sp\_helpuser

Создание нового пользователя и связывание его с учетной записью выполняется с помощью одной из двух хранимых процедур:

* sp\_adduser. Данная процедура оставлена для обеспечения совместимости с предыдущими версиями SQL Server и оперирует устаревшими понятиями.
* sp\_grantdbaccess. Эта хранимая процедура полностью соответствует понятиям системы безопасности SQL Server и пришла на смену предыдущей процедуре, начиная с версии SQL Server 7.0.

Процедура sp\_grantdbaccess имеет следующий синтаксис:

sp\_grantdbaccess [@loginame =] 'login' [,[@name\_in\_db =] 'name\_in\_db' [OUTPUT]]

Правом вызова указанной хранимой процедуры обладают члены фиксированной роли сервера sysadmin и члены фиксированных ролей базы данных db\_owner и db\_accessadmin. Параметр @loginame определяет имя учетной записи, которой предполагается предоставить доступ к текущей базе данных. Указанная учетная запись должна существовать на сервере.

С помощью параметра @name\_in\_db указывается имя, которое будет присвоено создаваемому пользователю. Это имя должно быть уникальным в пределах базы данных. Приведенный далее пример иллюстрирует использование хранимой процедуры sp\_grantdbaccess. В базе данных pubs создается новый пользователь с именем Admin, который связывается с учетной записью STORAGE\Admin:

USE pubs

EXEC sp\_grantdbaccess 'STORAGE\Admin', 'Admin'

Удаление пользователя выполняется с помощью системной хранимой процедуры sp\_revokedbaccess, имеющей синтаксис:

sp\_revokedbaccess [@name\_in\_db =] 'name'

Правом вызова указанной хранимой процедуры обладают члены фиксированной роли сервера sysadmin и члены фиксированных ролей базы данных db\_owner и db\_accessadmin. Единственный параметр процедуры определяет имя пользователя, которого необходимо удалить. Однако, прежде чем решиться на подобный шаг, следует убедиться, что пользователю не принадлежит никакой объект базы данных. Если же пользователь является владельцем одного из объектов базы данных, то следует либо удалить этот объект с помощью команды DROP, либо изменить владельца объекта с помощью системной хранимой процедуры sp\_changeobjectowner. Например, для удаления пользователя Admin достаточно будет выполнить команду:

EXEC sp\_revokedbaccess 'Admin'

Чтобы включить нового члена в фиксированную роль базы данных, необходимо вызвать системную хранимую процедуру sp\_addrolemember, имеющую синтаксис:

sp\_addrolemember [@rolename =] 'role' , [@membername =] 'security\_account'

* помощью параметра @rolename указывается имя роли, в которую требуется добавить нового члена. Указанная роль должна существовать в текущей базе данных. Например:

EXEC sp\_addrolemember 'db\_accessadmin', 'User1'

Для получения информации о членстве во всех ролях текущей базы данных можно использовать процедуру sp\_helprolemember. Например:

USE pubs

EXEC sp\_helprolemember

Исключение из фиксированной роли выполняется с помощью процедуры sp\_droprolemember. Например, для исключения из фиксированной роли db\_accessadmin пользователя User1 необходимо выполнить следующую команду:

USE pubs

EXEC sp\_droprolemember 'db\_accessadmin', 'User1'

Если фиксированные роли предназначены для наделения пользователей специальными правами в базе данных, то пользовательские роли служат лишь для группировки пользователей с целью облегчения управления их правами доступа к объектам. Создание пользовательской роли выполняется с помощью системной хранимой процедуры sp\_addrole, которая имеет синтаксис:

sp\_addrole [@rolename =] 'role' [, [@ownername =] 'owner']

или

CREATE ROLE <rolename>

Правом выполнения указанной хранимой процедуры обладают члены фиксированной роли сервера sysadmin и фиксированных ролей базы данных db\_owner и db\_securityadmin. По умолчанию владельцем роли становится владелец базы данных (пользователь dbo). Таким образом, пользователь dbo приобретает полный контроль над ролью. Если необходимо присвоить права владения ролью другому пользователю, то имя этого пользователя должно быть указано с помощью параметра @ownername. Указываемый пользователь должен иметься в базе данных. Владельцем пользовательской роли может также являться и роль приложения. В качестве примера рассмотрим создание пользовательской роли AccessDBUser, владельцем которой будет являться пользователь guest:

USE pubs

EXEC sp\_addrole 'AccessDBUser', 'guest'

Удаление пользовательской роли осуществляется с помощью хранимой процедуры sp\_droprole, имеющей следующий синтаксис:

sp\_droprole [rolename =] 'role'

* помощью единственного параметра процедуры указывается имя пользовательской роли, которую необходимо удалить из текущей базы данных. Однако перед подобной операцией нужно удалить из нее всех членов, для чего можно использовать хранимую процедуру sp\_droprolemember. Если роль владеет одним или более объектами базы данных, то уничтожить такую роль будет невозможно. Прежде чем приступить к удалению, необходимо либо передать права владения соответствующими объектами с помощью хранимой процедуры sp\_changeobjectowner, либо удалить эти объекты. С помощью приведенного ниже примера можно удалить пользовательскую роль

USE pubs

EXEC sp\_droprole 'AccessDBUser'

Для просмотра списка явных прав доступа возможен запуск системной хранимой процедуры sp\_helprotect, имеющей синтаксис:

sp\_helprotect [[@name =] 'object\_statement'] [,[@username=] 'security\_account'] [, [@grantorname =] 'grantor'] [, [@permissionarea =] 'type']

Например:

sp\_helprotect Clients (для объекта Clients)

sp\_helprotect @username = 'RFE\Joe'

(для

пользователя

RFE\Joe)

Данная процедура эффективна лишь для просмотра явных привилегий, назначенных с помощью оператора GRANT. Для просмотра всех привилегий, включая неявные, следует использовать SQL запрос.