Курс лекций по дисциплине «Хранение и защита информации в базах данных информационных систем»

Оглавление

1. Специфика защиты 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).

1.1 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 за всю корпорацию 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 в Калифорнии 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) значе- ния некоторого поля в БД также позволит выявить наличие такого зна- чения.

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 заработ-

ной 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.

1.2 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 кана-лов 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 со-

стоит в том, чтобы организовать скрытый канал для передачи информации от программно-аппаратного агента, функционирующего в среде Высокий, к другому программно-аппаратному агенту, функционирующемув 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 резуль-

тате 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.

1.3 SQL инъекции

Инъекцией SQL (SQL Injection) называют технологию взлома, которая добавляет в параметр динамического SQL 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 DDL DELETE, которая и будет выполнена в пакете с основной:

```
SELECT * FROM Customers WHERE CustomerlD ='Сидоров'; De-
lete 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 --'
```

Так как в инструкции отбираются все строки, далее все зависит только от того, как система обрабатывает множества строк. Однако независимо от этого, программа сделает совсем не то, что должна была сделать. 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--'
```

Значение 1=1 всегда истинно, таким образом, значение пароля 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 UserlD 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/p
asswd'),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,md
5(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 xp_ (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.

3. 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 досту- па к СУБД может использоваться отдельная подобная процедура. Ответ-

ственность 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 пользовали 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.

3.1 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 с ID связывается 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 входа может 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 злоумышленник может 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 пинкод, а уже затем ввести случайным образом сгенерированное настранице 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 КС,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 субъек- том доступа для средств защиты информации от несанкционированного доступа СУБД.

3.2 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.

Следует особо отметить, что 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-единственной 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). Привиле- гии управления сервером баз данных имеются у пользователей с имена- ми 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.

3.3 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 основных 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 частно- сти, он имеет право регистрации новых пользователей и предоставления им привилегий, как системных, так и объектных. Пользователь, имею- щий системные привилегии, является владельцем всех созданных им объектов, имеет по отношению к ним все привилегии и может предо- ставлять их полностью или частично другим пользователям. Минималь- ной 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.

Объекты / Субъекты	**Клиенты**	**Товары**	**Заказы**
Сидоров	чтение, вставка, модификация	чтение, вставка, модификация	чтение, вставка, модификация
Петров	чтение, вставка	чтение	чтение
Иванов	чтение	чтение	

Рис. 1 Модель безопасности на основе матрицы доступа

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), которым принадлежат права владения над соответствующими объектами 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 будет иметь 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 произволь-

ные 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, $S \subset O$) 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 $S \times O \times R$. Любая ячейка матрицы M содержит набор прав субъекта S к объекту 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 схематич-

ное изображение канала утечки в виде разрешенного доступа от имени разных пользователей к одному и тому же объекту будет выглядеть следующим образом: S_i -> (Запись в момент времени t_1) -> O и S_j -> (Чтение в момент времени t_2) -> O, где $i \neq j$ и $t_1 < t_2$. 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 становиться 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 Харриона-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-условных систем, 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 самом 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 общей сложности 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.

В формальную теорию защиты информации вводится понятие монитора безопасности. Концепция монитора безопасности обращений является достаточно естественной формализацией некого механизма, реализующего разграничение доступа в системе. Монитор безопасности обращений представляет собой фильтр, который разрешает или запрещает доступ, основываясь на установленных в системе правилах разграничения доступа. Монитор безопасности обращений удовлетворяет следующим свойствам:

1. Ни один запрос на доступ субъекта к объекту не должен выполняться в обход монитора;
2. Работа монитора должна быть защищена от постороннего вмешательства;
3. Представление монитора должно быть достаточно простым для возможности верификации корректности его работы.

Несмотря на то, что концепция монитора безопасности обращений является абстракцией, перечисленные свойства справедливы и для программных или аппаратных модулей, реализующих функции монитора обращений в реальных системах.

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 случаенабор привилегий зависит от реализации СУБД (определяется производителем). Выделяются 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 привилегии 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 эти 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 полеCOST отношения Tab1. Тогда операция назначения привилегий пользо- вателю 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 пользователямбез 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, рассматриваемое в следующем пункте.

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.

3.4 Мандатное разграничение доступа

К 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 классификации, 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 (принципи-

ально 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 MAC 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);
2. *-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 объек-

там. 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 A к субъектам и объектам уровня B и от субъектов и объектов уровня B к субъектам и объектам уровня C, то она может передаваться от субъектов и объектов уровня A к субъектам и объектам уровня C). 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 не- достатки 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.

3.5 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, основанной 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.

4. Обеспечение целостности данных

Под целостностью данных понимают соответствие информационной модели предметной области, т.е. данных, хранимых в базе данных, объектам реального мира и их взаимосвязям в каждый момент времени. Любое изменение в предметной области, значимое для построенной модели, должно отражаться в базе данных, и при этом должна сохраняться однозначная интерпретация информационной модели в терминах предметной области. Целостность БД не гарантирует достоверности содержащейся в ней информации, но обеспечивает по крайней мере правдоподобность этой информации, отвергая заведомо невероятные, невозможные значения. Таким образом, не следует путать целостность БД с достоверностью данных. Достоверность (или истинность) есть соответствие фактов, хранящихся в БД, реальному миру. Контроль целостности данных это способность СУБД или КС в целом обеспечить неизменность данных (данные, хранящиеся в системе, не отличаются в семантическом отношении от данных в исходных документах) в условиях случайного и (или) преднамеренного искажения (разрушения) или, иначе, под целостностью данных подразумевает отсутствие ненадлежащих изменений.

4.1 Принципы обеспечения целостности данных

Понятие «ненадлежащее изменение» введено Д. Кларком и Д. Вильсоном: ни одному пользователю КС, в том числе и авторизованному, не должны быть разрешены такие изменения данных, которые повлекут за собой их разрушение или потерю. В работах Кларка и Вильсона определены девять абстрактных теоретических принципов, выполнение которых позволит обеспечить целостность данных:

- корректность транзакций;
- авторизация пользователей;
- минимизация привилегий;
- разграничение функциональных обязанностей;
- аудит произошедших событий;
- объективный контроль;
- управление передачей привилегий;

- эффективное применение механизмов защиты;
- простота использования защитных механизмов.

По первому принципу данные могут изменяться только посредством «корректных» транзакций. Прямое (произвольным образом) изменение данных не допускается. В свою очередь корректность транзакций должна быть некоторым способом доказана. Второй принцип гласит, что изменение данных может осуществляться только авторизованными пользователями, имеющими определенные привилегии. Минимальность привилегий подразумевает, что пользователи (в конечном счете, субъекты) должны быть наделены теми и только теми привилегиями, которые минимально необходимы для выполнения тех или иных действий. Аудит произошедших событий (включая возможность восстановления полной картины происшедшего) является превентивной мерой в отношении потенциальных нарушителей и позволяет восстановить данные в случае их повреждения. Разграничение функциональных обязанностей подразумевает организацию работы с данными таким образом, что в каждой из ключевых стадий, составляющих единый критически важный с точки зрения целостности процесс, необходимо участие различных пользователей. Этим гарантируется, что один пользователь не может выполнить весь процесс целиком (или даже две его стадии) с тем, чтобы нарушить целостность данных. Принцип объективного контроля также является одним из краеугольных камней политики контроля целостности. Суть данного принципа заключается в том, что контроль целостности данных имеет смысл лишь тогда, когда эти данные отражают реальное положение вещей. Очевидно, что нет смысла заботиться о целостности данных, связанных с размещением боевого арсенала, который уже отправлен на переплавку. В связи с этим Кларк и Вильсон указывают на необходимость регулярных проверок, целью которых является выявление возможных несоответствий между защищаемыми данными и объективной реальностью, которую они отражают. Управление передачей привилегий необходимо для эффективной работы всей политики безопасности. Если схема назначения привилегий неадекватно отражает организационную структуру предприятия или не позволяет администраторам безопасности гибко манипулировать ею для обеспечения эффективности производственной деятельности, защита становится тяжким бременем и провоцирует попытки обойти ее там, где она мешает «нормальной» работе. В основу восьмого принципа контроля целостности заложен ряд идей, призванных обеспечить эффективное применение имеющихся механизмов обеспечения безопасности. На практике зачастую оказывается, что предусмотренные в системе механизмы безопасности или некорректно

используются, или полностью игнорируются системными администраторами. Простота использования защитных механизмов подразумевает, что самый безопасный путь эксплуатации системы будет также наиболее простым, и наоборот, самый простой - наиболее защищенным.

На практике наиболее часто употребляются две модели обеспечения целостности данных, модель целостности Кларка-Вильсона и модель Биба.

4.2 Модель Кларка-Вильсона

Модель целостности Кларка-Вильсона была предложена в 1987 г. как результат анализа практики бумажного документооборота в коммерческих компаниях, эффективной с точки зрения обеспечения целостности информации. Модель Кларка-Вильсона является описательной и не содержит каких бы то ни было строгих математических конструкций – скорее её целесообразно рассматривать как совокупность практических рекомендаций по построению системы обеспечения целостности в КС. Основные понятия данной модели – это корректность транзакций и разграничение функциональных обязанностей. Модель задает правила функционирования КС и определяет две категории данных и два класса операций над ними. Все содержавшиеся в системе данные подразделяются на контролируемые и на неконтролируемые элементы данных. Целостность первых поддерживается, а целостность вторых никак не контролируется. Введём следующие обозначения:

- S – множество субъектов;
- D – множество данных в автоматизированной системе (множество объектов);
- CDI (Constrained Data Items) – данные, целостность которых контролируется;
- UDI (Unconstrained Data Items) – данные, целостность которых не контролируется;

При этом $D = CDI \cup UDI$, $CDI \cap UDI = \varnothing$.

- TP (Transformation Procedure) – процедура преобразования, т.е. компонент, который может инициировать транзакцию – последовательность операций, переводящую систему из одного состояния в другое;
- IVP (Integrity Verification Procedure) – процедура проверки целостности CDI.

Правила модели Кларка-Вильсона:

1. В системе должны иметься IVP, способные подтвердить целостность любого CDI. Примером IVP может служить механизм подсчёта контрольных сумм;

2. Применение любой TP к любому CDI должно сохранять целостность этого CDI;
3. Только TP могут вносить изменения в CDI;
4. Субъекты могут инициировать только определённые TP над определёнными CDI. Данное требование означает, что система должна поддерживать отношения вида (s, t, d), где $s \in S$, $t \in TP$, $d \in CDI$. Если отношение определено, то субъект s может применить преобразование t к объекту d;
5. Должна быть обеспечена политика разделения обязанностей субъектов – т. е. субъекты не должны изменять CDI без вовлечения в операцию других субъектов системы;
6. Специальные TP могут превращать UDI в CDI;
7. Каждое применение TP должно регистрироваться в специальном CDI. При этом:
   - данный CDI должен быть доступен только для добавления информации;
   - в данный CDI необходимо записывать информацию, достаточную для восстановления полной картины функционирования системы;
8. Система должна распознавать субъекты, пытающиеся инициировать TP;
9. Система должна разрешать производить изменения в списках авторизации только специальным субъектам (например, администраторам безопасности). Данное требование означает, что тройки (s, t, d) могут модифицировать только определённые субъекты;

Безусловными достоинствами модели Кларка-Вильсона являются её простота и лёгкость совместного использования с другими моделями безопасности.

4.3 Модель Биба

Модель Биба была разработана в 1977 году как модификация модели Белла-ЛаПадулы, ориентированная на обеспечение целостности данных. Аналогично модели Белла-ЛаПадулы, модель Биба использует решётку классов безопасности, трактуемых в ней как решетку классов целостности.

Базовые правила модели Биба формулируются следующим образом:

1. Простое правило целостности (Simple Integrity, SI). Субъект с уровнем целостности XS может читать информацию из объекта с

уровнем целостности XO тогда и только тогда, когда XO преобладает над XS.

2. * - свойство (star-integrity). Субъект с уровнем целостности XS может писать информацию в объект с уровнем целостности XO тогда и только тогда, когда XS преобладает над XO.

Для первого правила существует мнемоническое обозначение No Read Down (NRD) - доступ на чтение дается, если уровень целостности (безопасности) объекта не ниже (а также включает в себя) уровень целостности (безопасности) субъекта, а для второго No Write Up (NWU) - доступ на запись дается, если уровень целостности (безопасности) субъекта не выше (а также включает в себя) уровня целостности (безопасности) объекта. Следовательно, состояние системы будет целостным тогда и только тогда, когда оно безопасно по чтению и записи.

Отдельного комментария заслуживает вопрос, что именно понимается в модели Биба под уровнями целостности. Действительно, в большинстве приложений целостность данных рассматривается как некое свойство, которое либо сохраняется, либо не сохраняется – и введение иерархических уровней целостности может представляться излишним. В действительности уровни целостности в модели Биба стоит рассматривать как *уровни достоверности*, а соответствующие информационные потоки – как передачу информации из более достоверной совокупности данных в менее достоверную и наоборот. То есть, модель Биба основывается на следующих допущениях: чем выше уровень безопасности объекта, тем выше его достоверность и чем выше уровень безопасности субъекта, тем более достоверную информацию он может вносить в систему.

Формальное описание модели Биба полностью аналогично описанию модели Белла-ЛаПадулы. К достоинствам модели Биба следует отнести её простоту, а также использование хорошо изученного математического аппарата. В то же время модель сохраняет все недостатки, присущие модели Белла-ЛаПадулы.

4.4 Совместное использование моделей безопасности

В реальных автоматизированных системах редко встречаются системы защиты, ориентированные исключительно на обеспечение конфиденциальности или исключительно на обеспечение целостности информации. Как правило, система защиты должна сочетать оба механизма – а значит, при построении и анализе этой системы будет необходимым совместное использование нескольких формальных моделей безопасности. Рассмотрим в качестве примера возможные варианты совместного использования моделей Белл-ЛаПадулы и Биба.

1. Две модели могут быть реализованы в системе независимо друг от друга. В этом случае субъектам и объектам независимо присваиваются уровни секретности и уровни целостности;
2. Возможно логическое объединение моделей за счёт выделения общих компонентов. В случае моделей Биба и Белла-ЛаПадулы таким общим компонентом является порядок разграничения доступа в пределах одного уровня секретности;
3. Возможно использование одной и той же решётки уровней как для секретности, так и для целостности. При этом субъекты и объекты с высоким уровнем целостности будут располагаться на низких уровнях секретности, а субъекты и объекты с низким уровнем целостности – на высоких уровнях секретности.

Если брать в расчет КС, то последняя реализация позволяет, например, разместить системные файлы на нижнем уровне иерархии, что обеспечит их максимальную целостность, не акцентируя внимание на излишней в данном случае секретности.

4.5 Операторы языка SQL задания ограничений целостности

Как мы уже знаем из курса «Модели данных и СУБД», существуют следующие виды ограничений:

1. Ограничители значений (domain constraints);
2. Ограничители ключей (key constraints);
3. Ограничители записи (entity constraints);
4. Ссылочная целостность (Declarative Referential Integrity, DRI).

Ограничители ключей задаются посредством запрета неопределенных Null значений для ключевых полей (по неопределенному значению невозможно произвести идентификацию записи) и требованием уникальности значения ключевого поля. При сравнении неопределенных значений не действуют стандартные правила сравнения: одно неопределенное значение никогда не считается равным другому неопределенному значению. В SQL во фразе WHERE для выявления равенства значения некоторого атрибута неопределенному значению применяют специальные предикаты:

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

Уникальность значений любого поля задается с помощью оператора UNIQUE. Для объявления уникальности совокупности полей применяется конструкция вида:

```
UNIQUE (<поле1>, <поле2>, …)
```

Для объявления первичного ключа используется оператор PRIMARY KEY. Дополнительно, с помощью специфичных для каждой СУБД операторов, может устанавливаться автонумерация ключевого поля.

Ограничители записи также задаются посредством запрета неопределенных значений, но уже не для ключевых, но важных в семантическом смысле полей. Тогда, по крайней мере, в отношении, имеющим хотя бы одно обязательное к заполнению поле, не будет полностью пустых кортежей.

Ссылочная целостность обеспечивает поддержку непротиворечивого состояния БД (согласованного состояния внешних ключей) в процессе модификации данных, а также при выполнении операций добавленияили удаления записей. В SQL для этого вводится оператор FOREIGN KEY

```
FOREIGN KEY(<список полей>) REFERENCES <имя табли-
цы>(<список полей>) ON DELETE CASCADE ON UPDATE CASCADE.
Например: FOREIGN KEY(client_id) REFERENCES clients(id) ON
DELETE CASCADE
```

Ограничители ключей, записи и ссылочная целостность определяют правила работы СУБД с реляционными структурами данных. Но с другой стороны, эти аспекты никак не касаются содержания базы данных. Для определения некоторых ограничений, которые связаны с содержани-ем базы данных, вводятся ограничители значений. Ограничители значе- ний задаются путем определения типа поля (домена), задания условия на значение, задания списка возможных значений и определения значения по умолчанию. Значение по умолчанию задается с помощью оператора

```
DEFAULT(<значение или выражение>) {FOR <имя поля>},
```

а ограничители значений вводятся с помощью оператора CHECK(<условие>). Внутри CHECK могут использоваться операторы сравнения, функции IN, BETWEEN, LIKE и другие функции SQL. Например:

```
DEFAULT(GetDate()) for date
CHECK(price between 0 and 100000)
CHECK(passport like 'MP%20')
CHECK(город in('Минск','Москва','Киев')) или
CHECK(город in(select capital from capitals))
```

Все рассмотренные выше ограничители целостности можно напрямую задать в инструкции определения таблицы CREATE TABLE, например (для MS SQL Server):

```
CREATE TABLE Publishers (Publisher_ID INTEGER IDENTI-
TY(1,1) NOT NULL PRIMARY KEY, Publisher VARCHAR(100) NOT
NULL);
```

```
CREATE TABLE Books(Book_ID INTEGER IDENTITY(1,1) NOT NULL
PRIMARY KEY, Name VARCHAR(100) NOT NULL, ISBN VARCHAR(14)
UNIQUE NULL, PublisherID INTEGER NULL FOREIGN KEY REFER-
ENCES Publishers(Publisher_ID), Publ_Year SMALLINT DE-
FAULT(Year(GetDate()) CHECK(Publ_Year>=1960 AND
Publ_Year<=YEAR(GetDate())), Pages SMALLINT NULL
CHECK(Pages BETWEEN 5 AND 10000))
```

В приведенных инструкциях оператор IDENTITY(1,1) определяет авто нумерацию записей. Первый его параметр позволяет задать начальное значение, второй – приращение.

Для анализа ошибок целесообразно именовать все ограничения, особенно если таблица содержит несколько ограничений одного типа, так как имя ограничения выводится в сообщении нарушения заданного ограничения. Для именования ограничений используется ключевое слово CONSTRAINT, после которого следует уникальное имя ограничения, затем тип ограничения и его тело, например:

```
CONSTRAINT PK_BOOKS PRIMARY KEY(Book_ID)
CONSTRAINT DEF_PYEAR DEFAULT(Year(GetDate()) FOR Publ_Year
CONSTRANT CH_PAGES CHECK(PAGES>=5 AND PAGES<=10000)
```

Именованные ограничения можно задать непосредственно в инструкции `CREATE TABLE` (через запятую после определения полей) или с помощью оператора изменения таблицы:

```
ALTER TABLE <имя таблицы> ADD CONSTRANT <имя ограничения>
<тело ограничения>
```

Удалить именованное ограничение можно с помощью оператора:

ALTER TABLE <имя таблицы> DROP CONSTRAINT <имя ограничения> {CASCADE|RESTRICT}

5. Восстановление целостного состояния БД

Возможность восстановления обеспечивается за счёт накопления избыточной информации. Во-первых, периодически выполняется резервное копирование базы данных. Во-вторых, информация о транзакциях, выполнявшихся в промежутке времени между двумя последовательными копированиями, сохраняется в специальном файле-журнале изменений базы данных. В-третьих, создаются контрольные точки, необходимыедля синхронизации процесса восстановления и уменьшения объема по- иска в файле журнала. Эта информация используется для восстановления согласованного состояния БД после любого сбоя. Поскольку транзакция является ключевым понятием при организации восстановления целостного состояния БД, вначале кратко рассмотрим основные свойства транзакций.

5.1 Понятие транзакции

Под транзакцией понимается выполняемая от имени определённого пользователя или процесса последовательность операций манипулирования данными, переводящая БД из целостного начального состояния в целостное конечное состояние. При этом промежуточные состояния БД не должны быть обязательно целостными. Транзакции присущи следующие свойства, известные как требования ACID (Atomicity, Consistency, Isolation и Durability):

Атомарность и согласованность. Транзакция как единица работы с данными обладает свойством атомарности (либо выполняются все предусмотренные транзакцией обновления данных, либо не выполняется ни одно) и согласованности (целостное начальное состояние БД преобразуется в целостное конечное состояние). Транзакция не может быть выполнена частично. Если в процессе обработки транзакции возникает какой-либо сбой, то все выполненные до этого момента обновления дан- ных должны быть отменены.

Долговечность. Транзакция обладает свойством долговечности: обновления данных, произведённые успешно завершившейся транзакцией, становятся постоянными и не могут быть утеряны или отменены ни при каких обстоятельствах. Говорят, что транзакция завершилась успешно, если в процессе исполнения не возникла какая-либо аварийная ситуация, и все проверки ограничений целостности дали положительный результат. В этом случае все произведённые обновления данных сохраняются (фиксируются) в постоянной памяти. Система гарантирует фиксацию обновлений успешно завершившейся транзакции даже в том случае, если

в следующий момент времени произойдёт системный сбой. В случае неуспешного завершения транзакции по любой причине система гарантирует отсутствие каких-либо следов её работы во внешней памяти.

Изолированность. Многопользовательская система, как правило, одновременно поддерживает сеансы работы нескольких пользователей. Некоторые из них могут пытаться получить доступ к одним и тем же данным в одном и том же интервале времени. В связи с этим возникает проблема управления одновременным доступом к БД. Должна быть выбрана такая стратегия управления, которая обеспечивала бы взаимную изоляцию пользователей. Два одновременно работающих пользователя не будут замечать друг друга, если на результаты работы любой транзакции каждого из них не будут влиять действия параллельно исполняющихся транзакций другого. Говорят, что транзакция обладает свойством изолированности, если ей недоступны промежуточные результаты других параллельно исполняющихся транзакций и её промежуточные результаты недоступны другим транзакциям. Другими словами, изолированная транзакция А может получить доступ к объекту, обрабатываемому транзакцией В только после завершения В, и наоборот.

Таким образом, понятие транзакции можно сформулировать как выполняемую от имени одного идентификатора субъекта неделимую последовательность операций над данными, обладающая свойствами атомарности, согласованности, изолированности и долговечности. В стандарте SQL определены следующие операторы управления транзакциями:

BEGIN TRANSACTION – сообщает диспетчеру транзакций о явном начале новой транзакции (неявно любая инструкция обновления данных запускается в рамках транзакции);

COMMIT TRANSACTION – сообщает диспетчеру транзакций об успешном завершении транзакции и требованию фиксации результатов во внешней памяти;

ROLLBACK TRANSACTION – сообщает диспетчеру транзакций о возникновению ошибки и требованию отката произведенных изменений.

Поддержание согласованности обеспечивается механизмом проверки ограничений целостности. Можно выделить два вида ограничений целостности: немедленно проверяемые и откладываемые. Немедленно проверяются ограничения целостности, затрагивающие один объект (отношение или домен). Например, не имеет смысла откладывать проверку ограничения целостности сущности или целостности значений при выполнении операции вставки кортежа в отношение. Немедленно проверяемые ограничения целостности соответствуют уровню отдельных инструкций обновления. При их нарушениях откат транзакции не нужен.

Достаточно отвергнуть соответствующую инструкцию. Откладываемые ограничения целостности - это ограничения, затрагивающие несколько отношений, или кортежей отношения. Их называют ограничениями на базу данных. Примером может служить ограничение ссылочной целостности или ограничение, затрагивающее несколько кортежей. Подобные ограничения должны (и могут) быть проверены лишь после выполнения некоторой логически замкнутой группы операций обновления. Таким образом, откладываемые ограничения соответствуют уровню транзакции. Их нарушение вызывает автоматическую отмену обновлений, т.е. замену COMMIT на ROLLBACK.

Поддержание изолированности - одна из приоритетных задач диспетчера транзакций. Изолированность достигается, если объект, обрабатываемый транзакцией, не изменяется непредсказуемо другими транзакциями, исполняемыми в том же интервале времени. Существует два основных метода управления параллельностью, позволяющих организовать безопасное одновременное выполнение транзакций: метод блокировки и метод временных меток. Оба этих метода относятся к пессимистическому подходу управления транзакциями, когда откладывается выполнение транзакций, способных в будущем войти в конфликт с другими транзакциями. Оптимистические методы основываются на предположении, что вероятность конфликта невысока и поэтому они допускают асинхронное выполнение транзакций. Проверка конфликтов производится на этапе фиксации транзакций.

5.2 Принципы восстановления данных

В основе восстановления данных лежат следующие два принципа:

1. Результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных;

2. Результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных.

Это, собственно, и означает, что восстанавливается последнее по времени согласованное состояние базы данных. Процедуры восстановления зависят от типа сбоя.

Перечислим ситуации, в результате которых база данных может оказаться в несогласованном состоянии:

Локальный сбой - это аварийное прекращение транзакции. Причиной может быть, например, попытка деления на ноль или нарушение ограничений целостности, или тупиковая ситуация. В этот же ряд следует поставить явное завершение транзакции оператором ROLLBACK. Если транзакция выполняла обновление данных, то состояние БД в момент

локального сбоя окажется несогласованным. Для восстановления целостности данных необходимо устранить изменения данных, произведённые прерванной транзакцией - произвести индивидуальный откат транзакции.

Мягкий сбой системы может произойти, например, вследствие аварийного отключения питания или при возникновении неустранимого сбоя процессора и т.п. В этом случае теряется содержимое оперативной памяти. Аварийно прерываются все существующие транзакции. Могут оказаться не зафиксированными в БД результаты транзакций, завершившихся оператором COMMIT. При перезагрузке системы должен быть выполнен откат всех не завершившихся к моменту сбоя транзакций. Закончившиеся, но незафиксированные транзакции должны быть автоматически исполнены повторно.

Жёсткий сбой – это физическое разрушение базы данных. Ситуация весьма маловероятная, но её последствия для организации владельца данных всегда катастрофические. Поэтому система должна быть в состоянии восстановить базу данных даже в этом случае.

В любом случае для восстановления согласованного состояния БД необходима некоторая информация. Рассмотрим, какая именно информация необходима для восстановления и как она должна использоваться. Все фрагменты БД, которые обрабатываются транзакциями, считываются системой в рабочие буферы базы данных в оперативной памяти. Транзакции выполняют все обновления данных в этих буферах. Таким образом, состояние рабочих буферов отражает текущее состояние той части базы данных, которая доступна для действующих транзакций.

Если происходит локальный сбой (или транзакция завершается оператором ROLLBACK), то необходимо восстановить состояние рабочих буферов на момент начала транзакции. Для отмены обновлений достаточно иметь полную информацию об операциях обновления, выполнявшихся транзакцией, а именно: тип операции, объект обновления, детали обновления. Тогда, проделав в обратном порядке обратные по смыслу операции, можно восстановить состояние изменявшихся транзакцией буферов БД.

Таким образом, для восстановления согласованного состояния БД,при программном откате транзакции или после локального сбоя должна сохраняться полная информация обо всех операциях обновления, выполненных транзакцией.

Эта информация хранится в специальном системном файле - журнале регистрации транзакций. В журнал для каждой транзакции записываются её уникальный идентификатор и детали всех операций обновления дан-

ных, выполненных транзакцией. Кроме того, каждой транзакции, завершившейся штатно (т.е. операторами COMMIT или ROLLBACK), сопоставляется запись об окончании транзакции.

Но в случае мягкого сбоя этого мало, так как мягкий сбой приводит к потере содержимого рабочих буферов БД. Нужно ещё располагать состоянием БД на какой-то момент времени t0 и знать, какие транзакции в этот момент существовали в системе. Тогда, если в момент $t_1 > t_0$ произойдёт мягкий сбой, то для восстановления системы достаточно будет загрузить в оперативную память зафиксированное в момент t0 состояние рабочих буферов БД, проанализировать записи журнала транзакций, сделанные в интервале $[t_0, t_1]$, и восстановить успешно завершённые или откатить незавершённые транзакции. С целью минимизации обращений к устройствам внешней памяти программы формируют записи файлов (логические) в своих рабочих буферах ОЗУ. Сброс записей во внешнюю память происходит автоматически, когда буфер заполняется. Незаполненный буфер можно вытолкнуть принудительно.

Итак, для обеспечения возможности восстановления системы после мягкого сбоя состояние рабочих буферов БД должно периодически фиксироваться во внешней памяти, и каждая запись журнала транзакций должна иметь временную метку. Однако и этого недостаточно. Нужно ещё, чтобы все записи журнала от транзакций, завершённых до момента сбоя, находились во внешней памяти. Буфер журнала имеет размер, достаточный для размещения записей от многих транзакций. Велика вероятность того, что к моменту мягкого сбоя он не будет сброшен во внешнюю память автоматически, и часть записей журнала будет утеряна, поэтому буфер журнала должен принудительно выталкиваться во внешнюю память при завершении транзакции. Только после этого транзакция считается закончившейся. Обновления, совершённые транзакцией, не могут попасть во внешнюю память раньше, чем соответствующая ей запись журнала. Это правило составляет суть протокола предварительной записи в журнал.

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 долж- на 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. Следует отметить, что скорость записи вфайл журнала может оказаться одним из важнейших факторов, опреде- ляющих общую производительность работы с БД.

Посмотрим теперь, как можно восстанавливать состояние базы данных, если в системе поддерживается журнал транзакций и протокол предварительной записи.

Индивидуальный откат транзакции. Индивидуальный откат производится либо по явно заданному оператору ROLLBACK, либо вследствие локального сбоя. Для осуществления отката создаётся список записей журнала от данной транзакции. Элементы списка размещены в порядке, обратном хронологическому. Список последовательно просматривается и для каждой записи выполняется противоположная по смыслу операция, восстанавливающая предыдущее состояние объекта базы данных. Начальным состоянием для процедуры отката является состояние буферов БД в момент прекращения транзакции. С точки зрения системы эта процедура является транзакцией. Поэтому обратные операции также регистрируются в журнале. Это "перестраховка" на случай мягкого сбоя системы в процессе отката. Имея эти записи, система сможет «доткатить» транзакцию, откат которой был прерван.

Восстановление после мягкого сбоя. Рабочие буферы базы данных, в отличие от буфера журнала, не выталкиваются во внешнюю память при каждом успешном завершении транзакции. Реально исполнение опера-

тора COMMIT сводится тому, что становится невозможной отмена проделанных транзакций изменений. Сам же изменённый объект может ещё долго существовать только в рабочем буфере БД. Может случиться так, что в системе произойдёт сбой после успешного выполнения COMMIT, но до того, как обновлённый транзакцией объект попадёт во внешнюю память. В соответствии с принципом долговечности транзакции система должна при перезагрузке установить эти обновления в БД, несмотря на то, что их уже нет в буфере оперативной памяти.

Кроме того, система может временно помещать в БД промежуточные результаты транзакций, если необходимо освободить место в буферах. При нормальной работе эти несогласованные изменения по завершении внесшей их транзакции отвергаются. Поскольку мягкий сбой прерывает все транзакции, промежуточные результаты оказываются "зафиксированными". При перезагрузке система должна откатить все прерванные транзакции и выполнить повторно все успешно завершившиеся, но не зафиксированные в БД. При этом транзакции, прерванные или завершившиеся оператором ROLLBACK до мягкого сбоя, в процессе восстановления не участвуют.

Текущее состояние БД в момент сбоя не может быть использовано как опорное для восстановления, поскольку неизвестно, к какому момен-ту времени оно относится. Для того чтобы иметь такое опорное состоя- ние БД, используется механизм создания контрольных точек. В течение некоторого интервала времени СУБД дожидается завершения очередных операций обновления во всех транзакциях и не допускает запуска новых операций. Исполнение действующих транзакций приостанавливается. В момент принятия контрольной точки, когда все транзакции приостановлены, в журнале создаётся специальная запись контрольной точки, содержащая список всех транзакций, существующих в системе. Затем во внешнюю память принудительно выталкивается содержимое буфера журнала и рабочих буферов БД. По завершении этого процесса исполнение приостановленных транзакций продолжается. Отметим, что транзакции, прерванные или завершившиеся оператором ROLLBACK до мягкого сбоя, в процессе восстановления не участвуют.

В процессе восстановления выполняются следующие действия.

- Создаётся два списка транзакций: отменяемых (UNDO) и исполняемых повторно (REDO). В список UNDO включаются все транзакции, указанные в записи контрольной точки. Список REDO остаётся пустым.
- Выполняется анализ записей журнала регистрации, начиная с записи контрольной точки.

- Если обнаружена запись о начале транзакции T, то эта транзакция добавляется в список UNDO.
- Если обнаружена запись о завершении транзакции T оператором COMMIT, то эта транзакция добавляется в список REDO.
- Журнал регистрации просматривается от конца до записи контрольной точки, и отменяются транзакции из списка UNDO.
- Журнал регистрации просматривается от записи контрольной точки до конца, и выполняются повторно транзакции из списка REDO.

По окончании этой процедуры система готова к работе.

Восстановление после жесткого сбоя. При жёстком сбое физическая база данных оказывается разрушенной. Поэтому для восстановления необходимо иметь её резервную копию. Обычно резервное копирование БД выполняется по факту переполнения журнала транзакций. Для этого в файле журнала устанавливается так называемая «жёлтая зона», по достижении которой запуск новых транзакций не производится. Система дожидается окончания всех существующих транзакций. После этого рабочие буферы журнала и базы данных выталкиваются во внешнюю память. Созданное таким образом состояние БД копируется на резервный носитель, а файл журнала очищается. Может быть также создана резервная копия журнала.

При восстановлении после жёсткого сбоя восстанавливается состояние БД на момент последнего копирования, а затем по текущему журналу регистрации повторно исполняются все транзакции, успешно завершившиеся до момента сбоя. Поскольку жёсткий сбой может не сопровождаться потерей буферов, после восстановления можно даже продолжить исполнение не завершившихся к моменту сбоя транзакций. Тем не менее, все незавершённые транзакции обычно откатываются.

5.3 Методы восстановления

Рассмотрим два метода восстановления, которые могут быть применены в случае, т.е. когда БД не была полностью разрушена, но лишь утратила согласованное состояние. Метод отложенного обновления и метод немедленного обновления отличаются друг от друга способом внесения обновлений во внешнюю память.

Метод отложенного обновления. При использовании этого метода обновления не заносятся в БД до тех пор, пока не начнется фаза фиксации транзакции. Если выполнение транзакции будет прекращено до достижения этой точки, никаких изменений в БД выполнено не будет, по-

этому не потребуется и их отмена. Однако в таком случае может потребоваться повторный прогон уже завершившихся транзакций, поскольку их результаты могли еще не достичь внешней памяти. Протокол восстановления будет состоять из следующих действий.

При запуске транзакции в журнал помещается запись: *Начало транзакции*. При выполнении операции вставки или обновления данных в файл журнала записываются новые значения всех измененных элементов данных (отметим, запись значений элементов данных до обновления не требуется). Реально запись изменений в саму БД не производится. Когда транзакция достигает своей конечной точки, в журнал помещается запись: *Транзакция завершена*. Все записи журнала по данной транзакции выводятся во внешнюю память, после чего выполняется фиксация внесенных транзакцией изменений. Для внесения действительных изменений в БД используется информация, помещенная в файл журнала.

В случае отмены выполнения транзакции, записи журнала по данной транзакции игнорируются, и произведенные изменения не фиксируются во внешней памяти. В соответствии с принципом WAL записи журнала по выполненной транзакции сохраняются во внешней памяти до того, как результаты транзакции будут зафиксированы. Поэтому, если отказ БД произойдет повторно в процессе действительного выполнения обновлений в БД, размещенные в журнале сведения будут сохранены и требуемые обновления можно будет выполнить позже. В случае отказа файл журнала анализируется с целью выявления транзакций, которые находились в процессе выполнения в момент отказа, начиная с последней строки. Файл журнала просматривается в обратном направлении вплоть до записи о последней выполненной контрольной точке. Все транзакции, для которых в файле журнала присутствуют записи *Начало транзакции*и *Транзакция завершена*, должны быть выполнены повторно. Процедура повторного прогона транзакций выполняет все операции записи в БД, используя информацию о состоянии элементов данных после обновления, содержащуюся в записях журнала по данной транзакции, причем в том порядке, в каком они были записаны в файле журнала. Если операции записи уже были успешно завершены до возникновения отказа, это не окажет никакого влияния на состояние элементов данных, поскольку они не могут быть испорчены, если будут записываться еще раз. Следовательно, данный метод гарантирует, что будут обновлены любые элементы данных, которые не были корректно обновлены до момента отказа.

Любая транзакция, для которой в файле журнала присутствуют записи *Начало транзакции* и *Отмена транзакции*, просто игнорируется, по-

скольку никаких реальных обновлений информации в БД по ней не выполнялось, а значит, не требуется и реального выполнения их отката. Если в процессе восстановления возникнет другой системный сбой, записи файла журнала могут быть использованы для восстановления БД еще раз. В таком случае не имеет значения, сколько раз каждая из строк журнала была использована для повторного внесения изменений в БД.

Метод немедленного обновления. При использовании этого протокола все изменения вносятся в БД сразу же после их выполнения в транзакции, не дожидаясь ее завершения. Помимо необходимости повторного прогона изменений, выполненных транзакциями, закончившимися до появления сбоя, в данном случае может потребоваться выполнить откат изменений, внесенных транзакциями, которые не были завершены к этому моменту.

При применении данного метода файл журнала используется с целью восстановления следующим образом. При запуске транзакции в журнал помещается запись: *Начало транзакции*. При выполнении любой операции обновления БД производится запись в файл журнала всех исходных и конечных значений всех измененных элементов данных. После выполнения записи в файл журнала производятся выполнение обновлений в буфере БД. Непосредственно в БД изменения будут внесены при очередной выгрузке буферов БД во внешнюю память. Когда транзакция завершает свое выполнение, в файл журнала заносится запись: *Транзакция завершена*.

В данном методе очень важно, чтобы в файл журнала все записи помещались до внесения соответствующих изменений в БД. Если изменения вначале будут внесены в БД и сбой в системе возникнет до размещения информации об этом в файл журнала, то менеджер восстановления не будет иметь возможности отменить (или повторить) данную опера- цию. При использовании протокола предварительной записи журналаменеджер восстановления всегда сможет безопасно предположить, что если для определенной транзакции в файле журнала отсутствует запись Т*ранзакция завершена*, значит, эта транзакция была активна в момент возникновения отказа и, следовательно, должна быть отменена. Если вы- полнение транзакции было прекращено, то для отмены выполненных ею изменений используются сведения об исходных значениях всех изме- ненных элементов данных, сохраненных в файле журнала. Поскольку транзакция может выполнить несколько изменений одного и того жеэлемента, отмена обновлений выполняется в обратном порядке. Незави- симо от того, были ли результаты выполнения транзакции внесены в са- му БД, наличие в записях журнала исходных значений полей гарантиру-

ет, что БД будет приведена в состояние, отвечающее началу отмененной транзакции.

На случай повторного отказа системы процедурой восстановления предусмотрено использование файла журнала для повторного прогона или отката транзакций. Для любой транзакции Т, для которой в файле журнала присутствуют записи *Начало транзакции* и *Транзакция завер- шена,* следует выполнить ее повторный прогон, используя для внесения в БД новые значения всех обновленных полей. Для любой транзакции, для которой в файле журнала присутствует запись *Начало транзакции*, но нет записи *Транзакция завершена*, необходимо выполнить откат внесен- ных ею изменений. На этот раз из записей файла журнала извлекается информация о значении измененных полей до их изменения, что позволяет привести базу данных в стояние, которое она имела до начала данной транзакции. Операции отмены выполняются в порядке, обратном порядку их записи в файл журнала.

Метод теневых страниц. Этот метод является альтернативой описанным выше схемам восстановления, построенным на использовании файла журнала. Он предусматривает организацию на время выполнения транзакции двух страниц - текущую и теневую. Когда транзакция начинает работу, обе страницы идентичны. Теневая страница никогда не изменяется и может быть использована для восстановления БД в случае отказа системы. В ходе выполнения транзакции текущая страница используется для записи в БД всех вносимых изменений. После заверше- ния транзакции текущая таблица становится теневой. Метод теневыхстраниц имеет ряд преимуществ перед методами использования журнала транзакций: исключается нагрузка, связанная с ведением журнала транзакций, процесс восстановления происходит существенно быстрее, поскольку нет необходимости в повторном прогоне или откате операций. Однако ему свойственны и определенные недостатки: фрагментация данных и необходимость периодического выполнения процедуры сборки мусора для возвращения в систему неиспользуемых блоков памяти.

5.4 Организация восстановления данных в СУБД MS SQL Server

SQL Server предлагает на выбор три модели восстановления, в основном отличающихся использованием журнала транзакций, и пять вариантов резервного копирования. Модели восстановления следующие:

1. Простая модель. Журнал транзакций не резервируется (не создается его резервная копия во внешней памяти);

2. Модель с неполным протоколированием. Массовые операции не заносятся в журнал транзакций. Журнал транзакций резервируется;
3. Полная модель. Все транзакции заносятся в журнал. Журнал транзакций резервируется.

Резервное копирование возможно следующих видов:

- Полное. Резервируются все данные;
- Дифференцированное. Производится резервирование всех страниц данных, измененных с момента последнего полного резервного копирования;
- Журнала транзакций. Производится резервирование всех транзакций в журнале.
- Файла или файловой группы. Производится резервирование всех данных, содержащихся в файле или файловой группе.
- Файловое дифференцированное. Производится резервирование всех страниц данных, модифицированных с момента последнего резервного копирования файла или файловой группы.

Восстановление всегда начинается с использования полной резервной копии. После этого из архивов дифференцированного и транзакционного резервирования восстанавливаются все транзакции, выполненные с момента создания полной резервной копии. Модель восстановления конфигурирует настройки базы данных SQL Server таким образом, чтобы обеспечить тот тип восстановления, который необходим базе данных. Ключевые отличия между разными моделями восстановления связаны с тем, в какой мере в них задействован журнал транзакций и какие данные в нем регистрируются.

Простая модель восстановления идеально подходит тем базам данных, которым требуется обеспечение атомарности транзакций, но не обязательна поддержка их живучести. Простая модель форсирует очищение сервером баз данных журнала в контрольных точках. При этом журнал будет хранить транзакции, запись которых в базу данных еще не подтверждена; пространство же, отведенное для хранения всех остальных транзакций, освобождается для повторного использования. Так как журнал транзакций в этой модели является только временным местом хранения, отпадает потребность в его резервировании. Эта модель восстановления имеет ряд преимуществ. Во-первых, журнал транзакций имеет маленькие размеры, однако расплачиваться за это придется потерей всех транзакций, выполненных с момента последнего полного или дифференцированного резервирования. План восстановления, основанный на про-

стой модели, позволяет выполнять полное резервирование раз в неделю, а дифференцированное – в конце каждого дня недели. Полная и дифференцированные резервные копии заменяются, когда будет выполнено следующее полное резервное копирование.

Соответственно, восстановление в простой модели предполагает две операции.

1. Восстановление из последней полной резервной копии;
2. Восстановление из последней (не обязательно) одной дифференцированной резервной копии.

Полная модель восстановления предлагает наиболее грубый и надежный план восстановления. В этой модели все транзакции, в том числе и массовые операции, протоколируются в журнале. Любая системная функция, такая как создание индекса, также протоколируется. Основным преимуществом этой модели является то, что все транзакции, выполненные в базе данных, могут быть восстановлены, вплоть до момента, предшествовавшего системному сбою. К недостаткам следует отнести медленное выполнение массовых операций, очень большой размер файла журнала транзакций и более длительное время выполнение операций резервирования и восстановления журнала транзакций.

Полная модель восстановления может использовать все пять типов резервирования базы данных. Чаще всего полная модель восстановления предполагает выполнение полного резервирования дважды в неделю, а дифференцированного – в конце каждого дня. Резервирование журнала транзакций выполняется регулярно на протяжении всего дня; при этом промежутки времени между отдельными сессиями могут колебаться от нескольких часов до нескольких минут. Восстановление базы данных в этой модели предполагает выполнение следующих операций.

- Восстановление из последней полной резервной копии;
- Восстановление из последней дифференцированной резервной копии, созданной после выполнения последнего полного резервирования (если таковая имеется);
- Восстановление всех резервных копий журнала транзакций, созданных с момента последнего полного или дифференцированного резервного копирования.

Если последней созданной резервной копией была полная, то ее восстановления будет достаточно. Если последней созданной резервной копией была дифференцированная, то перед ее восстановлением следует выполнить восстановление из последней полной резервной копии.

Модель восстановления с неполным протоколированием аналогична полной модели, за исключением того, что в журнал транзакций не заносятся следующие операции:

- массовые вставки;
- инструкции DML SELECT * INTO;
- операции над особо крупными объектами WRITETEXT и UPDATETEXT;
- инструкции CREATE INDEX (включая индексированные представления).

Компромисс в производительности операций в этой модели восстановления достигается за счет того, что массовые операции не рассматриваются как транзакции. Поскольку массовые операции в данной модели восстановления не регистрируются, если сбой произошел после массовой операции, но до создания архива транзакций, массовая операция будет утеряна и в восстановлении сможет участвовать только предыдущая резервная копия журнала транзакций. Исходя из этого, если используется модель восстановления с неполным протоколированием, за каждой массовой операцией должно немедленно следовать создание резервной копии журнала транзакций.

Эта модель восстановления может оказаться полезной только в том случае, если в базе данных выполняется большое количество массовых операций и очень важно повысить их производительность. Если же производительность выполнения массовых операций в базе данных достаточно высока, то лучше остановить свой выбор на полной модели восстановления.

Модель восстановления, установленная в системной базе model, применяется ко всем создаваемым базам данных. Для установки модели восстановления используется инструкция DDL ALTER DATABASE:

```
ALTER DATABASE имя_базы_данных SET Recovery параметр
```

Допустимыми параметрами в этой инструкции являются следующие: Full, Bulk-Logged и Simple. В следующем примере модель восстановления учебной базы данных Test изменяется на полную:

```
USE Test
ALTER DATABASE Test SET Recovery Full
```

Настоятельно рекомендуется в программном коде, создающем базу данных, явно указывать модель восстановления. Текущую модель восстановления, установленную в базе данных, можно определить с помощью представления каталога sys.database:

```
SELECT name, recovery_model_desc FROM sysdatabases
```

Несмотря на то что в базе данных обычно устанавливается только одна модель восстановления, никто не запрещает переключаться между разными моделями в зависимости от состава выполняемых операций с целью оптимизации производительности и удовлетворения текущих потребностей.

Для выполнения операции резервирования базы данных имеется утилита backup.

Если не принимать в расчет множество дополнительных параметров, то общий вид команды резервного копирования будет следующим:

```
BACKUP DATABASE имя_базы_данных
TO DISK = 'путь_к_файлу'
WITH NAME = 'имя_архива'
```

Следующая инструкция резервирует базу данных Test в файл на диске и присваивает архиву имя TestBackup:

```
BACKUP DATABASE Test
TO DISK = 'e:\TestBackup.bak'
WITH NAME = 'TestBackup'
```

Журнал транзакций также подлежит резервированию. Виртуально журнал можно представить себе как последовательный список транзакций, отсортированный по дате и времени. В то же время на физическом уровне SQL Server записывает различные части физического журнала в виртуальные блоки без какого-либо определенного порядка. Все транзакции в журнале можно разделить на две группы: активные транзакции (те, которые еще не подтверждены и не записаны в файл базы данных), и неактивные транзакции, которые предшествуют самой ранней активной транзакции. Активная часть журнала не обязательно содержит только неподтвержденные транзакции, она содержит все транзакции с момента начала самой старой неподтвержденной транзакции. Всего одна очень старая неподтвержденная транзакция может сделать активной очень большую часть журнала.

Выполнение резервирования журнала транзакций практически не отличается от полного или дифференцированного резервного копирования. Инструкция T-SQL имеет следующий вид:

```
BACKUP LOG Test
TO DISK = 'e:\TestBackupLog.bak'
WITH NAME = 'TestBackupLog'
```

При резервировании журнала транзакций используются те же параметры, которые используются при резервном копировании базы данных. Журнал транзакций не может быть зарезервирован, если в базе данных используется простая модель восстановления, или в базе данных исполь-

зуется модель восстановления с неполным протоколированием, или были повреждены файлы базы данных. В любом из этих случаев следует выполнить полное резервирование базы данных.

Обычно резервное копирование базы данных выполняется по строго определенному графику, однако невозможно заранее предсказать, какое количество дифференцированных и резервных копий и копий транзакций должно быть восстановлено. Так как каждый файл резервной копии требует отдельной инструкции RESTORE, невозможно создать корректный код, не включив в него обследование базы данных msdb и создание правильной последовательности восстановления. Инструкция RESTORE может выполнять воссоздание базы из полной, дифференцированной и транзакционной резервной копии. Ее общий синтаксис приведен ниже.

```
RESTORE DATABASE|LOG имя_базы_данных
FROM устройство_резервирования
WITH FILE = номер_файла, PASSWORD = пароль,
NORECOVERY|RECOVERY
```

Для воссоздания полной или дифференциальной резервной копии используется инструкция RESTORE DATABASE; если воссоздается журнал транзакций, используется инструкция RESTORE LOG. Параметр WITH FILE позволяет задать номер восстанавливаемой резервной копии в файле или на устройстве резервирования. Параметры RECOVERY/NORECOVERY являются жизненно важными в инструкции воссоздания. При каждом запуске SQL Server автоматически проверяется журнал транзакций. При этом откатываются все неподтвержденные транзакции и доводятся до конца все подтвержденные. Этот процесс получил название воссоздания (recovery) базы данных. Таким образом, если в инструкции воссоздания указан параметр NORECOVERY, SQL Server воссоздаст журнал, не обработав при этом ни одной транзакции; если же в инструкции будет указан параметр RECOVERY, транзакции будут обработаны. Например, выполним воссоздание базы данных Test

```
Use Master
RESTORE DATABASE Test
FROM DISK = 'e:\TestBackup.bak'
WITH FILE = 1, NORECOVERY
```

Продолжаем восстановление:

```
RESTORE LOG TestBackupLog
FROM DISK = 'e:\TestBackupLog.bak'
WITH FILE = 2, NORECOVERY
```

5.5 Создание отказоустойчивых систем

Для создания отказоустойчивых систем вышеперечисленных средств восстановления недостаточно, так как требуется обеспечить непрерывный режим работы сервера. Для повышения надежности и (или) производительности дисковой подсистемы жесткие диски объединяются в RAID массивы. Наиболее часто используемыми являются RAID массивы уровней 0, 1, 5, 6 и 10.

RAID 0 (Stripe). Режим, при использовании которого достигается максимальная производительность, но не надежность. Данные равномерно распределяются по дискам массива, диски объединяются в один, который может быть размечен на несколько. Распределенные операции чтения и записи позволяют значительно увеличить скорость работы, поскольку несколько дисков одновременно читают/записывают свою порцию данных. Пользователю доступен весь объем дисков, но это снижает надежность хранения данных, поскольку при отказе одного из дисков массив обычно разрушается и восстановить данные практически невозможно.

RAID 1 (Mirror). Несколько дисков (обычно 2), работающие синхронно на запись, то есть полностью дублирующие друг друга. Повышение производительности происходит только при чтении. Самый надежный способ защитить информацию от сбоя одного из дисков. Из-за высокой стоимости обычно используется при хранении очень важных данных. Высокая стоимость обусловлена тем, что лишь половина от общей емкости дисков доступна для пользователя.

RAID 10. Иногда также называется **RAID 1+0**, так как является комбинацией двух первых вариантов (массив RAID 0 из массивов RAID 1). Имеет все скоростные преимущества RAID 0 и преимущество надежности RAID 1, сохраняя недостаток – высокую стоимость дискового массива, так как эффективная ёмкость массива равна половине ёмкости использованных в нём дисков. Для создания такого массива требуется минимум 4 диска (их число должно быть чётным).

RAID 5. Массив, также использующий распределенное хранение данных аналогично RAID 0 (и объединение в один большой логический диск) плюс распределенное хранение кодов четности для восстановления данных при сбоях. Возможно как одновременное чтение, так и запись. Плюсом этого варианта является то, что доступная для пользователя емкость массива уменьшается на емкость лишь одного диска, хотя надежность хранения данных ниже, чем у RAID 1. По сути, является компромиссом между RAID 0 и RAID 1, обеспечивая достаточно высокую скорость работы при неплохой надежности хранения данных. Минимальное

количество дисков для такого массива – 3. При отказе одного диска из массива данные могут быть восстановлены без потерь в автоматическом режиме, хотя сам процесс восстановления происходит с полной нагрузкой на рабочие винчестеры, что при перегреве может привести к полному выходу тома из строя.

RAID 6. RAID 6 отличается от RAID 5 тем, что в каждом ряду дан- ных (по-английски stripe) имеет не один, а два блока контрольных сумм. Контрольные суммы – «многомерные», т.е. независимые друг от друга, поэтому даже отказ двух дисков в массиве позволяет сохранить исходные данные. Вычисление контрольных сумм по методу Рида-Соломона требует более интенсивных по сравнению с RAID 5 вычислений, поэтому раньше шестой уровень практически не использовался. Сейчас он поддерживается многими продуктами, так как в них стали устанавливать специализированные микросхемы, выполняющие все необходимые математические операции. Согласно некоторым исследованиям, восстановление целостности после отказа одного диска на томе RAID 5, составленном из дисков большого объема (500 гигабайт и выше), в 5% случаев заканчивается утратой данных. Другими словами, в одном случае из двадцати во время регенерации массива RAID 5 на диск резерва возможен выход из строя второго диска. RAID 6 и был призван решить указанный недостаток RAID 5.

Для предотвращения простоя сервера во время восстановления данных используется технология создания резервных серверов (Database Mirroring). Она позволяет защититься от незапланированного простоя, вызванного отказом сервера или сбоем базы данных и, как следует из ее названия, обеспечивает отказоустойчивость на уровне базы данных. Данную технологию можно использовать для одной или нескольких баз данных находящихся на одном и том же экземпляре SQL Server. При ее активации резервная база данных будет всегда обновляться текущими транзакциями, производимыми на основном сервере баз данных. При этом, что очень важно, Database Mirroring оказывает минимальное влияние на производительность.

Database Mirroring обеспечивает защиту от сбоев и дисковой подсистемы, так как и основной и резервный серверы имеют свои собственные дисковые подсистемы. Database Mirroring реализована с помощью трех систем: основной сервер, резервный сервер и сервер-свидетель.

Основной сервер (primary server) обеспечивает сервис баз данных в текущее время. По умолчанию, все клиентские подключения происходят именно к основному серверу. Задача резервного сервера (secondary server) состоит в поддержке резервной копии основного сервера. Свиде-

тель (witness), действующий независимо, отвечает за то, какой из серверов в настоящий момент является основным.

Зеркалирование баз данных работает следующим образом. Когда клиентское приложение записывает транзакцию на основной сервер, то перед изменением файла данных происходит запись изменений в журнал транзакций. Затем эти записи журнала транзакций отправляются на резервный сервер в журнал транзакций зеркальной базы данных. После того как резервный сервер зафиксирует эти записи в журнале транзакций он отправляет подтверждение основному серверу. Это позволяет обеим системам знать, что запись была принята и оба журнала транзакций имеют одни и те же данные. В случае операции фиксации, основной сер- вер ждет подтверждения от резервного сервера и только после этого от- правляет ответ клиентскому приложению, говоря о том, что данная опе- рация завершена. Если приложение теряет подключение к основномусерверу, система сделает еще одну попытку подключения к основному серверу. Если подключения не произойдет, следующая попытка подклю- чения будет произведена уже к резервному серверу.

6. Защита данных с помощью представлений, сохраненных 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.

6.1 Определение представлений

Представления (VIEW), представляют собой временные, (иначе виртуальные) таблицы и являются объектами базы данных, информация в которых не хранится постоянно, как в базовых таблицах, а формируется динамически при обращении к ним. По сути, представление - это именованный запрос, хранящийся в базе данных. Представление не требует для своего хранения дисковой памяти, за исключением памяти, необходимой для хранения определения самого представления. Представление не может существовать само по себе, а определяется только в терминах одной или нескольких таблиц. Применение представлений позволяет разработчику базы данных создать интерфейс приложения, не зависящий на все 100% от реально существующих таблиц, а также предоставить каждому пользователю или группе пользователей ограниченный набор данных, скрывая поля и записи с конфиденциальной информацией. Представления позволяют ограничить доступ к данным на уровне записей, дополняя, таким образом, дискреционную модель разграничения доступа. Поведение и внешний вид представления не отличается от базовой табли- цы. У пользователя создается впечатление, что он работает с настоящей, реально существующей таблицей.

Операторы создания, изменения и удаления представлений в стандарте языка и реализации в MS SQL Server совпадают и представлены следующими командами:

```
{CREATE| ALTER} VIEW имя_представления
    [(имя_столбца [,...n])] [WITH ENCRYPTION]
    AS SELECT … [WITH CHECK OPTION]
DROP VIEW имя_представления [,...n]
```

Рассмотрим назначение основных параметров. По умолчанию имена полей в представлении соответствуют именам полей в исходных таблицах. Явное указание имени поля требуется для вычисляемых полей или

при объединении нескольких таблиц, имеющих поля с одинаковыми именами. Имена полей перечисляются через запятую, в соответствии с порядком их следования в представлении.

Например, отобразим в представлении клиентов из Москвы:

```
CREATE VIEW Москвичи AS
SELECT КодКлиента, Фамилия, ГородКлиента
FROM Клиент WHERE ГородКлиента='Москва'
```

Обращение к представлению осуществляется с помощью оператора SELECT, как и к обычной таблице:

```
SELECT * FROM Москвичи
```

Представление можно использовать так же, как и любую другую таблицу. К представлению можно строить запрос, обновлять данные (если выполняются определенные требования), соединять с другими таблицами. Содержание представления не фиксировано и обновляется каждый раз, когда на него ссылаются в команде. Представления значительно расширяют возможности управления данными. В частности, это прекрасный способ разрешить доступ только к определенной информации в таблице, скрыв часть данных. Так, в вышеприведенном примере представление Москвичи ограничивает доступ пользователя к данным таблицы Клиент, позволяя видеть только часть значений.

Параметр WITH ENCRYPTION предписывает серверу шифровать SQL-код запроса, что гарантирует невозможность его несанкционированного просмотра и использования. Если при определении представления необходимо скрыть имена исходных таблиц и полей, а также алгоритм объединения данных, необходимо применить этот аргумент.

Параметр WITH CHECK OPTION предписывает серверу исполнять проверку изменений, производимых через представление, на соответствие условию, определенному во фразе WHERE оператора SELECT. Это означает, что не допускается выполнение изменений, которые при- ведут к исчезновению строки из представления. Использование аргумен- та WITH CHECK OPTION гарантирует, что любые сделанные изменения будут отображены в представлении. Если пользователь пытается выполнить изменения, приводящие к исключению строки из представления, при заданном аргументе WITH CHECK OPTION сервер выдаст сообще- ние об ошибке и все изменения будут отклонены. Например, выполним команду:

```
INSERT INTO Москвичи VALUES (12,'Петров', 'Самара')
```

Это допустимая команда в представлении, и строка будет добавлена с помощью представления Москвичи в таблицу Клиент. Однако, когда информация будет добавлена, строка исчезнет из представления, поскольку

название города отлично от Москвы. Для исключения подобных моментов добавим параметр WITH CHECK OPTION в определении представления Москвичи:

```
ALTER VIEW Москвичи
SELECT КодКлиента, Фамилия, ГородКлиента FROM Клиент
WHERE ГородКлиента='Москва' WITH CHECK OPTION
```

Для измененного представления вышеупомянутая вставка записи будет отклонена системой.

Не все представления в SQL могут быть обновляемыми. Обновляемое представление определяется следующими критериями:

- основывается только на одной базовой таблице;
- содержит первичный ключ этой таблицы;
- не содержит DISTINCT в своем определении;
- не использует GROUP BY в своем определении;
- по возможности не применяет в своем определении подзапросы;
- включает каждый столбец таблицы, имеющий атрибут NOT NULL;
- оператор SELECT просмотра не использует агрегирующие (итоговые) функции, соединения таблиц, хранимые процедуры и функции, определенные пользователем;
- основывается на одиночном запросе, поэтому объединение UNION не разрешено.

Если представление удовлетворяет этим условиям, к нему могут применяться операторы INSERT INTO, UPDATE, DELETE. Различия между обновляемыми представлениями и представлениями, предназначенными только для чтения, не случайны. Цели, для которых их используют, различны. С обновляемыми представлениями в основном обходятся точно так же, как и с базовыми таблицами. Фактически, пользователи не могут даже осознать, является ли объект, который они запрашивают, базовой таблицей или представлением, т.е. прежде всего это средство защиты для сокрытия конфиденциальных или не относящихся к потребностям данного пользователя частей таблицы. Представления в режиме «только для чтения» позволяют получать данные более рационально, объединяя в запросе множество базовых таблиц. Полученные представления могут затем использоваться в других запросах, что позволит избежать сложных предикатов и снизить вероятность ошибочных действий. Например, создадим представление с расчетом итоговых значений по типу товара и затем будем использовать его в запросе выборки:

```
CREATE VIEW Заказ(Код, Клиент, Товар, Количество, Цена,
Код_доставки) AS
SELECT КодЗаказа, Фамилия+" "+Имя, Товар, Количество, Це-
на*Количество, КодДоставки FROM Клиенты INNER JOIN Заказы
ON Клиенты.КодКлиента = Заказы.КодКлиента INNER JOIN Това-
ры ON Заказы.КодТовара = Товары.КодТовара ORDER BY Товар

CREATE VIEW Доставка_Итоги(Вид_доставки, Количе-
ство_товара, Сумма) AS SELECT ТипДоставки,
Count(Количество_товара), SUM(Цена)
FROM Доставка INNER JOIN Заказ ON Доставка.Код_доставки =
Заказ.Код_доставки GROUP BY ТипДоставки ORDER BY ТипДо-
ставки
```

6.2 Преимущества и недостатки представлений

Механизм использования представлений - мощное средство СУБД, позволяющее скрыть реальную структуру БД от некоторых пользователей за счет определения представлений. Рассмотрим основные преимущества применения представлений в подобной среде.

1. Независимость от данных. С помощью представлений можно создать согласованную, неизменную картину структуры базы данных, которая будет оставаться стабильной даже в случае изменения исходных таблиц (например, добавления или удаления полей, изменения связей, разделения таблиц, их реструктуризации или переименования). Если в таблицу добавляются или из нее удаляются не используемые в представлении поля, то изменять определение этого представления не потребуется. Если структура исходной таблицы переупорядочивается или таблица разделяется, можно создать представление, позволяющее работать с виртуальной таблицей прежнего формата. В случае разделения исходной таблицы, прежний формат может быть виртуально воссоздан с помощью представления, построенного на основе соединения вновь созданных таблиц – конечно, если это окажется возможным.
2. Повышение защищенности данных. Права доступа к данным могут быть предоставлены исключительно через ограниченный набор представлений, содержащих только то подмножество дан- ных, которое необходимо пользователю. Подобный подход поз- воляет существенно ужесточить контроль доступа отдельных ка- тегорий пользователей к информации в базе данных.
3. Снижение стоимости. Представления позволяют упростить структуру запросов за счет объединения данных из нескольких таблиц в единственную виртуальную таблицу. В результате многотаб-

личные запросы сводятся к простым запросам к одному представлению.

4. Дополнительные удобства. Создание представлений может обеспечивать пользователей дополнительными удобствами – например, возможностью работы только с действительно нужной частью данных. В результате можно добиться максимального упрощения той модели данных, которая понадобится каждому конечному пользователю.
5. Возможность настройки. Представления являются удобным средством настройки индивидуального образа базы данных. В результате одни и те же таблицы могут быть предъявлены пользователям в совершенно разном виде.
6. Обеспечение целостности данных. Если в операторе CREATE VIEW будет указана фраза WITH CHECK OPTION, то СУБД станет осуществлять контроль за тем, чтобы в исходные таблицы базы данных не была введена ни одна из строк, не удовлетворяю- щих предложению WHERE в определяющем запросе. Этот меха- низм гарантирует целостность данных в представлении.

Однако использование представлений в среде SQL не лишено недостатков:

1. Ограниченные возможности обновления. В большинстве случаев представления не позволяют вносить изменения в содержащиеся в них данные.
2. Структурные ограничения. Структура представления устанавливается в момент его создания. Если определяющий запрос представлен в форме SELECT * FROM …, то символ * ссылается на все столбцы, существующие в исходной таблице на момент со- здания представления. Если впоследствии в исходную таблицу базы данных добавятся новые столбцы, то они не появятся в дан- ном представлении до тех пор, пока это представление не будет удалено и вновь создано.
3. Снижение производительности. Использование представлений связано с определенным снижением производительности. В одних случаях влияние этого фактора совершенно незначительно, тогда как в других оно может послужить источником существенных проблем. Например, представление, определенное с помощью сложного многотабличного запроса, может потребовать значительных затрат времени на обработку, поскольку при его разрешении потребуется выполнять соединение таблиц всякий раз, ко-

гда понадобится доступ к данному представлению. Выполнение разрешения представлений связано с использованием дополнительных вычислительных ресурсов.

6.3 Transact-SQL

Перед тем, как перейти к изучению хранимых процедур, функций и триггеров, рассмотрим основные положения языка программирования, используемого для их написания. К сожалению, для данного языка нет стандарта, и он специфичен для каждой СУБД. В данном пособии будем рассматривать язык Transact-SQL (или кратко T-SQL), используемый в СУБД Microsoft SQL Server.

Начнем с объявления переменных. Как и в других высокоразвитых языках программирования, прежде чем начать использовать перемен-ную, ее необходимо создать - объявить. При объявлении переменной указывается ее имя и тип данных. Для объявления переменной предназначена команда DECLARE, имеющая следующий синтаксис:

```
DECLARE {@local_variable data_type} [,..n]
```

Имя переменной обязательно должно начинаться с символа @. Следующие символы могут быть любыми, исключая специальные символы. Ограничений на второй символ имени не накладывается (допустимо использование переменной с именем @). С помощью одной команды DECLARE можно объявить сразу несколько переменных, например:

```
DECLARE @aa int, @str varchar(25), @l_ int, @qwerty bit,
@@ float
```

По умолчанию только что созданные переменные содержат пустые значения NULL и перед использованием должны быть проинициализированы. Для присвоения значения переменной можно использовать операторы SET и SELECT. Оператор SET позволяет определить только одну переменную. Например:

```
SET @aa = 15
SET @str = 'asdf ghjkl'
```

Оператор SELECT позволяет определить сразу несколько переменных:

```
SELECT @aa = 15, @str = 'asdf ghjkl'
```

Оператор SELECT можно использовать также для просмотра значений переменных, например, для отладки кода. Удобство работы с переменными в Transact-SQL состоит в том, что переменные могут напрямую встраиваться в инструкции запросов.

Область определения переменных распространяется на пакет кода. Пакетом является часть кода, разделенная операторами GO. Клиентское

приложение отправляет серверу команды на выполнение в виде пакетов. После выполнения всех команд пакета (или в ходе выполнения пакета) сервер возвращает клиенту результат. После этого клиент может отправить следующий пакет и т.д.

Существуют также глобальные переменные. В большинстве языков программирования глобальной называется переменная, имеющая большую область определения. В Transact-SQL все не так. Глобальные переменные можно назвать системными. Они доступны только для чтения, и являются своеобразными датчиками состояния системы в текущем подключении и/или пакете. Глобальные переменные создать нельзя. Существует фиксированный набор глобальных переменных, и все они начинаются с двух символов @@. Наиболее часто используются на практике следующие переменные: @@Error, @@Identity, @@RowCount, @@DBTS, @NestLevel и @@ServerName, хранящие соответственно: код последней ошибки; последнее значение ключа, сгенерированное в текущем подключении; количество строк, возвращенных последней инструкцией Transact-SQL; текущее значение штампа времени в базе данных; текущее число вложенных хранимых процедур; имя текущего сервера.

Комментарии в Transact-SQL определяются с помощью символов -- (т.е. любой текст, следующий за символами --, воспринимается как комментарий) и /* */ (как и в C-подобных языках).

Выражения Transact-SQL, как и многих других языков программирования, состоят из операндов и операторов. Операнды - это исходные данные, используемые в выражении. Значения этих данных не меняются. Операторами же являются действия, которые необходимо выполнить над операндами для вычисления выражения (например, сложение или умножение).

Рассмотрим доступные к применению операторы:

Оператор присваивания: =

Арифметические операторы: + - / * % (остаток от деления). Опе- ратор + также используется для конкатенации строк.

Операторы сравнения: = != > < >= <= !< !> (восклицательныйзнак вызывает инверсию результата).

Побитовые операторы: & | ^

Символ ; - признак конца инструкции (SQL-92). Применение точки с запятой не требуется в T-SQL, но может использоваться (может понадобится в следующей версии).

Конструкция BEGIN...END предназначена для объединения двух и более команд в единый блок, воспринимаемый сервером как одно целое. В частности, подобная группировка команд используется при работе с

циклами и командами ветвления. Как и команды ветвления, так и команды цикла позволяют работать лишь с одной командой. Следующая команда, находящаяся вне блока, воспринимается как команда, не относящаяся к циклу или блоку ветвления.

Посредством конструкции IF...ELSE в Transact SQL реализуется ветвление алгоритмов. Типичным примером такого ветвления является выполнение набора команд в зависимости от соблюдения какого-то условия. Совместно с оператором IF можно применять функции EXISTS, IN и другие. Выражение IF EXISTS использует в качестве условия наличие какой-либо строки, возвращенной инструкцией SELECT. Этот метод работает намного быстрее, чем проверка условия @@RowCount > 0, потому что не требуется подсчет общего количества строк. Как только хотя бы одна строка удовлетворяет условию IF EXISTS(), управление переда- ется к следующему оператору. В следующем примере выражение IF EXISTS() используется для проверки, был ли открыт хотя бы один заказ или нет:

```
IF EXISTS(SELECT * FROM ORDERS WHERE Opened = 1)
BEGIN
    Print 'Обработка заказов';
END
```

Посредством команды WHILE в Transact SQL организуется цикл. Сразу отметим, что в SQL Server 2000 имеется всего один тип цикла - с предусловием, который и реализуется с помощью команды WHILE. Рассмотрим синтаксис команды WHILE:

```
WHILE boolean_expression
{ sql_ block }
[ BREAK ]
{ sql_ block }
[ CONTINUE ]
```

Команды continue и break расширяют возможности создания циклов. Первая из них позволяет немедленно вернуться к команде while. Далее условие проверяется в обычном порядке. Команда break позволяет немедленно выйти из цикла и продолжить выполнение сценария так, будто условие while не было выполнено.

Команда GOTO предназначена для выполнения перехода на указанную метку (label). Метки представляют собой символьную строку, удовлетворяющую стандартным правилам именования объектов. После имени метки ставится символ двоеточия, который и свидетельствует о том, что соответствующая строка является меткой. При выполнении оператора GOTO следует указать только имя метки без символа двоеточия.

Кроме оператора IF можно использовать функции CASE, COALESCE, ISNULL. Эти функции предназначены для определения воз-вращаемого значения в зависимости от значения начального параметра. Сначала рассмотрим использование функции CASE, имеющей следую- щий синтаксис:

```
CASE input_expression
WHEN when_expression THEN result_expression [,...n]
[ELSE else_result_expression]
END
```

Функция COALESCE предназначена для возвращения значения первого выражения из заданного списка, которое не равно NULL. В анализируемом выражении могут использоваться имена столбцов, переменные, константы, функции и т. д. Синтаксис функции COALESCE таков:

```
COALESCE(expression [,…n])
```

Аргумент expression определяет выражение, которое будет анализироваться. Конструкция [,...n] указывает, что в одной команде COALESCE можно задавать множество выражений. Рассмотрим простейший пример использования команды COALESCE:

```
DECLARE @Varl int, @Var2 int, @Var3 int
SET @Var2=1024
SELECT COALESCE(@Varl,@Var2,@Var3)
```

Функция ISNULL предназначена для замены пустого значения

```
ISNULL(expression, replacement_value)
```

например: `ISNULL(количество,1)`

Преобразование типов в Transact SQL выполняется с помощью двух функций:

```
CAST(expression AS data_type)
CONVERT(data_type[(length)], expression [, style])
```

Возможности функций примерно одинаковы, но функция CONVERT является более удобной для преобразования типов, связанных со временем (style позволяет дополнительно задать формат даты). Например:

```
DECLARE @var1 int
SET@var1 = 10
CAST(@var1 AS varchar(10))
SELECT GetDate() AS RawDate, CONVERT(nvarchar(25), Get-
Date(), 100) AS Date100, CONVERT(nvarchar(25), GetDate(),
1) AS Date1
```

Результат операции следующий:

2001-11-17 10:27:27.413 Nov 17 2001 10:27AM 11/17/01

Функция Str преобразуют данные между текстовым и числовым типами: Str(число, длина, точность). Например:

```
SELECT Str(123, 5, 2) AS str
```

Для обработки ошибок времени выполнения используется конструкция Try … Catch. Общая структура конструкции такова:

```
BEGIN TRY
   < программа >
END TRY
BEGIN CATCH
   < программа >
END CATCH;
```

Компилятор T-SQL трактует комбинацию TRY BEGIN CATCH как единую команду. Как и в любой другой инструкции, наличие терминатора пакета GO или точки с запятой между этими двумя командами приведет к необрабатываемой ошибке. Инструкция BEGIN CATCH должна следовать непосредственно за END TRY. Если при выполнении секции TRY происходит какая-либо ошибка, то управление немедленно предается секции CATCH. Если секция TRY выполняется без ошибок, то блок CATCH вообще не выполняется. Конструкция TRY…CATCH не может охватывать множество блоков инструкций BEGIN…END на языке Transact-SQL и не может охватывать конструкцию IF…ELSE. Приведем пример:

```
BEGIN TRY
   SELECT 'Первая попытка';
   RAISERROR('Имитация ошибки', 16, 1);
   SELECT 'Вторая попытка';
END TRY
BEGIN CATCH
   SELECT 'Секция обработки ошибки';
END CATCH;
SELECT 'Третья попытка';
```

Будет получен следующий результат:

```
Первая попытка
Секция обработки ошибок
Третья попытка
(1 row(s) affected)
```

В этом примере SQL Server выполняет секцию TRY, пока не встречает функцию raiserror, имитирующую ошибку. После этого управление передается в секцию CATCH. Следом за блоком CATCH выполняется следующая по порядку инструкция, выводящая сообщение о третьей попытке. Если в блоке TRY происходит ошибка и управление передается в секцию CATCH, туда же попадает и информация об ошибке. Эту информацию можно извлечь с помощью функции Error_Message(), возвращающей текст сообщения об ошибке и Error_Number(), возвращающей но-

мер ошибки. Эти функции были специально созданы для секции CATCH - вне этого блока они всегда возвращают пустое значение null.

Если обнаружена фатальная ошибка T-SQL, то выполнение пакета немедленно завершается без предоставления никакой возможности просмотреть содержимое переменной @@Error, обработать ошибку и, возможно, исправить ситуацию. Фатальные ошибки встречаются довольно редко, поэтому больших проблем, как правило, не создают. Чаще всего фатальные ошибки вызваны несколькими факторами:

- несовместимостью типов данных;
- недоступностью ресурсов SQL Server;
- синтаксическими ошибками;
- дополнительными настройками сервера, несовместимыми с конкретной задачей;
- отсутствием объектов или опечатками в их именах.

Чтобы получить список большинства сообщений о фатальных ошибках, можно выполнить следующий запрос:

```
SELECT Error, Severity, Description
FROM Master.dbo.SysMessages
WHERE Severity >= 19
ORDER BY Severity, Error
```

6.4 Хранимые процедуры

6.4.1 Понятие хранимой процедуры

Хранимые процедуры (Stored Procedure) представляют собой группы связанных между собой операторов SQL, сохраняемых в базе данных в откомпилированном виде. При этом одна процедура может быть использована в любом количестве клиентских приложений, что позволяет существенно сэкономить трудозатраты на создание прикладного программного обеспечения и эффективно применять стратегию повторного использования кода. Так же, как и любые процедуры в стандартных языках программирования, хранимые процедуры могут иметь входные и выходные параметры или не иметь их вовсе. Хранимые процедуры пишутся на специальном встроенном языке программирования, они могут включать любые операторы SQL, а также включают некоторый набор операторов, управляющих ходом выполнения программ, которые во многом схожи с подобными операторами процедурно ориентированных языков программирования. В коммерческих СУБД для написания текстов хранимых процедур используются собственные языки программирования,

так, в СУБД Oracle для этого используется язык PL/SQL, а в MS SQL Server используется язык Transact SQL. Выполнение в базе данных хранимых процедур вместо отдельных операторов SQL дает пользователю следующие преимущества:

- SQL инструкции прошли этап синтаксического анализа и находятся в исполняемом формате;
- хранимые процедуры поддерживают модульное программирование, так как позволяют разбивать большие задачи на самостоятельные, более мелкие и удобные в управлении части;
- хранимые процедуры могут вызывать другие хранимые процедуры и функции;
- хранимые процедуры могут быть вызваны из прикладных программ других типов;
- как правило, хранимые процедуры выполняются быстрее, чем последовательность отдельных операторов;
- хранимые процедуры проще использовать: они могут состоять из десятков и сотен команд, но для их запуска достаточно указать всего лишь имя нужной хранимой процедуры. Это позволяет уменьшить размер запроса, посылаемого от клиента на сервер, а значит, и нагрузку на сеть;
- хранение процедур в том же месте, где они исполняются, обеспечивает уменьшение объема передаваемых по сети данных и повышает общую производительность системы.

Применение хранимых процедур упрощает сопровождение программных комплексов и внесение изменений в них. Обычно все ограничения целостности в виде правил и алгоритмов обработки данных реализуются на сервере баз данных и доступны конечному приложению в виде набора хранимых процедур, которые и представляют интерфейс обработки данных. Для обеспечения целостности данных, а также в целях безопасности, приложение обычно не получает прямого доступа к данным – вся работа с ними ведется путем вызова тех или иных хранимых процедур. Подобный подход делает весьма простой модификацию алгоритмов обработки данных, тотчас же становящихся доступными для всех пользователей сети, и обеспечивает возможность расширения системы без внесения изменений в само приложение: достаточно изменить хра- нимую процедуру на сервере баз данных. Разработчику не нужно перекомпилировать приложение, создавать его копии, а также инструктировать пользователей о необходимости работы с новой версией. Пользова-

тели вообще могут не подозревать о том, что в систему внесены изменения.

Хранимые процедуры существуют независимо от таблиц или каких-либо других объектов баз данных. Они вызываются клиентской программой, другой хранимой процедурой или триггером. Разработчик может управлять правами доступа к хранимой процедуре, разрешая или запрещая ее выполнение. Изменять код хранимой процедуры разрешается только ее владельцу или члену фиксированной роли базы данных. При необходимости можно передать права владения ею от одного пользователя к другому.

6.4.2 Типы хранимых процедур

В SQL Server имеется несколько типов хранимых процедур.

- системные хранимые процедуры;
- пользовательские хранимые процедуры;
- Временные локальные и глобальные хранимые процедуры.

Системные хранимые процедуры предназначены для выполнения различных административных действий. Практически все действия по администрированию сервера выполняются с их помощью. Можно сказать, что системные хранимые процедуры являются интерфейсом, обеспечивающим работу с системными таблицами, которая, в конечном счете, сводится к изменению, добавлению, удалению и выборке данных из системных таблиц как пользовательских, так и системных баз данных. Системные хранимые процедуры имеют префикс sp_, хранятся в системной базе данных и могут быть вызваны в контексте любой другой базы данных.

Пользовательские хранимые процедуры реализуют те или иные действия, определенные пользователем. Пользовательские процедуры хранятся в конкретной базе данных. Вызов такой процедуры возможен в контексте той базы данных, где находится процедура.

Временные хранимые процедуры существуют лишь некоторое время, после чего автоматически уничтожаются сервером. Они делятся на локальные и глобальные. Локальные временные хранимые процедуры могут быть вызваны только из того соединения, в котором созданы. При создании такой процедуры ей необходимо дать имя, начинающееся с одного символа #. Как и все временные объекты, хранимые процедуры этого типа автоматически удаляются при отключении пользователя, перезапуске или остановке сервера. Глобальные временные хранимые процедуры доступны для любых соединений сервера, на котором имеется та-

кая же процедура. Для ее определения достаточно дать ей имя, начинающееся с символов ##. Удаляются эти процедуры при перезапуске или остановке сервера, а также при закрытии соединения, в контексте которого они были созданы.

Создание новой и изменение имеющейся хранимой процедуры осуществляется с помощью следующей команды:

```
{CREATE | ALTER } [PROCEDURE] имя_процедуры [;номер]
[{@имя_параметра тип_данных } [VARYING ] [=default] [OUT-
PUT] ] [,...n]
[WITH { RECOMPILE | ENCRYPTION }]
AS sql_оператор [...n]
```

Рассмотрим параметры данной команды.

Используя префиксы sp_, #, ##, создаваемую процедуру можно определить в качестве системной или временной. Как видно из синтаксиса команды, не допускается указывать имя владельца, которому будет принадлежать создаваемая процедура, а также имя базы данных, где она должна быть размещена. Таким образом, чтобы разместить создаваемую хранимую процедуру в конкретной базе данных, необходимо выполнить команду CREATE PROCEDURE в контексте этой базы данных. При обращении из тела хранимой процедуры к объектам той же базы данных можно использовать укороченные имена, т. е. без указания имени базы данных. Когда же требуется обратиться к объектам, расположенным в других базах данных, указание имени базы данных обязательно.

Номер в имени – это идентификационный номер хранимой процедуры, однозначно определяющий ее в группе процедур. Для удобства управления процедурами логически однотипные хранимые процедуры можно группировать, присваивая им одинаковые имена, но разные идентификационные номера.

Для передачи входных и выходных данных в создаваемой хранимой процедуре могут использоваться параметры, имена которых, как и имена локальных переменных, должны начинаться с символа @. В одной хранимой процедуре можно задать множество параметров, разделенных запятыми. В теле процедуры не должны применяться локальные переменные, чьи имена совпадают с именами параметров этой процедуры.

Для определения типа данных, который будет иметь соответствующий параметр хранимой процедуры, годятся любые типы данных SQL, включая определенные пользователем. Наличие ключевого слова OUTPUT означает, что соответствующий параметр предназначен для возвращения данных из хранимой процедуры. Однако это вовсе не означает, что параметр не подходит для передачи значений в хранимую процедуру. Указание ключевого слова OUTPUT предписывает серверу при

выходе из хранимой процедуры присвоить текущее значение параметра локальной переменной, которая была указана при вызове процедуры в качестве значения параметра. Отметим, что при указании ключевого слова OUTPUT значение соответствующего параметра при вызове про- цедуры может быть задано только с помощью локальной переменной. Не разрешается использование любых выражений или констант, допустимое для обычных параметров.

Ключевое слово VARYING применяется совместно с параметром OUTPUT, имеющим тип CURSOR. Оно определяет, что выходным параметром будет результирующее множество. =default означает, что параметру можно присвоить значение по умолчанию. Таким образом, при вызове процедуры можно не указывать явно значение соответствующего параметра.

Так как сервер кэширует план исполнения запроса и компилированный код, при последующем вызове процедуры будут использоваться уже готовые значения. Однако в некоторых случаях все же требуется выполнять перекомпиляцию кода процедуры. Указание ключевого слова RECOMPILE предписывает системе создавать план выполнения хранимой процедуры при каждом ее вызове.

Ключевое слово ENCRYPTION предписывает серверу выполнить шифрование кода хранимой процедуры, что может обеспечить защиту от использования авторских алгоритмов, реализующих работу хранимой процедуры.

Ключевое слово AS размещается в начале собственно тела хранимой процедуры, т.е. набора команд SQL, с помощью которых и будет реализовываться то или иное действие. В теле процедуры могут применяться практически все команды SQL, объявляться транзакции, устанавливаться блокировки и вызываться другие хранимые процедуры. Выход из хранимой процедуры может осуществляться посредством команды RETURN.

Удаление хранимой процедуры осуществляется командой:

```
DROP PROCEDURE {имя_процедуры} [,...n]
```

6.4.3 Выполнение хранимой процедуры

Для выполнения хранимой процедуры используется команда:

```
[[EXEC[UTE] имя_процедуры [номер]
[[@имя_параметра=]{значение | @имя_переменной}
  [OUTPUT ]|[DEFAULT ]][,...n]
```

Если вызов хранимой процедуры не является единственной командой в пакете, то присутствие команды EXECUTE обязательно. Более того, эта команда требуется для вызова процедуры из тела другой процедуры

или триггера. Использование ключевого слова OUTPUT при вызове процедуры разрешается только для параметров, которые были объявлены при создании процедуры с ключевым словом OUTPUT. Если указывает- ся ключевое слово DEFAULT, то будет использовано значение по умолчанию. Естественно, DEFAULT разрешается только для тех параметров, для которых определено значение по умолчанию.

Из синтаксиса команды EXECUTE видно, что имена параметров могут быть опущены при вызове процедуры. Однако в этом случае пользователь должен указывать значения для параметров в том же порядке, в каком они перечислялись при создании процедуры. Присвоить параметру значение по умолчанию, просто пропустив его при перечислении нельзя. Если же требуется опустить параметры, для которых определено значение по умолчанию, достаточно явного указания имен параметров при вызове хранимой процедуры. Более того, таким способом можно перечислять параметры и их значения в произвольном порядке. Отметим, что при вызове процедуры указываются либо имена параметров со значениями, либо только значения без имени параметра. Их комбинирование не допускается.

Команда RETURN позволяет завершить выполнение процедуры в ее середине и вернуть значение вызывающему пакету или клиенту. Возвращаемое значение 0 указывает на успешное выполнение процедуры и установлено по умолчанию. Компания Microsoft зарезервировала значения от -99 до -1 для служебного пользования. Разработчикам для возвращения состояния ошибки пользователю рекомендуется использовать значения -100 и меньше. Команду RETURN следует использовать для индикации успешного или аварийного завершения процедуры, т.е. ее не следует использовать для возвращения реальных данных. Если нужно получить не набор данных, а всего одно значение, лучше воспользоваться выходным параметром. При вызове хранимой процедуры, если ожидается выходное значение, команда EXEC должна использовать целочисленную переменную, например: EXEC @локальная_переменная = имя_хранимой_процедуры

В следующем примере хранимая процедура возвращает индикатор успешного или аварийного завершения в зависимости от значения входного параметра:

```
CREATE PROC ISItOK ( @OK VARCHAR(10) )
AS IF @0K = 'OK'
   RETURN 0
ELSE
   RETURN -100;
```

Пример вызыва процедуры:

```
DECLARE OReturnCode INT;
EXEC OReturnCode = IsITOK 'OK';
PRINT OReturnCode;
```

6.4.4 Примеры процедур

Вначале создадим процедуру без параметров, основанную на простом запросе выборки (получим названия и стоимость товаров, приобретенных Ивановым):

```
CREATE PROC my_proc1 AS
SELECT Товар.Название,Товар.Цена*Сделка.Количество AS Сто-
имость, Клиент.Фамилия
  FROM Клиент INNER JOIN (Товар INNER JOIN Сделка
  ON Товар.КодТовара=Сделка.КодТовара)
  ON Клиент.КодКлиента=Сделка.КодКлиента
  WHERE Клиент.Фамилия='Иванов'
```

Данная процедура по сути собственно не отличается от представления и возвращает набор данных. Для обращения к процедуре можно использовать команды:

```
EXEC my_proc1 или my_proc1
```

Создадим процедуру для уменьшения цены товара первого сорта на 10%.

```
CREATE PROC my_proc2 AS
UPDATE Товар SET Цена=Цена*0.9 WHERE Сорт='первый'
```

Данная процедура не возвращает никаких данных.

Создадим процедуру для получения названий и стоимости товаров, которые приобрел заданный клиент (с одним входным параметром):

```
CREATE PROC my_proc3 @k VARCHAR(20) AS
SELECT Товар.Название, Товар.Цена*Сделка.Количество AS
Стоимость, Клиент.Фамилия FROM Клиент INNER JOIN (Товар
INNER JOIN Сделка ON Товар.КодТовара=Сделка.КодТовара) ON
Клиент.КодКлиента=Сделка.КодКлиента WHERE
Клиент.Фамилия=@k
```

Для обращения к процедуре можно использовать команды:

```
EXEC my_proc3 'Иванов' или my_proc3 @k='Иванов'
Та же процедура с двумя входными параметрами
CREATE PROC my_proc4 @t VARCHAR(20), @p FLOAT AS
UPDATE Товар SET Цена=Цена*(1-@p) WHERE Тип=@t
```

Для обращения к процедуре можно использовать команды:

```
EXEC my_proc4 'Вафли',0.05   или   EXEC my_proc4
@t='Вафли', @p=0.05
```

Напишем процедуру с входными параметрами и значениями по умолчанию (уменьшим цену товара заданного типа в соответствии с указанным %):

```
CREATE PROC my_proc5 @t VARCHAR(20)='Конфеты',@p FLOAT=0.1
AS
UPDATE Товар SET Цена=Цена*(1-@p) WHERE Тип=@t
```

Для обращения к процедуре можно использовать команды:

```
EXEC my_proc5 'Вафли',0.05   или EXEC my_proc5
@t='Вафли', @p=0.05 или
EXEC my_proc5 @p=0.05    или      EXEC my_proc5
```

В последнем случае оба параметра (и тип, и проценты) не указаны при вызове процедуры, их значения берутся по умолчанию.

Процедура с входными и выходными параметрами. Создадим процедуру для определения общей стоимости товаров, проданных за конкретный месяц.

```
CREATE PROC my_proc6 @m INT, @s FLOAT OUTPUT AS
SELECT @s = Sum(Товар.Цена*Сделка.Количество)
  FROM Товар INNER JOIN Сделка ON То-
вар.КодТовара=Сделка.КодТовара
  GROUP BY Month(Сделка.Дата) HAVING Month(Сделка.Дата)=@m
```

Для обращения к процедуре можно использовать команды:

```
DECLARE @st FLOAT
EXEC my_proc6 1,@st OUTPUT
SELECT @st
```

Этот блок команд позволяет определить стоимость товаров, проданных в январе (входной параметр месяц указан равным 1).

Создадим процедуру для определения общего количества товаров, приобретенных фирмой, в которой работает заданный сотрудник. Сначала разработаем процедуру для определения фирмы, где работает сотрудник.

```
CREATE PROC my_proc7 @n VARCHAR(20), @f VARCHAR(20) OUTPUT
AS SELECT @f=Фирма FROM Клиент WHERE Фамилия=@n
```

Затем вызовем эту процедуру в другой процедуре

```
CREATE PROC my_proc8 @fam VARCHAR(20), @kol INT OUTPUT AS
DECLARE @firm VARCHAR(20)
EXEC my_proc7 @fam, @firm OUTPUT
SELECT @kol = Sum(Сделка.Количество)
  FROM Клиент INNER JOIN Сделка
  ON Клиент.КодКлиента=Сделка.КодКлиента
  GROUP BY Клиент.Фирма HAVING Клиент.Фирма = @firm
```

Вызов процедуры осуществляется с помощью команды:

```
DECLARE @k INT
EXEC my_proc8 'Иванов',@k OUTPUT
SELECT @k
```

Еще пример процедуры:

```
CREATE PROCEDURE dbo.GetBadPrintedTasks(@SmenaID int,
@BadPrints int output) AS
SELECT @BadPrints=COUNT(код_операции)
   FROM dbo.Производство WHERE (dbo.Производство.код_смены
= @SmenaID)
   GROUP BY код_смены
```

И еще один пример, демонстрирующий получение значения первичного ключа записи после ее вставки в БД с помощью функции SCOPE_IDENTITY(). В данном примере вначале осуществляется поиск записи по ее первичному ключу. Если запись найдена, то производится ее модификация, если нет, то производится вставка новой записи:

```
CREATE PROCEDURE dbo.pr_ProductSave
 @ID int OUTPUT = null, @Product_Name varchar(30),
@ID_Type int
AS IF NOT EXISTS(SELECT * FROM Продукция WHERE
код_продукции = @ID)
  BEGIN
   INSERT INTO Продукция(название, тип)
   VALUES(@Product_Name, @ID_Type)
   SET @ID = SCOPE_IDENTITY()
  END
 ELSE
  BEGIN
   UPDATE Продукция SET название = @Product_Name, тип =
   @ID_Type WHERE(код_продукции = @ID)
  END
```

Вызов процедуры следующий:

```
DECLARE @ID INT
EXEC dbo.pr_ProductSave @ID OUTPUT, 'Цукерка', 100
SELECT @ID
```

6.5 Функции и триггеры

6.5.5 Понятие функции пользователя

При реализации на языке SQL сложных алгоритмов, которые могут потребоваться более одного раза, сразу встает вопрос о сохранении разработанного кода для дальнейшего применения. Эту задачу можно было бы реализовать с помощью хранимых процедур, однако их архитектура не позволяет использовать процедуры непосредственно в выражениях,

т.к. они требуют промежуточного присвоения возвращенного значения переменной, которая затем и указывается в выражении. Естественно, подобный метод применения программного кода не слишком удобен. Возможность создания пользовательских функций была предоставлена в среде MS SQL Server, начиная с версии 2000. В других реализациях SQL в распоряжении пользователя имелись лишь встроенные функции, обеспечивающие выполнение наиболее распространенных задач: поиск максимального или минимального значения и др.

Как и процедуры, функции пользователя представляют собой самостоятельные объекты базы данных, располагаются в определенной базе данных и доступны только в ее контексте. В SQL Server имеются следующие классы функций пользователя:

- Scalar – функции возвращают обычное скалярное значение, каждая может включать множество команд, объединяемых в один блок с помощью конструкции BEGIN...END;
- Inline – функции содержат всего одну команду SELECT и возвращают пользователю набор данных в виде значения типа данных TABLE;
- Multi-statement – функции также возвращают пользователю значение типа данных TABLE, содержащее набор данных, однако в теле функции находится множество команд SQL (INSERT, UPDATE и т.д.). Именно с их помощью и формируется набор данных, который должен быть возвращен после выполнения функции.

В отличие от хранимых процедур, пользовательские функции могут применяться в запросах так же, как и системные встроенные функции. Пользовательские функции, возвращающие таблицы, могут стать альтернативой представлениям. Представления ограничены одним выражением SELECT, а пользовательские функции способны включать дополнительные выражения, что позволяет создавать более сложные и мощные конструкции.

Удаление любой функции осуществляется командой:

```
DROP FUNCTION имя_функции [,...n]
```

Скалярные функции. Создание и изменение скалярной функции выполняется с помощью команды:

```
{CREATE|ALTER} FUNCTION имя_функции
([{@имя_параметра скаляр_тип_данных [=default]}[,...n]])
RETURNS скаляр_тип_данных
[WITH {ENCRYPTION | SCHEMABINDING} [,...n] ]
[AS] BEGIN
    <тело_функции>
```

```
RETURN скаляр_выражение
END
```

Рассмотрим назначение параметров команды.

Функция может содержать один или несколько входных параметров либо не содержать ни одного. Каждый параметр должен иметь уникальное в пределах создаваемой функции имя и начинаться с символа "@". После имени указывается тип данных параметра. Дополнительно можно указать значение, которое будет автоматически присваиваться параметру (DEFAULT), если пользователь явно не указал значение соответствующего параметра при вызове функции.

С помощью конструкции RETURNS скаляр_тип_данных указывается, какой тип данных будет иметь возвращаемое функцией значение.

Дополнительные параметры, с которыми должна быть создана функция, могут быть указаны посредством ключевого слова WITH. Благодаря ключевому слову ENCRYPTION код команды, используемый для создания функции, будет зашифрован, и никто не сможет просмотреть его. Эта возможность позволяет скрыть логику работы функции. Кроме того, в теле функции может выполняться обращение к различным объектам базы данных, а потому изменение или удаление соответствующих объек-тов может привести к нарушению работы функции. Чтобы избежать это- го, требуется запретить внесение изменений, указав при создании этой функции ключевое слово SCHEMABINDING.

Между ключевыми словами BEGIN...END указывается набор команд, они и будут являться телом функции.

Когда в ходе выполнения кода функции встречается ключевое слово RETURN, выполнение функции завершается и как результат ее вычисления возвращается значение, указанное непосредственно после слова RETURN. Отметим, что в теле функции разрешается использование множества команд RETURN, которые могут возвращать различные значения. В качестве возвращаемого значения допускаются как обычные константы, так и сложные выражения. Единственное условие – тип данных возвращаемого значения должен совпадать с типом данных, указанным после ключевого слова RETURNS.

Для примера создадим и применим функцию скалярного типа для вычисления суммарного количества товара, поступившего за определенную дату:

```
CREATE FUNCTION sales(@data DATETIME) RETURNS INT
AS BEGIN
  DECLARE @c INT
  SET @c=(SELECT SUM(количество) FROM Сделка WHERE
дата=@data)
```

```
  RETURN (@c)
END
```

В качестве входного параметра используется дата. Функция возвращает значение целого типа, полученное из оператора SELECT путем суммирования количества товара из таблицы Сделка. Условием отбора записей для суммирования является равенство даты сделки значению входного параметра функции.

```
Обращение к функции следующее:
DECLARE @kol INT
SET @kol=user1.sales('02.11.01')
SELECT @kol
Или просто:  SELECT user1.sales('02.11.01')
```

Функции Inline. Создание и изменение функции этого типа выполняется с помощью команды:

```
{CREATE|ALTER} FUNCTION имя_функции
([{@имя_параметра скаляр_тип_данных [=default]}[,...n]])
RETURNS TABLE [WITH {ENCRYPTION|SCHEMABINDING} [,...n] ]
[AS] RETURN [() SELECT_оператор ()]
```

Основная часть параметров, используемых при создании табличных функций, аналогична параметрам скалярной функции. Тем не менее, создание табличных функций имеет свою специфику. После ключевого слова RETURNS всегда должно указываться ключевое слово TABLE. Таким образом, функция данного типа должна строго возвращать значение типа данных TABLE. Структура возвращаемого значения типа TABLE не указывается явно при описании собственно типа данных. Вместо этого сервер будет автоматически использовать для возвращаемого значения TABLE структуру, возвращаемую запросом SELECT, который является единственной командой функции.

Особенность функции данного типа заключается в том, что структура значения TABLE создается автоматически в ходе выполнения запроса, а не указывается явно при определении типа после ключевого слова RETURNS.

Возвращаемое функцией значение типа TABLE может быть использовано непосредственно в запросе, т.е. в разделе FROM.

Создадим и применим функцию табличного типа для определения двух наименований товара с наибольшим остатком:

```
CREATE FUNCTION itog() RETURNS TABLE
AS RETURN (SELECT TOP 10 Товар.Название FROM Товар INNER
JOIN Склад ON Товар.КодТовара=Склад.КодТовара ORDER BY
Склад.Остаток DESC)
```

Использовать функцию для получения 10 наименований товара с наибольшим остатком можно следующим образом:

```
SELECT Название FROM user1.itog()
```

Другой пример:

```
USE MyDB
GO
CREATE FUNCTION dbo.fn_GetLastTask() RETURNS TABLE
AS RETURN (SELECT TOP 1 Задание.код_задания AS ID,
Задание.время, Продукция.название
FROM Задание INNER JOIN Продукция ON Задание.код_продукции
= Продукция.код_продукции ORDER BY ID DESC)
GO
```

6.5.6 Определение триггера в стандарте языка SQL

Триггер представляет собой специальный тип хранимых процедур, запускаемых сервером автоматически при попытке изменения данных в таблицах, с которыми триггеры связаны. Триггеры используются для проверки целостности данных, а также для отката транзакций. Таким образом, триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных. Применение триггеров большей частью весьма удобно для пользователей базы данных. И все же их использование часто связано с дополнительными затратами ресурсов на операции ввода/вывода. В том случае, когда тех же результатов можно добиться с помощью хранимых процедур или прикладных программ, применение триггеров нецелесообразно. Право на создание триггера имеет только владелец базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

Чаще всего триггеры применяются для поддержания целостности данных в базе данных, так как с помощью ограничений целостности и значений по умолчанию не всегда можно добиться нужного уровня функциональности. Часто требуется реализовать сложные алгоритмы проверки данных, отслеживать изменения значений таблицы, чтобы нужным образом изменить связанные данные.

Каждый триггер привязывается к конкретной таблице и выполняется в составе соответствующей транзакции модификации данных. В случае обнаружения ошибки или нарушения целостности данных в коде триггера можно произвести откат транзакции. Тем самым внесение изменений (событие, вызвавшее триггер) отменяется. Отменяются также все изменения, уже сделанные триггером. В отличие от обычной процедуры,

триггер выполняется неявно в каждом случае возникновения триггерного события.

Триггер не имеет аргументов. Основной формат команды CREATE TRIGGER показан ниже:

```
CREATE TRIGGER имя_триггера
  BEFORE | AFTER <триггерное_событие> ON <имя_таблицы>
  [REFERENCING <список_псевдонимов>]
  [FOR EACH {ROW|STATEMENT}]
  [WHEN(условие_триггера)]
  <тело_триггера>
```

Триггерные события состоят из вставки, удаления и обновления строк в таблице. В последнем случае для триггерного события можно указать конкретные имена столбцов таблицы. Время запуска триггера определяется с помощью ключевых слов BEFORE (триггер запускается до выполнения связанных с ним событий) или AFTER (после их выполнения). Выполняемые триггером действия задаются для каждой строки (FOR EACH ROW), охваченной данным событием, или только один раз для каждого события (FOR EACH STATEMENT).

Обозначение <список_псевдонимов> относится к таким компонентам, как старая или новая строка (OLD/NEW) либо старая или новая таблица (OLD TABLE/NEW TABLE). Ясно, что старые значения не применимы для событий вставки, а новые – для событий удаления.

При условии правильного использования триггеры могут стать очень мощным механизмом. Но им присущи и некоторые недостатки:

- скрытая функциональность: перенос части функций в базу данных и сохранение их в виде одного или нескольких триггеров иногда приводит к сокрытию от пользователя некоторых функциональных возможностей (только разработчик знает все). Хотя это в определенной степени упрощает его работу, но, к сожалению, может стать причиной незапланированных, потенциально нежелательных и вредных побочных эффектов, поскольку в этом случае пользователь не в состоянии контролировать все процессы, происходящие в базе данных;
- влияние на производительность: перед выполнением каждой команды по изменению состояния базы данных СУБД должна проверить триггерное условие с целью выяснения необходимости запуска триггера для этой команды. Выполнение подобных вычислений сказывается на общей производительности СУБД, а в моменты пиковой нагрузки ее снижение может стать особенно заметным.

- Неправильно написанные триггеры могут привести к серьезным проблемам, таким, например, как появление "мертвых" блокировок. Триггеры способны длительное время блокировать множество ресурсов, поэтому следует обратить особое внимание на сведение к минимуму конфликтов доступа.

6.5.7 Реализация триггеров в среде MS SQL Server

В реализации СУБД MS SQL Server используется следующий оператор создания или изменения триггера:

```
{CREATE|ALTER} TRIGGER имя_триггера
ON {имя_таблицы|имя_представления} [WITH ENCRYPTION] FOR
  {AFTER|INSTEAD OF} {DELETE|INSERT|UPDATE|WITH APPEND}
  AS    sql_операторы
```

Триггер может быть создан только в текущей базе данных, но допускается обращение внутри триггера к другим базам данных, в том числе и расположенным на удаленном сервере. Имя триггера должно быть уникальным в пределах базы данных. Дополнительно можно указать имя владельца. При указании аргумента WITH ENCRYPTION сервер выполняет шифрование кода триггера, чтобы никто, включая администратора, не мог получить к нему доступ и прочитать его.

Типы триггеров. В SQL Server существует два параметра, определяющих поведение триггеров:

- AFTER. Триггер выполняется после успешного выполнения вызвавших его команд. Если же команды по какой-либо причине не могут быть успешно завершены, триггер не выполняется. Следует отметить, что изменения данных в результате выполнения запроса пользователя и выполнение триггера осуществляется в теле одной транзакции: если произойдет откат триггера, то будут отклонены и пользовательские изменения. Можно определить несколько AFTER-триггеров для каждой операции (INSERT, UPDATE, DELETE). Если для таблицы предусмотрено выполнение нескольких AFTER-триггеров, то с помощью системной хранимой процедуры sp_settriggerorder можно указать, какой из них будет выполняться первым, а какой последним. По умолчанию в SQL Server все триггеры являются AFTER-триггерами.
- INSTEAD OF. Триггер вызывается вместо выполнения команд. В отличие от AFTER-триггера INSTEAD OF-триггер может быть определен как для таблицы, так и для представления. Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF триггер.

Триггеры различают по типу событий, на которые они реагируют. Существует три типа триггеров:

- INSERT TRIGGER – запускается при попытке вставки данных с помощью команды INSERT;
- UPDATE TRIGGER – запускается при попытке изменения данных с помощью команды UPDATE;
- DELETE TRIGGER – запускается при попытке удаления данных с помощью команды DELETE.

Допускается создание триггера, реагирующего на две или на все три команды. Аргумент WITH APPEND позволяет создавать несколько триггеров каждого типа.

Программирование триггера. При выполнении команд добавления, изменения и удаления записей сервер создает две специальные таблицы: inserted и deleted. В них содержатся списки строк, которые будут вставлены или удалены по завершении транзакции. Структура таблиц inserted и deleted идентична структуре таблиц, для которой определяется триггер. Для каждого триггера создается свой комплект таблиц inserted и deleted, поэтому никакой другой триггер не сможет получить к ним доступ. В зависимости от типа операции, вызвавшей выполнение триггера, содержимое таблиц inserted и deleted может быть разным:

- INSERT – в таблице inserted содержатся все строки, которые пользователь пытается вставить в таблицу; в таблице deleted не будет ни одной строки; после завершения триггера все строки из табли- цы inserted переместятся в исходную таблицу;
- DELETE – в таблице deleted будут содержаться все строки, которые пользователь попытается удалить; триггер может проверить каждую строку и определить, разрешено ли ее удаление; в таблице inserted не окажется ни одной строки;
- UPDATE – при ее выполнении в таблице deleted находятся старые значения строк, которые будут удалены при успешном завершении триггера. Новые значения строк содержатся в таблице inserted. Эти строки добавятся в исходную таблицу после успешного выполнения триггера.

Для получения информации о количестве строк, которое будет изменено при успешном завершении триггера, можно использовать глобальную переменную @@ROWCOUNT; она возвращает количество строк, обработанных последней командой. Следует подчеркнуть, что триггер запускается не при попытке изменить конкретную строку, а в момент

выполнения команды изменения. Одна такая команда воздействует на множество строк, поэтому триггер должен обрабатывать все эти строки.

Если триггер обнаружил, что из 100 вставляемых, изменяемых или удаляемых строк только одна не удовлетворяет тем или иным условиям, то никакая строка не будет вставлена, изменена или удалена. Такое поведение обусловлено требованиями транзакции – должны быть выполнены либо все модификации, либо ни одной.

Триггер выполняется как неявно определенная транзакция, поэтому внутри триггера допускается применение команд управления транзакциями. В частности, при обнаружении нарушения ограничений целостности для прерывания выполнения триггера и отмены всех изменений, которые пытался выполнить пользователь, необходимо использовать команду ROLLBACK TRANSACTION.

Для получения списка полей, измененных при выполнении команд INSERT или UPDATE, вызвавших выполнение триггера, можно использовать функцию COLUMNS_UPDATED(). Она возвращает двоичное число, каждый бит которого, начиная с младшего, соответствует одному столбцу таблицы (в порядке следования столбцов при создании таблицы). Если бит установлен в значение "1", то соответствующий столбец был изменен. Кроме того, факт изменения столбца определяет и функция UPDATE (имя_столбца).

Для удаления триггера используется команда

```
DROP TRIGGER {имя_триггера} [,...n]
```

Приведем примеры использования триггеров**.** Будем использовать триггер для реализации ограничений на значение. В добавляемой в таблицу «Сделка» записи количество проданного товара должно быть не больше, чем его остаток из таблицы «Склад».

Команда вставки записи в таблицу «Сделка» может быть, например, такой:

```
INSERT INTO Сделка VALUES (3,1,-299,'01/08/2002')
```

Создаваемый триггер должен отреагировать на ее выполнение следующим образом: необходимо отменить команду, если в таблице Склад величина остатка товара оказалась меньше продаваемого количества товара с введенным кодом (в примере код товара=3). Во вставляемой записи количество товара указывается со знаком "+", если товар поставляется, и со знаком "-", если он продается. Представленный триггер настроен на обработку только одной добавляемой записи:

```
CREATE TRIGGER Триггер_ins ON Сделка FOR INSERT
AS
IF @@ROWCOUNT=1
```

```
BEGIN
  IF NOT EXISTS(SELECT * FROM inserted
      WHERE inserted.количество<=ALL(SELECT Склад.Остаток
    FROM Склад,Сделка WHERE Склад.КодТовара=
    Сделка.КодТовара))
    BEGIN
      ROLLBACK TRAN
    PRINT 'Отмена поставки: товара на складе нет'
    END
END
```

Далее, определим триггер для сбора статистических данных. Например, с помощью инструкции:

```
INSERT INTO Сделка VALUES (3,1,200,'01/08/2002')
```

в таблицу «Сделка» добавляется товар с кодом 3 от клиента с кодом 1 в количестве 200 единиц. При продаже необходимо соответствующим образом изменить количество его запаса на складе. Триггер обрабатывает только одну добавляемую строку:

```
ALTER TRIGGER Триггер_ins ON Сделка FOR INSERT
AS
DECLARE @x INT, @y INT
IF @@ROWCOUNT=1
--в таблицу Сделка добавляется запись о поставке товара
BEGIN
--количество проданного товара должно быть не
--меньше, чем его остаток из таблицы Склад
  IF NOT EXISTS(SELECT * FROM inserted WHERE -
inserted.количество< =ALL(SELECT Склад.Остаток FROM
Склад,Сделка WHERE Склад.КодТовара = Сделка.КодТовара And
Склад.КодТовара = inserted.КодТовара))
    BEGIN
       ROLLBACK TRAN
       PRINT 'откат товара нет '
    END
  ELSE
  --если запись о товаре уже была в таблице
  --Склад, то определяется код и количество
  --товара из добавленной в таблицу Сделка записи
  BEGIN
    SELECT @y=i.КодТовара, @x=i.Количество
    FROM Сделка C, inserted I WHERE
    C.КодТовара=i.КодТовара
    --и производится изменения количества товара в таблице
Склад
    UPDATE Склад SET Остаток=остаток+@x WHERE КодТовара=@y
  END
END
```

Создадим триггер для обработки операции удаления записи из таблицы «Сделка», например, посредством следующей команды:

```
DELETE FROM Сделка WHERE КодСделки=4
```

Для товара, код которого указан при удалении записи, необходимо откорректировать его остаток на складе. Триггер обрабатывает только одну удаляемую запись.

```
CREATE TRIGGER Триггер_del ON Сделка FOR DELETE
AS
IF @@ROWCOUNT=1 -- удалена одна запись
BEGIN
  DECLARE @y INT,@x INT
--определяется код и количество товара из
--удаленной из таблицы Склад записи
  SELECT @y=КодТовара, @x=Количество FROM deleted
--в таблице Склад корректируется количество товара
     UPDATE Склад SET Остаток=Остаток-@x WHERE
КодТовара=@y
END
```

Создадим два триггера для подсчета количества записей в таблице, один для вставки записей, один – для удаления. Такие триггеры могут быть полезны в случаях, когда количество записей в таблице становится очень большим и его необходимо постоянно отслеживать. Для их работы необходима вспомогательная таблица RecordCounter с полями TableName и RecordCount, которая собственно и будет содержать количество записей. Триггер вставляет запись во вспомогательную таблицу, если таблица, генерирующая триггерное событие, ранее не отслеживалась.

```
CREATE TRIGGER [dbo].[RecordCountAdd] ON [dbo].[Orders]
   AFTER INSERT
AS BEGIN
  SET NOCOUNT ON;
  IF EXISTS(SELECT * FROM dbo.RecordCounter
  WHERE TableName = 'Orders')
UPDATE dbo.RecordCounter SET RecordCount = RecordCount + 1
WHERE TableName = 'Orders';
  ELSE
  BEGIN
    DECLARE @c INT
    SELECT @c = Count(*) FROM dbo.Orders
    INSERT INTO dbo.RecordCounter(TableName, RecordCount)
    VALUES('Orders', @c);
  END
END
```

```
CREATE TRIGGER [dbo].[RecordCountAdd] ON [dbo].[Orders]
   AFTER DELETE
AS BEGIN
  SET NOCOUNT ON;
  IF EXISTS(SELECT * FROM dbo.RecordCounter
  WHERE TableName = 'Orders')
    UPDATE dbo.RecordCounter
    SET RecordCount = RecordCount-1
    WHERE TableName = 'Orders';
  ELSE
BEGIN
  DECLARE @c INT
  SELECT @c = Count(*) FROM dbo.Orders
  INSERT INTO dbo.RecordCounter(TableName, RecordCount)
  VALUES('Orders', @c);
  END
END
```

7. Система защиты Microsoft Access

Система защиты СУБД Microsoft Access базируется на использовании следующих средств:

1. Шифрование/дешифрование;
2. Отображение и скрытие объектов в окне базы данных;
3. Использование пароля БД;
4. Использование защиты на уровне пользователя.

Шифрование базы данных — это простейший способ защиты. При шифровании базы данных ее файл сжимается и становится недоступным для чтения с помощью служебных программ или текстовых редакторов. Шифрование незащищенной базы данных неэффективно, поскольку каждый сможет открыть такую базу данных и получить полный доступ ко всем ее объектам. Шифрование обычно применяется при электронной передаче базы данных или сохранении ее на дискету, кассету или компакт-диск.

Другим способом защиты объектов в базе данных от посторонних пользователей является скрытие объектов в окне базы данных. Этот способ защиты является наименее надежным, поскольку относительно просто можно отобразить любые скрытые объекты.

Другим простейшим способом защиты является установка пароля для открытия базы данных (.mdb). После установки пароля при каждом открытии базы данных будет появляться диалоговое окно, в которое требуется ввести пароль. Только те пользователи, которые введут правильный пароль, смогут открыть базу данных. Этот способ более надежен, чем предыдущий (Microsoft Access шифрует пароль, поэтому к нему нет доступа при непосредственном чтении файла базы данных), но он действует только при открытии базы данных. После открытия базы данных все объекты становятся доступными для пользователя (пока не определены другие типы защиты, описанные ниже в этом разделе). Данный способ не является надёжным способом защиты баз данных. Существует достаточное количество бесплатных и платных утилит, отображающих пароль. В том числе доступны исходники кода на VBA, позволяющие прочитать такой пароль.

7.1 Использование защиты на уровне пользователя

Наиболее гибкий и распространенный способ защиты базы данных называют защитой на уровне пользователя, позволяющей установить различные уровни доступа к важным данным и объектам в базе данных

для различных пользователей. Данный вид защиты реализован в версиях Microsoft Access по 2003 включительно. Начиная с Access 2007, Microsoft реализует другой путь защиты на основе цифровых подписей. Защита на уровне пользователей Microsoft Access реализует избирательный подход разграничения доступа. При активизации данного вида защиты в базе данных Microsoft Access администратор базы данных или владелец объекта предоставляет определенные разрешения отдельным пользователям и группам пользователей на следующие объекты: таблицы, запросы, формы, отчеты и макросы. Учетные записи системы безопасности определяют пользователей и группы пользователей, которым разрешен доступ к объектам. Эта информация, которую называют сведениями о рабочей группе, сохраняется в специальном файле рабочей группы. Файл рабочей группы содержит сведения о пользователях, входящих в рабочую группу. Эти сведения включают имена учетных записей пользователей, их пароли и имена групп, в которые входят пользователи. Для создания файла рабочих групп и подключения его к БД используется специальное приложение «Администратор рабочих групп».

После входа в систему анализируется файл рабочей группы, в котором каждый пользователь идентифицируется уникальным кодом. Группам и пользователям предоставляются разрешения, определяющие возможность их доступа к каждому объекту базы данных. Существуют два типа разрешений на доступ: явные и неявные. Разрешения называются явными, если они непосредственно присвоены учетной записи пользователя; такие разрешения не влияют на разрешения других пользователей. Неявными называются разрешения на доступ, присвоенные учетной записи группы. Пользователь, включенный в такую группу, получает все разрешения, предоставленные группе; удаление пользователя из этой группы лишает его всех разрешений, присвоенных данной группе. Разрешения хранятся в самой БД. Раздельное хранение учетных записей и матрицы доступа (смотри Рис. 2) повышает надежность системы защиты. Если пользователь с данным идентификатором не обнаруживается в присоединенном файле рабочих групп, доступ к объектам отклоняется.

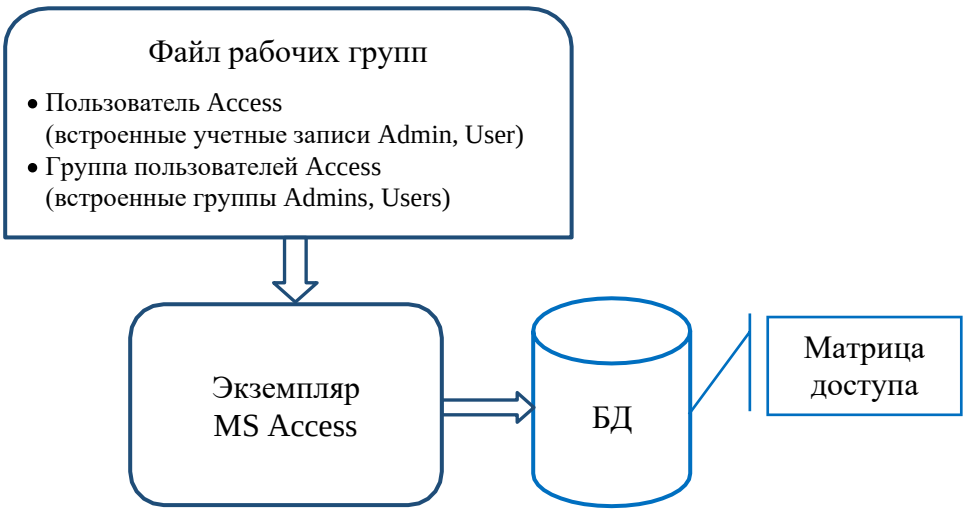

Рис. 2 Схема безопасности СУБД MS Access

При попытке пользователя выполнить какую-либо операцию с защищенным объектом базы данных его текущие разрешения определяются комбинацией явных и неявных разрешений на доступ. На уровне пользователей всегда действуют минимальные ограничения из налагаемых явными разрешениями для пользователя и для всех групп, к которым принадлежит данный пользователь. Поэтому простейшим способом управления рабочей группой является создание новых групп и определение разрешений на доступ для этих групп, а не для индивидуальных пользователей. После этого изменение разрешений для отдельных пользователей осуществляется путем добавления пользователей в группы или удаления их из групп. Кроме того, при необходимости предоставить новые разрешения, они предоставляются сразу всем членам группы в одной операции.

В Microsoft Access определены две стандартные группы: администраторы (группа «Admins») и пользователи (группа «Users»), но допускается определение дополнительных групп. Члены группы «Users» могут быть допущены только к просмотру данных в таблицах. Члены группы «Admins» имеют разрешения на доступ ко всем объектам базы данных. Если для системы защиты достаточно группы администраторов и группы пользователей, то нет необходимости создавать другие группы. В этом случае необходимо присвоить соответствующие разрешения на доступ стандартной группе «Users» и добавить дополнительных администраторов в стандартную группу «Admins». Каждый новый пользователь автоматически добавляется в группу «Users». Типовые разрешения на доступ для группы «Users» могут включать «Чтение данных» и «Обновление данных» для таблиц и запросов и «Открытие/запуск» для форм и отчетов

В случае необходимости более разветвленной структуры управления для различных групп пользователей, имеется возможность создания новых групп, присвоения группам различных наборов разрешений на доступ и добавления новых пользователей в соответствующие группы. Например, можно организовать защиту базы данных «Заказы» с помощью создания учетных записей «Управляющие» для руководства фирмы, «Представители» для торговых представителей и «Персонал» для административного персонала. После этого следует присвоить наибольшее количество разрешений группе «Управляющие», промежуточное количество группе «Представители» и минимальное группе «Персонал». При создании учетной записи для нового сотрудника эта запись будет добавляться в одну из групп, и сотрудник автоматически получит разрешения, принадлежащие этой группе.

Когда пользователь в первый раз запускает Access после установки Microsoft Office, Access автоматически создает файл рабочей группы, который идентифицируется по указанными пользователем имени и названию организации. Относительное расположение файла рабочей группы записывается в следующие параметры реестра:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\10.0\Access\Je
t\4.0\Engines\SystemDB и
HKEY_USERS\.DEFAULT\Software\Microsoft\Office\10.0\Access\
Jet\4.0\Engines\SystemDB
```

Так как получение этой информации не представляет особого труда, существует вероятность того, что кто-то сможет создать другую версию файла рабочей группы и приобрести неотъемлемые разрешения на доступ с помощью стандартной учетной записи администратора, т.к. программа установки автоматически добавляет в группу «Admins» стандартную учетную запись пользователя «Admin». Чтобы избежать этого, при активизации защиты на уровне пользователя необходимо создать новый файл рабочей группы и присоединить его к защищаемой БД. Созданный файл рабочей группы используется при открытии не только защищаемой, но любой другой базы данных, что вызывает серьёзные неудобства. Если планируется использовать файл рабочей группы для конкретной БД, следует создать ярлык для открытия этой БД и в командной строке которого указать параметр /wrkgrp и полное имя файла рабочей группы. Например:

```
"C:\Program Files\MSOffice2003\OFFICE11\MSACCESS.EXE"
/WrkGrp C:\test\gr.mdw
```

Назначение разрешений. Назначать и изменять разрешения могут следующие пользователи:

- члены группы «Admins»;
- владелец объекта;
- любой пользователь, получивший на этот объект разрешения администратора.

Пользователи, являющиеся членами группы «Admins» или владельцами объектов и не имеющие разрешения на выполнение какого-либо действия, имеют возможность назначить их себе сами.

Пользователь, создавший таблицу, запрос, форму, отчет или макрос, является владельцем этого объекта. Кроме того, пользователи, входящие в группу «Admins», могут также сменить владельца объекта или заново создать объект, что является альтернативным способом смены владельца объекта. Для создания объекта достаточно импортировать или экспортировать этот объект в другую базу данных или сделать копию объекта. Этот прием является простейшим способом смены владельца объектов, в том числе и всей базы данных.

Некоторые разрешения на доступ автоматически предоставляют другие разрешения. Например, разрешение «обновление данных» на таблицу автоматически предоставляет разрешения «чтение данных» и «чтение макета», необходимые для изменения данных в таблице. Разрешения «изменение макета» и «чтение данных» автоматически предоставляют разрешение «чтение макета». Для макросов разрешение «чтение макета» влечет предоставление разрешения «открытие/запуск». При изменении объекта и его последующем сохранении разрешения на доступ к нему сохраняются. Однако если объект сохраняется под новым именем, он становится новым объектом и, следовательно, получает разрешения, установленные по умолчанию для данного типа объектов, а не разрешения исходного объекта.

Установка и снятие защиты. В действительности, система защиты Microsoft Access всегда включена. До активизации защиты на уровне пользователя Microsoft Access автоматически подключает всех пользователей с помощью встроенной учетной записи пользователя «Admin» с пустым паролем. Microsoft Access неявно использует данную учетную запись как учетную запись администратора рабочей группы, а также как владельца всех создаваемых баз данных и таблиц, форм, отчетов и макросов.

Чтобы активировать защиту на уровне пользователя необходимо ввести пароль пользователя при запуске Microsoft Access. Для активации появления окна ввода пароля достаточно назначить пароль пользователю «Admin» (вначале он пустой). Соответственно, для отключения вывода

диалогового окна ввода пароля следует снять пароль записи «Admin». Поскольку учетные записи пользователя «Admin» совершенно одинаковы для всех экземпляров Microsoft Access, то первым шагом при организации системы защиты является определение учетных записей администратора и владельца (или единой учетной записи, являющейся записью и администратора, и владельца). После этого следует удалить учетную запись пользователя «Admin» из группы «Admins». Если этого не сделать, любой пользователь Microsoft Access сможет подключиться к рабочей группе с помощью учетной записи «Admin» и получить все разрешения на доступ к таблицам, запросам, формам, отчетам и макросам рабочей группы. Кроме того, группе Users по умолчанию присвоены все права доступа. Так как любой пользователь Access входит в эту группу, то он неявно также получает все права доступа. Поэтому, вторым шагом является минимизация прав группы Users до минимально требуемого уровня. Следующим важным шагом является смена владельца всех объектов БД с учетной записи «Admin» на новую учетную запись владельца путем создания новой базы данных и импорта в нее всех объектов из исходной базы данных. В заключение производится настройка разрешений.

В группу «Admins» разрешается добавлять произвольное число учетных записей, однако владельцем базы данных может быть только одна учетная запись - та, что была активной при создании базы данных, либо та, что была активной при передаче права владельца путем создания новой базы данных и импорта в нее всех объектов из исходной базы данных. Однако учетные записи групп могут являться владельцами таблиц, запросов, форм, отчетов и макросов базы данных. Для входа в Microsoft Access могут быть использованы только учетные записи пользователя; вход с помощью учетной записи группы невозможен.

Для снятия защиты на уровне пользователя необходимо создать новую БД, в ярлыке прописать путь к этой БД, к файлу рабочих групп защищённой БД и при открытии указать имя и пароль владельца. Затем необходимо импортировать в созданную БД все объекты защищённой БД, после чего сменить владельца всех объектов на встроенную учетную запись Admin. В заключение остается отключить окно появления запроса пароля, сняв пароль записи «Admin».

Защита Access является относительно ненадежной. Имеются специализированные программы, позволяющие узнать имя и пароль владельца БД. При отсутствии файла рабочих групп его тоже можно восстановить. Для этого потребуется узнать имена и идентификаторы владельцев объектов БД. Поскольку эта информация содержится в файле базы данных, то и она может быть извлечена с помощью таких программ как AOPR.

Иногда для взлома БД Access совсем не обязательно использовать программы, позволяющие определить пароль БД или пользователя. Часто программисты совсем не заботятся о сокрытии пароля в тексте программы. Запустив программу, работающую с защищённой БД, можно от-крыть в шестнадцатеричном редакторе WinHex виртуальную память это- го приложения. Проведя поиск Unicode строк вида 'User ID='; 'Password='; 'Database Password=' или 'pwd=' можно найти имя пользова- теля, его пароль и пароль базы данных. Можно и вовсе проигнорировать наличие защиты. Существуют программы, например, программа восста- новления БД Access AccessRecovery, которая создаёт новый файл без за- щиты и переносит в него таблицы, запросы, формы, макросы, отчеты и код модулей.

7.2 Методы противодействия взлому защиты Access.

Защита с использованием пароля БД, содержащего непечатные символы. В первую очередь этот способ нацелен на противодействие определению паролей с помощью специальных программ. Способ основан на том, что пароль БД формата Access 2000 и 2002-2003 - текстовая строка в формате Unicode. При этом, нет ни каких ограничений на её содержимое. Стандартный способ установки и использования пароля БД подразумевает его ввод с клавиатуры в диалоговом окне. Если строка пароля содержит непечатные символы, то они не будут корректно отображены программой, открывающей пароли БД. С другой стороны, этот пароль нельзя ввести в диалоговом окне при открытии БД в MS Access, что требует написание специальной программы загрузчика. В спецификации баз данных и в справке по DAO 3.60 указано, что максимальное число символов в пароле - 14. Но на самом деле их может быть 20. Задание пароля в 20 символов с наличием несколько непечатных символов приводит большинство взламывающих программ в полное замешательство.

Защита путем модификации файла БД и его расширения. Данный способ защиты основан на модификации первых байт файла. Таким образом, перед открытием БД с помощью дополнительной программы в её файл записывается правильный заголовок, хранимый в программе доступа, а после закрытия возвращается неправильный. При попытке открыть файл БД с помощью Access появляется сообщение об ошибке. Этот способ защиты хорошо совместить со способом изменения расширения файла. Например, можно взять заголовок dbf файла и записать его в начало mdb файла. Далее изменить расширение файла на dbf. Указанный метод не достаточно эффективен, так как программу, работающую с БД, можно прервать искусственно и на диске останется не защищённая

БД. Поэтому стоит его использовать только в сочетании с другими способами.

Защита изменением версии БД. Этот способ - дальнейшее развитие идеи модификации заголовка файла с целью противодействия программам, читающим пароли. Метод основан на том, что для работы с БД Access 97 и выше программы используют разные алгоритмы чтения па- роля и при этом пытаются самостоятельно определить версию mdb фай- ла. Для определения версии можно использовать последовательность из 40 байт, начиная с 122 байта от начала файла. Если в БД Access 97 впи- сать эту последовательность от Access 2003, то MS Access, библиотеки доступа ADO и DAO будут нормально работать с этим файлом, а боль- шинство взламывающих пароли программ нет. Аналогичный результат может быть получен при пересадке байтовой последовательности из Access 97 в Access 2003.

8. Архитектура системы безопасности SQL Server

Система безопасности MS SQL Sever реализует дискреционный и ролевой подходы к управлению доступом и имеет два уровня: уровень сервера и уровень базы данных (смотри 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.

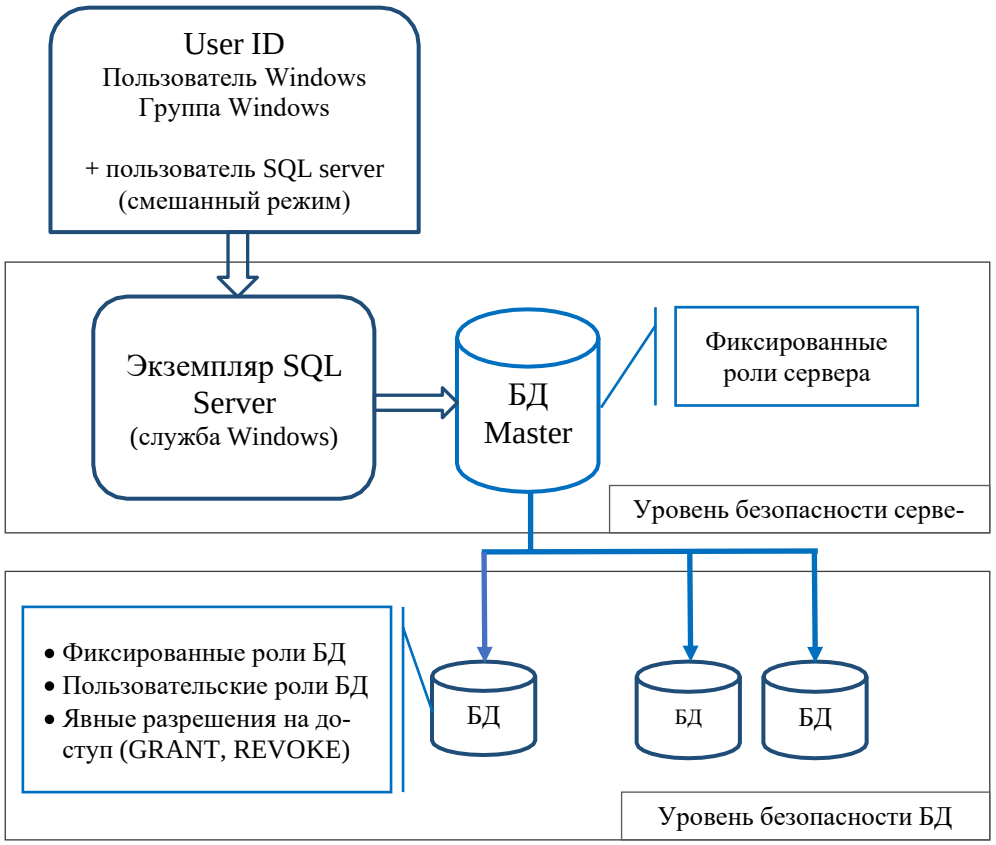

Рис. 3 Обобщенная схема безопасности СУБД MS SQL Server

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).

8.1 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).

Не следует путать регистрационные записи SQL Server с учетными записями Windows в этом сервере баз данных. Эти два типа регистрации совершенно различны. Пользователям SQL Server не нужен доступ к каталогам, где хранятся файлы базы данных, или к любым другим файлам, так как реальный доступ к файлам осуществляет процесс SQL Server, а не сам пользователь. В то же время права доступа к файлам нужны са- мому процессу SQL Server, следовательно, ему нужна учетная запись Windows. Здесь также доступны два варианта:

- Учетная запись локального администратора. SQL Server может использовать эту учетную запись для получения доступа к компьютеру, на котором установлен. Этот вариант идентичен установке на обособленный сервер, так как в этом случае нет возможности поддерживать сетевую систему безопасности Windows, необходимую для распределенной обработки;
- Учетная запись пользователя домена (рекомендуется). SQL Server может использовать учетную запись пользователя доменаWindows, созданную специально для него. Учетной записи поль- зователя SQL Server могут быть назначены административные привилегии для локального сервера, и посредством него он может получить доступ к сети для поддержания взаимодействия с други- ми серверами.

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 Administrators)	Члены этой роли имеют абсолютные права в SQL Server. Никто не имеет больших прав доступа, чем члены данной роли.
Setupadmin (Setup Administrators)	Этой роли предоставлены права управления связанными серверами, конфигурирования хранимых процедур, запускаемых автоматически при старте SQL Server, а также право добавлять учетные записи в роль setupadmin.
Serveradmin (Server Administrators)	Обычно в эту роль включаются пользователи, которые должны выполнять администрирование сервера. Имеют право на останов сервера (SHUTDOWN), изменять параметры работы служб (sp_conf igure), применять изменения (RECONFIGURE), управлять полнотекстовым поиском (sp_fulltext_service).
Securityadmin (Security Administrators)	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-

	занными серверами, включать учетные записи в роль securityadmin и читать журнал ошибок SQL Server.
Dbcreator (Database Creators)	Члены этой роли могут создавать новые базы данных, удалять и переименовывать имеющиеся, восстанавливать резервные копии базы данных и журнала транзакций.

Для добавления учетной записи пользователя в ту или иную фиксированную роль сервера можно воспользоваться хранимой процедурой sp_addsrvrolemember, имеющей следующий синтаксис:

```
sp_addsrvrolemember [@loginame =] 'login' , [@rolename =]
'role'
```

Например, рассмотрим добавление учетной записи Windows с именем Admin компьютера STORAGE в фиксированную роль сервера sysadmin:

```
EXEC sp_addsrvrolemember 'STORAGE\Admin', 'sysadmin'
```

Хотя в общем случае требуется, чтобы на сервере существовала учетная запись, которую предполагается включить в одну из фиксированных ролей, есть одно исключение. Дело в том, что пользователь Windows может получить доступ к SQL Server как член группы Windows, которой предоставлен доступ к серверу. Такие учетные записи можно включать в роли сервера без предварительного предоставления доступа непосредственно учетной записи. Для получения информации о том, какая учетная запись в какую фиксированную роль сервера включена, используется следующая системная хранимая процедура:

```
sp_helpsrvrolemember [ [@srvrplename =] 'role']
```

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_dropsrvroiemember, имеющая синтаксис:

```
sp_dropsrvrolemember [@loginame =] 'login', [@rolename =]
'role'
```

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.

8.1.1 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 предполагает, что процесс регистрации пользователей в сети достаточно защищен, и поэтому 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 досту- па.

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 без- опасности домена во время подключения пользователя. Если админи- стратор внес какие-то изменения в учетную запись пользователя, напри- мер, 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 про- цедура 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 бал 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 севере 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 ком-

плексу 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                                             NT Login
Ox0105000000000005150000000FCE31531A931         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.

8.1.2 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 эти прямые ворота в системе безопасности открывают 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 все компьютерах идентификатор безопасности 0x01.

Идентификатор безопасности хранится в столбце sid таблицы sysxlogins системной базы данных Master. В ней хранится информацияоб учетных записях как SQL Server, так и Windows. Для учетных записейSQL Server максимальный размер идентификатора безопасности состав- ляет 16 байт, для учетных записей Windows - 28 байт. Каждая строка этой таблицы соответствует одной учетной записи. Таким образом, втаблице sysxlogins может содержаться довольно много строк с информа- цией об учетных записях. Для более удобной работы с локальными учет- ными записями можно использовать представление syslogins, включаю- щее 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'
```

Параметр шифрования skip_encryption указывает серверу хранить пароль в системной таблице sysxlogins без всякого шифрования. В то же время SQL Server ожидает, что пароль обязательно будет зашифрован, поэтому он не распознает созданный таким образом пароль. Следовательно, следует избегать использования этого параметра.

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        SID
Den         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
```

В принципе, 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 вы- бранному значению. Список используемых идентификаторов безопасно- сти локального сервера можно просмотреть с помощью следующего за- проса:

```
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.

8.2 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:

db_securityadmin	Члены роли могут управлять правами доступа к объектам базы данных других пользователей и членством их в ролях
db_owner	Члены роли имеют права владельца, т. е. могут выполнять любые действия
db_denydatawriter	Членам этой роли запрещено изменение данных независимо от выданных им разрешений
db_denydatareader	Членам данной роли запрещен просмотр данных независимо от выданных им разрешений
db_ddladmin	Члены роли могут создавать, изменять и удалять объекты базы данных
db_datawriter	Пользователи, включенные в эту роль, могут изменять

	данные в любой таблице или представлении базы данных
db_datareader	Пользователи, включенные в эту роль, могут читать данные из любых таблиц и представлений базы данных
db_backupoperator	Члены роли выполняют резервное копирование базы данных
db accessadmin	Члены роли имеют право управлять пользователями базы данных: создавать, удалять и изменять

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 объек- том. Запрет привилегии замещает собой ее предоставление, а предостав- ление 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 вли- яют 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 связывает- ся 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'
```

С 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 информаци 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 просто `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'
```

С 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 следующий SQL запрос:

```
WHERE principal_type_desc <> 'DATABASE_ROLE'
UNION
--role members
SELECT rm.member_principal_name, rm.principal_type_desc,
p.class_desc, p.object_name, p.permission_name,
p.permission_state_desc,rm.role_name
FROM  perms_cte p right outer JOIN (
    select role_principal_id, dp.type_desc as
principal_type_desc, member_principal_id,
user_name(member_principal_id) as member_principal_name,
user_name(role_principal_id) as role_name--,*
    from sys.database_role_members rm
    INNER JOIN sys.database_principals dp
    ON rm.member_principal_id = dp.principal_id
) rm
ON rm.role_principal_id = p.principal_id
order by 1
```

Для перемещения БД на другой компьютер необходимо вначале отсоединить БД. Для этого на компьютере источнике вначале следует сменить контекст на системную БД Master, затем вызвать хранимую процедуру sp_detach_db имеющую один обязательный параметр – имя отсоединяемой базы данных:

```
USE [master]
GO
sp_detach_db 'test'
go
```

После перемещения файлов БД в место назначения производится обратная процедура. Если место, куда скопированы файлы, отличается от каталога по умолчанию для сервера на компьютере назначения, то требуется назначить данному каталогу полные права доступа группе SQLServer2005MSSQLUser$MyPC$SQLEXPRESS (примечание: между символами доллара находится имя сервера). Если сервер запущен от имени некоторого пользователя, то права назначаются ему, а не вышеуказанной группе. Подсоединение БД производится с помощью хранимой процедуры sp_attach_db. Первым ее аргументом идет имя БД, затем путь к файлу БД с расширением mdf, затем путь к файлу БД с расширением ldf, например:

```
use [master]
go
sp_attach_db 'test', 'D:\Service\Database
files\MSSQL.1\Data\test.mdf', 'D:\Service\Database
files\MSSQL.1\Data\test_log.ldf'
go
```

Следующим шагом является восстановление осиротевших пользователей. Покажем процедуру восстановления на примере пользователя operator. Вначале производится добавление учетных записей БД.

```
exec sp_addlogin 'operator', 'password', 'test'
go
```

После этапа добавления учетных записей производится восстановление (генерация) их системных идентификаторов. Для этого требуются специальные полномочия, которые позволяют реконфигурацию настроек сервера. В сервере версии 2000 их можно получить, вызвав процедуру sp_configure 'allow update', 1 и затем вызвать команду RECONFIGURE WITH OVERRIDE. Системный идентификатор получают с помощью функции SUSER_SID('логин'). Например:

```
USE test
GO
sp_configure 'allow update', 1
RECONFIGURE WITH OVERRIDE
GO

USE Test
UPDATE sysusers SET sid=SUSER_SID('operator') WHERE name =
'operator'
GO
```

В заключение восстанавливаются исходные настройки сервера

```
sp_configure 'allow update', 0
RECONFIGURE
GO
```

Для серверов версии 2005 и выше необходимо запустить сервер в монопольном режиме (ключ -m) и затем провести процедуру обновления SID осиротевших пользователей.

В заключение рассмотрим пример создания БД в SQL Server 2005 и выполнение действий по администрированию БД. Эти действия выполним в консоли, используя клиентскую программу sqlcmd.

```
cmd
sqlcmd -S lab48-sk\SQLEXPRESS
```

После соединения все команды выполняются в клиенте. Для начала войдем в систему с правами администратора. Получим список пользователей, включенных в фиксированную роль сервера sysadmin:

```
sp_helpsrvrolemember 'sysadmin'
```

Включим пользователя RFE\Joe в роль sysadmin:

```
sp_addsrvrolemember 'RFE\Joe', 'sysadmin'
```

В случае смешанной аутентификации сервера добавим непривилегированного пользователя RFE\User (база по умолчанию test):

```
sp_addlogin 'RFE\User', 'mypassword', 'test'
```

В случае аутентификации Windows просто предоставим доступ пользователю RFE\User к серверу

```
sp_grantlogin 'RFE\User'
Добавим пользователя RFE\User в список пользователей базы
данных test (с именем user)
USE test
go
sp_grantdbaccess 'RFE\User', 'user'
```

Включим RFE\User в фиксированную роль БД

```
sp_addrolemember 'db_datareader', 'user'
Добавим еще одного пользователя Operator
sp_grantlogin 'RFE\Operator'
sp_grantdbaccess 'RFE\Operator', 'operator'
Предоставим ему явные права доступа
GRANT SELECT, INSERT, UPDATE, DELETE ON Clients
TO operator
GRANT SELECT, INSERT, UPDATE, DELETE ON Orders TO operator
GRANT SELECT, INSERT, UPDATE, DELETE ON Goods TO operator
DENY UPDATE(Goods.price) ON Goods TO operator
Просмотрим список назначенных явных прав доступа
exec sp_helprotect @username='operator'
```

Литература

1. Смирнов, С.Н. Безопасность систем баз данных / С.Н. Смирнов. – М.: Гелиос-АРВ, 2007. – 352с.
2. Цирлов, В. Л. Основы информационной безопасности. Краткий курс / В. Л. Цирлов. – Феникс, 2008. - 256 с.
3. Герасименко В.А., Малюк А.А. Основы защиты информации. М.: МИФИ, 2001 г.
4. Neerja Bhatnagar. Security in Relational databases / N. Bhatnagar // - In: Handbook of Information and Communication Security. ed. by P. Stavroulakis, M. Stamp.- Springer. – 2010. - pp. 257 - 272.
5. Pernul, G. Database Security / G. Pernul // In: Advances in Computers, Vol. 38. ed. by M. C. Yovits. - Academic Press. – 1994. - pp. 1 - 74.
6. Хорев П.Б. Программно-аппаратная защита информации: учебное пособие / П.Б. Хорев. - М.: Форум, 2013. - 352 с.
7. Белов Е.Б. [и др]. Основы информационной безопасности: учебное пособие для студентов вузов, обуч. по спец. в обл. информ. безопасности / Е.Б. Белов – М: Горячая линия - Телеком, 2006. – 544 с.
8. Барсуков В.С. Современные технологии безопасности: Интегральный подход. / В.С. Барсуков, Водолазский В.В. – М: Но- лидж, 2000. – 496 с.
9. Романец Ю.В. Защита информации в компьютерных системах и сетях / Ю.В. Романец, П.А. Тимофеев, В.Ф. Шаньгин. – М: Радио и связь, 1999. - 328 с.
10. Голиков А.М. Основы информационной безопасности: учебное пособие / А.М. Голиков. – Томск: Томск. гос. ун-т систем упр. и радиоэлектроники, 2007. – 288 с.
11. Защита информации. Основные термины и определения. ГОСТ Р 50922 – 2006. – М: Стандартинформ, 2008.
12. Защита информации. Обеспечение информационной безопасности в организации. Основные термины и определения. ГОСТ Р 53114-2008. – М: Стандартинформ, 2009.
13. Михеева, В. Microsoft Access 2003 / В. Михеева и И. Харитонова. – СПб: БХВ, 2004. – 1072 с.
14. Нильсен, Пол. SQL Server 2005. Библия пользователя. Пер с англ./ Пол Нильсен. – Москва, СПб, Киев: Вильямс (Диалектика), 2008. – 1232 с.
15. Мамаев, Е. Microsoft SQL Server 2000. Наиболее полное руководство / Е. Мамаев – СПб: БХВ, 2005. – 1280 с.