МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАН

ФЕРГАНСКИЙ ФИЛИАЛ ТАШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ИМЕНИ МУХАММАДА АЛ-ХОРАЗМИЙ

КАФЕДРА
«ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ

по предмету

«БЕЗОПАСНОСТЬ БАЗЫ ДАННЫХ»

Для студентов направления:

5330500 – «Компьютер инжиниринг»

Фергана - 2018 г

Методические указания предназначены для студентов обучения бакалавриатуры по направлению **5330500** – «Компьютер инжиниринг» обсуждена и одобрена на заседании совета Ферганского филиала Ташкентского университета информационных технологий им.Мухаммада ал-Хоразмий от _____ 20____ г. протокол № ____.

Составитель:

Акрамова Г.А. – ФФ ТУИТ, ассистент кафедры «Информационные технологии»

Рецензенты:

Абдукодиров А.Г. – ФФ ТУИТ, к.ф-м.н. зав.кафедрой «Информационные технологии»

Холмурзаев А.А. –ФерПИ, к.т.н. зав.кафедрой «Архитектура»

Методические указания обсуждены и одобрены на заседании кафедры «Информационные технологии» (протокол № ___ от «__» _____ 2018 г.) и рекомендована к рассмотрению на Учебно-Методическом Совете факультета «_____».

Зав. кафедрой: _____ **А.Г. Абдукодиров**

Методические указания обсуждена и утверждена на заседании Учебно-Методического Совета факультета «_____» (протокол № ___ от «___» _____ 2018 г.)

Председатель учебно-методического совета факультета: _____ **Тожибоев И.Т.**

«СОГЛАСОВАНО»

Начальник учебно-методического отдела: _____ **Умаров Ш.А.**

Лабораторная работа №1.Описание языков программирование повышение безопасности баз данных.

Цель работы: Изучение среды языков программирование по повышению безопасности базы данных.

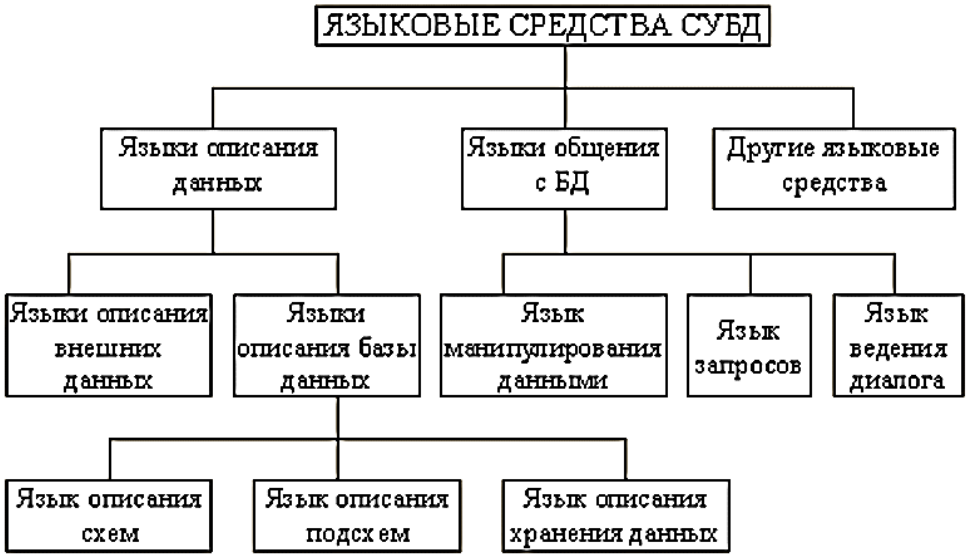
***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть

Для работы с базами данных используются специальные языки баз данных. Чаще всего выделяется два языка:

– *язык определения данных (ЯОД)* – служит для определения логической структуры БД;

– *язык манипулирования данными (ЯМД)* – содержит набор операторов манипулирования данными (добавление данных в БД, удаление, модификация, выборка и т.д.).



Во многих СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных.

Стандартным языком реляционных СУБД является язык SQL (Structured Query Language, query – вопрос) – структурированный язык запросов, оперирует не отдельными записями, а группами записей.

Реляционные СУБД (relation – отношение): 1970 г., показана возможность управления данными благодаря их описанию в терминах математической теории отношений – гибкая и простая реляционная модель данных стала доминирующей среди разработчиков и пользователей СУБД.

Объектно-реляционные БД – объектно-ориентированные возможности (определение новых типов данных и функций их обработки) встраиваются в реляционное основание.

Язык SQL сочетает средства ЯОД и ЯМД, то есть позволяет определять схему реляционной БД и манипулировать данными.

Использование языка SQL обеспечивает:

– организацию данных – возможность изменять структуру представления данных, устанавливать соотношения между элементами БД;

– чтение данных (пользователем или приложением);

– обработку данных – добавление новых данных, удаление, модификация;

– правление доступом – ограничение возможности пользователя по чтению и изменению данных и защита их от несанкционированного доступа;

– целостность данных – защита БД от разрушения в результате несогласованных действий или отказа системы;

– совместное использование данных – пользователями, работающими параллельно (чтобы они не мешали друг другу).

– FoxPro (Фокс-про́) — один из диалектов языка программирования xBase, применяемый в одноименном программном пакете. Как язык программирования, в основном применяется для разработки Файл-серверных реляционных СУБД, хотя существует, за счет гибких и богатых средств языка, возможность разработки и других классов программ.

В настоящее время используется в среде разработки Microsoft Visual FoxPro.

К язык программирования, оптимизированный для работы с массивами, модификаторами действий. Среди особенностей также отсутствие циклов, зависимые переменные и K-tree.

Реализован для Windows, Solaris, Linux.

Изначально язык программирования K-tree был разработан Артуром Уитни, Kx Systems, но его реализация от Kx проприетарная. Kona — альтернативная свободная реализация, не аффилированная с Kx Systems. 8 апреля 2011 года лицензия интерпретатора Kona изменена с проприетарной на свободную лицензию ISC (используемой также в проекте OpenBSD).

Артур Уитни ушел из группы разработки языка J в самом её начале и занялся разработкой собственного языка, который он назвал K. Одним из разногласий между Уитни и Иверсоном было чрезмерное (по мнению Уитни) усложнение языка J понятиями ранга, идею которых он сам и выдвинул в свое время, представив в 1982 году на конференции по APL в Гейдельберге. Однако в K он отказался от рангов и операторы просто действуют поэлементно. Кроме того Уитни счел множество возможностей языка J избыточными (комплексные числа, трехмерная графика). Язык K получился проще, компактнее, и оказался отлично приспособлен к сфере баз данных. Компания Уитни (Kx Systems) разработала на этом языке реляционную базу данных под названием kdb, являющуюся на сегодняшний день продуктом-лидером в этой области и превосходящую, в частности, широко разрекламированный Oracle по скорости на тестах TPC. При этом дистрибутив kdb полностью (вместе с интерпретатором K, примерами), занимает всего 200 килобайт.

Transact-SQL (T-SQL) — процедурное расширение языка SQL, созданное компанией Microsoft (для Microsoft SQL Server) и Sybase (для Sybase ASE).

SQL был расширен такими дополнительными возможностями как:

– управляющие операторы,
– локальные и глобальные переменные,
– различные дополнительные функции для обработки строк, дат, математики и т. п.,
– поддержка аутентификации Microsoft Windows.

Язык Transact-SQL является ключом к использованию MS SQL Server. Все приложения, взаимодействующие с экземпляром MS SQL Server, независимо от их реализации и пользовательского интерфейса, отправляют серверу инструкции Transact-SQL.

PL/Perl

PL/Perl — это вариант языка программирования Perl используемый при написании триггеров и хранимых процедур популярного сервера БД PostgreSQL.

Как и для PL/Perl и PL/Python разрешается использовать лишь подмножество Tcl — запрещаются все операции ввода-вывода за пределы базы данных.

PL/pgSQL (Procedural Language/PostGres Structured Query Language — процедурное расширение языка SQL, используемое в СУБД PostgreSQL. Этот язык предназначен для написания функций, триггеров и правил и обладает следующими особенностями:

- добавляет управляющие конструкции к стандарту SQL;
- допускает сложные вычисления;
- может использовать все объекты БД, определенные пользователем;
- прост в использовании.

PL/Python — это вариант языка программирования Python используемый при написании триггеров и хранимых процедур популярного сервера БД PostgreSQL.

Включён в состав этого сервера баз данных начиная с версии 7.2, позже PL/pgSQL, PL/Tcl и PL/Perl. Как и для Tcl и Perl разрешается использовать лишь подмножество Python — запрещаются все операции ввода-вывода за пределы базы данных.

В связи с параллельной поддержкой и использованием 2-й и 3-й версии языка Python в настоящее время в PostgreSQL можно использовать две версии PL/Python — plpython2u или plpython3u.

PL/SQL (Procedural Language / Structured Query Language) — язык программирования, процедурное расширение языка SQL, разработанное корпорацией Oracle. Базируется на языке Ада[1].

PL/SQL встроен в следующие СУБД: Oracle Database (начиная с версии 7), TimesTen (англ.) (с версии 11.2.1) и IBM DB2 (с версии 9.7). Также PL/SQL используется как встроенный язык для средства быстрой разработки Oracle Forms и инструмента разработки отчётов Oracle Reports.

PL/Tcl — это вариант языка программирования Tcl используемый при написании триггеров и хранимых процедур популярного сервера БД PostgreSQL.

Как и для PL/Perl и PL/Python разрешается использовать лишь подмножество Tcl — запрещаются все операции ввода-вывода за пределы базы данных.

Progress 4GL — это высокоуровневый язык программирования четвёртого поколения (англ. 4th Generation Language; аббревиатура 4GL), разработанный в Progress Software Corporation. Progress 4GL является высокоуровневым, процедурным языком разработки приложений, который позволяет удовлетворять всем требованиям, предъявляемым к современным приложениям, в то же время уменьшая сложность и повышая производительность их разработки. лалалала 4GL содержит все необходимые программные конструкции для решения самых различных аспектов программирования сложных приложений без необходимости прибегать к менее эффективным и менее переносимым языкам третьего поколения. Кроме этого, 4GL обеспечивает поддержку и переход между тремя основными принципами программирования: структурированным, событийно-управляемым и объектно-ориентированным, — от Вас не требуется осваивать новые принципы программирования для того, чтобы успешно работать с PROGRESS. Для завершения процесса разработки промышленного приложения Вам потребуются средства разработки не только логики взаимодействия с пользователем, но также потребуются средства для решения таких важных задач, как:

- Автоматический контроль транзакций и блокирование записей
- Получение и обработка информации из баз данных
- Сложные вычисления и обработка данных
- Пакетная обработка

- Генерация отчетов
- Целостность базы данных и требования безопасности
- Поддержка двухбайтовых кодировок

Язык 4GL содержит все функции и операторы, необходимые для удовлетворения вышеперечисленных требований. Но, в отличие от остальных инструментальных средств, менее ориентированных на разработку приложения в архитектуре клиент/сервер, PROGRESS не требует использования различных языков программирования для отдельного программирования обработки данных на клиенте, серверных процессов и пакетной обработки на сервере. Всё это уменьшает стоимость затрат по изучению языка и продолжению разработки.

Используется в СУБД Progress. Кроме традиционных SQL-запросов, реализация поддержки которых не очень удобна, позволяет использовать выражения FOR EACH, FIND, FIND FIRST.

Кроме того, существует возможность сокращённого написания операторов. Например, объявление переменной

RPG (Report Program Generator) — язык программирования, синтаксис которого был изначально сходен с командным языком механических табуляторов компании IBM. Был разработан для облегчения перехода инженеров, обслуживавших эти табуляторы на новую технику и переноса данных, первоначально был реализован для IBM 1401. Широко использовался в 1960-х и 1970-х гг.

Наиболее распространённой версией языка, по всей видимости являлась RPG II.

Компания IBM продолжает поддержку языка и в настоящее время, так как на нём написан громадный объём кода, который невыгодно переводить на другие языки программирования.

В версии RPG IV, выпущенной в 2001 году, введены элементы объектного программирования.

Кроме мэйнфреймов и машин AS/400 от IBM, RPG был реализован на платформах Digital VAX, Sperry Univac BC/7, Univac system 80, Siemens BS2000, Burroughs B1700, Hewlett Packard HP3000, ICL 2900 series, Honeywell 6220, WANG VS, IBM PC (DOS).

Компилятор Visual RPG, разработанный сторонним производителем, обеспечивает работу под Windows и поддержку GUI. Существуют также реализации для OpenVMS и других, более экзотических платформ.

SQL/PSM — стандарт для SQL/Persistent Stored Modules (постоянно хранимые модули), разработанный Американским национальным институтом стандартов (ANSI) в качестве расширения SQL. Впервые был принят в 1996. Стандарт поддерживает процедурное программирование в дополнение к выражениям запроса языка SQL.

Расширение SQL/PSM закреплено стандартом ISO/IEC 9075-4:2003. SQL/PSM стандартизирует процедурное расширение для SQL, включая управление потоком выполнения, обработку условий, обработку флагов состояний, курсоры и локальные переменные, а также присваивание выражений переменным и параметрам. Более того, SQL/PSM формализует объявление и поддержку постоянных подпрограмм языков баз данных (например, «хранимых процедур»).

Порядок выполнения работы

На языке SQL Server или SQL Server 2015 написать программу для повышение безопасности баз данных, указанным преподавателем.

Вариант выполнения работы

- Название темы варианта
- Цель работы

- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Обеспечение информационной безопасности СУБД
2. Поясните аспекты информационной безопасности для СУБД?
3. Уровень полномочий субъекта.
4. Для чего нужна мандатная защита?

Лабораторная работа №2.Принципы реализации SQL Server и свойства аутентификации.

Цель работы: Изучение принципов реализации SQL Server и средства аутентификации.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть.

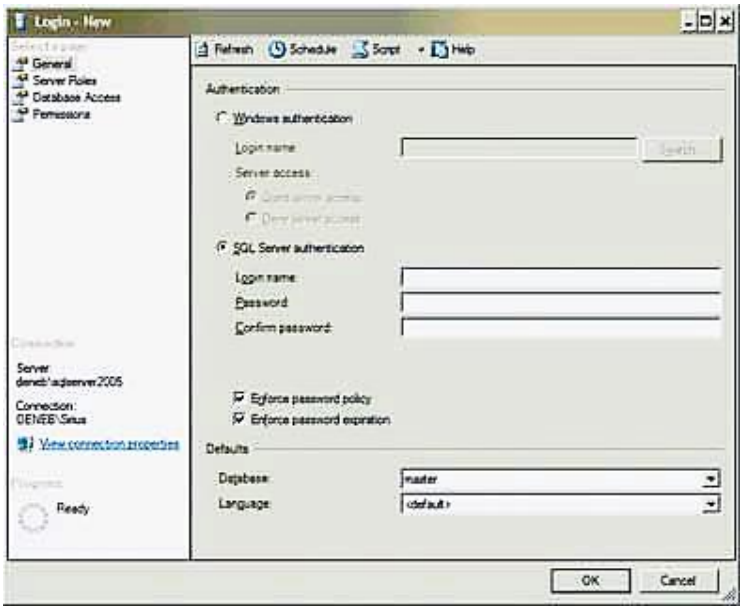
Microsoft SQL Server — система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

Аутентификация - это процесс проверки права пользователя на доступ к тому или иному ресурсу. Чаще всего аутентификация осуществляется с помощью ввода имени и пароля.

SQL Server 2015 поддерживает два режима аутентификации: с помощью Windows и с помощью SQL Server. Первый режим позволяет реализовать решение, основанное на однократной регистрации пользователя и едином пароле при доступе к различным приложениям (*Single SignOn solution, SSO*). Подобное решение упрощает работу пользователей, избавляя их от необходимости запоминания множества паролей и тем самым снижая риск их небезопасного хранения (вспомним стикеры с паролями, наклеенные на мониторы). Кроме того, данный режим позволяет использовать средства безопасности, предоставляемые операционной системой, такие как применение групповых и доменных политик безопасности, правил формирования и смены паролей, блокировка учетных записей, применение защищенных протоколов аутентификации с помощью шифрования паролей (Kerberos или NTLM).

Аутентификация с помощью SQL Server предназначена главным образом для клиентских приложений, функционирующих на платформах, отличных от Windows. Этот способ считается менее безопасным, но в SQL Server 2015 он поддерживает шифрование всех сообщений, которыми обмениваются клиент и сервер, в том числе с помощью сертификатов, сгенерированных сервером. Шифрование также повышает надежность этого способа аутентификации. Для учетной записи SQL Server можно указать такой параметр, как необходимость сменить пароль при первом соединении с сервером. Если SQL Server 2015 работает под управлением Windows Server 2003, можно воспользоваться

такими параметрами учетной записи, как проверка срока действия пароля и локальная парольная политика Windows.

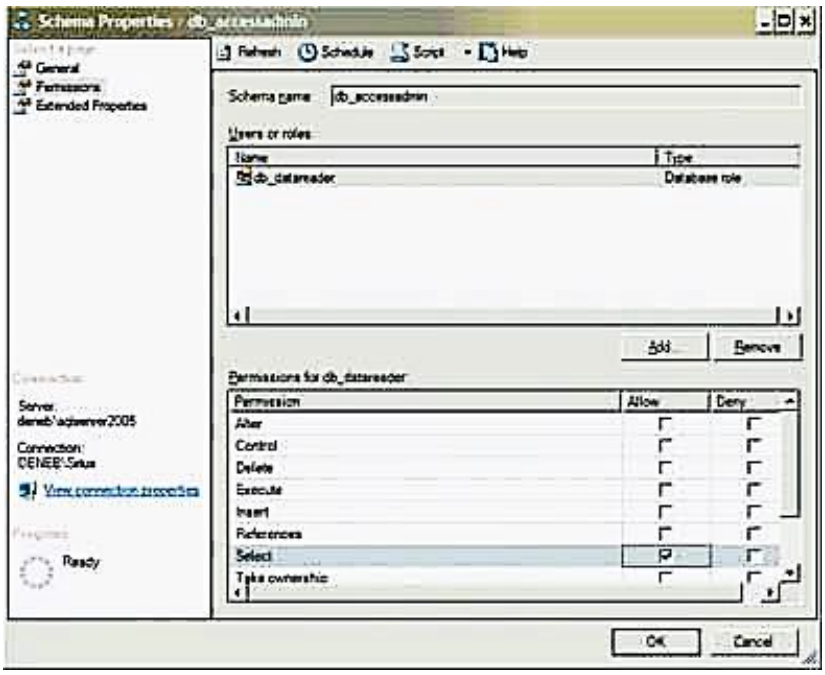


Установка правил для пароля пользователя

Схемы, не имеющие отношения к пользователям

Вот уже не первый десяток лет принцип распределения прав доступа к объектам баз данных в большинстве серверных СУБД основан на наличии у каждого объекта базы данных пользователя-владельца, который может предоставлять другим пользователям права доступа к объектам базы данных. При этом набор объектов, принадлежащих одному и тому же пользователю, называется схемой. Данный способ владения объектами создавал определенные неудобства при сопровождении приложений, использующих базы данных. Так, при увольнении сотрудника, создавшего объекты, используемые многими пользователями, и удалении соответствующей учетной записи приходилось вносить изменения в серверный код (а нередко и в код клиентского приложения). Понимание возможности возникновения этих проблем привело к распространению небезопасных, но простых в применении способов управления учетными записями пользователей. Вплоть до хранения их имен и паролей в обычных таблицах, что резко повышало угрозу несанкционированного доступа к данным и приложениям.

В SQL Server 2015 концепция ролей расширена: эта СУБД позволяет полностью отделить пользователя от схем и объектов базы данных. Теперь объекты базы данных принадлежат не пользователю, а схеме, не имеющей никакого отношения ни к каким учетным записям и тем более к административным привилегиям. Таким образом, схема становится механизмом группировки объектов, упрощающим предоставление пользователям прав на доступ к объектам.



Установка прав для схем данных

Роли

Для упрощения управления правами доступа в большинстве серверных СУБД применяется механизм ролей - наборов прав доступа к объектам базы данных, присваиваемых некоторой совокупности пользователей. При использовании ролей управление распределением прав доступа к объектам между пользователями, выполняющими одинаковые функции и применяющими одни и те же приложения, существенно упрощается: создание роли и однократное назначение ей соответствующих прав осуществляется намного быстрее, нежели определение прав доступа каждого пользователя к каждому объекту. SQL Server 2015 позволяет создавать так называемые вложенные роли, то есть присваивать одной роли другую со всеми ее правами. Это упрощает управление не только правами пользователей, но и самими ролями, создавая, к примеру, сходные между собой группы ролей. SQL Server 2015 также поддерживает так называемые роли для приложений (*application roles*), которые могут использоваться для ограничения доступа к объектам базы данных в тех случаях, когда пользователи обращаются к данным с помощью конкретных приложений. В отличие от обычных ролей, роли для приложений, как правило, неактивны и не могут быть присвоены пользователям. Их применение оказывается удобным в том случае, когда требования безопасности едины для всех пользователей, при этом не требуется аудит или иная регистрация деятельности конкретных пользователей в базе данных.

Рекомендации по управлению доступом

Ниже следует ряд несложных рекомендаций, касающихся применения средств управления доступом при создании и развертывании решений на основе SQL Server 2015.

Применение схем и ролей. Общие принципы

Одним из обязательных этапов проектирования баз данных должно быть детальное определение способов доступа к объектам базы данных. В частности, на этом этапе необходимо определить, каким образом использовать схемы для группировки объектов базы данных с точки зрения доступа к ним, какие роли создать, определить возможные группы пользователей и роли, которые следует им присвоить. Кроме того, на этом же этапе важно принять решение о том, как будут использоваться представления и хранимые

процедуры для обеспечения безопасного доступа к данным (например, для ограничения непосредственного доступа к таблицам).

Управление доступом в службах отчетов

При применении служб отчетов (*Reporting Services*) необходимо отдельно позаботиться о правилах доступа к объектам базы данных для пользователей, применяющих приложения, обращающиеся к этим службам. Так, указанной категории пользователей должны быть доступны как сами описания отчетов, так и те объекты базы данных, на основании которых эти отчеты генерируются, и в этом случае наиболее уместно создание отдельной роли, обладающей указанными правами. Следует заметить, что по умолчанию службы отчетов используют службы Internet Information Services (*IIS*) и средства безопасности Windows для аутентификации пользователей на сервере отчетов. Данный режим считается наиболее безопасным, поскольку может позволить запретить анонимный доступ к web-приложениям, выполняющимся под управлением IIS.

Управление доступом в службах уведомлений

Службы уведомлений (*Notification Services*) выполняются от имени собственной учетной записи, которая, с одной стороны, должна обладать определенными правами для обеспечения доставки уведомлений, с другой стороны, указанный набор прав должен быть минимально необходимым (как минимум, такая учетная запись не должна входить в группу Administrators).

Управление доступом в интеграционных службах

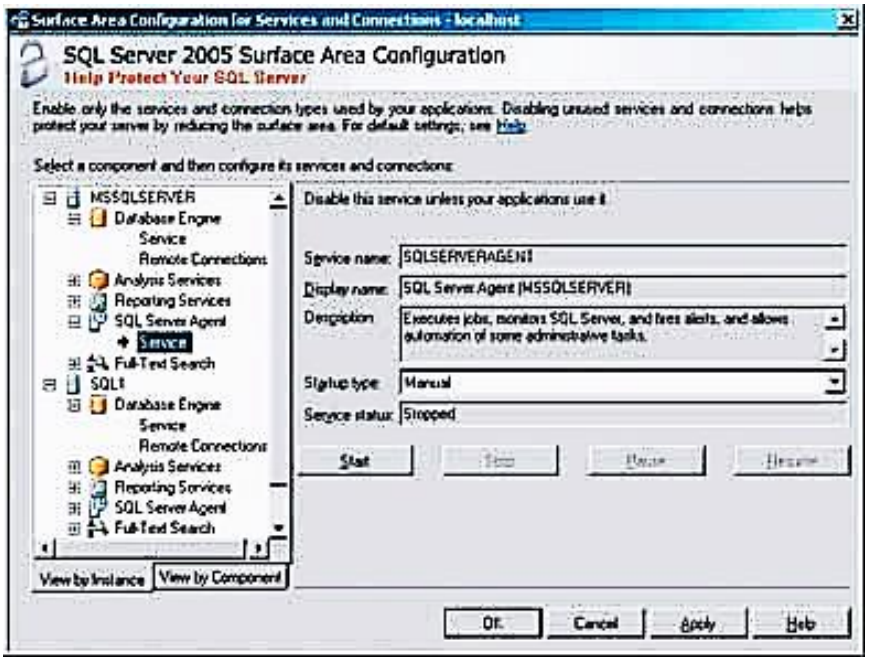
Для управления доступом в интеграционных службах в SQL Server 2015 введены новые роли: db_dt sadmin,db_dt sltuser и db_dtsoperator. Они позволяют осуществлять контроль доступа к пакетам интеграционных служб, хранящихся в базе данных msdb.

Управление доступом в службах репликации

Для управления доступом в службах репликации SQL Server 2015 была создана новая модель безопасности этих служб, позволяющая более гибко управлять учетными записями агента репликации (*Replication Agent*). Теперь каждый агент репликации может выполняться под собственной учетной записью, что позволяет наделить каждого агента именно теми правами, которые требуются для его функционирования, без присвоения избыточных привилегий. При этом, если серверы, участвующие в процессе репликации, находятся в одном домене, для указанных учетных записей рекомендуется использовать аутентификацию Windows.

Управление доступом для SQL Server Agent

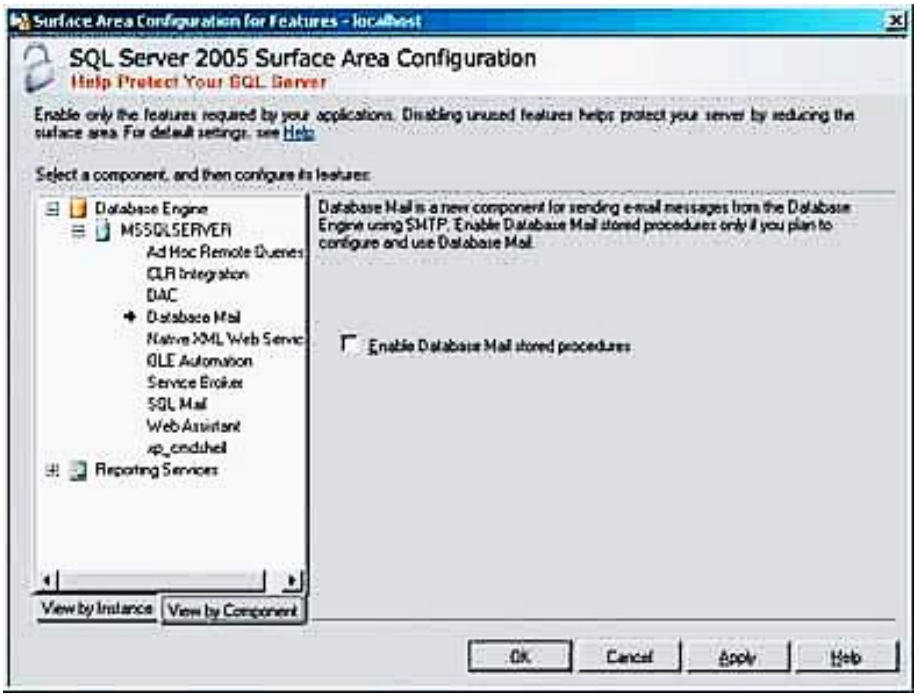
Агент SQL Server Agent должен выполняться от имени учетной записи Windows, обладающей рольюsysadmin, но не принадлежащей группе Administrators и имеющей разрешение на увеличение квот памяти для процесса. Сам SQL Server Agent должен выполняться как служба операционной системы, протоколироваться как сервис. Кроме того, можно создать специальную прокси-запись и ассоциировать ее с соответствующими правами Windows для доступа к внешним ресурсам. Для управления службой SQL Server Agent можно использовать утилиту Surface Area Configuration.



Управление службой SQL Server Agent

Управление доступом для Database Mail

Database Mail - это новый компонент SQL Server 2015, предназначенный для поддержки протокола SMTP (*Simple Mail Transport Protocol*). Для управления доступом к нему рекомендуется создавать настраиваемые профили, позволяющие указать, каким образом пользователи базы данных msdb имеют доступ к данному профилю (самим пользователям следует присвоить роль *DatabaseMailUserRole* базы данных msdb). Для управления службой Database Mail можно также использовать утилиту Surface Area Configuration (рис. 4).



Управление службой Database Mail

Управление доступом к базе данных с помощью протокола HTTP

Поскольку HTTP-доступ является одним из наиболее распространенных способов атак на приложения извне, следует разрешить доступ к HTTP Endpoints только тем пользователям или группам, которым он действительно необходим. Кроме того, на сервере баз данных следует отключить учетную запись Windows Guest, поскольку она позволяет пользователям подключаться к компьютеру без ввода пароля.

Порядок выполнения работы

На языке SQL Server или SQL Server 2015 написать программу для повышение безопасности баз данных, указанным преподавателем.

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Обеспечение информационной безопасности СУБД
2. Поясните аспекты информационной безопасности для СУБД?
3. Уровень полномочий субъекта.
4. Для чего нужна мандатная защита?

Лабораторная работа №3.Настройка схемы базы данных с защитой паролем

Цель работы: Изучение среды настройка базы данных с зашитой паролю.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть

В среде SQL Server используются ключи шифрования для защиты данных, информации об учетных данных и соединениях, которые хранятся в серверной базе данных. В SQL Server существует два вида ключей: симметричные и асимметричные. В симметричных ключах для шифрования и расшифровки данных используется одинаковый пароль. При использовании асимметричных ключей один пароль применяется для шифрования данных (*открытый* ключ), а другой для расшифровки данных (*закрытый* ключ).

Используемые в SQL Server ключи шифрования представляют собой сочетание открытых, закрытых и симметричных ключей, которые используются для защиты конфиденциальных данных. Симметричный ключ создается во время инициализации SQL Server, при первом запуске экземпляра SQL Server. Этот ключ используется SQL Server для шифрования конфиденциальных данных, которые хранятся в SQL Server. Открытые и закрытые ключи создаются операционной системой и используются для защиты симметричного ключа. Пара из открытого и закрытого ключей создается для каждого экземпляра SQL Server, который сохраняет конфиденциальные данные в базе данных.

Применения ключей шифрования SQL Server и базы данных

Существует два основных применения ключей в SQL Server: главный ключ службы создается на экземпляре SQL Server и для него, а главный ключ базы данных используется для базы данных.

Главный ключ службы автоматически создается при первом запуске экземпляра SQL Server и используется для шифрования пароля связанного сервера, учетных данных или главного ключа базы данных. Главный ключ службы шифруется с помощью ключа локального компьютера и API-интерфейса защиты данных Windows. Этот API-интерфейс использует ключ, получаемый из учетных данных Windows, которые соответствуют учетной записи службы для SQL Server и учетным данным компьютера. Главный ключ службы может быть расшифрован лишь той учетной записью службы, под которой он был создан, или участником, имеющим доступ к учетным данным компьютера.

Главный ключ базы данных — это симметричный ключ, который применяется для защиты закрытых ключей сертификатов и асимметричный ключей, которые есть в базе данных. Он также может использоваться для шифрования данных, но из-за ограниченной длины не может применяться на практике для шифрования данных так же широко, как симметричный ключ.

При создании этот ключ зашифровывается с помощью алгоритма «Triple DES» и пользовательского пароля. Чтобы разрешить автоматическое шифрование главного ключа, копия этого ключа зашифровывается с помощью главного ключа службы. Ключ хранится как в базе данных, где используется ключ, так и в системной базе данных master.

Копия, которая хранится в базе данных master, автоматически обновляется при каждом изменении главного ключа. Но это поведение по умолчанию может быть изменено с помощью параметра DROP ENCRYPTION BY SERVICE MASTER KEY инструкции ALTER MASTER KEY. Главный ключ базы данных, который не зашифрован с помощью главного ключа службы, следует открывать с помощью инструкции OPEN MASTER KEY и пароля.

Управление ключами SQL Server и базы данных

Управление ключами шифрования заключается в создании новых ключей базы данных, создании резервной копии ключей сервера и базы данных и знании порядка восстановления, удаления и смены ключей.

Чтобы управлять симметричными ключами, можно использовать средства, входящие в SQL Server, для выполнения следующих действий:

- Резервное копирование копии ключа сервера и базы данных, чтобы использовать их при восстановлении установки сервера или в ходе запланированного переноса.
- Восстановление ранее сохраненного ключа в базе данных. Это позволяет новому экземпляру сервера обращаться к существующим данным, которые первоначально шифровались не им.
- Удаление зашифрованных данных из базы данных в маловероятной ситуации, когда не удается обратиться к зашифрованным данным.
- Повторное создание ключей и повторное шифрование данных в маловероятной ситуации, когда ключ становится известен посторонним. Для повышения безопасности следует периодически повторно создавать ключи (например, раз в несколько месяцев) для защиты сервера от атак с целью расшифровки ключа.
- Добавление или удаление экземпляра сервера из масштабного развертывания, когда несколько серверов используют одну базу данных и ключ, допускающий обратимое шифрование для этой базы данных.

Важная информация по безопасности

Для доступа к объектам, защищенным главным ключом службы, необходима учетная запись службы SQL Server, которая использовалась для создания ключа, или учетная запись компьютера. То есть, компьютер привязан к системе, в которой был создан ключ. Можно изменить учетную запись службы SQL Server *или* учетную запись компьютера, не теряя доступа к ключу. Однако если изменить обе учетные записи, доступ к главному ключу службы будет потерян. Если доступ к главному ключу службы будет потерян без одного из этих элементов, то не удастся расшифровать данные и объекты, зашифрованные с помощью первоначального ключа.

Соединения, защищенные с помощью главного ключа службы, не могут быть восстановлены без него.

Для доступа к объектам и данным, защищенным главным ключом базы данных, требуется только пароль, использованный для защиты ключа.

Microsoft SQL Server 2005 Compact Edition (SQL Server Compact Edition) Database Engine позволяет ограничить доступ к локальной базе данных, защитив ее паролем. Для защищаемой базы данных SQL Server Compact Edition создается один пароль. Каждому пользователю базы данных пароль задать нельзя. Пароли для баз данных SQL Server Compact Edition:

- включают до 40 символов;
- состоят из букв, символов, цифр или их комбинаций;
- не подлежат восстановлению.

Примечание

Наличие пароля не запрещает чтения открытого текста в файле базы данных. Чтобы хранить данные в зашифрованном формате, ограничив тем самым программный доступ к базе данных, используйте сочетание защиты паролем и шифрования.

Создание защищенных паролем баз данных

Чтобы создать защищенную паролем базу данных, укажите свойство пароля при создании базы данных. Защищенные паролем базы данных создаются следующими способами.

- Использование синтаксиса SQL Чтобы создать защищенную паролем базу данных с помощью синтаксиса SQL, укажите ее пароль в инструкции **CREATE DATABASE**. Пароль следует помещать за ключевым словом **DATABASEPASSWORD** и заключать в одинарные кавычки, как показано в следующем примере:
- CREATE DATABASE "secure.sdf" DATABASEPASSWORD '<myPassword>'
- Использование ADO .NET Чтобы создать защищенную паролем базу данных с помощью метода SqlCeEngine.CreateDatabase, в строке подключения необходимо указать свойство пароля, как показано в следующем примере:
- "data source=\ssce.sdf; password=<myPassword>"
- Дополнительные сведения см. в описании класса System.Data.SqlServerCe.SqlCeEngine в пакете для разработки программного обеспечения (SDK) Microsoft .NET Compact Framework в Microsoft Visual Studio 2005.
- Использование OLE DB Чтобы создать зашифрованную базу данных с помощью поставщика OLE DB для SQL Server Compact Edition, необходимо передать свойство поставщика **DBPROP_SSCE_ENCRYPTDATABASE** как VARIANT_TRUE, а также указать пароль с помощью свойства поставщика **DBPROP_SSCE_DBPASSWORD**.

Доступ к защищенным паролем базам данных

Чтобы открыть защищенную паролем базу данных, необходимо указать пароль. Для доступа к защищенным паролем базам данных используются следующие методы.

– Использование поставщика данных для SQL Server Compact Edition.Чтобы получить доступ к защищенной паролем базе данных с помощью метода SqlCeConnection.Open, в строке подключения необходимо указать свойство пароля.

Например:

– "data source=\ssce.sdf; password=<myPassword>"

Дополнительные сведения см. в описании класса **System.Data.SqlServerCe.SqlCeConnection** в пакете для разработки программного обеспечения .NET Compact Framework SDK в Visual Studio 2005.

Использование OLE DB SQL Server Compact Edition поддерживает механизм управления доступом на уровне файлов, который требует ввода пароля для доступа к защищенной паролем базе данных SQL Server Compact Edition. Данный пароль необходимо вводить при каждом открытии базы данных. Пароль указывается с помощью свойства **DBPROP_SSCE_DBPASSWORD** из набора свойств поставщика **DBPROPSET_SSCE_DBINIT**.

Это свойство служит для указания пароля создаваемой базы данных. Зашифрованные базы данных всегда защищаются паролем.

Порядок выполнения работы

На языке SQL Server или SQL Server 2015 написать программу для повышение безопасности баз данных, указанным преподавателем.

Вариант выполнения работы

– Название темы варианта
– Цель работы
– Теоретическую часть
– Схему
– Текст программы(скрипт)

Контрольные вопросы:

1. Архитектуры системы безопасности в MS SQL Server
2. Алгоритм проверки аутентификации пользователя в MS SQL Server

Лабораторная работа №4. Реализация прав доступа для использования SQL Server

Цель работы: Изучение способов реализации права доступа к базам данных в среде SQL Server.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть

Безопасность сервера во многом зависит от того, как хорошо вы сможете настроить права доступа на объекты. Если где-то предоставить пользователю чуть-чуть лишнего, так сразу жди проблем. Нет, пользователь не будет использовать твои ошибки. Ими

воспользуюсь я, или другой хакер. И тогда распрощайся со своими таблицами с данными или всей базой данных. Наша жизнь беспощадна не только в реале, но и в виртуале.

Почему-то под безопасностью базы данных подразумевается защита от вторжения извне, т.е. совершенное злостным хакером, напившимся бочкой пива :). Нет, такие взломы происходят слишком редко. Я работаю сейчас программистом в достаточно крупной конторе, и администратор вообще не задумывается о защите портов сервака, на котором открыто все, что угодно. На одном сервере крутится куча баз, программ и даже FTP сервер и за 5 лет его ни разу не взломали :). Благо я уломал этого админа установить WEB сервер на отдельное железо, а то если бы народ узнал IP адрес нашего главного сервера, то его любой ламер смог бы проскриптить. Ни база данных, ни Windows не патчились уже несколько лет.

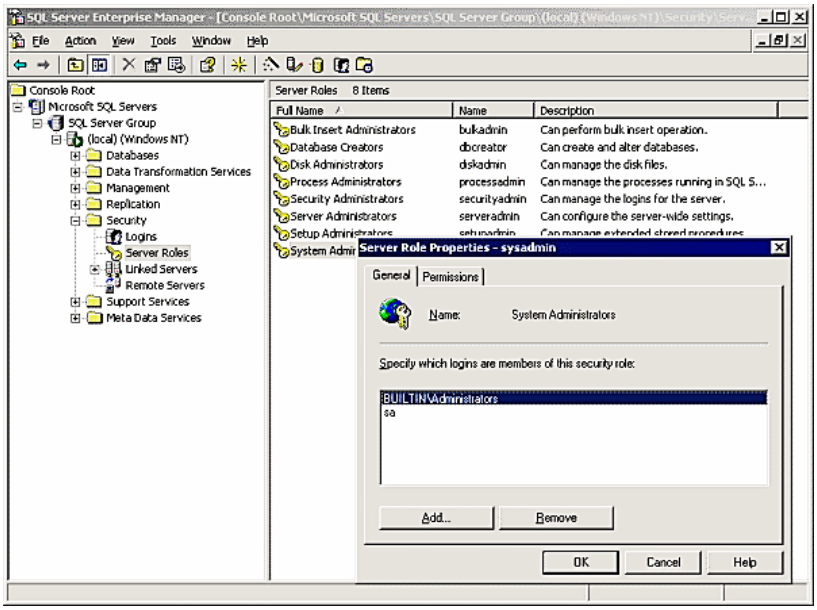
А вот внутренние проблемы из-за неправильной политики безопасности возникают каждый день. Все пользователи входят в систему с правами администратора и могут творить, что угодно. Это действительно проблема, потому что излишние права дают ламерам возможность показать свою безграмотность в полной мере. Поэтому мы будем рассматривать безопасность вне зависимости от того, откуда идет угроза – извне (от хакера) или изнутри (от ушастого юзера).

Чтобы у тебя не было проблем, мы рассмотрим все необходимые основы безопасности не только защиты от хакеров, но и от особо ушастых. В качестве примера я выбрал MS SQL Server, потому что он содержит все, что есть в других базах (Oracle, MySQL и т.д.) и имеет дополнительные возможности по управлению безопасностью. Кто-то может тут подумать, что это делает MS круче. Нет, дополнительные возможности иногда избыточны и только добавляют проблем.

Серверные роли

В Windows и других ОС для управления правами существуют группы и пользователи. С помощью групп мы можем объединять пользователей в кучу и назначать права им всем сразу. Это проще, чем назначать права каждому в отдельности. В базах данных для этих целей существует понятие «роль». Допустим, что 100 пользователей должны иметь право читать данные из определенной таблицы. Давать каждому из них это право напряжесто. Намного проще создать роль, которой разрешено читать, а потом всех нужных пользователей включить в нее. Результат подобен группировке.

В SQL сервере бывает два типа ролей – серверные и баз данных (вторые мы рассмотрим позже). Серверные роли заранее определены, и их изменять нельзя. Открой в Enterprise Manager ветку Security/Server role, и в правой части окна можно будет увидеть список встроенных ролей. Что может делать пользователь соответствующей роли можно определить по описанию.



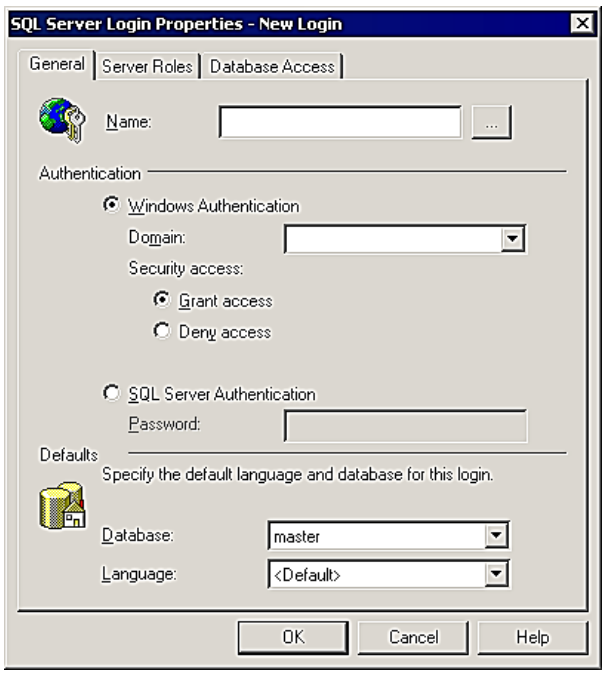
Серверные роли в MS SQL Server и окно редактирования роли

Для добавления уже существующего пользователя в роль, нужно щелкнуть по строке роли дважды и в появившемся окне можно добавлять пользователей в роль или удалять. На закладке Permission более подробно описано, что может делать выделенный пользователь.

Пользователи

Для управления пользователями открой в Enterprise Manager ветку Security/Logins. В правой части появится список всех пользователей сервера. По умолчанию доступ имеют администраторы домена и встроенная учетная запись sa.

Для добавления нового пользователя, щелкни правой кнопкой в пустом месте правой половины окна и в появившемся меню выбери New login. Перед нами откроется окно добавления нового пользователя. В самом верху окна выбирается имя пользователя. Если нужно выбрать уже существующего юзера домена или компьютера, то щелкни по кнопке (…) справа от поля ввода, и ты увидишь окно поиска пользователя в домене.

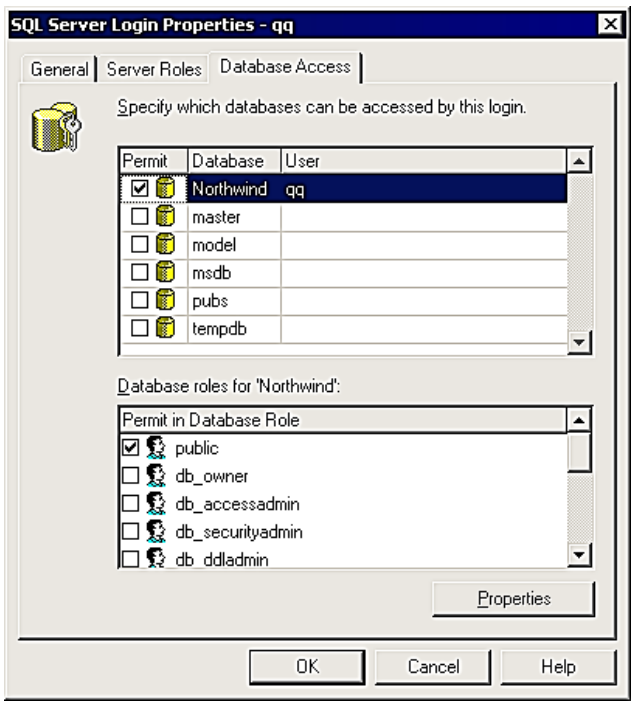


Добавление нового пользователя

Чуть ниже выбирается тип аутентификации – Windows или SQL Server. Если выбрать Windows, то пароль указывать не надо, потому что сервер сам возьмет его из системы. Но зато можно выбрать один из переключателей – Grant access (разрешить доступ) или Deny access (запретить). Во втором случае пользователь будет прописан в базе, но подключиться он не сможет – запрещено однако.

Если выбрать аутентификацию SQL Server, то нужно будет задать пароль, потому что в этом случае, он будет храниться в системных таблицах сервера баз данных. Обрати внимание – даже если в настройках сервера указана только аутентификация Windows, записи SQL сервера создавать разрешено, но вот войти в систему с этими записями будет невозможно.

На закладке Server Roles можно указать, какой серверной роли будет принадлежать юзер. Таким образом, уже на этапе создания можно включить юзеров в нужные роли.



Доступ пользователя к базам данных

На закладке Database Access указываем базы, с которыми может работать юзер. Здесь окно разделено на две части: в верхней половине можно выбирать базу данных, к которой разрешен доступ, а в нижнем списке выбирается роль базы данных. В зависимости от выбранной роли в базе, пользователю будут доступны те или иные права. Один пользователь может входить в несколько ролей.

Давай для примера создадим учетную запись qq, которой будет разрешен доступ к базе данных Northwind. Это стандартная тестовая база данных, которая создается при установке сервера. Сохрани изменения. Теперь открой ветку Databases/Northwind/Users и увидишь список пользователей, которым разрешен доступ к выбранной базе данных. Обрати внимание, что запись qq присутствует здесь. В других базах она будет отсутствовать, потому что к ним доступ нашего нового пользователя запрещен.

Настройка прав доступа к базам данных

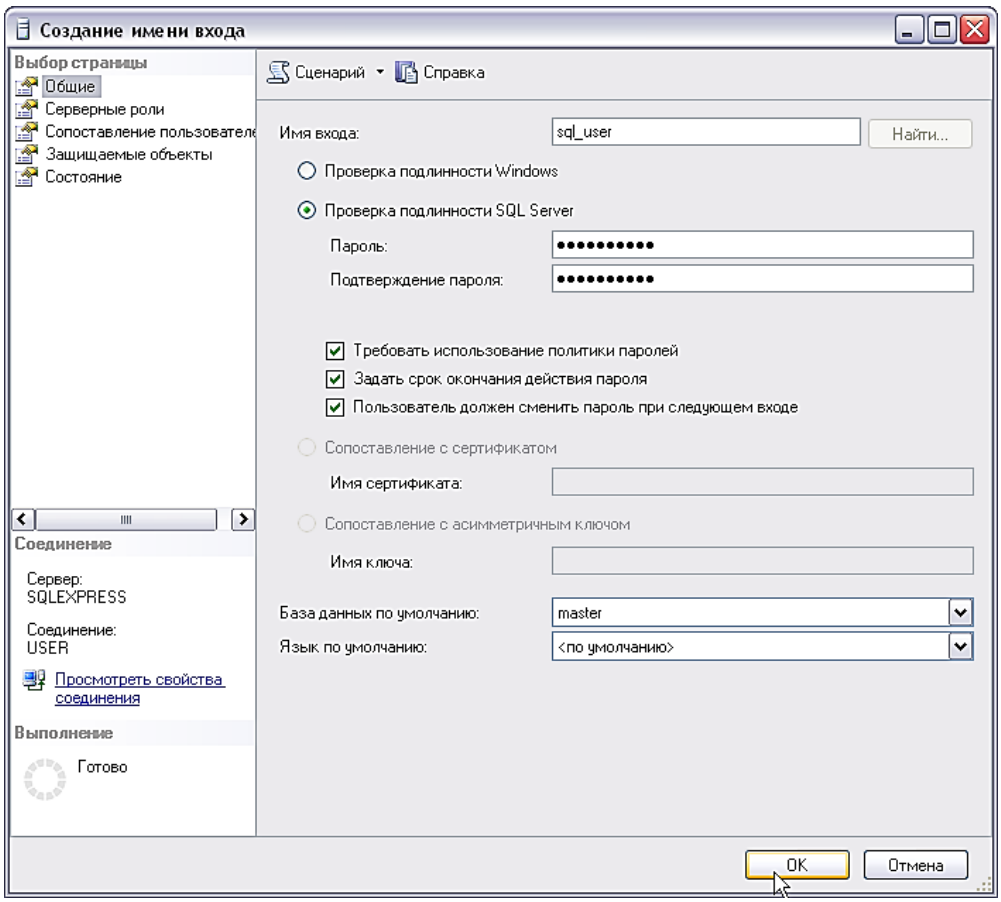
Управление доступом к базам данных можно производить несколькими способами:

– с помощью утилиты DB Администратор, поставляемой в комплекте установки (см. Руководство пользователя, глава Управление доступом к базам данных);

– с помощью SQL Server Management Studio (см. ниже).

Создание пользователей и настройка прав доступа к базам данных в SQL Server Management Studio

Создание и редактирование свойств пользователей осуществляется из раздела *Безопасность* дерева объектов SQL Server Management Studio. Для добавления нового пользователя необходимо выделить раздел *Имена входа*, в контекстном меню выбрать пункт "Создать имя входа…". Откроется окно "Создание имени входа" (Рис. 1).



Задать проверку подлинности пользователя - *Проверка подлинности Windows* или *Проверка подлинности SQL server*.

Для создания пользователя с аутентификацией Windows необходимо выбрать пользователя или группу пользователей по кнопке "Найти…".

Для создания пользователя с аутентификацией SQL необходимо ввести имя пользователя в графе *Имя пользователя*, указать пароль и подтверждение пароля. Пользователь с SQL аутентификацией создается в случае использования SQL аутентификации базы данных (подробнее см. Руководство пользователя, глава Свойства подключения к базе данных).

На странице ***Сопоставление пользователей*** отмечаются галочками базы данных, для которых установлен доступ для данного пользователя и членство в роли в базе данных. Новый пользователь в базе данных создается с правами Пользователя (подробнее о возможностях групп пользователей см. Руководство пользователя, глава Управление доступом к базам данных). Чтобы дать пользователю права Администратора базы данных, необходимо в списке *Членство в роли базы данных для: <имя_базы>* отметить галочкой роль ‘db_owner’.

Также возможно добавление существующего пользователя из раздела *Безопасность → Пользователи* выделенной базы. Для этого необходимо выбрать пункт контекстного меню "Создать пользователя…", задать *Имя пользователя,* ввести вручную или выбрать по кнопке Имя *входа* созданного ранее пользователя. Чтобы дать пользователю права Администратора базы данных, необходимо в списке *Членство в роли базы данных* отметить галочкой роль ‘db_owner’.

Порядок выполнения работы

На языке SQL Server или SQL Server 2015 написать программу для повышение безопасности баз данных, указанным преподавателем.

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Проблемы проверки подлинности безопасности системы СУБД
2. Архитектуры системы безопасности в MS SQL Server
3. Режимы защиты безопасности MS SQL Server

Лабораторная работа №5.Обеспечить повторное использование объектов в базе данных.

Цель работы: Изучение среды языков программирование по повышению безопасности базы данных.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

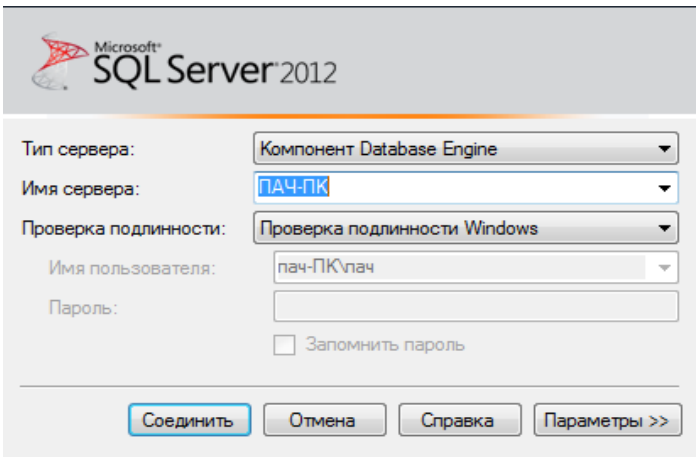
Теоретическая часть

SQL Server 2015 эффективно и быстро выделяет страницы для объектов и повторно использует место на диске, освобожденное после удаления строк. Эти действия происходят внутри системы и используют структуры данных, невидимые для пользователей. Однако эти процессы и структуры иногда упоминаются в сообщениях SQL Server.

Этот раздел посвящен обзору алгоритмов и структур данных, применяемых для выделения места на диске. Он также содержит сведения, необходимые пользователям и администраторам для понимания терминов, употребляемых в сообщениях SQL Server.

Запустил «Среда SQL Server Management Studio» через меню «Пуск».

В появившемся окне указал данные предоставленные на (Рис. 1.1)



В появившемся окне разработки нажал ПКМ на каталоге «Базы данных» и в появившемся контекстном меню выбрал «Создать базу данных…» (Рис. 1.1)

В появившемся окне ввел данные показанные на Рис. 1.2.

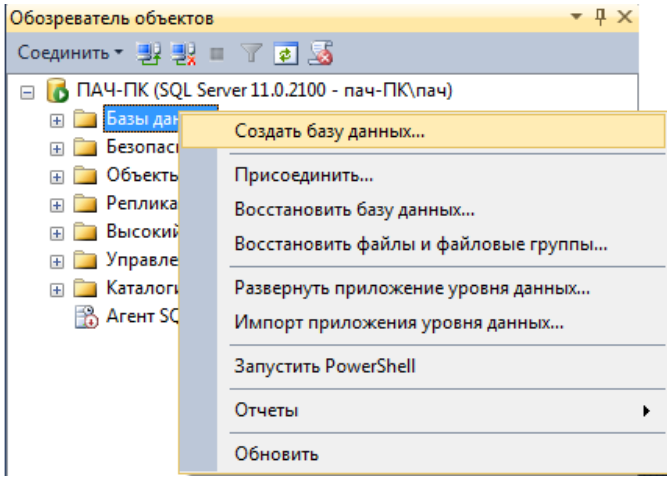


Рис. 1.2

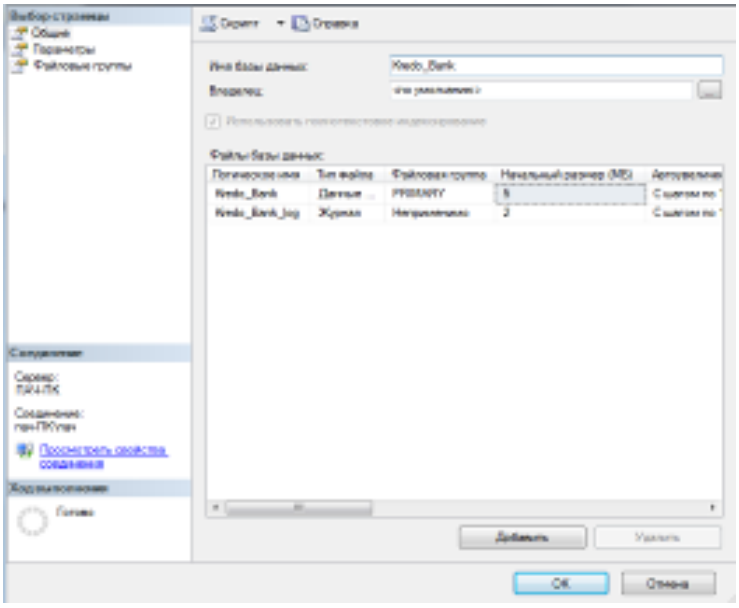


Рис. 1.3.

Создание и заполнение таблиц

При помощи клика ПКМ на каталоге «Таблицы», и выбора из контекстного меню пункта «Создать таблицу…» создал таблицы:

Валюта, в которую вошли такие элементы: Код валюты, Наименование, Обменный курс.

Результат создания показан на Рис. 2.1 и Рис. 2.2.

Имя столбца	Тип данных	Разрешить значения NULL
[Код валюты]	int	☐
Наименование	ntext	☐
[Обменный курс]	smallmoney	☐
		☐

Рис. 2.1

Код валюты	Наименование	Обменный курс
643	Российский рубль	0,2510
840	Доллар США	8,1700
978	Евро	10,9500
NULL	*NULL*	*NULL*

Рис. 2.2

Вкладчики, в которую вошли такие элементы: ФИО вкладчика, Адрес, Телефон, Паспортные данные, Дата вклада, Дата возврата, Код вклада, Сумма вклада, Сумма возврата, Отметка о возврате вклада, Код сотрудника.

Результат создания показан на Рис. 2.3 и Рис. 2.4.

ФИО вкладчика	Адрес	Телефон	Паспортные данные	Дата вклада	Дата возврата	Код вклада	Сумма вклада	Сумма возврата	Отметка о возврат...	Код сотрудника
Мироненко Сергей Дмитриевич	г. Харьков, ул....	+380658697459	МТ458632	2013-10-10	2023-10-10	3	10000,0000	18004,3800	Невозвращённ	18968531
Иванов Петр Русланович	г. Харьков, ул....	+380975236845	МТ789236	2010-01-02	2014-01-02	2	5000,0000	5500,6800	Невозвращённ	12158881
Сидоренко Мария Петровна	г. Харьков, ул....	+380501236983	МН523678	2009-06-05	2012-12-05	5	200000,0000	249057,5300	Возвращённ	45612584
Александрова Наталья Сергеевна	г. Харьков, ул....	+380523698758	МН476325	2010-12-10	2011-06-10	1	2000,0000	2020,1600	Возвращённ	45612584
Ольховский Юрий Андреевич	г. Харьков, ул....	+380256898770	МН145896	2012-05-09	2014-05-09	5	2000,0000	2200,0000	Невозвращённ	35698783
Явтушенко Татьяна Николаевна	г. Харьков, ул....	+380569874566	МТ125874	2012-10-05	2013-10-05	4	50000,0000	52500,0000	Возвращённ	21348962
Струп Дмитрий Александрович	г. Харьков, ул....	+380978564823	МТ145897	2013-11-10	2018-11-10	1	25000,0000	27501,3700	Невозвращённ	35698783
Горулько Виктор Владимирович	г. Харьков, ул....	+380458723684	МН789668	2011-02-05	2026-02-05	3	15000,0000	18013,1500	Невозвращённ	18968531
Ковтун Николай Викторович	г. Харьков, ул....	+380856978523	МТ121566	2002-10-18	2012-10-18	2	50000,0000	75013,7000	Возвращённ	35698783
Набок Мария Николаевна	г. Харьков, ул....	+380123685478	МТ152155	2009-05-21	2013-05-21	4	15000,0000	18002,0500	Возвращённ	45612584

Рис. 2.3

Имя столбца	Тип данных	Разрешить .
[ФИО вкладчика]	varchar(50)	☐
Адрес	ntext	☐
Телефон	varchar(MAX)	☑
[Паспортные данные]	varchar(MAX)	☐
[Дата вклада]	date	☐
[Дата возврата]	date	☐
[Код вклада]	int	☐
[Сумма вклада]	money	☐
[Сумма возврата]	money	☐
[Отметка о возврате вк...	ntext	☐
[Код сотрудника]	int	☐

Рис. 2.4

Сотрудники, в которую вошли такие элементы: Код сотрудника, ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Код должности. Результат создания показан на Рис. 3.5 и Рис. 3.6.

Код сотрудника	ФИО	Возраст	Пол	Адрес	Телефон	Паспортные д...	Код должности
12158881	Стропова Ната...	29	Ж	г. Харьков, ул. ...	+380665898948	MT589678	2570219
18968531	Прокопенко С...	35	М	г. Харьков, ул. ...	+380968527950	MH589326	1013681
18979881	Зеленый Иван ...	29	М	г. Харьков, ул. ...	+380668579504	MH454874	8219838
18998991	Мироненко Ан...	22	Ж	г. Харьков, ул. ...	+380635896828	MT454697	9052369
21348962	Замай Петр Ал...	30	М	г. Харьков, ул. ...	+380506897512	MH415456	2570219
35698783	Струп Дмитри...	26	М	г. Харьков, ул. ...	+380975588688	MT215614	2570219
45612584	Костенко Олег...	23	М	г. Харьков, ул....	+380985671235	MH446545	2570219
47868588	Варжеленко Р...	17	М	г. Харьков, ул. ...	+380968527950	MT458669	2570219
52369875	Яковенко Юри...	19	М	г. Харьков, ул. ...	+380678597950	MT541546	4883207
56987125	Шевченко Дми...	22	М	г. Харьков, ул. ...	+380935856989	MH151447	4883207
89756988	Задорожний Д...	28	М	г. Харьков, ул. ...	+380678958989	MT455455	2570219

Рис. 3.5

Имя столбца	Тип данных	Разрешить ...
[Код сотрудника]	int	☐
ФИО	varchar(50)	☑
Возраст	tinyint	☐
Пол	char(1)	☐
Адрес	ntext	☐
Телефон	varchar(MAX)	☑
[Паспортные данные]	varchar(MAX)	☐
[Код должности]	int	☐

Рис. 3.6

Должности, в которую вошли такие элементы: Код должности, Наименование должности, Оклад, Обязанности, Требования.

Результат создания показан на Рис. 3.7 и Рис. 3.8.

Имя столбца	Тип данных	Разрешить ...
[Код должности]	int	☐
[Наименование должн...	ntext	☐
Оклад	money	☐
Обязаности	ntext	☐
Требования	ntext	☐

Рис. 3.7

Код должности	Наименовани...	Оклад	Обязаности	Требования
1013681	Начальник отд...	32000,0000	Привлечение ...	Высшее эконо...
2570219	Менеджер по ...	8200,0000	Активный пои...	Опыт работы в...
4883207	Кассир	4900,0000	Прием и выда...	Высшее, средн...
8219838	Финансовый к...	16000,0000	Активный пои...	Опыт привлеч...
9052369	Программист ...	14000,0000	Разработка от...	Навыки прогр...

Рис. 3.8

Вклады (Код вклада, Наименование вклада, Минимальный срок вклада, Минимальная сумма вклада, Код валюты, Процентная ставка, Дополнительные условия).

Результат создания показан на Рис. 3.9 и Рис 3.9.1.

Код вклада	Наименовани...	Минимальны...	Минимальна...	Код валюты	Процентная с...	Дополнитель...
1	Вклад до востр...	1	1,0000	840	2	Является попо...
2	Пополняемый...	12	1000,0000	978	5	Счет предусма...
3	Целевой вклад...	120	1000,0000	840	8	Пополняем.
4	Срочный вклад	12	1,0000	643	5	Не пополняем.
5	Валютная рента	1	100000,0000	840	7	Пополняем, к...

Рис. 3.9

	Имя столбца	Тип данных	Разрешить ...
🔑	[Код вклада]	int	☐
	[Наименование вклада]	ntext	☐
	[Минимальный срок вк...	int	☐
	[Минимальная сумма в...	money	☐
	[Код валюты]	int	☐
	[Процентная ставка]	tinyint	☐
	[Дополнительные усло...	ntext	☐

Рис. 3.9.1

Порядок выполнения работы

На языке SQL Server или SQL Server 2015 написать программу для повышение безопасности баз данных, указанным преподавателем.

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Проблемы проверки подлинности безопасности системы СУБД
2. Архитектуры системы безопасности в MS SQL Server
3. Режимы защиты безопасности MS SQL Server

Лабораторная работа №6.Основные характеристики системы рабочих групп пользователей для обеспечения безопасности баз данных.

Цель работы: Изучение основных характеристики системы рабочих групп пользователей для обеспечения безопасности баз данных.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть.

Пользователь БД (user) - это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. На каждом этапе развития базы данных (проектирование, реализация, эксплуатация, модернизация и развитие, полная реорганизация) с ней связаны разные категории пользователей.

Существуют различные категории пользователей:

Конечные пользователи. Это основная категория пользователей, в интересах которых создается БД. В зависимости от особенностей создаваемой БД круг конечных пользователей может различаться. Это могут быть случайные пользователи, которые обращаются за информацией к БД время от времени и регулярные пользователи. В качестве случайных пользователей могут рассматриваться, например, клиенты фирмы, просматривающие каталог продукции или услуг. Регулярными пользователями могут быть сотрудники, которые работают со специально разработанными для них программами, которые обеспечивают автоматизацию их деятельности при выполнении служебных обязанностей.

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

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

Разработчики и администраторы приложений. Это группа пользователей, которая функционирует во время проектирования, создания и реорганизации БД. Администраторы приложений координируют работу разработчиков при разработке конкретного приложения или группы приложений, объединенных в функциональную подсистему.

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

База данных взаимодействует в соответствующей среде со множеством пользователей. Пользователи могут предъявлять противоречивые требования к базе данных. Следовательно, возникает проблема координации деятельности пользователей и

управления целостностью данных и защитой БД. Необходимость решения этой проблемы вызвало необходимость *администрирования* в базы данных.

К основным *функциям* группы администратора БД относят:

1. *Анализ предметной области*: описание предметной области, выявление ограничений целостности, определение статуса (доступности, секретности) данных, определение потребностей пользователей.
2. *Проектирование структуры БД*: описание информационного содержания и внутренней структуры БД.
3. *Задание ограничений целостности при описании структуры БД*:

– определение ограничений целостности, вызванных структурой БД;

– разработка процедур обеспечения целостности БД при вводе и коррекции данных;

– определение ограничений целостности при параллельной работе пользователей в многопользовательском режиме.

4. *Первоначальная загрузка и ведение БД*
5. *Защита данных:*

– определение системы паролей, принципов регистрации пользователей, создание групп пользователей, обладающих одинаковыми правами доступа к данным;

– тестирование системы защиты;

– исследование случаев нарушения системы защиты;

– разработка средств фиксации доступа к данным и попыток нарушения системы защиты

– разработка принципов защиты конкретных данных и объектов проектирования

6. *Обеспечение восстановления БД*: разработка организационных средств архивирования и принципов восстановления БД; разработка дополнительных программных средств и технологических процессов восстановления БД после сбоев.
7. *Анализ обращений пользователей*: сбор статистики по характеру запросов, времени их выполнения.
8. *Анализ эффективности функционирования БД*: анализ показателей функционирования БД, планирование реструктуризации.
9. *Работа с конечными пользователями*: сбор информации об изменении предметной области, об оценке работы БД, обучение и консультирование пользователей
10. *Подготовка и поддержание системных средств: анализ существующих на рынке* программных средств и возможность их использования, проверка работоспособности закупаемых программных средств.
11. *Организационно-методическая работа по проектированию БД*: выбор или создание методики проектирования БД; определение целей и направления развития системы в целом; планирование этапов развития БД; обеспечение возможностей комплексной отладки множества приложений, взаимодействующих с БД и т.д.

Порядок выполнения работы

На языке SQL Server или SQL Server 2015 написать программу для повышение безопасности баз данных, указанным преподавателем.

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Алгоритм проверки аутентификации пользователя в MS SQL Server
2. Режимы защиты безопасности MS SQL Server

Лабораторная работа №7.Защита ключей базы данных SQL Server.

Цель работы: Ознакомление с зашитыми функциями SQL Server и их основными возможностями.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть.

SQL Server используются ключи шифрования для защиты данных, информации об учетных данных и соединениях, которые хранятся в серверной базе данных. SQL Server существует два вида ключей: *симметричный* и *асимметричный*. В симметричных ключах для шифрования и расшифровки данных используется одинаковый пароль. При использовании асимметричных ключей один пароль применяется для шифрования данных (открытый ключ), а другой для расшифровки данных (закрытый ключ).

Используемые в SQL Server ключи шифрования представляют собой сочетание открытых, закрытых и симметричных ключей, которые используются для защиты конфиденциальных данных. Симметричный ключ создается во время инициализации SQL Server, при первом запуске экземпляра SQL Server. Этот ключ используется SQL Server для шифрования конфиденциальных данных, которые хранятся в SQL Server. Открытые и закрытые ключи создаются операционной системой и используются для защиты симметричного ключа. Пара из открытого и закрытого ключей создается для каждого экземпляра SQL Server, который сохраняет конфиденциальные данные в базе данных.

Применения ключей шифрования SQL Server и базы данных

SQL Server поддерживает два основных варианта применения ключей: *главный ключ службы* (SMK) создается на экземпляре SQL Server и для этого экземпляра, а *главный ключ базы данных* (DMK) используется для базы данных.

Главный ключ службы автоматически создается при первом запуске экземпляра SQL Server и используется для шифрования пароля связанного сервера, учетных данных или главного ключа базы данных. Главный ключ службы шифруется с помощью ключа локального компьютера и API-интерфейса защиты данных Windows. Этот API-интерфейс

использует ключ, получаемый из учетных данных Windows, которые соответствуют учетной записи службы для SQL Server и учетным данным компьютера. Главный ключ службы может быть расшифрован лишь той учетной записью службы, под которой он был создан, или участником, имеющим доступ к учетным данным компьютера.

Главный ключ базы данных — это симметричный ключ, который применяется для защиты закрытых ключей сертификатов и асимметричный ключей, которые есть в базе данных. Он также может использоваться для шифрования данных, но из-за ограниченной длины не может применяться на практике для шифрования данных так же широко, как симметричный ключ.

При создании этот ключ зашифровывается с помощью алгоритма «Triple DES» и пользовательского пароля. Чтобы разрешить автоматическое шифрование главного ключа, копия этого ключа зашифровывается с помощью главного ключа службы. Ключ хранится как в базе данных, где используется и в master системной базы данных.

Копия главного ключа базы данных, хранящихся в master Системная база данных автоматически обновляется при каждом изменении главного ключа. Тем не менее, это значение по умолчанию можно изменить с помощью DROP ENCRYPTION BY SERVICE MASTER KEY параметр ALTER MASTER KEY инструкции. Главный ключ базы данных, который не зашифрован с помощью главного ключа службы, следует открывать с помощью инструкции OPEN MASTER KEY и пароля.

Управление ключами SQL Server и базы данных

Управление ключами шифрования заключается в создании новых ключей базы данных, создании резервной копии ключей сервера и базы данных и знании порядка восстановления, удаления и смены ключей.

Чтобы управлять симметричными ключами, можно использовать средства, входящие в SQL Server, для выполнения следующих действий:

- Резервное копирование копии ключа сервера и базы данных, чтобы использовать их при восстановлении установки сервера или в ходе запланированного переноса.
- Восстановление ранее сохраненного ключа в базе данных. Это позволяет новому экземпляру сервера обращаться к существующим данным, которые первоначально шифровались не им.
- Удаление зашифрованных данных из базы данных в маловероятной ситуации, когда не удается обратиться к зашифрованным данным.
- Повторное создание ключей и повторное шифрование данных в маловероятной ситуации, когда ключ становится известен посторонним. Для повышения безопасности следует периодически повторно создавать ключи (например, раз в несколько месяцев) для защиты сервера от атак с целью расшифровки ключа.
- Добавление или удаление экземпляра сервера из масштабного развертывания, когда несколько серверов используют одну базу данных и ключ, допускающий обратимое шифрование для этой базы данных.

Важная информация по безопасности

Для доступа к объектам, защищенным главным ключом службы, необходима учетная запись службы SQL Server, которая использовалась для создания ключа, или учетная запись компьютера. То есть, компьютер привязан к системе, в которой был создан ключ. Можно изменить учетную запись службы SQL Server *или* учетную запись компьютера, не теряя доступа к ключу. Однако если изменить обе учетные записи, доступ к главному ключу службы будет потерян. Если доступ к главному ключу службы будет потерян без одного из этих элементов, то не удастся расшифровать данные и объекты, зашифрованные с помощью первоначального ключа.

Соединения, защищенные с помощью главного ключа службы, не могут быть восстановлены без него.

Для доступа к объектам и данным, защищенным главным ключом базы данных, требуется только пароль, использованный для защиты ключа.

Модель шифрования SQL Server

Модель шифрования SQL Server в основном предоставляет функции управления ключами шифрования, соответствующие стандарту ANSI X9.17. В этом стандарте определены несколько уровней ключей шифрования, использующихся для шифрования других ключей, которые в свою очередь применяются для шифрования собственно данных. В таблице перечислены уровни ключей шифрования SQL Server и ANSI X9.17.

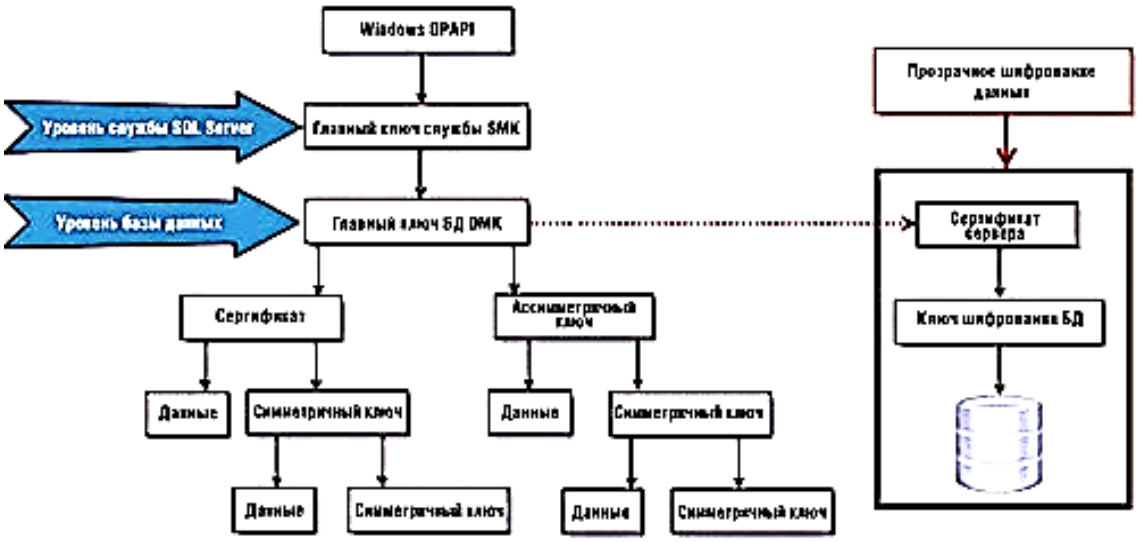
Таблица — Уровни шифрования ключей SQL Server и ANSI X9.17

Уровень SQL Server	Уровень ANSI X9.17	Описание
SMK	Главный ключ	SMK — ключ верхнего уровня, используемый для шифрования DMK. SMK шифруется с применением Windows DPAPI
DMK	Ключ шифрования ключей	DMK — симметричный ключ, используемый для шифрования симметричного ключа, асимметричного ключа и сертификата. Для каждой базы данных может быть определен только один DMK
Симметричные ключи, асимметричные ключи и сертификаты	Ключ данных	Симметричные ключи, асимметричные ключи и сертификаты используются для шифрования данных

Главный ключ службы Service master key(SMK) — ключ верхнего уровня и предок всех ключей в SQL Server. SMK — асимметричный ключ, шифруемый с использованием Windows Data Protection API (DPAPI). SMK автоматически создается, когда шифруется какой-нибудь объект, и привязан к учетной записи службы SQL Server. SMK используется для шифрования главного ключа базы данных Database master key (DMK).

Второй уровень иерархии ключей шифрования — DMK. С его помощью шифруются симметричные ключи, асимметричные ключи и сертификаты. Каждая база данных располагает лишь одним DMK.

Следующий уровень содержит симметричные ключи, асимметричные ключи и сертификаты. Симметричные ключи — основное средство шифрования в базе данных. Microsoft рекомендует шифровать данные только с помощью симметричных ключей. Кроме того, в SQL Server 2015 и более новых версиях есть сертификаты уровня сервера и ключи шифрования базы данных для прозрачного шифрования данных. На рисунке 1 показана иерархия ключей шифрования для SQL Server 2015 и более новых версий.



Иерархия ключей шифрования в SQL Server 2015 и более новых версиях.

После знакомства с иерархией ключей шифрования SQL Server мы рассмотрим способы реализации шифрования, доступные в SQL Server. Также я покажу, как применить некоторые из них.

Шифрование на уровне ячеек

Начиная с SQL Server 2005, можно шифровать или расшифровывать данные на сервере. Делать это можно различными способами. Например, можно шифровать данные в базах данных одним из следующих методов.

– Пароль. Это наименее надежный способ, так как для шифрования и расшифровки данных используется одна и та же парольная фраза. Если хранимые процедуры и функции не зашифрованы, то доступ к парольной фразе возможен через метаданные.

– Сертификат. Этот способ обеспечивает надежную защиту и высокое быстродействие. Сертификат можно связать с пользователем; подписать его необходимо с помощью DMK.

– Симметричный ключ. Достаточно надежен, удовлетворяет большинству требований к безопасности данных и обеспечивает достаточное быстродействие. Для шифрования и расшифровки данных используется один ключ.

– Асимметричный ключ. Обеспечивает надежную защиту, так как применяются различные ключи для шифрования и расшифровки данных. Однако это негативно влияет на быстродействие. Специалисты Microsoft не рекомендуют использовать его для шифрования крупных значений. Асимметричный ключ может быть подписан с использованием DMK или создан с помощью пароля.

SQL Server располагает встроенными функциями для шифрования и расшифровки на уровне ячеек. Функции шифрования:

– ENCRYPTBYKEY, использует симметричный ключ для шифрования данных;

– ENCRYPTBYCERT, использует открытый ключ сертификата для шифрования данных;

– ENCRYPTBYPASSPHRASE, использует парольную фразу для шифрования данных;

– ENCRYPTBYASYMKEY, использует асимметричный ключ для шифрования данных.

Функции расшифровки:

- DECRYPTBYKEY, использует симметричный ключ для расшифровки данных;
- DECRYPTBYCERT, использует открытый ключ сертификата для расшифровки данных;
- DECRYPTBYPASSPHRASE, использует парольную фразу для расшифровки данных;
- DECRYPTBYASYMKEY, использует асимметричный ключ для расшифровки данных;
- DECRYPTBYKEYAUTOASYMKEY, использует асимметричный ключ, который автоматически расшифровывает сертификат.

SQL Server располагает двумя системными представлениями, с помощью которых можно получить метаданные для всех симметричных и асимметричных ключей, существующих в экземпляре SQL Server. Как видно из названий, sys.symmetric_keys возвращает метаданные для симметричных, а sys.asymmetric_keys — для асимметричных ключей. Еще одно полезное представление — sys.openkeys. В этом представлении каталога содержится информация о ключах шифрования, открытых в текущем сеансе.

Практическая часть

Демонстрация шифрования на уровне ячеек

Далее в статье будет показано, как использовать некоторые функции шифрования, расшифровки и представления. Но сначала пройдем по этапам создания тестовой базы данных, в которой есть таблица с номерами кредитных карт.

Подготовка. В первую очередь создайте базу данных EncryptedDB с помощью среды SQL Server Management Studio (SSMS) или выполнив программный код T-SQL:

```
USE [master]
GO
CREATE DATABASE [EncryptedDB]
GO
```

Затем запустите код T-SQL для создания таблицы с именем CreditCardInformation в базе данных EncryptedDB:

```
USE [EncryptedDB]
GO
CREATE TABLE [dbo].[CreditCardInformation]
([PersonID] [int] PRIMARY KEY,
[CreditCardNumber] [varbinary](max))
GO
```

Эта таблица будет содержать ложную информацию о кредитных картах. Номера кредитных карт будут сохранены в столбце двоичных переменных, потому что они будут шифроваться.

Затем используйте следующий программный код для создания главного ключа DMK базы данных EncryptedDB, шифруемого с помощью парольной фразы $tr0nGPa$$w0rd:

```
USE [EncryptedDB]
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '$tr0nGPa$$w0rd'
GO
```

В первой демонстрации данные шифруются с использованием симметричного ключа, который будет зашифрован с помощью асимметричного ключа. Для этого необходимо создать асимметричный ключ, зашифровать его парольной фразой $tr0nGPa$$w0rd, создать симметричный ключ и зашифровать симметричный ключ с помощью только что созданного асимметричного ключа. Выполнить эти задачи можно, запустив программный код в листинге1.

Теперь мы можем приступить к шифрованию данных. Для этого необходимо сначала открыть симметричный ключ, только что созданный с помощью команды OPEN SYMMETRIC KEY, за которой следует имя симметричного ключа. Затем вы указываете, что нужно расшифровать его с использованием заданного асимметричного ключа. Программный код выглядит следующим образом:

```
USE [EncryptedDB]
GO
OPEN SYMMETRIC KEY MySymmetricKey
DECRYPTION BY ASYMMETRIC KEY MyAsymmetricKey
WITH PASSWORD = 'StrongPa$$w0rd!'
GO
```

После выполнения этого кода направьте запрос в представление sys.openkeys, чтобы убедиться, что ключ открыт:

```
USE [EncryptedDB]
GO
SELECT * FROM [sys].[openkeys]
```

Будут получены результаты, аналогичные показанным на рисунке 2. Наконец, необходимо ввести несколько номеров кредитных карт в таблицу CreditCardInformation, запустив код из листинга 2. Затем направьте запрос к таблице CreditCardInformation:

```
USE [EncryptedDB]
GO
SELECT * FROM [dbo].[ CreditCardInformation]
```

Как показано на рисунке 3, все данные в столбце CreditCardNumber представлены в двоичном формате. С помощью функции DECRYPTBYKEY можно просмотреть зашифрованные данные:

```
USE [EncryptedDB]
GO
SELECT [PersonID]
CONVERT([nvarchar](32), DECRYPTBYKEY(CreditCardNumber))
AS [CreditCardNumber]
FROM [dbo].[CreditCardInformation]
GO
```

Порядок выполнения работы

На языке SQL Server или SQL Server 2015 написать программу для повышение безопасности баз данных, указанным преподавателем.

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Проблемы проверки подлинности безопасности системы СУБД
2. Архитектуры системы безопасности в MS SQL Server
3. Режимы защиты безопасности MS SQL Server

Лабораторная работа №8.Создание цифровых сертификатов и ключей.

Цель работы: Изучение основных способов создание цифровых сертификатов и ключей.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть

Цифровые сертификаты представляют собой эффективное средство определения подлинности, которое можно применять при аутентификации пользователей в процессе регистрации, для обеспечения безопасного обмена информацией в Internet и определения происхождения программного обеспечения. Одной из главных областей применения цифровых сертификатов является шифрование данных и электронная подпись сообщений электронной почты. С помощью последней получатели письма могут удостовериться, что письмо было послано именно данным отправителем и не изменено. Разобравшись с технологией цифровых сертификатов, можно научиться определять оптимальный тип иерархии центров сертификации CA (Certificate Authority) и какой вариант CA использовать — внешний или внутренний. Для тех, кто решит, что лучше использовать собственный CA, ниже приводится описание связанных с этим проблем, а также действий, которые следует предпринять для создания сервера сертификации, необходимого для выдачи собственных сертификатов. Начнем с краткого обзора частей инфраструктуры открытых ключей PKI (Public Key Infrastructure).

Цифровой сертификат – открытый ключ с встроенной в нем информацией о владельце секретного ключа (имя человека или название организации, адрес, эл. почта и т. д.).

Инфраструктура с открытым ключем (PKI – Public Key Infrastructure) это система для создания и управления цифровыми сертификатами.

Удостоверяющий центр (Validation Authority) подписывает своим ключем цифровой сертификат, подтверждая, что он действительно принадлежит тому, чьи данные он содержит.

Имеются несколько известных всем удостоверяющих центров (Validation Authority), подписи которых распознаются браузерами и операционными системами как заслуживающие доверия. То есть, чтобы, например, создать сайт, который сможет безопасно собирать и обрабатывать персональную информацию пользователей, надо создать секретный и открытый ключи, послать открытый ключ вместе с необходимой информацией о владельце в удостоверяющий центр, получить подписанный ими цифровой сертификат, разместить его на сайте.

Подтверждение цифрового сертификата стоит денег, в зависимости от крутости удостоверяющего центра, иногда весьма немалых, поэтому для тестовых сайтов можно сделать само подписанный сертификат. При этом, браузер посетителя будет сильно ругаться, что сайт поддельный а сертификат подписан неизвестно кем (что есть правда), и ему нет доверия, но безопасное соединение работать будет нормально. Сертификат представляет собой объект безопасности с цифровой подписью, который содержит открытый (и необязательно закрытый) ключ для SQL Server. Можно использовать сертификаты, сформированные внешними средствами, или сертификаты, созданные SQL Server.

SQL Server Сертификаты соответствуют стандарту сертификатов IETF X.509v3.

Сертификаты полезны благодаря возможности выполнить как экспорт ключей в файлы сертификатов X.509, так и импорт из них. Синтаксис, применяемый при создании сертификатов, позволяет вводить параметры создания для сертификатов, например дату окончания действия.

Использование сертификата в SQL Server. Сертификаты могут использоваться для защиты соединений, при зеркальном отображении баз данных, подписывании пакетов и других объектов, шифровании данных или соединений. В следующей таблице перечислены дополнительные ресурсы для сертификатов в SQL Server.

Раздел	Описание
CREATE CERTIFICATE (Transact-SQL)	Содержит описание команды создания сертификатов.
Определение источника пакетов с помощью цифровых подписей	Содержит сведения об использовании сертификатов для подписывания программных пакетов.
Использование сертификатов для конечной точки зеркального отображения базы данных (Transact-SQL)	Содержит сведения об использовании сертификатов с зеркальным отображением базы данных.

Асимметричные ключи. Асимметричные ключи используются для защиты симметричных ключей. Их можно также использовать для ограниченного шифрования данных и цифрового подписывания объектов базы данных. Асимметричный ключ состоит из закрытого ключа и соответствующего открытого ключа. Дополнительные сведения об асимметричных ключах см. в разделе CREATE ASYMMETRIC KEY (Transact-SQL).

Асимметричные ключи можно импортировать из файлов ключа для строгого имени, но их нельзя экспортировать. Они также не имеют даты окончания действия. Асимметричные ключи нельзя применять для шифрования соединений.

Использование асимметричного ключа в SQL Server. Асимметричные ключи можно использовать для защиты данных или подписывания незашифрованного текста. В следующей таблице перечислены дополнительные ресурсы для асимметричных ключей в SQL Server.

Раздел	Описание
CREATE ASYMMETRIC KEY (Transact-SQL)	Содержит описание команды создания асимметричных ключей.
SIGNBYASYMKEY (Transact-SQL)	Отображает параметры для подписывания объектов.

Инструменты. Microsoft предоставляет средства и программы, которые создают сертификаты и файлы ключа для строгого имени. Эти средства обеспечивают более гибкий процесс создания ключа, чем синтаксис SQL Server. С помощью этих средств можно создать ключи RSA большей длины, а затем импортировать их в SQL Server. В следующей таблице показано, где найти эти средства.

Инструмент	Назначение
makecert	Создает сертификаты.
sn	Создает строгие имена для симметричных ключей.

Практическая часть

Инструкция CREATE CERTIFICATE (Transact-SQL). Добавляет сертификат в базу данных SQL Server. Эта функция несовместима с экспортом базы данных с использованием платформы приложения уровня данных (DACFx). Необходимо удалить все сертификаты перед экспортом.

Синтаксис

```
-- Syntax for SQL Server and Azure SQL Database

CREATE CERTIFICATE certificate_name [ AUTHORIZATION user_name ]
   { FROM <existing_keys> | <generate_new_keys> }
   [ ACTIVE FOR BEGIN_DIALOG =  { ON | OFF } ]

<existing_keys> ::=
   ASSEMBLY assembly_name
   | {
      [ EXECUTABLE ] FILE = 'path_to_file'
      [ WITH PRIVATE KEY ( <private_key_options> ) ]
     }
   | {
      BINARY = asn_encoded_certificate
      [ WITH PRIVATE KEY ( <private_key_options> ) ]
     }
<generate_new_keys> ::=
   [ ENCRYPTION BY PASSWORD = 'password' ]
   WITH SUBJECT = 'certificate_subject_name'
   [ , <date_options> [ ,...n ] ]

<private_key_options> ::=
     {
      FILE = 'path_to_private_key'
       [ , DECRYPTION BY PASSWORD = 'password' ]
       [ , ENCRYPTION BY PASSWORD = 'password' ]
     }
   |
     {
      BINARY = private_key_bits
       [ , DECRYPTION BY PASSWORD = 'password' ]
       [ , ENCRYPTION BY PASSWORD = 'password' ]
     }
```

```
<date_options> ::=
   START_DATE = 'datetime' | EXPIRY_DATE = 'datetime'
-- Syntax for Azure SQL Data Warehouse and Parallel Data Warehouse

CREATE CERTIFICATE certificate_name
   { <generate_new_keys> | FROM <existing_keys> }
   [ ; ]

<generate_new_keys> ::=
   WITH SUBJECT ='certificate_subject_name'
   [ , <date_options> [ ,...n ] ]

<existing_keys> ::=
   {
    FILE ='path_to_file'
    WITH PRIVATE KEY
      (
       FILE ='path_to_private_key'
       , DECRYPTION BY PASSWORD ='password'
      )
   }

<date_options> ::=
   START_DATE ='datetime' | EXPIRY_DATE ='datetime'
```

Аргументы

имя_сертификата

— Это имя для сертификата в базе данных.

АВТОРИЗАЦИЯ *имя_пользователя*

— Это имя пользователя, которому принадлежит этот сертификат.

СБОРКА *assembly_name*

Указывает подписанную сборку, уже загруженную в базу данных.

[ИСПОЛНЯЕМЫЙ ФАЙЛ] ФАЙЛ = "*путь_к_файлу*"

Указывает полный путь, включающий имя файла, к файлу, зашифрованному по правилам DER и содержащему сертификат. Если используется параметр EXECUTABLE, то файл является библиотекой DLL, заверенной с использованием данного сертификата. *путь_к_файлу* может быть локальным путем или UNC-путь в сетевую папку. К файлу осуществляется в контексте безопасности SQL Server учетной записи службы. Эта учетная запись должна иметь соответствующие разрешения на доступ в файловой системе.

WITH PRIVATE KEY

Указывает, что закрытый ключ сертификата загружен в SQL Server. Это предложение действительно лишь в случае, когда сертификат создается из файла. Для загрузки закрытого ключа сборки, используйте ALTER CERTIFICATE.

ФАЙЛ = "*path_to_private_key*"

Указывает полный путь к закрытому ключу, включая имя файла. *path_to_private_key* может быть локальным путем или UNC-путь в сетевую папку. К файлу осуществляется в контексте безопасности SQL Server учетной записи службы. Эта учетная запись должна иметь соответствующие разрешения на доступ в файловой системе.

Примечание

Этот параметр недоступен в автономной базе данных.

asn_encoded_certificate

Биты закодированного сертификата ASN, указанного в качестве двоичной константы.

ДВОИЧНЫЙ =*private_key_bits*

Область применения: начиная с SQL Server 2012 до SQL Server 2017.

Биты закрытого ключа, указанного в качестве двоичной константы. Биты могут находиться в зашифрованном виде. Если они зашифрованы, пользователь должен предоставить пароль для расшифровки. Проверка политики паролей для данного пароля не выполняется. Биты закрытого ключа должны быть в формате файла PVK.

DECRYPTION BY PASSWORD = "*key_password*"

Указывает пароль, необходимый для расшифровки закрытого ключа, получаемого из файла. Это предложение необязательно, если закрытый ключ защищен пустым паролем. Не рекомендуется сохранять закрытый ключ в файл без защиты паролем. Если требуется пароль, но пароль не указан, инструкция завершится ошибкой.

ENCRYPTION BY PASSWORD = "*пароль*"

Указывает пароль, используемый для шифрования закрытого ключа. Этот аргумент нужно использовать лишь в том случае, если необходимо зашифровать сертификат с помощью пароля. Если это предложение опущено, закрытый ключ шифруется с помощью главного ключа базы данных.

пароль должен соответствовать требованиям политики паролей Windows компьютера, на котором выполняется экземпляр SQL Server. Дополнительные сведения см. в разделе Password Policy.

Тема = "*certificate_subject_name*"

Термин *субъекта* ссылается на поле в метаданных сертификата, как определено в стандарте X.509. Субъект должен быть не более 64 символов, и это ограничение действует для SQL Server в Linux. Для SQL Server в Windows, субъекта может содержать до 128 символов. Вопросы, которые больше 128 символов усекаются, если они хранятся в каталоге, но в большом двоичном объекте (BLOB), содержащий сертификат хранит полное имя субъекта.

Start_date = "*datetime*"

Дата, начиная с которой сертификат действителен. Если не указан, START_DATE устанавливается равной текущей дате. Значение START_DATE имеет формат времени UTC. Это значение можно указать в любом формате, который можно преобразовать в формат даты и времени.

EXPIRY_DATE = "*datetime*"

Дата истечения срока действия сертификата. Если не указана, EXPIRY_DATE будет присвоено через год после START_DATE. Значение EXPIRY_DATE имеет формат времени UTC. Это значение можно указать в любом формате, который можно преобразовать в формат даты и времени. SQL ServerКомпонент Service Broker проверяет дату истечения срока действия. Однако срок действия не выполняется, когда этот сертификат используется для шифрования.

ACTIVE FOR BEGIN_DIALOG = { **ON** | {OFF}

Делает сертификат доступным для инициатора диалога с компонентом Компонент Service Broker. Значение по умолчанию — ON.

Замечания

Сертификат — это защищаемый объект уровня базы данных, соответствующий стандарту X.509 и поддерживающий поля X.509 V1. Инструкция CREATE CERTIFICATE может загрузить сертификат из файла или сборки. Она также может создать пару ключей и самостоятельно подписанный сертификат.

Закрытый ключ должен быть <= 2500 байт в зашифрованном виде. Закрытые ключи, созданные SQL Server 1024 бит long с использованием SQL Server 2014 и 2048 бит

длиннее начиная с SQL Server 2016. Закрытые ключи, импортированные из внешнего источника, имеют минимальную длину в 384 бит и максимальную длину в 4 096 бит. Длина импортируемого закрытого ключа должна быть кратной 64 бит. Для сертификатов, используемых для прозрачного шифрования данных, размер закрытого ключа ограничен 3456 битами.

Хранится весь серийный номер сертификата, но отображаются только первые 16 байтов в представлении каталога sys.certificates.

Всего поля издателя сертификата сохраняется, но только первые 884 байты sys.certificates представления каталога.

Закрытый ключ должен соответствовать открытый ключ, заданный *имя_сертификата*.

При создании сертификата из контейнера загрузка закрытого ключа необязательна. Однако в случае, когда SQL Server создает самостоятельно подписанный сертификат, закрытый ключ создается всегда. По умолчанию закрытый ключ шифруется главным ключом базы данных. Если главный ключ базы данных не существует, а пароль не указан, инструкция завершается ошибкой.

Параметр ENCRYPTION BY PASSWORD не требуется, если закрытый ключ зашифрован главным ключом базы данных. Используйте этот параметр только в том случае, если закрытый ключ зашифрован с паролем. Если пароль не указан, закрытый ключ будет зашифрован с использованием главного ключа базы данных. Если не удается открыть главный ключ базы данных, отсутствие этого предложения вызовет ошибку.

Если закрытый ключ зашифрован главным ключом базы данных, указывать пароль расшифровки не требуется.

Примечание

Встроенные функции шифрования и цифровой подписи не проверяют даты истечения срока действия сертификатов. Пользователи этих функций должны самостоятельно определять, когда следует проверять сроки действия сертификатов.

Двоичное описание сертификата можно создать с помощью CERTENCODED (Transact-SQL) и CERTPRIVATEKEY (Transact-SQL) функции. Пример, использующий **CERTPRIVATEKEY** и **CERTENCODED** для копирования сертификата в другую базу данных, см. пример Б в разделе CERTENCODED (Transact-SQL) .

Permissions

Требуется разрешение CREATE CERTIFICATE в базе данных. Только имена входа Windows, SQL Server сертификаты могут принадлежать имена входа и ролям приложений. Сертификаты не могут принадлежать группам и ролям.

Порядок выполнения работы

На языке SQL Server или SQL Server 2015 написать программу для повышение безопасности баз данных, указанным преподавателем.

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Примеры для выполнения работы

А. Создание само заверяющего сертификата

В следующем примере будет создан сертификат под названием Shipping04. Закрытый ключ этого сертификата защищен паролем.

```
CREATE CERTIFICATE Shipping04
  ENCRYPTION BY PASSWORD = 'pGFD4bb925DGvbd2439587y'
  WITH SUBJECT = 'Sammamish Shipping Records',
  EXPIRY_DATE = '20201031';
GO
```

Б. Создание сертификата из файла

В следующем примере будет создан сертификат в базе данных путем загрузки пары ключей из файлов.

```
CREATE CERTIFICATE Shipping11
   FROM FILE = 'c:\Shipping\Certs\Shipping11.cer'
   WITH PRIVATE KEY (FILE = 'c:\Shipping\Certs\Shipping11.pvk',
   DECRYPTION BY PASSWORD = 'sldkflk34et6gs%53#v00');
GO
```

В. Создание сертификата из заверенного исполняемого файла

```
CREATE CERTIFICATE Shipping19
   FROM EXECUTABLE FILE = 'c:\Shipping\Certs\Shipping19.dll';
GO
```

Возможно также создание сборки из файла библиотеки dll, а затем создание сертификата из сборки.

```
CREATE ASSEMBLY Shipping19
   FROM ' c:\Shipping\Certs\Shipping19.dll'
   WITH PERMISSION_SET = SAFE;
GO
CREATE CERTIFICATE Shipping19 FROM ASSEMBLY Shipping19;
GO
```

Г. Создание само заверяющего сертификата

В следующем примере создается сертификат под названием Shipping04 без указания пароль шифрования. В этом примере можно использовать с Хранилище данных SQL Azure и Параллельное хранилище данных.

```
CREATE CERTIFICATE Shipping04
  WITH SUBJECT = 'Sammamish Shipping Records';
GO
```

Контрольные вопросы:

1. Проблемы проверки подлинности безопасности системы СУБД
2. Архитектуры системы безопасности в MS SQL Server
3. Режимы защиты безопасности MS SQL Server

Лабораторная работа №9.Свойства резервного копирования ключей базы данных

Цель работы: Изучение создание и использование резервного копирование ключей базы данных.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть

Для создания резервных копий базы данных можно составить план обслуживания базы данных. Если задача резервного копирования определяется с помощью среды SQL Server Management Studio, можно создать соответствующий скрипт Transact-QLBACKUP, нажав кнопку Скрипт и выбрав место назначения для этого скрипта.

Параметры

Резервный набор данных

Параметры панели **Резервный набор данных** позволяют указать необязательные сведения о резервном наборе данных, созданном операцией резервного копирования.

Название

Укажите имя резервного набора данных. Система автоматически предложит имя по умолчанию на основе имени базы данных и типа резервной копии.

Description

Вводить описание резервного набора данных.

Срок действия резервного набора данных истекает

Выбирать один из следующих параметров истечения срока действия. Этот параметр будет отключен, если в качестве назначения резервного копирования выбран URL-адрес.

Далее указать количество дней до истечения срока действия резервного набора данных, после чего его можно будет перезаписать. Это значение может быть задано в диапазоне от 0 до 99 999 дней. Значение 0 означает, что срок действия резервного набора данных не ограничен.

Значение по умолчанию для срока действия резервного набора данных задается параметром. Срок хранения носителей резервных копий по умолчанию (дней). Чтобы задать этот параметр, щелкните правой кнопкой мыши имя сервера в обозревателе объектов и выберите пункт Свойства, а затем страницу Параметры базы данных в диалоговом окне Свойства сервера.

Вкл. Укажите дату истечения срока действия резервного набора данных, после которого набор можно будет перезаписать.

Сжатие

SQL Server 2015 Enterprise (и более поздние версии) поддерживает сжатие резервной копии.

Установка сжатия резервной копии

В выпуске SQL Server 2015 Enterprise (или более поздней версии) выберите одно из следующих значений сжатия резервных копий.

Использовать параметр сервера по умолчанию Щелкнуть для использования настроек уровня сервера, установленных по умолчанию.

Значения по умолчанию устанавливаются в параметре конфигурации сервера backup compression default. Сведения о том, как просмотреть текущую настройку этого параметра, см. в разделе Параметр конфигурации сервера "Просмотр или настройка параметра сжатия резервных копий по умолчанию".

Сжимать резервные копии Щелкните для сжатия резервной копии, независимо от уровня сервера по умолчанию.

** Важно. ** По умолчанию сжатие существенно повышает загрузку ЦП, что может помешать выполнению других операций. Поэтому может потребоваться создать сжатые резервные копии с низким приоритетом в сеансе, для которого использование ЦП ограничивается регулятором ресурсов. Дополнительные сведения см. ниже в подразделе

Использование регулятора ресурсов для ограничения загрузки ЦП при сжатии резервной копии (Transact-SQL).

Не сжимать резервные копии щелкнуть для создания резервной копии без сжатия, независимо от уровня сервера по умолчанию.

Шифрование

Для создания шифрованной резервной копии установите флажок Зашифровать файл резервной копии. Выберите алгоритм шифрования для шага шифрования и выберите сертификат или асимметричный ключ из списка существующих сертификатов или асимметричных ключей. Доступны следующие алгоритмы шифрования:

- AES 128
- AES 192
- AES 256
- Triple DES

Параметр шифрования отключен, если вы решили присоединить резервную копию к существующему резервному набору данных.

Рекомендуется регулярно создавать резервные копии сертификата или ключей и сохранять их в местоположении, отличном от местоположения шифруемой резервной копии.

Поддерживаются только ключи, относящиеся к расширенному управлению ключами.

Практическая часть

Службы SQL Server 2014 Integration Services (SSIS)включает в себя базу данных SSISDB. Создайте запрос представления в базе данных SSISDB для просмотра объектов, настроек и рабочих данных, которые хранятся в каталоге **SSISDB**. Этот раздел содержит инструкции для выполнения резервного копирования и восстановления базы данных.

В каталоге **SSISDB** хранятся пакеты, которые развернуты на сервере службы Службы Integration Services. Дополнительные сведения о каталоге см. в разделе Каталог служб SSIS.

Создание резервной копии базы данных служб SSIS

1. Открывать среду SQL Server Management Studio и установите соединение с экземпляром SQL Server.

2. Создайте резервную копию главного ключа для базы данных SSISDB с помощью инструкции BACKUP MASTER KEY Transact-SQL. Ключ хранится в указанном файле. Используйте пароль для шифрования главного ключа базы данных в файле.

Дополнительные сведения об инструкции см. в разделе BACKUP MASTER KEY (Transact-SQL).

В следующем примере главный ключ экспортируется в файл c:\temp directory\RCTestInstKey. Пароль LS2Setup! используется для шифрования главного ключа.

- backup master key to file = 'c:\temp\RCTestInstKey'
- encryption by password = 'LS2Setup!'

3. Выполните резервное копирование базы данных SSISDB с помощью диалогового окна **Создание резервной копии базы данных** в SQL Server Management Studio. Дополнительные сведения см. в разделе Как создать резервную копию базы данных (среда SQL Server Management Studio).

4. Создайте скрипт CREATE LOGIN для ## MS_SSISServerCleanupJobLogin ##, выполнив следующие действия. Дополнительные сведения см. в разделе CREATE LOGIN (Transact-SQL).

A) В обозревателе объектов среды SQL Server Management Studioразверните узел **Безопасность**, а затем узел **Имена входа**.

B) Щелкните правой кнопкой мыши **##MS_SSISServerCleanupJobLogin##**и выберите **Внести в скрипт имена входа как> СОЗДАТЬ в > В новом окне редактора запросов**.

5. Если планируется восстановление базы данных SSISDB из копии на экземпляре SQL Server, где каталог SSISDB еще не создан, создайте скрипт CREATE PROCEDURE для sp_ssis_startup следующим образом. Дополнительные сведения см. в статье CREATE PROCEDURE (Transact-SQL).

A) В обозревателе объектов разверните узел **Базы данных**, а затем узел **master** > **Программирование** > **Хранимые процедуры**.

B) Щелкните правой кнопкой мыши **dbo.sp_ssis_startup**и выберите **Внести в скрипт хранимые процедуры как> СОЗДАТЬ в > В новом окне редактора запросов**.

6. Убедитесь, что агент SQL Server запущен.

7. Если планируется восстановление базы данных SSISDB из копии на экземпляре SQL Server, где каталог SSISDB еще не создан, создайте скрипт для задания обслуживания сервера SSIS следующим образом. Скрипт создается в агенте SQL Server автоматически при создании каталога SSISDB. Задание помогает очистить журналы операции с вышедшим сроком сохранения и удалить старые версии проектов.

A) В обозревателе объектов разверните узел **Агент SQL Server**, а затем узел **Задания**

B) Щелкнуть правой кнопкой мыши задание по обслуживанию служб SSIS и выберите **Внести в скрипт задание как> СОЗДАТЬ в> В новом окне редактора запросов**.

Восстановление базы данных служб SSIS

1. Если база данных SSISDB восстанавливается из копии в экземпляре SQL Server, где каталог SSISDB никогда не создавался, включите среду CLR с помощью хранимой процедуры sp_configure. Дополнительные сведения см. в разделах sp_configure (Transact-SQL) и Параметр clr enabled.

- use master
- sp_configure 'clr enabled', 1
- reconfigure

2. Если база данных SSISDB восстанавливается из копии на экземпляре SQL Server, где каталог SSISDB никогда не создавался, создайте асимметричный ключ и имя входа из асимметричного ключа и предоставьте разрешение UNSAFE для имени входа.

Create Asymmetric key MS_SQLEnableSystemAssemblyLoadingKey
FROM Executable File = 'C:\Program Files\Microsoft SQL Server\110\DTS\Binn\Microsoft.SqlServer.IntegrationServices.Server.dll'

Хранимые процедуры CLR Службы Integration Services требуют предоставления разрешения UNSAFE для имени входа, поскольку имени входа необходим дополнительный доступ к ресурсам, на которые существуют ограничения, например API-интерфейс Microsoft Win32. Дополнительные сведения о коде разрешения UNSAFE см. в разделе Creating an Assembly.

- Create Login MS_SQLEnableSystemAssemblyLoadingUser
- FROM Asymmetric key MS_SQLEnableSystemAssemblyLoadingKey
- Grant unsafe Assembly to MS_SQLEnableSystemAssemblyLoadingUser

3. Восстановите базу данных SSISDB из резервной копии с помощью диалогового окна **Восстановление базы данных** в SQL Server Management Studio. Дополнительные сведения см. в следующих разделах:

- Восстановление базы данных (страница "Общие")
- Восстановление базы данных (страница "Файлы")
- Восстановление базы данных (страница "Параметры")

4. Выполните скрипт, созданный в разделе Создание резервной копии базы данных служб SSIS для ##MS_SSISServerCleanupJobLogin##, sp_ssis_startup и заданий по обслуживанию служб SSIS. Убедитесь, что агент SQL Server запущен.

5. Выполнить следующую инструкцию для установки автоматического выполнения процедуры sp_ssis_startup. Дополнительные сведения см. в разделе sp_procoption (Transact-SQL).

EXEC sp_procoption N'sp_ssis_startup','startup','on'

6. Сопоставить пользователя SSISDB ##MS_SSISServerCleanupJobUser## (база данных SSISDB) с ##MS_SSISServerCleanupJobLogin## с помощью диалогового окна **Свойства имени входа** в среде SQL Server Management Studio.

7. Восстановите главный ключ с помощью одного из следующих методов. Дополнительные сведения о шифровании см. в разделе Encryption Hierarchy.

Метод 1

Используйте этот метод при наличии резервной копии главного ключа базы данных и пароля, используемого для шифрования этого ключа.

- Restore master key from file = 'c:\temp\RCTestInstKey'
- Decryption by password = 'LS2Setup!' -- 'Password used to encrypt the master key during SSISDB backup'
- Encryption by password = 'LS3Setup!' -- 'New Password'
- Force

Если главный ключ базы данных еще не зашифрован главным ключом сервера, то в среде SQL Server Management Studio отобразится следующее предупреждающее сообщение. Пропустить это предупреждающее сообщение.

Не удается расшифровать текущий главный ключ. Ошибка пропущена, так как был указан параметр FORCE.

Аргумент FORCE указывает, что следует продолжить процесс восстановления, даже если текущий главный ключ базы данных закрыт. Для каталога SSISDB это сообщение будет отображаться, поскольку главный ключ базы данных не является открытым в экземпляре, где осуществляется восстановление базы данных.

Метод 2

Этот метод следует использовать при наличии исходного пароля, который использовался для создания SSISDB.

- open master key decryption by password = 'LS1Setup!' --'Password used when creating SSISDB'
- Alter Master Key Add encryption by Service Master Key

8. Определить, совместимы ли схема каталога SSISDB и двоичные файлы Службы Integration Services (сборка ISServerExec и SQLCLR), запустив catalog.check_schema_version.

9. Для подтверждения того, что база данных SSISDB успешно восстановлена, выполнить такие операции с каталогом SSISDB, как запуск пакетов, развернутых на сервере Службы Integration Services. Дополнительные сведения см. разделе Выполнение пакета на сервере служб SSIS с использованием среды SQL Server Management Studio.

Перемещение базы данных служб SSIS

Следовать инструкциям по перемещению пользовательских баз данных. Дополнительные сведения см. в статье Move User Databases.

Обязательно создать резервную копию главного ключа базы данных SSISDB и защитите файл резервной копии. Дополнительные сведения см. в статье Создание резервной копии каталога SSISDB.

Убедится в том, что соответствующие объекты служб Integration Services (SSIS) созданы в новом экземпляре SQL Server, где каталог SSISDB еще не был создан.

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Проблемы проверки подлинности безопасности системы СУБД
2. Архитектуры системы безопасности в MS SQL Server
3. Режимы защиты безопасности MS SQL Server

Лабораторная работа №10.Шифрование данных с использованием SQL Server.

Цель работы: Изучение основы шифрование и использование метода шифрование в среде SQL Server.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть

Шифрование представляет собой способ скрытия данных с помощью ключа или пароля. Это делает данные бесполезными без соответствующего ключа или пароля для дешифрования. Шифрование не решает проблемы управления доступом. Однако оно повышает защиту за счет ограничения потери данных даже при обходе системы управления доступом. Например, если компьютер, на котором установлена база данных, был настроен неправильно и злоумышленник смог получить конфиденциальные данные, то украденная информация будет бесполезна, если она была предварительно зашифрована. В SQL Server можно шифровать соединения, данные и хранимые процедуры. В следующей таблице содержатся дополнительные сведения о шифровании в SQL Server. Несмотря на то, что шифрование является полезным средством обеспечения безопасности, его не следует применять ко всем данным или соединениям. При решении о внедрении шифрования необходимо проанализировать, как пользователи получают доступ к данным. Если пользователи получают доступ к данным через открытую сеть, то шифрование может потребоваться для повышения безопасности. Однако если весь доступ осуществляется по безопасной внутренней сети, то шифрование не требуется. Использование шифрования включает политику управления паролями, ключами и сертификатами.

SQL Server содержит функции шифрования и расшифровки данных с помощью сертификата и асимметричного или симметричного ключа. Все они содержатся во внутреннем хранилище сертификатов. Хранилище использует иерархию шифрования, обеспечивающую безопасность сертификатов и ключей на уровне, находящемся выше в иерархии. Эта область функций SQL Server называется секретным хранилищем. Самым быстрым режимом шифрования, поддерживаемым функциями шифрования, является шифрование с помощью симметричного ключа. Этот режим подходит для управления большими томами данных. Симметричные ключи могут быть зашифрованы сертификатами, паролями или другими симметричными ключами.

Ключи и алгоритмы

SQL Server поддерживает несколько алгоритмов шифрования симметричным ключом, включая DES, Triple DES, RC2, RC4, 128-разрядный RC4, DESX, 128-разрядный AES, 192-разрядный AES и 256-разрядный AES. Алгоритмы реализуются с помощью API-интерфейса Windows Crypto. В пределах соединения с базой данных SQL Server может поддерживать несколько открытых симметричных ключей. Открытый ключ получается из хранилища и доступен для расшифровки данных. Не нужно указывать, какой симметричный ключ будет использоваться для расшифровки фрагмента данных. Каждое зашифрованное значение содержит идентификатор ключа (идентификатор GUID ключа), использованного для его шифрования. Если расшифрован и открыт верный ключ, то ядро сопоставляет зашифрованный поток байтов с открытым симметричным ключом. Этот ключ затем используется для выполнения расшифровки и возвращения данных. Если верный ключ не открыт, возвращается значение NULL.

Практическая часть

Для выполнения приведенных ниже шагов необходимы следующие разрешения.

- Разрешение CONTROL на базу данных.
- Разрешение CREATE CERTIFICATE на базу данных. Сертификаты могут принадлежать только именам входа Windows, именам входа SQL Server и ролям приложений. Сертификаты не могут принадлежать группам и ролям.
- Разрешение ALTER на таблицу.

– Некоторые разрешения на ключ, также не должно быть запрета на разрешение VIEW DEFINITION.

Использование Transact-SQL

Для применения приведенных ниже примеров требуется главный ключ базы данных. Если в вашей базе данных главного ключа еще нет, создайте его, выполнив указанный ниже оператор и указав пароль:

– CREATE MASTER KEY ENCRYPTION BY

– PASSWORD = '<some strong password>';

Всегда создавайте резервную копию главного ключа базы данных. Дополнительные сведения о создании главных ключей баз данных см. в статье CREATE MASTER KEY (Transact-SQL).

Шифрование столбца данных с помощью простого симметричного шифрования

1. В **обозревателе объектов** подключится к экземпляру компонента Компонент Database Engine.
2. На стандартной панели выберать пункт **Создать запрос**.
3. Скопировать следующий пример в окно запроса и нажмите кнопку

Выполнить:

```
USE AdventureWorks2012;
GO
CREATE CERTIFICATE Sales09
  WITH SUBJECT = 'Customer Credit Card Numbers';
GO
CREATE SYMMETRIC KEY CreditCards_Key11
   WITH ALGORITHM = AES_256
   ENCRYPTION BY CERTIFICATE Sales09;
GO
-- Create a column in which to store the encrypted data.
ALTER TABLE Sales.CreditCard
   ADD CardNumber_Encrypted varbinary(128);
GO
-- Open the symmetric key with which to encrypt the data.
OPEN SYMMETRIC KEY CreditCards_Key11
  DECRYPTION BY CERTIFICATE Sales09;

-- Encrypt the value in column CardNumber using the
-- symmetric key CreditCards_Key11.
-- Save the result in column CardNumber_Encrypted.
UPDATE Sales.CreditCard
SET CardNumber_Encrypted = EncryptByKey(Key_GUID('CreditCards_Key11')
   , CardNumber, 1, HashBytes('SHA1', CONVERT( varbinary
   , CreditCardID)));
GO
-- Verify the encryption.
-- First, open the symmetric key with which to decrypt the data.

OPEN SYMMETRIC KEY CreditCards_Key11
```

```
  DECRYPTION BY CERTIFICATE Sales09;
GO
-- Now list the original card number, the encrypted card number,
-- and the decrypted ciphertext. If the decryption worked,
-- the original number will match the decrypted number.
SELECT CardNumber, CardNumber_Encrypted
   AS 'Encrypted card number', CONVERT(nvarchar,
   DecryptByKey(CardNumber_Encrypted, 1 ,
   HashBytes('SHA1', CONVERT(varbinary, CreditCardID))))
   AS 'Decrypted card number' FROM Sales.CreditCard;
GO
```

Шифрование столбца данных с помощью симметричного шифрования, включающего структуру проверки подлинности

1. В **обозревателе объектов** подключится к экземпляру компонента Компонент Database Engine.

2. На стандартной панели выбирать пункт **Создать запрос**.

3. Скопировать следующий пример в окно запроса и нажимать кнопку **Выполнить**.

```
USE AdventureWorks2012;
GO
CREATE CERTIFICATE HumanResources037
  WITH SUBJECT = 'Employee Social Security Numbers';
GO
CREATE SYMMETRIC KEY SSN_Key_01
   WITH ALGORITHM = AES_256
   ENCRYPTION BY CERTIFICATE HumanResources037;
GO
USE [AdventureWorks2012];
GO
-- Create a column in which to store the encrypted data.
ALTER TABLE HumanResources.Employee
   ADD EncryptedNationalIDNumber varbinary(128);
GO
-- Open the symmetric key with which to encrypt the data.
OPEN SYMMETRIC KEY SSN_Key_01
  DECRYPTION BY CERTIFICATE HumanResources037;
-- Encrypt the value in column NationalIDNumber with symmetric
-- key SSN_Key_01. Save the result in column EncryptedNationalIDNumber.
UPDATE HumanResources.Employee
SET EncryptedNationalIDNumber = EncryptByKey(Key_GUID('SSN_Key_01'),
NationalIDNumber);
GO
-- Verify the encryption.
-- First, open the symmetric key with which to decrypt the data.
OPEN SYMMETRIC KEY SSN_Key_01
  DECRYPTION BY CERTIFICATE HumanResources037;
GO
-- Now list the original ID, the encrypted ID, and the
-- decrypted ciphertext. If the decryption worked, the original
```

```
-- and the decrypted ID will match.
SELECT NationalIDNumber, EncryptedNationalIDNumber
   AS 'Encrypted ID Number',
   CONVERT(nvarchar, DecryptByKey(EncryptedNationalIDNumber))
   AS 'Decrypted ID Number'
   FROM HumanResources.Employee;
GO
```

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Проблемы проверки подлинности безопасности системы СУБД
2. Архитектуры системы безопасности в MS SQL Server
3. Режимы защиты безопасности MS SQL Server

Лабораторная работа №11.Применение прозрачных методов шифрования данных

Цель работы: Изучение основных методов прозрачного шифрование и их применение.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть

Прозрачное шифрование данных (TDE) позволяет шифровать файлы данных SQL Server, База данных SQL Azureи Хранилище данных SQL Azure; это называется шифрованием хранящихся данных. Чтобы защитить базу данных, можно принять ряд мер предосторожности, например спроектировать систему безопасности, проводить шифрование конфиденциальных ресурсов и поместить серверы базы данных под защиту брандмауэра. Однако если будет похищен физический носитель (например, диск или ленты резервной копии), злоумышленник может легко восстановить или подключить базу данных и получить доступ к данным. Одним из решений может стать шифрование конфиденциальных данных в базе данных и защита ключей, используемых при шифровании, с помощью сертификата. Это не позволит использовать данные ни одному человеку, не имеющему ключей, но такой тип защиты следует планировать заранее.

Функция прозрачного шифрования данных выполняет шифрование и дешифрование ввода-вывода в реальном времени для файлов данных и журналов. При шифровании используется ключ шифрования базы данных (DEK), который хранится в загрузочной записи базы данных, где можно получить к нему доступ при восстановлении. Ключ шифрования базы данных является симметричным ключом, защищенным сертификатом, который хранится в базе данных master на сервере, или асимметричным ключом, защищенным модулем расширенного управления ключами. Функция прозрачного шифрования данных защищает "неактивные" данные, то есть файлы данных и журналов.

Благодаря ей обеспечивается соответствие требованиям различных законов, постановлений и рекомендаций, действующих в разных отраслях. Это позволяет разработчикам программного обеспечения шифровать данные с помощью алгоритмов шифрования AES и 3DES, не меняя существующие приложения.

Функция прозрачного шифрования данных не обеспечивает шифрование каналов связи. Дополнительные сведения о способах шифрования данных, передаваемых по каналам связи, см. в разделе Включение шифрования соединений в ядре СУБД (диспетчер конфигурации SQL Server).

Сведения о прозрачном шифровании данных (TDE)

Шифрование файла базы данных проводится на уровне страниц. Страницы в зашифрованной базе данных шифруются до записи на диск и дешифруются при чтении в память. Прозрачное шифрование данных не увеличивает размер зашифрованной базы данных.

Сведения, применимые к база данных SQL

При использовании прозрачного шифрования данных с База данных SQL версии 12 База данных SQL автоматически создает для вас сертификат на уровне сервера, хранящийся в базе данных master. Чтобы переместить базу данных TDE в База данных SQL, вам необходимо расшифровать ее, переместить, а затем повторно включить TDE в целевой База данных SQL. Пошаговые инструкции по прозрачному шифрованию данных в База данных SQLможно найти в разделе Transparent Data Encryption with Azure SQL Database.

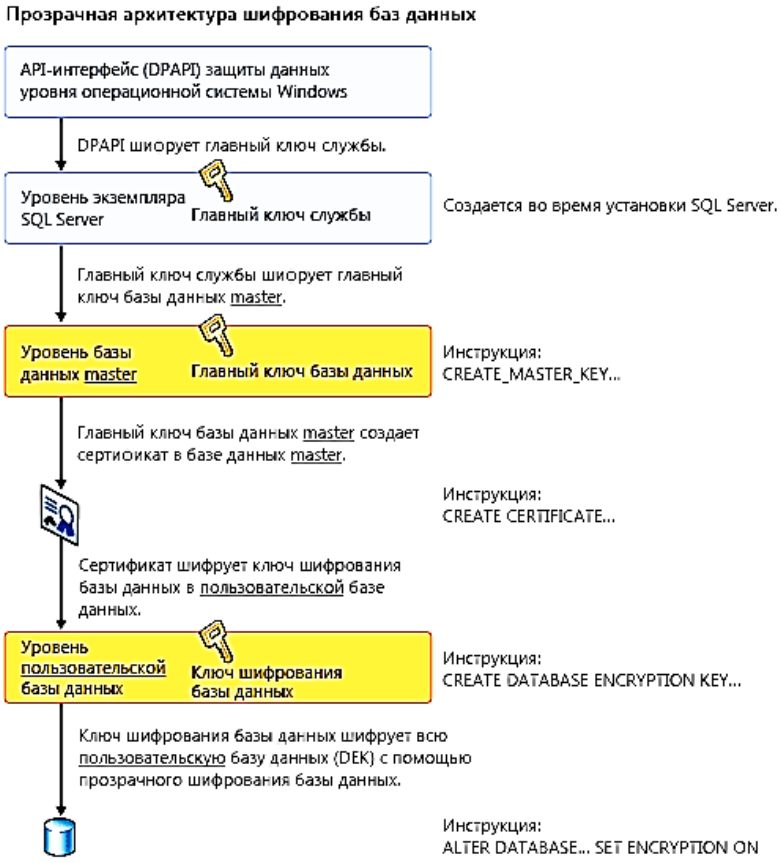
Сведения, применимые к SQL Server

После защиты базы данных ее можно восстановить с помощью правильного сертификата. Дополнительные сведения о сертификатах см. в разделе SQL Server Certificates and Asymmetric Keys.

При включении функции прозрачного шифрования данных необходимо немедленно создать резервную копию сертификата и закрытого ключа, связанного с этим сертификатом. В случае если сертификат окажется недоступен или понадобится восстановить базу данных на другом сервере либо присоединить ее к другому серверу, необходимо иметь копии сертификата и закрытого ключа. Иначе будет невозможно открыть базу данных. Сертификат шифрования следует сохранить, даже если функция прозрачного шифрования в базе данных будет отключена. Даже если база данных не зашифрована, части журнала транзакций могут по-прежнему оставаться защищенными, а для некоторых операций будет требоваться сертификат до выполнения полного резервного копирования базы данных. Сертификат, который превысил свой срок хранения, все еще может быть использован для шифрования и расшифровки данных с помощью прозрачного шифрования данных.

Иерархия шифрования

На рисунке ниже показана архитектура прозрачного шифрования данных. При использовании прозрачного шифрования данных в База данных SQL пользователь может настраивать только элементы уровня базы данных (ключ шифрования базы данных и фрагменты ALTER DATABASE).



Использование прозрачного шифрования данных

Чтобы использовать прозрачное шифрование данных, выполните следующие действия.

Область применения: SQL Server.

- Создайте главный ключ
- Создайте или получите сертификат, защищенный главным ключом
- Создайте ключ шифрования базы данных и защитите его с помощью сертификата
- Задайте ведение шифрования базы данных

В следующем примере демонстрируется шифрование и дешифрование базы данных AdventureWorks2012 с помощью сертификата с именем MyServerCert, установленного на сервере.

```
SQL
USE master;
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<UseStrongPasswordHere>';
go
CREATE CERTIFICATE MyServerCert WITH SUBJECT = 'My DEK Certificate';
go
USE AdventureWorks2012;
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE MyServerCert;
GO
```

```
ALTER DATABASE AdventureWorks2012
SET ENCRYPTION ON;
GO
```

Операции шифрования и дешифрования запланированы в фоновых потоках SQL Server. Состояние этих операций можно просмотреть в представлениях каталога и динамических административных представлениях в списке, представленном далее в этом разделе.

Файлы резервных копий баз данных, в которых включено TDE, также шифруются с помощью ключа шифрования базы данных. Поэтому для восстановления таких резервных копий необходимо иметь сертификат, защищающий ключ шифрования базы данных. Это значит, что помимо резервного копирования базы данных обязательно необходимо сохранять резервные копии сертификатов серверов, чтобы не допустить потери данных. Если сертификат станет недоступным, это приведет к потере данных. Дополнительные сведения см. в статье SQL Server Certificates and Asymmetric Keys.

Команды и функции

Чтобы в следующих инструкциях можно было использовать сертификаты TDE, они должны быть зашифрованы главным ключом базы данных. Если они зашифрованы только паролем, то они будут отклонены инструкциями как шифраторы.

Если после использования сертификатов в TDE включить защиту паролем, база данных станет недоступной после перезапуска.

В следующей таблице представлены ссылки и объяснения команд и функций TDE.

Команда или функция	Назначение
CREATE DATABASE ENCRYPTION KEY (Transact-SQL)	Создает ключ, используемый для шифрования базы данных.
ALTER DATABASE ENCRYPTION KEY (Transact-SQL)	Изменяет ключ, используемый для шифрования базы данных.
DROP DATABASE ENCRYPTION KEY (Transact-SQL)	Удаляет ключ, использовавшийся для шифрования базы данных.
Параметры ALTER DATABASE SET (Transact-SQL)	Объясняет параметр **ALTER DATABASE**, который используется для включения TDE.

Представления каталога и динамические административные представления

В следующей таблице показаны представления каталогов и динамические административные представления TDE.

Представление каталога или динамическое административное представление	Назначение
sys.databases (Transact-SQL)	Представление каталога со сведениями о базе данных.
sys.certificates (Transact-SQL)	Представление каталога с сертификатами из базы данных.
sys.dm_database_encryption_keys (Transact-SQL)	Динамическое административное представление со сведениями о ключах шифрования, используемых в базе данных, и состоянии шифрования базы данных.

Разрешения

Для каждой функции или команды TDE действуют отдельные требования к разрешениям, описанные в таблицах выше.

Для просмотра метаданных, связанных с TDE, необходимо разрешение VIEW DEFINITION на сертификат.

Во время проверки базы данных на повторное шифрование, выполняемой для операции шифрования, операции обслуживания базы данных отключаются. Для выполнения операции обслуживания базы данных можно использовать однопользовательский режим. Дополнительные сведения см. в разделе Установка однопользовательского режима базы данных.

Состояние шифрования базы данных можно определить с помощью динамического административного представления sys.dm_database_encryption_keys. Дополнительные сведения см. выше в подразделе "Представления каталога и динамические административные представления" этого раздела.

В режиме TDE все файлы и файловые группы зашифрованы. Если какие-либо файловые группы в базе данных помечены признаком READ ONLY, то операция шифрования базы данных завершится сбоем.

Если база данных используется в зеркальном отображении базы данных или в доставке журналов, то зашифрованы будут обе базы данных. Транзакции журналов при передаче между ними будут передаваться в зашифрованном виде.

Полнотекстовые индексы будут шифроваться после включения шифрования базы данных. Полнотекстовые индексы, созданные до версии SQL Server 2008, будут импортированы в базу данных во время обновления до SQL Server 2008 или более поздней версии, и к ним будет применено прозрачное шифрование данных.

Совет

Чтобы отслеживать изменения в состоянии прозрачного шифрования для базы данных, используйте аудит базы данных SQL или подсистему аудита SQL Server. Для SQL Server прозрачное шифрование данных отслеживается в группе действий аудита DATABASE_CHANGE_GROUP, которую можно найти в списке Действия и группы действий подсистемы аудита SQL Server.

Ограничения

При первом шифровании базы данных, изменении ключа или дешифровании базы данных не допускаются следующие операции:

- удаление файла из файловой группы в базе данных;
- удаление базы данных;
- перевод базы данных в режим «вне сети»;
- отсоединение базы данных;
- перевод базы данных или файловой группы в состояние READ ONLY.

При выполнении инструкций CREATE DATABASE ENCRYPTION KEY, ALTER DATABASE ENCRYPTION KEY, DROP DATABASE ENCRYPTION KEY или ALTER DATABASE...SET ENCRYPTION не допускаются следующие операции:

- удаление файла из файловой группы в базе данных;
- Удаление базы данных.
- перевод базы данных в режим «вне сети»;
- отсоединение базы данных;
- перевод базы данных или файловой группы в состояние READ ONLY;

- использование команды ALTER DATABASE;
- запуск резервного копирования базы данных или файла базы данных;
- запуск восстановления базы данных или файла базы данных;
- создание моментального снимка.

Следующие операции или условия запрещают выполнение инструкций CREATE DATABASE ENCRYPTION KEY, ALTER DATABASE ENCRYPTION KEY, DROP DATABASE ENCRYPTION KEY или ALTER DATABASE...SET ENCRYPTION.

- база данных предназначена только для чтения, или в ней есть файловые группы, доступные только для чтения;
- выполняется команда ALTER DATABASE;
- выполняется какая-либо операция резервного копирования;
- база данных находится в состоянии восстановления или в состоянии «вне сети»;
- идет создание моментального снимка;
- выполняется задача обслуживания базы данных.

При создании файлов базы данных быстрая инициализация файлов недоступна при включенном TDE.

Чтобы зашифровать ключ шифрования базы данных с помощью асимметричного ключа, используемый асимметричный ключ должен находиться в поставщике расширенного управления ключами.

Прозрачное шифрование данных и журналы транзакций

Включение TDE в базе данных приводит к «обнулению» оставшейся части виртуального журнала транзакций и принудительному началу нового виртуального журнала транзакций. Это гарантирует, что после включения шифрования базы данных в журналах транзакций не останется простого текста. Состояние шифрования файла журнала можно определить, просмотрев столбец encryption_state в представлении sys.dm_database_encryption_keys, как показано в примере:

```
USE AdventureWorks2012;
GO
/* The value 3 represents an encrypted state
  on the database and transaction logs. */
SELECT *
FROM sys.dm_database_encryption_keys
WHERE encryption_state = 3;
GO
```

Все данные, записанные в журнале транзакций до изменения ключа шифрования базы данных, будут зашифрованы с помощью предыдущего ключа шифрования базы данных.

После двукратного изменения ключа шифрования базы данных необходимо выполнить резервное копирование журнала перед следующим изменением ключа шифрования базы данных.

Прозрачное шифрование данных и системная база данных tempdb

Системная база данных tempdb будет шифроваться, если с помощью TDE шифруется любая другая база данных в экземпляре SQL Server. Это может влиять на производительность незашифрованных баз данных в том же экземпляре SQL Server.

Дополнительные сведения о системной базе данных tempdb см. в разделе База данных tempdb.

Прозрачное шифрование данных и репликация

Репликация данных не выполняется автоматически в зашифрованном виде из базы данных с включенным TDE. Необходимо отдельно включить TDE, если нужно защитить базы данных распространителя и подписчика. При репликации моментальных снимков и начальном распределении данных для репликации транзакций и репликации слиянием данные могут храниться в незашифрованных промежуточных файлах, например файлах BCP. Во время репликации транзакций или репликации слиянием шифрование можно включить для защиты каналов связи. Дополнительные сведения см. в разделе Включение шифрования соединений в ядре СУБД (диспетчер конфигурации SQL Server).

Прозрачное шифрование данных и данные FILESTREAM

Данные FILESTREAM не шифруются, даже если включено прозрачное шифрование данных.

Прозрачное шифрование и расширение буферного пула

Файлы, касающиеся расширения буферного пула, не шифруются, если база данных зашифрована с помощью прозрачного шифрования данных. Необходимо использовать средства шифрования на уровне файловой системы, такие как Bitlocker или файловая система EFS для файлов с расширением BPE.

Прозрачное шифрование данных и In-Memory OLTP

Прозрачное шифрование данных можно включить в базе данных, которая содержит объекты OLTP в памяти. В SQL Server 2016 и База данных SQL Azure записи журнала выполняющейся в памяти OLTP шифруются, если включено TDE. В SQL Server 2014 записи журнала выполняющейся в памяти OLTP шифруются, если включено TDE, однако файлы в файловой группе MEMORY_OPTIMIZED_DATA не шифруются.

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Проблемы проверки подлинности безопасности системы СУБД
2. Архитектуры системы безопасности в MS SQL Server
3. Режимы защиты безопасности MS SQL Server

Лабораторная работа №12.Транзакции и параллельные вычисления для защиты базы данных.

Цель работы: Изучение свойство транзакции и параллельное вычисление для защиты базы данных.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть

Параллелизм – возможность параллельной обработки в СУБД многих параллельных транзакций к одним и тем же данным в одно и то же время. СУБД должно осуществлять правильное восстановление данных при отмене транзакций или в случае системного сбоя и гарантировать, что пользователи при параллельной работе с данными мешать друг другу не будут.

Основные проблемы, возникающие при параллельной обработке транзакций:

1. Потеря результатов обновления. Возникает когда одновременно, но с некоторым сдвигом во времени выполняются 2 транзакции, заключающиеся в чтении данных из одного и того же кортежа с последующим изменением данных в них.

Если 2 транзакции одновременно изменяют и фиксируют изменения одних и тех же данных, то сохраняются результаты той транзакции, которая завершилась последней по времени.

2. Незафиксированные зависимости, или т.н. «грязное чтение» (Dirty Read).

Появляется, если с помощью некой транзакции T2 осуществляется чтение или изменение некоторого кортежа, только что измененного другой транзакцией T1, однако изменения еще не были подтверждены. Если данные транзакции T1 подтверждены не будут, то результаты транзакции T2 окажутся неверными.

3. Несогласованные данные (неповторяемое чтение Fuzzy Read).

Возникает, когда одна транзакция читает кортеж, а другая изменяет его и фиксирует изменения до окончания первой.

4. Строки-призраки (Phantom). Возникает при работе не с одним кортежем, а при выполнении групповых операции (вычисление среднего, подсчет баланса, т. е. когда обрабатывается несколько строк одновременно). В таких сложных обработках, как правило,некий массив кортежей читается неоднократно. Если между первым и вторым чтениями была удалена или добавлена запись в массив строк, при повторном считывании массива появляется лишняя строка (фантом), что может привести к неверным результатам.

Т.о. СУБД должна гарантировать, что если 2 транзакции выполняются параллельно, то результаты их выполнения должны быть аналогичны тем, которые были бы при их последовательном выполнении, каждый пользователь должен чувствовать себя так, как будто он работает монопольно (концепция сериализации транзакций).

Для решения проблем при параллельной работе используется механизм блокировок: одна или набор записей блокируются на момент выполнения транзакции. Поэтому желательно делать транзакции покороче, чтобы обеспечить равномерную работу пользователей в сети.

На практике используются 2 вида блокировок:

— без взаимного доступа (монопольная, exclusive locks)

— с взаимным доступом (разделяемая, shared locks)

Транзакция, предназначенная для чтения, извлечения записи, должна накладывать s-locks на читаемые записи. Транзакция, предназначенная для изменения данных, должна накладывать x-locks на эти записи. Если этой же транзакцией до этого была наложена s-locks,то эта блокировка поменяется на x-locks.Это приводит к тому, что в первом случае, только при чтении, другие пользователи могут одновременно читать ту же запись, а при изменении данных другим пользователям запрещено видеть неподтвержденные данные, пока они не будут зафиксированы.

Блокировки, как правило, задаются неявно:

- запрос на извлечение данных (select) по умолчанию считается запросом с s-locks
- любой запрос на обновление (insert, delete, update) по умолчанию x-locks

Некоторые СУБД поддерживают режим сосуществования s-locks, накладываемых разными клиентами на один и тот же ресурс. Т. к. если несколько человек одновременно

читают одни и те же данные, то при отказе наложившего s-locks формально получается, что данные уже можно изменять, хотя другие пользователи продолжают чтение.

Для осуществления s-locks и x-locks системе нужны ресурсы (память), которая отводится для установки защиты на те или иные кортежи. Обычно в СУБД возможное число одновременных блокировок – настраиваемая величина. Как только на одну и ту же таблицу набирается некоторое количество блокировок, система переводит ее из режима блокировки записей в режим блокировки всей таблицы. Это позволяет экономить ресурсы.

Для заданного набора транзакций любой порядок их выполнения называется графиком запуска. Если транзакции выполняются друг за другом, последовательно, то такой график запуска – последовательный.

Блокировки нарушают график запуска, что может приводить к разным результатам деятельности нескольких параллельно работающих пользователей. Для того, чтобы не нарушать последовательность транзакций, используют механизм упорядочения транзакций.

Во многих коммерческих СУБД на сегодня существуют и другие методы осуществления параллельной работы кроме блокировок:
1. Явные блокировки (SQL Server, SyBase): в некоторые команды (напр. select) включены ключевые слова, которые явным образом позволяют блокировать кортежи, страницы, таблицы на этапе проектирования.

2. Уровни изоляции транзакций. Существуют специальные команды, с помощью которых можно информировать СУБД о том, что некоторая программа не будет повторно считывать данные в момент транзакции, позволяя тем самым СУБД снять блокировку до окончания транзакции.

3. Параметры блокировки. Администратор БД во многих СУБД, как правило, вручную имеет возможность менять размер блокируемого участка. Кроме того, он может изменять число блокировок, интервал и т.п.

1. Степень дробления блокировок. Блокировке может подвергаться не только отдельная запись, но и несколько записей вплоть до блокировки всей таблицы. Кроме того, возможна теоретически блокировка на уровне одного поля. На практике блокировка поля не используется из-за ресурсоемкости, а выигрыш во времени невелик. Чаще всего блокировки осуществляются на уровне страниц (1.8 кБ) т.к. в реляционных СУБД извлечение данных также ведется страницами.

2. Взаимные блокировки. Из-за того, что несколько пользователей могут по очереди подключаться и блокировать несколько записей в рамках одной транзакции, возможно возникновение тупиковых ситуаций: транзакция T1 обработала одну запись и обратилась ко второй записи, а транзакция T2 сначала обработала и заблокировала вторую запись и хочет обратиться к первой. Каждая из транзакций должна обратиться к следующей записи для обновления, а запись заблокирована. Транзакции не могут закончиться, что мешает работать и другим пользователям. В этом случае в системе предусмотрены специальные периодически подключающиеся сервисы, которые сканируют оперативную память, ищут взаимные блокировки и жертвуют одной транзакцией в пользу завершения другой.

3. Уровни изоляции. Один из способов вместо блокировок – т.н. уровни изоляции. Термин «уровни изоляции» используется для описания степени вмешательства параллельных транзакций в работу некоторой данной.

Уровней изоляции может быть насколько. В стандарте SQL принято 4, в порядке возрастания приоритета:

1. Read Uncommitted – незавершенное чтение. Не имеет ограничений на чтение данных, все данные видны пользователям, обеспечивается максимальный параллелизм, минимальная изоляция, разрешается делать только одно изменение (одному из пользователей). Предотвращается только пропавшее обновление.

2. Read Committed – завершенное чтение. Гарантирует отсутствие пропавших обновлений и грязного чтения. Но также обеспечивает высокий уровень параллелизма и разрешает модифицировать несколько данных.

3. Repeatable read – повторяемое чтение. Ограничивает пропавшие обновления, грязное чтение, неповторяемое чтение. Приемлем в тех случаях, когда необходимы множественные изменения от нескольких клиентов. При этом автоматически устанавливаются блокировки на множество строк. Фактически каждая транзакция видит только первоначальную версию данных.

4. Serializable – Последовательное преобразование. Предотвращает все возможные проблемы параллельных транзакций, т. ч. и фантомы. Транзакции выполняются так, как будто они выполняются последовательно. На практике практически не встречается, по умолчанию чаще всего Read Committed.

4. Временные отметки Метод временных отметок, применяемый для контроля целостности данных, полностью отличается от блокировок. Он не предусматривает какого-либо ожидания конфликтных транзакций, они просто отменяются и запускаются заново.

В его основе лежит понятие временной отметки – уникального идентификатора, включающего помимо прочего момент запуска транзакции с помощью системных часов, который и лежит в основе этой метки.

Практическая часть

SQL Server предлагает множество средств управления поведением транзакций. Пользователи в основном должны указывать только начало и конец транзакции, используя команды SQL или API (прикладного интерфейса программирования). Транзакция определяется на уровне соединения с базой данных и при закрытии соединения автоматически закрывается. Если пользователь попытается установить соединение снова и продолжить выполнение транзакции, то это ему не удастся. Когда транзакция начинается, все команды, выполненные в соединении, считаются телом одной транзакции, пока не будет достигнут ее конец.

SQL Server поддерживает три вида определения транзакций:

- явное;
- автоматическое;
- подразумеваемое.

По умолчанию SQL Server работает в режиме автоматического начала транзакций, когда каждая команда рассматривается как отдельная транзакция. Если команда выполнена успешно, то ее изменения фиксируются. Если при выполнении команды произошла ошибка, то сделанные изменения отменяются и система возвращается в первоначальное состояние.

Когда пользователю понадобится создать транзакцию, включающую несколько команд, он должен явно указать транзакцию.

Сервер работает только в одном из двух режимов определения транзакций: автоматическом или подразумевающемся. Он не может находиться в режиме исключительно явного определения транзакций. Этот режим работает поверх двух других.

Для установки режима автоматического определения транзакций используется команда:

```
SET IMPLICIT_TRANSACTIONS  OFF
```

При работе в режиме неявного (подразумевающегося) начала транзакций SQL Server автоматически начинает новую транзакцию, как только завершена предыдущая. Установка режима подразумевающегося определения транзакций выполняется посредством другой команды:

```
SET IMPLICIT_TRANSACTIONS  ON
```

Явные транзакции

Явные транзакции требуют, чтобы пользователь указал начало и конец транзакции, используя следующие команды:

- начало транзакции: в журнале транзакций фиксируются первоначальные значения изменяемых данных и момент начала транзакции;
- BEGIN TRAN[SACTION]
- [имя_транзакции |
- @имя_переменной_транзакции
- [WITH MARK [‘описание_транзакции’]]]
- конец транзакции: если в теле транзакции не было ошибок, то эта команда предписывает серверу зафиксировать все изменения, сделанные в транзакции, после чего в журнале транзакций помечается, что изменения зафиксированы и транзакция завершена;
- COMMIT [TRAN[SACTION]
- [имя_транзакции |
- @имя_переменной_транзакции]]
- создание внутри транзакции точки сохранения: СУБД сохраняет состояние БД в текущей точке и присваивает сохраненному состоянию имя точки сохранения;
- SAVE TRAN[SACTION]
- {имя_точки_сохранеия |
- @имя_переменной_точки_сохранения}
- прерывание транзакции; когда сервер встречает эту команду, происходит откат транзакции, восстанавливается первоначальное состояние системы и в журнале транзакций отмечается, что транзакция была отменена. Приведенная ниже команда отменяет все изменения, сделанные в БД после оператора BEGIN TRANSACTION или отменяет изменения, сделанные в БД после точки сохранения, возвращая транзакцию к месту, где был выполнен оператор SAVE TRANSACTION.
- ROLLBACK [TRAN[SACTION]
- [имя_транзакции |
- @имя_переменной_транзакции
- | имя_точки_сохранения
- |@имя_переменной_точки_сохранения]]

Функция @@TRANCOUNT возвращает количество активных транзакций.
Функция @@NESTLEVEL возвращает уровень вложенности транзакций.

```
BEGIN TRAN
SAVE TRANSACTION  point1
```

Пример 1. Использование точек сохранения

```
В точке point1 сохраняется первоначальное состояние таблицы Товар
DELETE FROM Товар WHERE КодТовара=2
  SAVE TRANSACTION  point2
В точке point2 сохраняется состояние таблицы Товар без товаров с кодом 2.
DELETE FROM Товар WHERE КодТовара=3
  SAVE TRANSACTION  point3
В точке point3 сохраняется состояние таблицы Товар без товаров с кодом 2 и с кодом 3.
DELETE FROM Товар WHERE КодТовара<>1
  ROLLBACK TRANSACTION point3
```

Происходит возврат в состояние таблицы без товаров с кодами 2 и 3, отменяется последнее удаление.
SELECT * FROM Товар
Оператор SELECT покажет таблицу Товар без товаров с кодами 2 и 3.
ROLLBACK TRANSACTION point1
Происходит возврат в первоначальное состояние таблицы.
SELECT * FROM Товар
COMMIT
Первоначальное состояние сохраняется.

Вложенные транзакции

Вложенными называются транзакции, выполнение которых инициируется из тела уже активной транзакции.

Для создания вложенной транзакции пользователю не нужны какие-либо дополнительные команды. Он просто начинает новую транзакцию, не закрыв предыдущую. Завершение транзакции верхнего уровня откладывается до завершения вложенных транзакций. Если транзакция самого нижнего (вложенного) уровня завершена неудачно и отменена, то все транзакции верхнего уровня, включая транзакцию первого уровня, будут отменены. Кроме того, если несколько транзакций нижнего уровня были завершены успешно (но не зафиксированы), однако на среднем уровне (не самая верхняя транзакция) неудачно завершилась другая транзакция, то в соответствии с требованиями ACID произойдет откат всех транзакций всех уровней, включая успешно завершенные. Только когда все транзакции на всех уровнях завершены успешно, происходит фиксация всех сделанных изменений в результате успешного завершения транзакции верхнего уровня.

Каждая команда COMMIT TRANSACTION работает только с последней начатой транзакцией. При завершении вложенной транзакции команда COMMIT применяется к наиболее "глубокой" вложенной транзакции. Даже если в команде COMMIT TRANSACTION указано имя транзакции более высокого уровня, будет завершена транзакция, начатая последней.

Если команда ROLLBACK TRANSACTION используется на любом уровне вложенности без указания имени транзакции, то откатываются все вложенные транзакции, включая транзакцию самого высокого (верхнего) уровня. В команде ROLLBACK TRANSACTION разрешается указывать только имя самой верхней транзакции. Имена любых вложенных транзакций игнорируются, и попытка их указания приведет к ошибке. Таким образом, при откате транзакции любого уровня вложенности всегда происходит откат всех транзакций. Если же требуется откатить лишь часть транзакций, можно использовать команду SAVE TRANSACTION, с помощью которой создается точка сохранения.

```
BEGIN TRAN
INSERT Товар (Название, остаток)
VALUES ('v',40)
  BEGIN TRAN
  INSERT Товар (Название, остаток)
  VALUES ('n',50)
  BEGIN TRAN
   INSERT Товар (Название, остаток)
   VALUES ('m',60)
ROLLBACK TRAN
```

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Проблемы проверки подлинности безопасности системы СУБД
2. Архитектуры системы безопасности в MS SQL Server
3. Режимы защиты безопасности MS SQL Server

Лабораторная работа №13.Применение защиты безопасности для реализации политики безопасности в SQL Server

Цель работы: Изучение метода защиты безопасности для реализации политики безопасности в среде SQL Server.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть

Защиту SQL Server можно рассматривать как последовательность шагов, затрагивающих четыре области: платформу, проверку подлинности, объекты (включая данные) и приложения, получающие доступ к системе. В приведенных ниже разделах описано создание и реализация эффективного плана обеспечения безопасности.

Дополнительные сведения о безопасности SQL Server см. на веб-сайте SQL Server. Они включают руководство с рекомендациями и контрольный список безопасности. Кроме того, этот веб-сайт содержит сведения о пакетах обновлений и файлы для загрузки.

Безопасность платформы и сети

Платформа для SQL Server включает в себя физическое оборудование и сетевые компьютеры, с помощью которых клиенты соединяются с серверами базы данных, а также двоичные файлы, применяемые для обработки запросов базы данных.

Физическая безопасность

Рекомендуется строго ограничивать доступ к физическим серверам и компонентам оборудования. Например, оборудование сервера базы данных и сетевые устройства должны находиться в закрытых охраняемых помещениях. Доступ к резервным носителям также следует ограничить. Для этого их рекомендуется хранить в отдельных охраняемых помещениях.

Реализация физической сетевой безопасности начинается с запрета доступа неавторизованных пользователей к сети. Следующая таблица содержит дополнительные сведения об источниках сведений по сетевой безопасности.

Безопасность операционной системы

В состав пакетов обновления и отдельных обновлений для операционной системы входят важные дополнения, позволяющие усилить безопасность. Все обновления для операционной системы необходимо устанавливать только после их тестирования с приложениями базы данных.

Кроме того, эффективную безопасность можно реализовать с помощью брандмауэров. Брандмауэр, распределяющий или ограничивающий сетевой трафик, можно настроить в соответствии с корпоративной политикой информационной безопасности. Использование брандмауэра повышает безопасность на уровне операционной системы, обеспечивая узкую область, на которой можно сосредоточить меры безопасности. Следующая таблица содержит дополнительные сведения по использованию брандмауэра с SQL Server.

Уменьшение контактной зоны является мерой безопасности, предполагающей остановку или отключение неиспользуемых компонентов. Уменьшение контактной зоны повышает уровень безопасности за счет уменьшения числа возможных способов атаковать систему. Важную роль в ограничении контактной зоны SQL Server играет запуск необходимых служб по принципу «минимума прав доступа», согласно которому службам и пользователям предоставляются только необходимые для работы права. Следующая таблица содержит дополнительные сведения по службам и доступу к системе.

Если в системе SQL Server используются службы IIS, необходимы дополнительные действия для обеспечения безопасности контактной зоны платформы. Следующая таблица содержит сведения о SQL Server и службах IIS.

Безопасность файлов операционной системы SQL Server

SQL Server использует файлы операционной системы для работы и хранения данных. Оптимальным решением для обеспечения безопасности файлов будет ограничение доступа к ним. Следующая таблица содержит сведения об этих файлах.

Сведения о	См.
SQL Server программные файлы	Расположение файлов для экземпляра по умолчанию и именованных экземпляров SQL Server

SQL Server позволяют повысить безопасность. Для определения новейшего доступного пакета обновления для SQL Serverперейдите на веб-сайт SQL Server .

С помощью приведенного ниже скрипта можно определить установленный в системе пакет обновления.

```
SELECT CONVERT(char(20), SERVERPROPERTY('productlevel'));
GO
```

Безопасность участников и объектов базы данных

Участники — это отдельные пользователи, группы и процессы, которым предоставлен доступ к ресурсам SQL Server. Защищаемые объекты — это сервер, база данных и объекты, которые содержит база данных. У каждого из них существует набор разрешений, с помощью которых можно уменьшить контактную зону SQL Server . Следующая таблица содержит сведения об участниках и защищаемых объектах.

Сведения о	См.
Пользователи, роли и процессы сервера и базы данных	Участники (компонент Database Engine)
Безопасность объектов сервера и базы данных	Защищаемые объекты
Иерархия безопасности SQL Server	Иерархия разрешений (компонент Database Engine)

Шифрование и сертификаты

Шифрование не решает проблемы управления доступом. Однако оно повышает безопасность, ограничивая потерю данных даже в тех редких случаях, когда средства управления доступом удается обойти. Например, если главный компьютер, на котором установлена база данных, был настроен неправильно, и злонамеренный пользователь смог получить конфиденциальные данные (например, номера кредитных карточек), то украденная информация будет бесполезна, если она была предварительно зашифрована. В следующей таблице содержатся дополнительные сведения о шифровании в SQL Server.

Сведения о	См.
Иерархия шифрования в SQL Server	Иерархия средств шифрования
Реализация безопасных соединений	Включение шифрования соединений в компоненте Database Engine (диспетчер конфигураций SQL Server)
Функции шифрования	Криптографические функции (Transact-SQL)

Сертификаты — это совместно используемые на двух серверах программные «ключи», которые позволяют обеспечить безопасную передачу данных с помощью надежной проверки подлинности. SQL Server позволяет создавать и использовать сертификаты для повышения безопасности объектов и соединений. Следующая таблица содержит дополнительные сведения об использовании сертификатов с SQL Server.

Сведения о	См.
Создание сертификата, который будет использоваться SQL Server	CREATE CERTIFICATE (Transact-SQL)
Использование сертификатов при зеркальном отображении базы данных	Использование сертификатов для конечной точки зеркального отображения базы данных (Transact-SQL)

Безопасность приложений

SQL Server рекомендуется разрабатывать защищенные клиентские приложения.

Дополнительные сведения об обеспечении безопасности клиентских приложений на сетевом уровне см. в разделе Client Network Configuration.

Средства, программы, представления и функции безопасности SQL Server

SQL Server предусмотрены средства, программы, представления и функции, которые используются для настройки и управления безопасностью.

Средства и программы безопасности SQL Server

Следующая таблица содержит сведения о средствах и программах SQL Server, с помощью которых можно настраивать и администрировать безопасность.

Сведения о	См.
Соединение с SQL Server	Использование среды SQL Server Management Studio
Соединение с SQL Server и запуск запросов из командной строки	Программа sqlcmd
Настройка сети и управление SQL Server	Диспетчер конфигурации SQL Server

Сведения о	См.
Включение и отключение компонентов с помощью средства управления на основе политики	Администрирование серверов с помощью управления на основе политик
Управление симметричными ключами для сервера отчетов	Программа rskeymgmt (SSRS)

Представления каталога и функции безопасности SQL Server

В компоненте Компонент Database Engine сведения о безопасности доступны через несколько представлений и функций, оптимизированных для наибольшей производительности и полезности. Следующая таблица содержит сведения о представлениях и функциях безопасности.

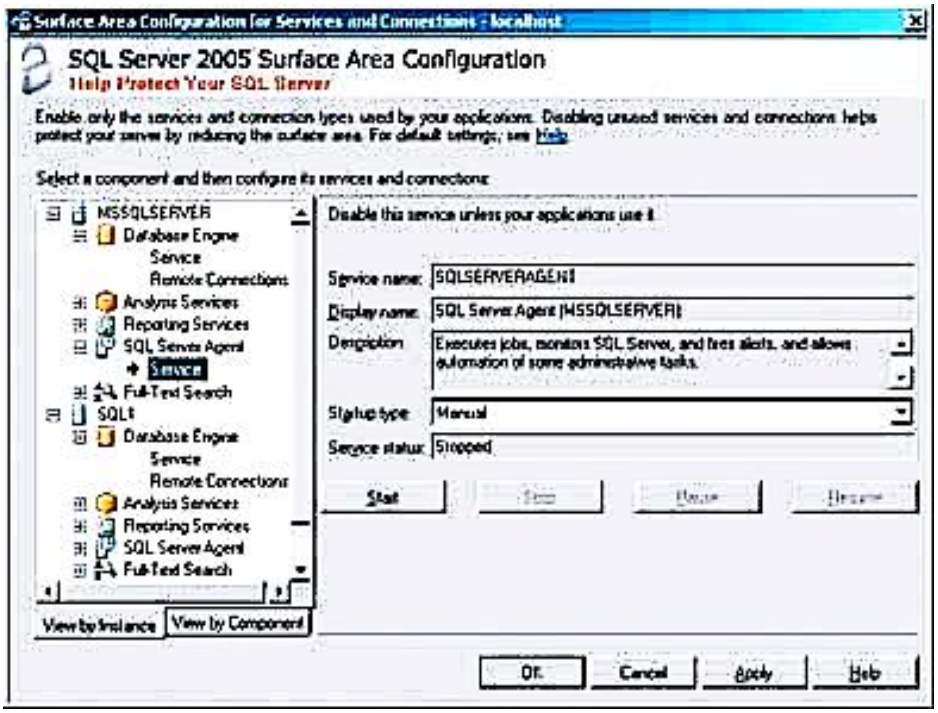
Сведения о	См.
SQL Server представления каталога безопасности, которые возвращают сведения о разрешениях, участниках, ролях и других сущностях уровня базы данных и сервера. Кроме того, существуют представления каталога, содержащие сведения о ключах шифрования, сертификатах и учетных данных.	Представления каталога безопасности (Transact-SQL)
SQL Server , которые возвращают сведения о текущем пользователе, разрешениях и схемах.	Функции безопасности (Transact-SQL)
SQL Server .	Динамические представления управления и функции, связанные с безопасностью (Transact-SQL)

Практическая часть

Управление доступом для SQL Server Agent

Агент SQL Server Agent должен выполняться от имени учетной записи Windows, обладающей ролью sysadmin, но не принадлежащей группе Administrators и имеющей разрешение на увеличение квот памяти для процесса. Сам SQL Server Agent должен выполняться как служба операционной системы, протоколироваться как сервис. Кроме того, можно создать специальную прокси-запись и ассоциировать ее с соответствующими правами Windows для доступа к внешним ресурсам.

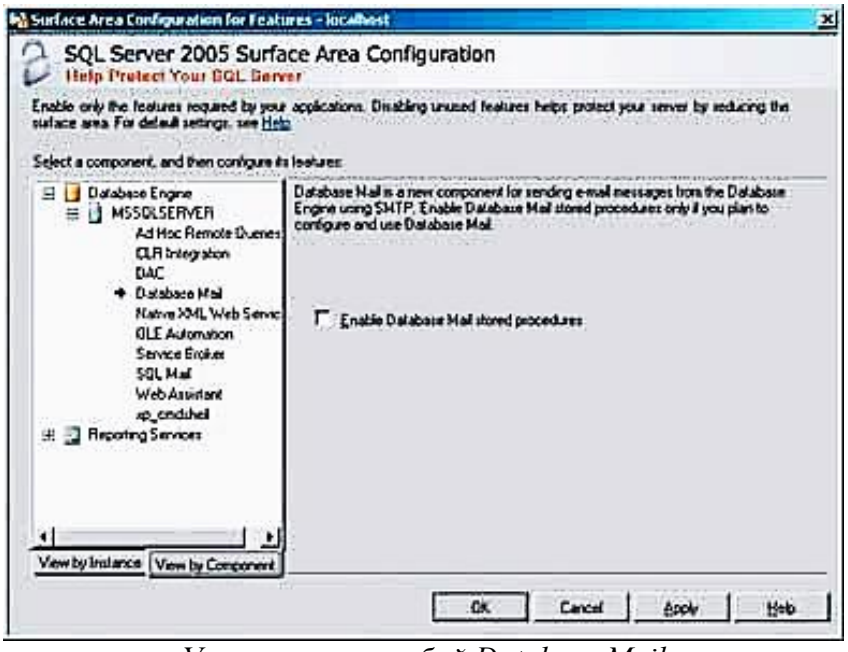
Для управления службой SQL Server Agent можно использовать утилиту Surface Area Configuration (рис. 1).



Управление службой SQL Server Agent

Управление доступом для Database Mail

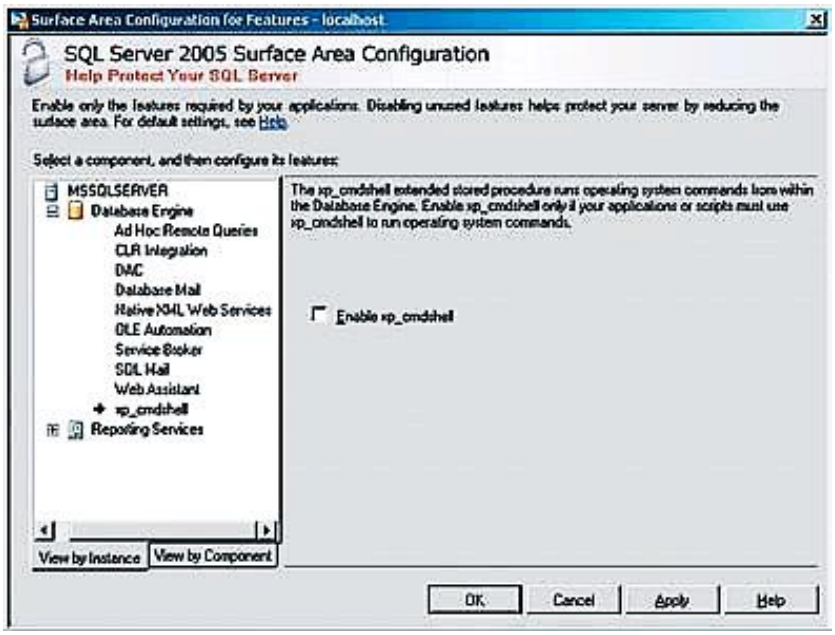
Database Mail — это новый компонент SQL Server 2005, предназначенный для поддержки протокола SMTP (*Simple Mail Transport Protocol*). Для управления доступом к нему рекомендуется создавать настраиваемые профили, позволяющие указать, каким образом пользователи базы данных msdb имеют доступ к данному профилю (самим пользователям следует присвоить роль *DatabaseMailUserRole* базы данных msdb). Для управления службой Database Mail можно также использовать утилиту Surface Area Configuration (рис. 2).



Управление службой Database Mail

Отключение ненужных служб

Многие из служб SQL Server отключены по умолчанию, и при отсутствии реальной необходимости их применения рекомендуется оставить их в неактивном состоянии. Так, поддержка применения Microsoft.NET Framework в ядре СУБД должна быть включена только в тех случаях, когда использование управляемого кода в приложении действительно необходимо. Не рекомендуется без необходимости включать такие службы и опции, как Database Mail и SQL Mail, AdHoc Remote Queries, Web Assistant, Remote DAC (*Dedicated Administrator Connection*), делать доступной хранимую процедуру xp_cmdshell для выполнения команд операционной системы из ядра СУБД и средства применения COM-расширений функциональности сервера (*OLE Automation extended stored procedures*), создавать точки доступа по протоколу HTTP (*HTTP Endpoints*).



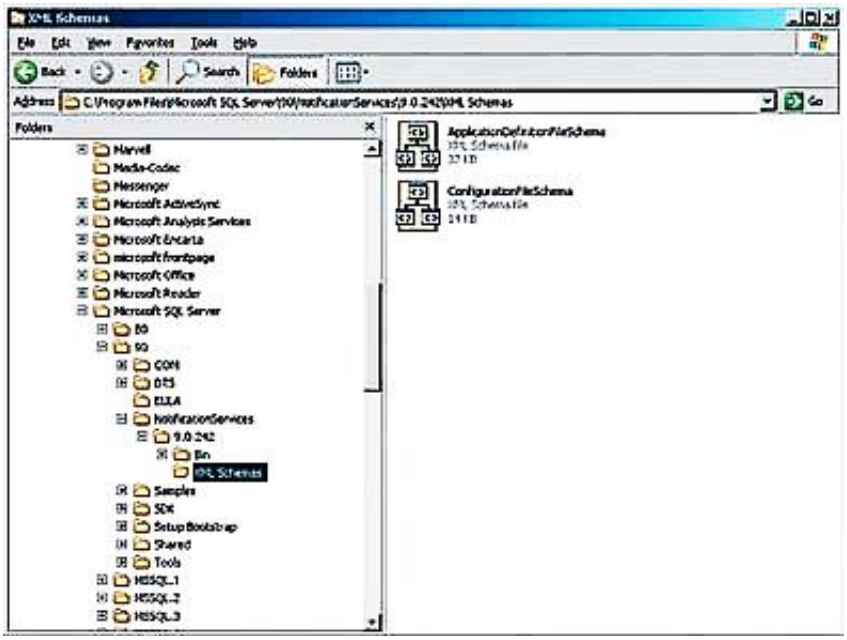
Управление доступностью служб SQL Server 2015

Применение брандмауэра при HTTP-доступе к серверу

Поскольку атаки с применением протокола HTTP являются сегодня одним из самых распространенных видов атак, предоставлять доступ к SQL Server по протоколу HTTP через Интернет следует только в случае реальной необходимости. Не стоит напоминать, что такой доступ должен быть реализован только через брандмауэр.

Обеспечение безопасности файлов и папок

Некоторые службы SQL Server (например, службы уведомлений) используют ряд файлов и папок для хранения конфигурационной информации и данных приложения. Ядро служб уведомлений (или иных служб SQL Server) должно иметь доступ к этим файлам и папкам, но пользователи не должны иметь возможности вносить изменения в содержащиеся в них файлы. Для защиты файлов и папок, используемых службами уведомлений, можно использовать разрешения NTFS для ограничения доступа к папкам и файлам либо механизмы Encrypted File System (*EFS*) для шифрования файлов и папок и предотвращения неавторизованного доступа.



Управление доступностью служб SQL Server 2015

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Проблемы проверки подлинности безопасности системы СУБД
2. Архитектуры системы безопасности в MS SQL Server
3. Режимы защиты безопасности MS SQL Server

Лабораторная работа №14.Внедрение аутентификации пользователей и программного обеспечения на SQL Server

Цель работы: Изучение основных способов создание цифровых сертификатов и ключей.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть

Аутентификация заключается в предоставлении ответа на следующий вопрос: "Имеет ли данный пользователь законное право на доступ к системе?" Таким образом, данная концепция безопасности определяет процесс проверки подлинности учетных данных пользователя, чтобы не допустить использование системы несанкционированными пользователями. Аутентификацию можно реализовать путем запроса, требуя, чтобы пользователь предоставил, например, следующее:

- нечто, что известно пользователю (обычно пароль);

– нечто, что принадлежит пользователю, например, идентификационную карту;

– физические характеристики пользователя, например, подпись или отпечатки пальцев.

Наиболее применяемый способ подтверждения аутентификации реализуется посредством использования имени пользователя и пароля. Система проверяет достоверность этой информации, чтобы решить, имеет ли данный пользователь законное право на доступ к системе или нет. Этот процесс может быть усилен применением шифрования.

Шифрование данных представляет собой процесс кодирования информации таким образом, что она становится непонятной, пока не будет расшифрована пользователем.

Аутентификация

Система безопасности компонента Database Engine состоит из двух разных подсистем безопасности:

– системы безопасности Windows;

– системы безопасности SQL Server.

Система безопасности Windows определяет безопасность на уровне операционной системы, т.е. метод, посредством которого пользователи входят в систему Windows, используя свои учетные записи Windows. (Аутентификация посредством этой подсистемы также называется аутентификацией Windows.)

Система безопасности SQL Server определяет дополнительную безопасность, требуемую на уровне системы баз данных, т.е. способ, посредством которого пользователи, уже вошедшие в операционную систему, могут подключаться к серверу базы данных. Система безопасности SQL Server определяет регистрационное имя входа (или просто называемое логином) в SQL Server, которое создается в системе и ассоциируется с определенным паролем. Некоторые регистрационные имена входа в SQL Server идентичны существующим учетным записям Windows. (Аутентификация посредством этой подсистемы также называется аутентификацией SQL Server.)

На основе этих двух подсистем безопасности компонент Database Engine может работать в одном из следующих режимов аутентификации: в режиме Windows и в смешанном режиме.

Режим Windows требует, чтобы пользователи входили в систему баз данных исключительно посредством своих учетных записей Windows. Система принимает данные учетной записи пользователя, полагая, что они уже были проверены и одобрены на уровне операционной системы. Такой способ подключения к системе баз данных называется доверительным соединением (trusted connection), т.к. система баз данных доверяет, что операционная система уже проверила подлинность учетной записи и соответствующего пароля.

Смешанный режим позволяет пользователям подключаться к компоненту Database Engine посредством аутентификации Windows или аутентификации SQL Server. Это означает, что некоторые учетные записи пользователей можно настроить для использования подсистемы безопасности Windows, а другие, вдобавок к этому, могут использовать также и подсистему безопасности SQL Server. Аутентификация SQL Server предоставляется исключительно в целях обратной совместимости. Поэтому зачастую следует использовать аутентификацию Windows.

Реализация режима аутентификации

Выбор одного из доступных режимов аутентификации осуществляется посредством среды SQL Server Management Studio. Чтобы установить режим аутентификации Windows, щелкните правой кнопкой сервер баз данных в окне Object Explorer и в контекстном меню выберите пункт Properties. Откроется диалоговое окно Server Properties, в котором нужно

выбрать страницу Security, а на ней Windows Authentication Mode. Для выбора смешанного режима в этом же диалоговом окне Server Properties вам нужно выбрать Server and Windows Authentication Mode.

После успешного подключения пользователя к компоненту Database Engine его доступ к объектам базы данных становится независимым от использованного способа аутентификации для входа в базу данных - аутентификации Windows или SQL Server аутентификации.

Шифрование данных

Шифрование - это процесс приведения данных в запутанное непонятное состояние, вследствие чего повышается уровень их безопасности. Обычно конкретная процедура шифрования осуществляется с использованием определенного алгоритма. Наиболее важный алгоритм шифрования называется **RSA**, по первым буквам фамилий его создателей - Rivers, Shamir и Adelman.

Компонент Database Engine обеспечивает безопасность данных посредством иерархии уровней шифрования и инфраструктуры управления ключами. Каждый уровень защищает следующий за ним уровень шифрования, используя комбинацию сертификатов, асимметричных и симметричных ключей:

На рисунке выше главный сервисный ключ задает ключ, который управляет всеми другими ключами и сертификатами. Главный сервисный ключ создается автоматически при установке компонента Database Engine. Этот ключ зашифрован с помощью *API-интерфейса защиты данных Windows (DPAPI - Data Protection API).*

Важным свойством главного сервисного ключа является то, что он управляется системой. Хотя системный администратор может выполнять разные задачи по обслуживанию ключа, ему следует выполнять лишь одну из них - резервное копирование главного сервисного ключа, чтобы его можно было восстановить в случае повреждения.

Как можно видеть на рисунке, главный сервисный ключ базы данных является корневым объектом шифрования на уровне базы данных для всех ключей, сертификатов и данных. Каждая база данных имеет один главный ключ базы данных, который создается посредством инструкции **CREATE MASTER KEY**. Поскольку главный ключ базы данных защищен главным сервисным ключом системы, то система может автоматически расшифровывать главный ключ базы данных.

Существующий главный ключ базы данных можно использовать для создания пользовательских ключей. Существует три формы пользовательских ключей:

- симметричные ключи;
- асимметричные ключи;

– сертификаты.

Эти формы пользовательских ключей рассмотрены в следующих далее подразделах.

Симметричные ключи

В системе шифрования с использованием симметричного ключа оба участника обмена - отправитель и получатель сообщения - применяют один и тот же ключ. Иными словами, для шифрования информации и ее обратного расшифровывания используется этот единственный ключ. Использование симметричных ключей имеет много преимуществ и один недостаток. Одним из преимуществ использования симметричных ключей является то обстоятельство, что с их помощью можно защитить значительно больший объем данных, чем с помощью двух других типов ключей. Кроме этого, использование ключей этого типа намного быстрее, чем использование несимметричных ключей. Но с другой стороны, использование симметричного ключа в среде распределенной системы может сделать задачу обеспечения целостности шифрования почти невыполнимой, поскольку один и тот же ключ применяется на обоих концах обмена данными. Язык Transact-SQL поддерживает несколько инструкций и системных функций применительно к симметричным ключам. В частности, для создания симметричного ключа применяется инструкция **CREATE SYMMETRIC KEY**, а для удаления существующего симметричного ключа - инструкция **DROP SYMMETRIC KEY**. Прежде чем симметричный ключ можно использовать для шифрования данных или для защиты другого ключа, его нужно открыть. Для этого используется инструкция **OPEN SYMMETRIC KEY.**

После того как вы откроете симметричный ключ, вам нужно для шифрования использовать системную функцию **EncryptByKey**. Эта функция имеет два входных параметра: идентификатор ключа и текст, который требуется зашифровать. Для расшифровки зашифрованной информации применяется системная функция **DecryptByKey.**

Асимметричные ключи

В случае если у вас имеется распределенная система или если использование симметричных ключей не обеспечивает достаточного уровня безопасности данных, то следует использовать асимметричные ключи. ***Асимметричный ключ*** состоит из двух частей: личного закрытого ключа (private key) и соответствующего общего открытого ключа (public key). Каждый из этих ключей может расшифровывать данные, зашифрованные другим ключом. Благодаря наличию личного закрытого ключа асимметричное шифрование обеспечивает более высокий уровень безопасности данных, чем симметричное шифрование. Язык Transact-SQL поддерживает несколько инструкций и системных функций применительно к асимметричным ключам. В частности, для создания нового асимметричного ключа применяется инструкция **CREATE ASYMMETRIC KEY**, а для изменения свойств асимметричного ключа используется инструкция **ALTER ASYMMETRIC KEY**. Для удаления асимметричного ключа применяется инструкция **DROP ASYMMETRIC KEY.**

После того как вы создали асимметричный ключ, для шифрования данных используйте системную функцию **EncryptByAsymKey**. Эта функция имеет два входных параметра: идентификатор ключа и текст, который требуется зашифровать. Для расшифровки информации, зашифрованной с использованием асимметричного ключа, применяется системная функция **DecryptByAsymKey.**

Сертификаты

Сертификат открытого ключа, или просто сертификат, представляет собой предложение с цифровой подписью, которое привязывает значение открытого ключа к определенному лицу, устройству или службе, которая владеет соответствующим

открытым ключом. Сертификаты выдаются и подписываются *центром сертификации (Certification Authority - CA)*. Сущность, которая получает сертификат из центра сертификации (CA), является субъектом данного сертификата (certificate subject).

Между сертификатами и асимметричными ключами нет значительной функциональной разницы. Как первый, так и второй используют алгоритм RSA. Основная разница между ними заключается в том, что асимметричные ключи создаются вне сервера.

Сертификаты содержат следующую информацию:

- значение открытого ключа субъекта;
- информацию, идентифицирующую субъект;
- информацию, идентифицирующую издателя сертификата;
- цифровую подпись издателя сертификата.

Основным достоинством сертификатов является то, что они освобождают хосты от необходимости содержать набор паролей для отдельных субъектов. Когда хост, например, безопасный веб-сервер, обозначает издателя как надежный центр авторизации, этот хост неявно доверяет, что данный издатель выполнил проверку личности субъекта сертификата.

Сертификаты предоставляют самый высший уровень шифрования в модели безопасности компонента Database Engine. Алгоритмы шифрования с использованием сертификатов требуют большого объема процессорных ресурсов. По этой причине сертификаты следует использовать при реальной необходимости.

Наиболее важной инструкцией применительно к сертификатам является **инструкция CREATE CERTIFICATE**. Использование этой инструкции показано в примере ниже:

```
USE SampleDb;
CREATE MASTER KEY
ENCRYPTION BY PASSWORD = '12345!'
GO
CREATE CERTIFICATE cert01
   WITH SUBJECT = 'Сертификат для схемы dbo';
```

Чтобы создать сертификат без параметра ENCRYPTION BY, сначала нужно создать главный ключ базы данных. (Все инструкции CREATE CERTIFICATE, которые не содержат этот параметр, защищаются главным ключом базы данных.) По этой причине первой инструкцией в примере выше является инструкция CREATE MASTER KEY. После этого инструкция CREATE CERTIFICATE используется для создания нового сертификата cert01, владельцем которого является объект dbo базы данных SampleDb, если этот объект является текущим пользователем.

Редактирование пользовательских ключей

Наиболее важными представлениями каталога применительно к шифрованию являются следующие:

- sys.symmetric_keys
- sys.asymmetric_keys
- sys.certificates
- sys.database_principals

Первые три представления каталога предоставляют информацию обо всех симметричных ключах, всех асимметричных ключах и всех сертификатах, установленных в текущей базе данных, соответственно. Представление каталога sys.database_principals предоставляет информацию обо всех принципалах в текущей базе данных. ***Принципалы (principals)*** - это субъекты, которые имеют разрешение на доступ к определенной

сущности. Типичными принципалами являются учетные записи Windows и SQL Server. Кроме этих принципалов, также существуют группы Windows и роли SQL Server. Группа Windows - это коллекция учетных записей и групп Windows. Присвоение учетной записи пользователя членство в группе дает этому пользователю все разрешения, предоставленные данной группе. Подобным образом роль является коллекцией учетных записей.

Представление каталога sys.database_principals можно соединить с любым из первых трех, чтобы получить информацию о владельце определенного ключа. В примере ниже показано получение информации о существующих сертификатах. Подобным образом можно получить информацию о симметричных и асимметричных ключах.

```
USE SampleDb;
SELECT p.name, c.name, certificate_id
   FROM sys.certificates c, sys.database_principals p
   WHERE p.principal_id = p.principal_id
```

Results | Messages

	name	name	certificate...
1	public	cert01	256
2	dbo	cert01	256
3	guest	cert01	256
4	INFORMATION_SCHEMA	cert01	256
5	sys	cert01	256
6	db_owner	cert01	256
7	db_accessadmin	cert01	256
8	db_securityadmin	cert01	256
9	db_ddladmin	cert01	256
10	db_backupoperator	cert01	256
11	db_datareader	cert01	256
12	db_datawriter	cert01	256
13	db_denydatareader	cert01	256
14	db_denydatawriter	cert01	256

Расширенное управление ключами SQL Server

Следующим шагом к обеспечению более высокого уровня безопасности ключей является использование ***расширенного управления ключами (Extensible Key Management - EKM)***. Основными целями расширенного управления ключами являются следующие:

- повышение безопасности ключей посредством выбора поставщика функций шифрования;
- общее управление ключами по всему приложению.

Расширенное управление ключами позволяет сторонним разработчикам регистрировать свои устройства в компоненте Database Engine. Когда такие устройства зарегистрированы, регистрационные имена (logins) в SQL Server могут использовать хранящиеся в них ключи шифрования, а также эффективно использовать продвинутые возможности шифрования, поддерживаемые этими модулями. Расширенное управление ключами позволяет защитить данные от доступа администраторов базы данных (за исключением членов группы sysadmin). Таким образом система защищается от пользователей с повышенными привилегиями. Данные можно зашифровывать и расшифровывать, используя инструкции шифрования языка Transact-SQL, а SQL Server может использовать внешнее устройство расширенного управления ключами для хранения ключей.

Способы шифрования данных

SQL Server поддерживает два способа шифрования данных:

- шифрование на уровне столбцов;
- прозрачное шифрование данных.

Шифрование на уровне столбцов позволяет шифровать конкретные столбцы данных. Для реализации этого способа шифрования используется несколько пар сопряженных функций. Далее мы не будем рассматривать этот метод шифрования, поскольку его реализация является сложным ручным процессом, требующим внесения изменений в приложение.

Прозрачное шифрование данных является новой возможностью базы данных, которая зашифровывает файлы базы данных автоматически, не требуя внесения изменений в какие-либо приложения. Таким образом можно предотвратить доступ к информации базы данных несанкционированным лицам, даже если они смогут получить в свое распоряжение основные или резервные файлы базы данных.

Файлы базы данных зашифровываются на уровне страниц. Страницы зашифрованной базы данных шифруются перед тем, как они записываются на диски и расшифровываются при считывании с диска в память.

Как и большинство других методов шифрования, прозрачное шифрование данных основано на ключе шифрования. В нем используется симметричный ключ, посредством которого и защищается база данных.

Применения прозрачного шифрования для защиты определенной базы данных реализуется в четыре этапа:

1. Используя инструкцию CREATE MASTER KEY, создается главный ключ базы данных.

2. С помощью инструкции CREATE CERTIFICATE создается сертификат.

3. Используя инструкцию **CREATE DATABASE ENCRYPTION KEY**, создается ключ шифрования.

4. Выполняется конфигурирование базы данных для использования шифрования. (Этот шаг можно реализовать, присвоив параметру ENCRYPTION инструкции ALTER DATABASE значение ON.)

Настройка безопасности компонента Database Engine

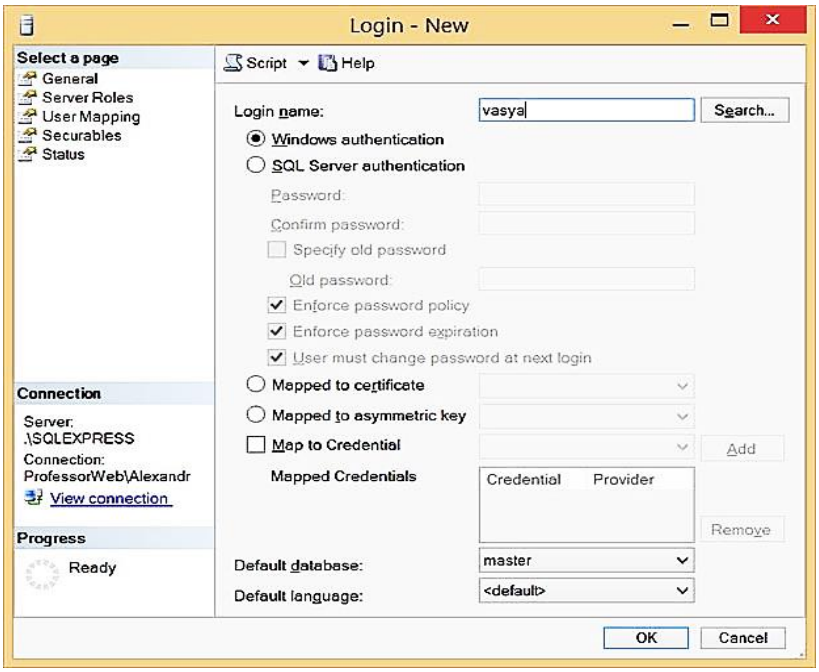
Настройку безопасности компонента Database Engine можно выполнить одним из следующих способов:

- с помощью среды управления Management Studio сервера SQL Server;
- используя инструкции языка Transact-SQL.

Эти два метода рассматриваются в последующих подразделах.

Управление безопасностью с помощью среды Management Studio

Чтобы с помощью среды Management Studio создать новое регистрационное имя, разверните в обозревателе объектов Object Explorer узел сервера, затем разверните папку "Security", в этой папке щелкните правой кнопкой папку "Logins" и в контекстном меню выберите опцию New Login. Откроется диалоговое окно Login - New:



Первым делом нужно решить, какой способ аутентификации применять: Windows или SQL Server. В случае выбора аутентификации Windows, в качестве регистрационного имени (Login name) необходимо указать действительное имя пользователя Windows в форме domain\user_name (домен\имя_пользователя). А если выбрана аутентификация SQL Server, необходимо ввести новое регистрационное имя (Login name) и соответствующий пароль (Password). Можно также указать базу данных и язык по умолчанию. База данных по умолчанию - это база данных, к которой пользователь автоматически подключается сразу же после входа в компонент Database Engine. Выполнив все эти действия, пользователь может входить в систему под этой новой учетной записью.

Управление безопасностью посредством инструкций Transact-SQL

Для управления безопасностью компонента Database Engine применяются три инструкции языка Transact-SQL: CREATE LOGIN, ALTER LOGIN и DROP LOGIN. Инструкция **CREATE LOGIN** создает новое регистрационное имя входа в SQL Server. Синтаксис этой инструкции следующий:

```
CREATE LOGIN login_name
   { WITH option_list1 |
   FROM {WINDOWS [ WITH option_list2 [,...] ]
      | CERTIFICATE certname | ASYMMETRIC KEY key_name }}
```

В параметре login_name указывается создаваемое регистрационное имя. Как можно видеть в синтаксисе этой инструкции, в предложении WITH можно указать один или несколько параметров для регистрационного имени или указать в предложении FROM сертификат, асимметричный ключ или учетную запись пользователя Windows, связанную с соответствующим регистрационным именем.

В списке option_list1 указывается несколько параметров, наиболее важным из которых является параметр password, который задает пароль для данного регистрационного имени. (Другие возможные параметры - DEFAULT_DATABASE, DEFAULT_LANGUAGE и CHECK_EXPIRATION.)

Как видно из синтаксиса инструкции CREATE LOGIN, предложение FROM может содержать один из следующих параметров:

Параметр WINDOWS

Указывает, что данное регистрационное имя соотносится с существующей учетной записью пользователя Window. Этот параметр можно указать с другими подпараметрами, такими как default_database и default_language.

Параметр CERTIFICATE

Задает имя сертификата для привязки к данному регистрационному имени.

Параметр ASYMMETRIC KEY

Задает имя асимметричного ключа для привязки к данному регистрационному имени. (Сертификат и асимметричный ключ уже должны присутствовать в базе данных master.)

В примерах ниже показано создание разных форм регистрационного имени. В следующем примере создается регистрационное имя "Vasya" с паролем "12345!":

```
USE SampleDb;
CREATE LOGIN Vasya WITH PASSWORD = '12345!';
```

В примере ниже создается регистрационное имя "Vasya", которое сопоставляется с учетной записью пользователя Windows с таким же самым именем пользователя:

```
USE SampleDb;
CREATE LOGIN [ProfessorWeb\vasya] FROM WINDOWS;
```

Для конкретной системной среды нужно должным образом изменить имя компьютера и имя пользователя (в примере выше это ProfessorWeb и vasya соответственно).

Вторая инструкция языка Transact-SQL для обеспечения безопасности **ALTER LOGIN** - изменяет свойства определенного регистрационного имени. С помощью этой инструкции можно изменить текущий пароль и его конечную дату действия, параметры

доступа, базу данных по умолчанию и язык по умолчанию. Также можно задействовать или отключить определенное регистрационное имя.

Наконец, инструкция **DROP LOGIN** применяется для удаления существующего регистрационного имени. Однако регистрационное имя, которое ссылается (владеет) на другие объекты, удалить нельзя.

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Проблемы проверки подлинности безопасности системы СУБД
2. Архитектуры системы безопасности в MS SQL Server
3. Режимы защиты безопасности MS SQL Server

Лабораторная работа №15.Выполнять резервное копирование баз данных на SQL Server

Цель работы: Ознакомление с резервными копированием базы данных на SQL Server.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть

Microsoft SQL Server позволяет создавать резервные копии и восстанавливать базы данных. Компонент резервного копирования и восстановления SQL Server предоставляет необходимую защиту важных данных, которые хранятся в базах данных SQL Server. Хорошо продуманная стратегия резервного копирования и восстановления защищает базы от потери данных при повреждениях, происходящих из-за различных сбоев. Проверьте выбранную стратегию, выполнив восстановление баз данных из набора резервных копий; это поможет эффективно среагировать на реальные проблемы.

Резервная копия — это копия данных, используемая для восстановления данных. Резервные копии позволяют восстанавливать данные после сбоя. При правильном создании резервных копий можно будет восстановить базу после многих сбоев, таких как:

- сбой носителя;
- ошибки пользователей (например, удаление таблицы по ошибке);
- сбои оборудования (например, поврежденный дисковый накопитель или безвозвратная потеря данных на сервере);
- стихийные бедствия.

Кроме того, резервные копии баз данных полезны и для выполнения повседневных административных задач, например для копирования базы данных с одного сервера на другой, настройки зеркального отображения баз данных и архивирования.

Типы резервного копирования SQL Server

Тип резервной копии	Описание
Полная	Все файлы данных и часть журнала транзакций
Журнал транзакций	Любые изменения базы данных, записанные в файлах журнала
Заключительные фрагменты журнала	Активная часть журнала
Разностная	Части базы данных, которые изменились с момента выполнения полного резервного копирования базы данных
Файл (файловая группа)	Указанные файлы или файловые группы
Частичная	Первичная файловая группа, все файловые группы, доступные для чтения и записи, и любые указанные файловые группы, доступные только для чтения
Доступная только для копирования	База данных или журнал (не оказывается влияния на последовательность резервного копирования)

В SQL Server предоставляется несколько методов резервного копирования для удовлетворения требований всевозможных сфер бизнеса и разнообразных применений баз данных.

Полные резервные копии

Полная резервная копия базы данных содержит файлы данных и часть журнала транзакций. Полная резервная копия представляет базу данных на момент создания резервной копии и служит основным источником данных в случае сбоя системы. При осуществлении полного резервного копирования базы данных сервером SQL Server выполняются следующие действия:

- резервное копирование всех данных в базе данных;
- резервное копирование всех изменений, которые возникают во время выполнения резервного копирования;
- резервное копирование всех транзакций, не зафиксированных в журнале транзакций.

Сервером SQL Server используются части журнала транзакций, которые были записаны в файл резервной копии для обеспечения согласованности данных при восстановлении резервной копии. Восстановленная база данных совпадает с состоянием базы данных на момент завершения резервного копирования за исключением всех незафиксированных транзакций. При восстановлении базы данных производится откат незафиксированных транзакций. Если база данных доступна только для чтения, возможно, полных резервных копий будет достаточно для предотвращения потери данных.

Резервные копии журнала транзакций

В резервные копии журнала транзакций записываются все изменения базы данных. Резервное копирование журналов транзакций обычно выполняется при создании полных резервных копий базы данных. Обратите внимание на следующие факты, касающиеся резервных копий журналов транзакций:

- не следует выполнять резервное копирование журнала, если хотя бы раз не создавалась полная резервная копия базы данных;
- журналы транзакций невозможно восстановить без соответствующей резервной копии базы данных;
- при использовании простой модели восстановления невозможно создать резервные копии журналов транзакций.

При резервном копировании журнала транзакций сервером SQL Server выполняется следующее:

- Создаются резервные копии журнала транзакций от последней успешно выполненной инструкции BACKUP LOG до конца текущего журнала транзакций.
- Усекается журнал транзакций до начала активной части журнала транзакций, и отбрасываются сведения в неактивной части.

Активная часть журнала транзакций начинается с момента самой последней открытой транзакции и продолжается до конца журнала транзакций.

Резервные копии заключительных фрагментов журнала

Резервная копия заключительных фрагментов журнала — это резервная копия журнала транзакций, включающая часть журнала, которая ранее не подвергалась резервному копированию (известна как активная часть журнала). Резервное копирование заключительных фрагментов журнала осуществляется без усечения журнала и обычно используется, когда файлы данных становятся недоступными для базы данных, но файл журнала не поврежден.

Разностные резервные копии

Разностное резервное копирование следует выполнять для минимизации времени, которое необходимо для восстановления часто изменяемой базы данных. Разностное резервное копирование возможно только в том случае, когда создана полная резервная копия базы данных. Когда создаются разностные резервные копии, сервером SQL Server выполняются следующие действия:

- Создаются резервные копии частей базы данных, которые изменились с момента выполнения полного резервного копирования базы данных.
- Создаются резервные копии всех операций, происходивших во время разностного резервного копирования, а также всех транзакций, не зафиксированных в журнале транзакций.

Резервные копии файлов и файловых групп

Если выполнение полного резервного копирования очень больших баз данных нецелесообразно с практической точки зрения, можно создать резервные копии файлов и файловых групп базы данных. Когда создаются резервные копии файлов и файловых групп, сервером SQL Server выполняются следующие действия:

- Создаются резервные копии только файлов базы данных, которые указаны в параметре FILE или F1LEGROUP.

– Разрешается резервное копирование конкретных файлов базы данных вместо всей базы данных.

При создании резервных копий файлов и файловых групп необходимо:

– указать логические файлы и файловые группы;

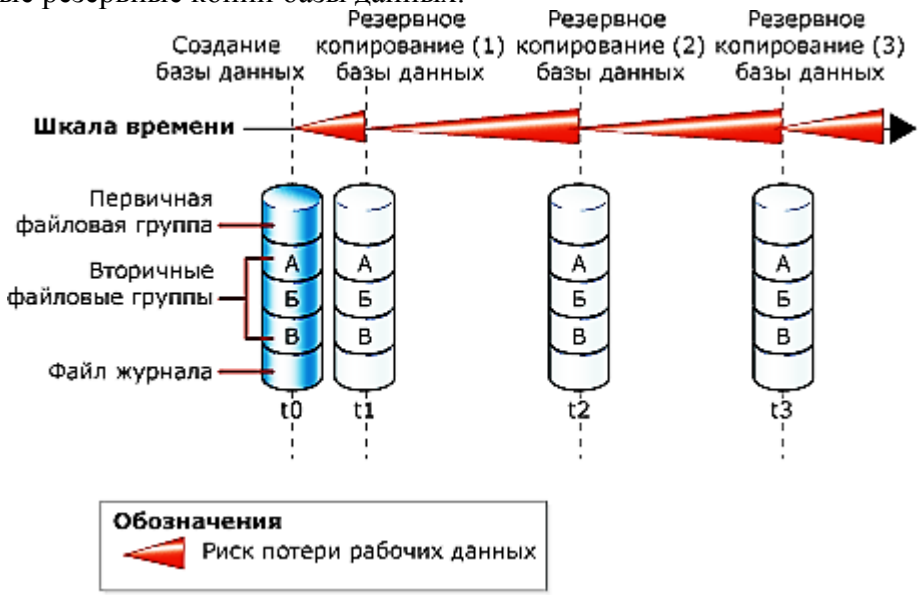
– создать резервные копии журнала транзакций, чтобы восстанавливаемые файлы согласовывались с остальной базой данных;

– создать план резервного копирования каждого файла на циклической основе, чтобы обеспечить регулярное резервное копирование всех файлов и файловых групп базы данных.

Практическая часть

Резервные копии баз данных при простой модели восстановления

При использовании простой модели восстановления после создания каждой резервной копии база данных уязвима для потенциальной потери данных в случае аварийной ситуации. Потенциальные потери работы растут с каждым внесенным изменением до следующего создания резервной копии, после чего объем потенциальных потерь работы снова уменьшается до нуля и начинается новый цикл. Потенциальные потери работы становятся тем больше, чем больше времени прошло со времени создания последней резервной копии. На следующем рисунке показана вероятность потери данных при использовании стратегии резервного копирования, в которой применяются только полные резервные копии базы данных.



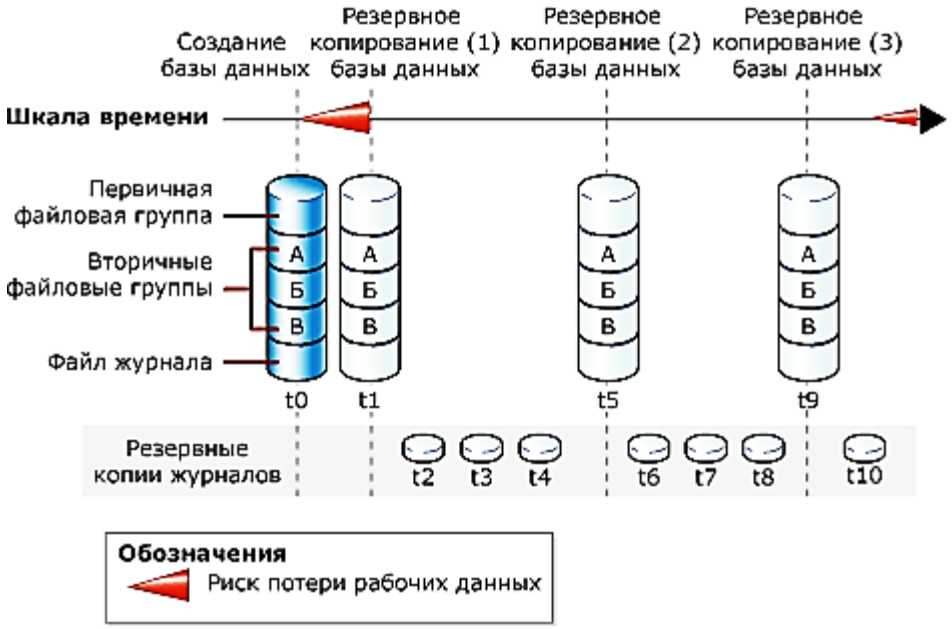
Пример (Transact-SQL)

В следующем примере показано, как создать полную резервную копию базы данных с помощью предложения WITH FORMAT, чтобы перезаписать все существующие резервные копии и создать новый набор носителей.

```
-- Back up the AdventureWorks2012 database to new media set.
BACKUP DATABASE AdventureWorks2012
    TO                          DISK                          =
'Z:\SQLServerBackups\AdventureWorksSimpleRM.bak'
    WITH FORMAT;
GO
```

Резервные копии баз данных при модели полного восстановления

Для баз данных, в которых используются полная модель восстановления и модель восстановления с неполным протоколированием, создание резервных копий баз данных необходимо, но недостаточно для нормальной работы. Требуется также создание резервных копий журнала транзакций. На следующем рисунке показан минимальный вариант полной стратегии резервного копирования, доступный при использовании модели полного восстановления.



Пример (Transact-SQL)

В следующем примере показано, как создать полную резервную копию базы данных с помощью предложения WITH FORMAT, чтобы перезаписать все существующие резервные копии и создать новый набор носителей. Затем в примере производится резервное копирование журнала транзакций. В реальной ситуации, возможно, придется создать ряд обычных резервных копий журнала. В этом примере образец базы данных **AdventureWorks2012** должен быть переключен на модель полного восстановления.

```
USE master;
ALTER DATABASE AdventureWorks2012 SET RECOVERY FULL;
GO
-- Back up the AdventureWorks2012 database to new media set
(backup set 1).
BACKUP DATABASE AdventureWorks2012
  TO                              DISK                              =
'Z:\SQLServerBackups\AdventureWorks2012FullRM.bak'
  WITH FORMAT;
GO
--Create a routine log backup (backup set 2).
BACKUP     LOG     AdventureWorks2012     TO     DISK     =
'Z:\SQLServerBackups\AdventureWorks2012FullRM.bak';
GO
```

Восстановление базы данных с помощью полной резервной копии

Из полной резервной копии базы данных путем ее восстановления можно за один этап восстановить всю базу данных в любом местоположении. В резервную копию включается достаточная часть журнала транзакций, чтобы можно было восстановить базу

данных в состоянии на момент окончания резервного копирования. Восстановленная база данных соответствует состоянию исходной базы данных на момент окончания резервного копирования, за вычетом незавершенных транзакций. Согласно модели полного восстановления затем следует выполнить восстановление всех последующих резервных копий журналов транзакций. После восстановления базы данных выполняется откат незавершенных транзакций.

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Проблемы проверки подлинности безопасности системы СУБД
2. Архитектуры системы безопасности в MS SQL Server
3. Режимы защиты безопасности MS SQL Server

Лабораторная работа №16.Выдача записей для дополнительного доступа к модулям путем сравнения сертификатов в базе данных.

Цель работы: Изучение способа выдача записей для дополнительного доступа к модулям путем сравнения сертификатов в базе данных.

***Необходимые принадлежности*:** Персональный компьютер, Программное обеспечение, принтер.

Теоретическая часть

SQL Server поддерживает явное олицетворение другого участника с помощью автономной инструкции EXECUTE AS, а также неявное олицетворение с помощью предложения EXECUTE AS в модулях. Олицетворение участников уровня сервера и имен входа выполняется с помощью автономной инструкции EXECUTE AS путем использования инструкции EXECUTE AS LOGIN. Олицетворение участников уровня базы данных и имен пользователей выполняется с помощью автономной инструкции EXECUTE AS путем использования инструкции EXECUTE AS USER. Неявные олицетворения, выполняемые через предложение EXECUTE AS в модулях, олицетворяют указанного пользователя или имя учетной записи на уровне базы данных или сервера. Олицетворение зависит от уровня модуля: это либо модуль уровня базы данных (например, хранимая процедура или функция), либо модуль уровня сервера (например, триггер уровня сервера).

Основные сведения об области олицетворения

При олицетворении участника с помощью инструкции EXECUTE AS LOGIN или с помощью предложения EXECUTE AS внутри модуля области сервера областью олицетворения является весь сервер. Это означает, что после смены контекста становится доступен любой ресурс внутри сервера, к которому имеет доступ олицетворенное имя входа учетной записи.

Однако при олицетворении участника с помощью инструкции EXECUTE AS USER или с помощью предложения EXECUTE AS внутри модуля области базы данных область олицетворения по умолчанию ограничена базой данных. Это означает, что ссылки на объекты вне области базы данных будут возвращать ошибку. Причина такого поведения по умолчанию рассматривается в следующем сценарии.

Владелец базы данных может, имея полные права внутри базы данных, не иметь никаких прав вне области базы данных. Поэтому SQL Server не разрешит владельцу базы данных олицетворять или давать кому-либо возможность олицетворить другого пользователя, чтобы тот получил доступ к ресурсам вне области текущих разрешений владельца базы данных.

Например, рассмотрим две базы данных на одном сервере, принадлежащие разным владельцам. **Database1** принадлежит Бобу, а **Database2** — Фреду. Боб не хочет, чтобы у Фреда был доступ к его базе данных, и наоборот. Как владелец **Database1** Боб может создать в своей базе данных пользователя для Фреда, и, имея все разрешения внутри **Database 1**, Боб может олицетворить пользователя Фред. Однако накладываемые SQL Server ограничения безопасности не позволяют Бобу получить доступ к базе данных Фреда в контексте олицетворения. Без наличия этих ограничений по умолчанию Боб получил бы анонимный доступ к данным Фреда. Именно поэтому область олицетворений уровня базы данных ограничена по умолчанию базой данных.

Однако в некоторых случаях может понадобиться выборочно расширить область олицетворения за пределы базы данных. Например, если приложение использует две базы данных и требует доступа к одной базе данных из другой.

Рассмотрим случай маркетингового приложения, которое вызывает хранимую процедуру с именем **GetSalesProjections** в базе данных **Marketing**. В этой хранимой процедуре задан переключатель контекста выполнения. Хранимая процедура запрашивает в базе данных **Sales** сведения о продажах из таблицы **SalesStats**. По умолчанию этот сценарий не сработает, поскольку контекст выполнения, установленный внутри одной базы данных, является недопустимым вне этой базы данных. Однако разработчики маркетингового приложения не предоставляют пользователям этого приложения прямой доступ к базе данных **Sales** или права на какой-либо объект внутри нее. Идеальным решением будет олицетворение пользователя, обладающего необходимыми разрешениями в базе данных **Sales**, с помощью предложения EXECUTE AS в хранимой процедуре. Однако текущие ограничения по умолчанию это запрещают. В SQL Server можно выборочно расширять текущую область олицетворения базы данных путем настройки доверительной модели отношений между двумя базами данных. Однако, перед тем как знакомиться с описанием этой модели и способами выборочного расширения области олицетворения, необходимо разобраться в принципах проверки подлинности и понять роль средств проверки подлинности в SQL Server.

Основные сведения о средствах проверки подлинности

Проверка подлинности — это процесс, посредством которого участник доказывает системе свою подлинность. Средство проверки подлинности — это объект, выполняющий проверку подлинности отдельного участника, то есть подтверждающий его подлинность. Например, при подключении к SQL Server подлинность имени входа, заданного для текущего соединения, проверяется экземпляром SQL Server.

Рассмотрим случай, когда пользователь явно переключает контекст на уровне сервера с помощью инструкции EXECUTE AS LOGIN. Для этого обязательно наличие разрешений на олицетворение на уровне сервера. Эти разрешения позволяют участнику, вызвавшему инструкцию EXECUTE AS LOGIN, олицетворять указанное имя входа в пределах всего экземпляра SQL Server. Следовательно, эта инструкция позволяет ему имитировать вход в систему под олицетворенным именем входа. Владелец разрешений области уровня сервера — **sysadmin**, который является обладателем экземпляра SQL Server. При олицетворении уровня сервера средство проверки подлинности — это **sysadmin** или сам экземпляр SQL Server.

Однако рассмотрим случай, когда контекст устанавливается инструкцией EXECUTE AS USER или предложением EXECUTE AS в модуле области базы данных. В этих случаях проверяются разрешения на олицетворение внутри области базы данных. Область по умолчанию разрешений IMPERSONATE для пользователей — это сама база данных, владельцем которой является **dbo**. Кроме того, владелец базы данных является средством проверки подлинности этих олицетворений. Именно владелец базы данных устанавливает подлинность олицетворенного пользователя и подтверждает его подлинность. Поскольку владельцу базы данных принадлежит вся база данных, олицетворенный контекст считается подлинным внутри всей указанной базы данных. Однако вне этой базы данных олицетворенный контекст является недопустимым.

Применение средств проверки подлинности

С помощью средств проверки подлинности проверяется допустимость установленного контекста внутри определенной области. Чаще всего средствами проверки подлинности являются системный администратор (SA), экземпляр SQL Server, а также **dbo** (при работе с базами данных). Средство проверки подлинности — это, в конечном итоге, владелец области, внутри которой установлен контекст для отдельного пользователя или имени входа. Сведения о средстве проверки подлинности получаются из информации о маркере, поддерживаемой для имени входа и пользователя, и доступны через представления sys.user_token и sys.login_token. Дополнительные сведения см. в разделе Основные сведения о контексте выполнения.

Контекст выполнения, принадлежащий владельцу области как ее средству проверки подлинности, действителен внутри всей этой области. Это обусловлено тем, что владелец области, например базы данных, является неявно доверенным для всех сущностей внутри области. Контекст так же действителен в других областях, например в других базах данных или непосредственно экземплярах SQL Server, в которых средство проверки подлинности является доверенным. Следовательно, действительность олицетворенного пользовательского контекста вне области базы данных зависит от того, является ли средство проверки подлинности доверенным для контекста внутри целевой области. Эта «доверенность» устанавливается путем выдачи средству проверки подлинности разрешения AUTHENTICATE, если целевая область является другой базой данных, или путем выдачи разрешения AUTHENTICATE SERVER, если целевая область — экземпляр SQL Server.

Расширение области олицетворения

Для расширения области олицетворения от текущей базы данных до целевой области, например до другой базы данных или экземпляра SQL Server, необходимо выполнить следующие условия.

- Средство проверки подлинности должно быть доверенным в целевой области.
- База данных-источник должна быть помечена как заслуживающая доверия.

Доверие средству проверки подлинности

На основании предыдущего примера с базами данных **Sales** и **Marketing** ниже показано, как хранимая процедура **GetSalesProjections** в базе данных **Marketing** обрабатывает данные из таблицы **SalesStats** в базе данных **Sales**. Хранимая процедура содержит предложение EXECUTE AS USER **MarketingExec**. Владельцем базы данных **Sales** является **SalesDBO**, владельцем базы данных **Marketing** — **MarketingDBO**.

Если хранимая процедура **GetSalesProjections** вызывается пользователем, то предложение EXECUTE AS неявно переключает контекст выполнения хранимой процедуры с вызвавшего пользователя на пользователя **MarketingExec**. Средством проверки подлинности для этого контекста является **MarketingDBO** — владелец базы данных **Marketing**. По умолчанию процедура имеет доступ к любым ресурсам внутри базы данных **Marketing**, к которой разрешен доступ пользователю **MarketingExec**. Однако, чтобы получить доступ к таблице базы данных **Sales**, база данных **Sales** должна доверять средству проверки подлинности **MarketingDBO**.

Для этого в базе данных **Sales** нужно создать пользователя с именем **MarketingDBO**, сопоставленным с именем входа **MarketingDBO**, и выдать пользователю разрешение AUTHENTICATE в базе данных **Sales**. В результате внутри базы данных действительным является любой контекст выполнения, средство проверки подлинности которого обладает этим разрешением. Поскольку средству проверки подлинности **MarketingDBO** предоставлено разрешение AUTHENTICATE в базе данных **Sales**, контекст для пользователя **MarketingExec**, установленный предложением EXECUTE AS в хранимой процедуре **GetSalesProjections** в базе данных **Marketing**, является доверенным в базе данных **Sales**.

Доверие базе данных

В SQL Server доверительная модель продвинулась на шаг вперед в обеспечении дополнительной безопасности и гранулярности в расширении области олицетворения уровня базы данных. Как вариант, доверительные отношения между целевой областью и средством проверки подлинности контекста можно устанавливать с помощью разрешения AUTHENTICATE, но, кроме того, можно определить, доверяет ли экземпляр SQL Server базе данных-источнику и ее содержимому.

Предположим, что участник **MarketingDBO** является владельцем другой базы данных с именем **Conference**. Предположим также, что владельцу базы данных с именем **MarketingDBO** нужно, чтобы контексты выполнения, указанные внутри базы данных **Marketing**, обладали доступом к ресурсам в базе данных **Sales**. Однако при этом у контекстов, установленных в базе данных **Conference**, не должно быть доступа к базе данных **Sales.**

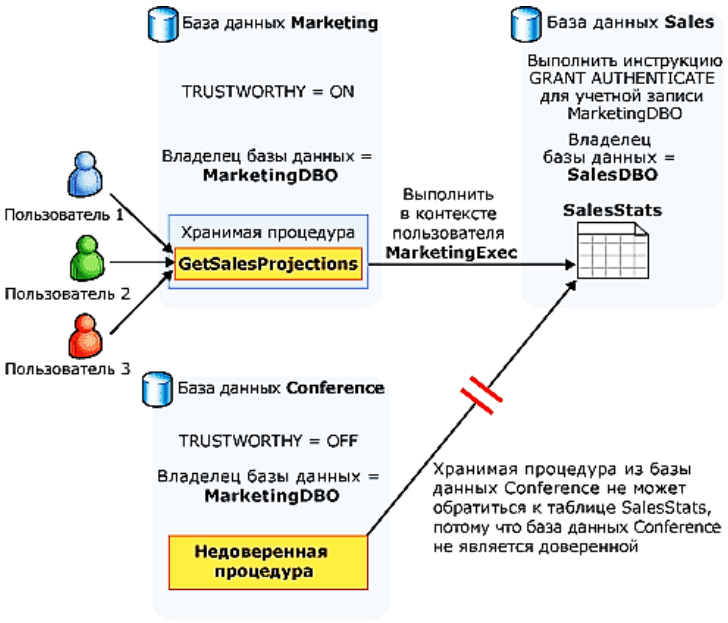
Чтобы эти требования выполнялись, база данных, содержащая модуль, в котором используется олицетворенный контекст для доступа к ресурсам вне базы данных, должна быть помечена как заслуживающая доверия. Свойство TRUSTWORTHY показывает, доверяет ли экземпляр SQL Server базе данных и ее содержимому. Свойство TRUSTWORTHY решает две задачи:

1. Оно уменьшает опасность, исходящую от подключенных к экземпляру SQL Server баз данных, которые могут содержать опасные модули, выполняемые в контексте пользователя с обширными правами доступа.
2. Для этого подключенные базы данных по умолчанию не помечаются как заслуживающие доверия. Кроме того, для доступа к ресурсам вне базы данных с помощью потенциально опасных модулей необходимо, чтобы база данных была помечена как заслуживающая доверия. Свойство TRUSTWORTHY в базе данных могут задавать только члены предопределенной роли сервера **sysadmin.**
3. Это позволяет администратору экземпляра SQL Server выявлять среди баз данных, принадлежащих одному владельцу (который является доверенным средством проверки подлинности в некоторой области), те базы данных, которым следует предоставить доступ к внешним ресурсам, и те, которым его предоставлять не следует.

Именно для этого предназначено свойство TRUSTWORTHY. Например, предположим, что олицетворенные контексты из базы данных Database 1 должны быть доверенными, тогда как контексты из другой базы данных, Database 2, не должны быть доверенными, и все они принадлежат одному владельцу, который является доверенным средством проверки подлинности целевой области. Свойству TRUSTWORTHY можно присвоить значение ON для **Database 1** и OFF для **Database 2**; таким образом, модули в **Database 2** не будут иметь доступа к ресурсам вне этой базы данных.

Ниже приведен пример управления доступом к ресурсам вне области базы данных-источника с помощью свойства TRUSTWORTHY. Пользователь **MarketingDBO** обладает разрешениями AUTHENTICATE в базе данных **Sales** и является владельцем баз данных **Marketing** и **Conference**. Хранимая процедура **GetSalesProjections** в базе данных **Marketing** может успешно обращаться к базе данных **Sales**, поскольку она соответствует двум требованиям безопасности: средство проверки подлинности **MarketingDBO** является доверенным в целевой области, а база данных-источник **Marketing** заслуживает доверия. Попытки обратиться к базе данных **Sales** из базы данных **Conference** запрещаются, поскольку при этом выполняется только одно требование:

– средство проверки подлинности **MarketingDBO** является доверенным в целевой области.



При каждой попытке обратиться к ресурсам вне области базы данных с помощью олицетворенного контекста экземпляр SQL Server проверяет, заслуживает ли доверия база данных, из которой пришел запрос, а также является ли доверенным средство проверки подлинности.

Сертификаты и асимметричные ключи в качестве средств проверки подлинности

Чтобы расширить доступ олицетворения контекста внутри базы данных до ресурсов вне области базы данных, следует использовать владельца базы данных в качестве средства проверки подлинности. Для этого владелец базы данных должен быть доверенным для внешнего ресурса, кроме того, база данных должна сама по себе заслуживать доверия. Впрочем, этот подход неявно подразумевает, что если владелец доверенной базы данных обладает разрешениями AUTHENTICATE или AUTHENTICATE SERVER в целевой области и вызывающая база данных заслуживает доверия, то любой олицетворенный контекст, установленный в этой базе данных, действителен во всей целевой области, которая доверяет владельцу базы данных.

Может потребоваться уровень доверия с более высокой гранулярностью. Предположим, что необходимо доверять только небольшому числу модулей в базе данных-источнике, при этом доступ к целевому ресурсу осуществляется с помощью предложения EXECUTE AS, а база данных-источник является доверенной не целиком. Например, допустим, что пользователю **SalesDBO** нужно установить доступ хранимой процедуры **GetSalesProjections** к таблице **SalesStats** только от имени пользователя **MarketingExec**, но при этом у любого другого пользователя базы данных **Marketing** с правами на олицетворение **MarketingExec** не должно быть доступа к базе данных **Sales.** Для этого недостаточно сделать пользователя **MarketingDBO** доверенным и пометить базу данных **Marketing** как заслуживающую доверия. Для выполнения этого требования доверительная модель обеспечивает дополнительный уровень гранулярности, позволяя использовать в качестве средств проверки подлинности сертификаты или асимметричные ключи. Это дает техническое преимущество, обеспечиваемое электронной подписью. Дополнительные сведения об электронной подписи см. в разделе ADD SIGNATURE (Transact-SQL).

Применение подписей

Подписи модуля удостоверяют, что код внутри модуля может изменять только пользователь с доступом к закрытому ключу, с помощью которого подписан этот модуль. Учитывая гарантии, обеспечиваемые процессом подписи, можно доверять сертификату

или асимметричному ключу, указанному в подписи. Точнее, можно доверять владельцу сертификата или асимметричного ключа, а не только владельцу базы данных.

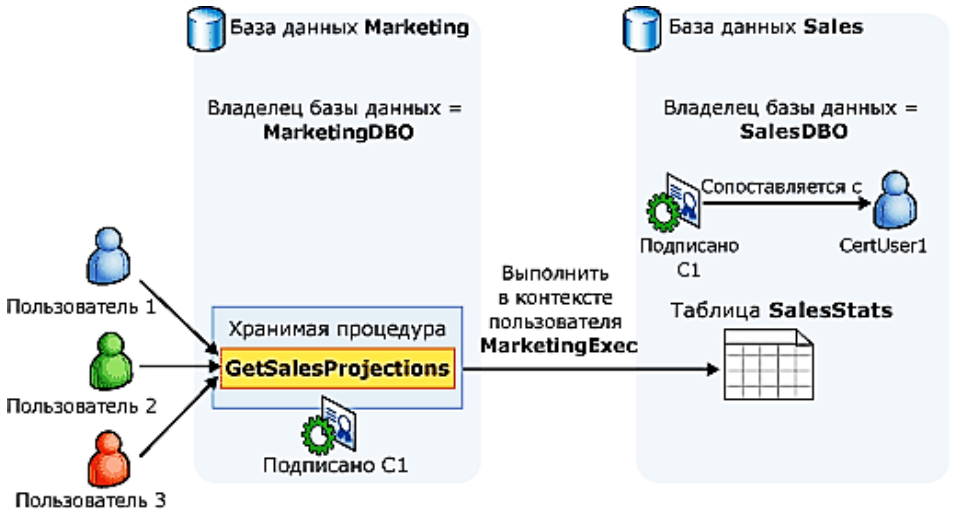
Подписанный модуль становится доверенным после выдачи разрешения AUTHENTICATE или AUTHENTICATE SERVER пользователю в целевой области, сопоставленной с сертификатом или асимметричным ключом.

Таким образом, контекст выполнения, установленный внутри модуля, подписанного с помощью доверенного сертификата, действителен в целевой области, в которой является доверенным этот сертификат.

Например, допустим, что процедура **GetSalesProjections** подписана с помощью сертификата с именем **C1**. Сертификат **C1** должен существовать в базе данных **Sales**, с сертификатом **C1** должен быть сопоставлен пользователь, например **CertUser1**. Пользователь **CertUser1** обладает разрешениями AUTHENTICATE в базе данных **Sales.**

При вызове процедуры проверяется подлинность процедуры (не была ли она подделана в момент подписания процедуры). Если подпись заверена, то для контекста, который устанавливается в модуле предложением EXECUTE AS, в качестве средства проверки подлинности используется **C1**. Если подпись не проходит проверку, то средство проверки подлинности не добавляется в лексему и попытка доступа к внешним ресурсам завершается неуспешно.

В следующем примере показано управление доступом к ресурсам вне области исходной базы данных с помощью подписанного модуля. Процедура **GetSalesProjections** в базе данных **Marketing** подписана с помощью сертификата **C1**. Сертификат **C1** существует в базе данных **Sales**, с ним сопоставлен пользователь **CertUser**. Пользователь **CertUser1** обладает разрешениями AUTHENTICATE в базе данных **Sales.**



Доверие этому средству проверки подлинности проверяется аналогично проверке, выполняемой, когда владелец базы данных является средством проверки подлинности. Таким образом, это проверка разрешений AUTHENTICATE SERVER или AUTHENTICATE. Однако поскольку доверие устанавливается на гранулярном уровне и модуль нельзя изменить без изменения подписи, то нет необходимости проверять свойство TRUSTWORTHY в базе данных.

Это уменьшает опасность от подключенных баз данных с вредоносным кодом. Атакующий будет вынужден подписать модуль закрытым ключом, который соответствует уже доверенному сертификату. Однако атакующий не имеет доступа к этому ключу. Кроме того, если существующий доверенный модуль будет изменен, либо если будет создан новый модуль, то он не будет иметь действительной доверенной подписи.

Правила расширения области олицетворения базы данных

В итоге область олицетворения контекста, установленного внутри базы данных, можно расширить на другие области тогда и только тогда, когда выполняются следующие условия:

- Средство проверки подлинности (владелец базы данных, сертификат или асимметричный ключ), с помощью которого подписан модуль, должен быть доверенным в целевой области. Для этого нужно предоставить разрешения AUTHENTICATE или AUTHENTICATE SERVER участнику, сопоставленному с владельцем базы данных, сертификатом или ассиметричным ключом.
- Если средством проверки подлинности является владелец базы данных, то база данных-источник должна быть помечена как заслуживающая доверия. Для этого свойству базы данных TRUSTWORTHY следует присвоить значение ON.

Выбор механизма доверия в зависимости от выполняемой задачи

Как механизм на основе владельца базы данных, так и механизм на основе подписей имеют свои достоинства и недостатки. Какой из механизмов лучше — зависит от поставленных задач и рабочей среды.

Подход к установлению доверия, основанный на владельце базы данных

Достоинства и недостатки этого подхода:

- Не требует вникать в суть таких криптографических понятий, как сертификаты или подписи.
- Не дает такой же гранулярности, как подход, основанный на подписях.
- При подключении базы данных к экземпляру SQL Server свойство TRUSTWORTHY в базе данных принимает значение OFF. Модули, которым доверяет владелец базы данных, не будут действительны до тех пор, пока администратор явно не задаст для свойства TRUSTWORTHY значение ON. Из этого неявно следует, что, для того чтобы подключенная база данных могла функционировать должным образом и обращаться к другим базам данных, предполагается некоторое вмешательство системного администратора.

Вариант выполнения работы

- Название темы варианта
- Цель работы
- Теоретическую часть
- Схему
- Текст программы(скрипт)

Контрольные вопросы:

1. Проблемы проверки подлинности безопасности системы СУБД
2. Архитектуры системы безопасности в MS SQL Server
3. Режимы защиты безопасности MS SQL Server