

ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ МОЛДОВЫ
ФАКУЛЬТЕТ МАТЕМАТИКИ И ИНФОРМАТИКИ
ДЕПАРТАМЕНТ ИНФОРМАТИКИ

Татьяна ПАША

БЕЗОПАСНОСТЬ БАЗ ДАННЫХ

курс лекций

Утверждено
Советом факультета
Математики и Информатики

Кишинэу, 2022
CEP USM

CZU 004.65.056.5(075.8)

П 220

Рекомендовано: *Департаментом Информатики МолдГУ*
Утверждено: *Советом Факультета Математики и Информатики МолдГУ*

Рецензент: Ион АНДРИЕШ*, др., конф. унив.*
Аурелиу ЗГУРЕАНУ*, др., конф. унив.*

DESCRIEREA CIP A CAMEREI NAŢIONALE A CĂRŢII

Паша, Татьяна.

Безопасность баз данных: Курс лекций / Татьяна Паша; Государственный Университет Молдовы, Факультет Математики и Информатики, Департамент Информатики. – Кишинэу: CEP USM, 2022. – 92 p.: fig., tab.

Bibliogr.: p. 87. – 50 ex.

ISBN 978-9975-159-64-7.

004.65.056.5(075.8)

П 220

ISBN 978-9975-159-64-7

© Татьяна ПАША, 2022
@ USM, 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5

I. ОСНОВНЫЕ ПОНЯТИЯ БЕЗОПАСНОСТИ БАЗ ДАННЫХ

1.1. Системы управления базами данных. Введение в безопасность баз данных 7
1.2. Конфиденциальность и целостность данных 12
1.3. Угрозы, уязвимости и риски безопасности 14
1.4. Аудит 16

II. УПРАВЛЕНИЕ ДОСТУПОМ К БАЗЕ ДАННЫХ

2.1. Политика управления 18
2.2. Создание, изменение и удаление идентификаторов и пользователей базы данных 21
2.3. Предоставление и отзыв привилегий 28
2.4. Типы ролей. Создание, назначение и отзыв ролей 29
2.5. Реализация политики паролей 34

III. ВНУТРЕННЯЯ БЕЗОПАСНОСТЬ ДАННЫХ

3.1. Ограничения целостности 37
3.2. Представления 41
3.3. Шифрование данных 45

IV. МЕТОДЫ АУДИТА БАЗ ДАННЫХ

4.1. Аудит входа в систему на уровне SQL Server 49
4.2. Аудит манипулирования данными DML (Data Manipulation Language) 50
4.3. Аудит действий DDL (Data Definition Language) 54
4.4. Аудит изменения привилегий, пользователей (user)/данных входа в систему (login) и других атрибутов безопасности 57
4.5. Аудит ошибок в базе данных и SQL Server 59
4.6. Аудит команд SELECT для конфиденциальных данных 61

4.7. Аудит использования базы данных в нерабочее время 62
4.8. Аудит изменений исходного кода хранимых процедур и триггеров 63

V. УЯЗВИМОСТЬ БАЗЫ ДАННЫХ

5.1. Уязвимость базы данных в интернете 64
5.2. SQL Injection 66

VI. МЕТОДЫ БЕЗОПАСНОСТИ ДАННЫХ. РИСКИ И ПРОЦЕДУРЫ ВОССТАНОВЛЕНИЯ ДАННЫХ

6.1. Резервные копии базы данных 72
6.2. Повреждение базы данных 78
6.3. Методы восстановления базы данных 79

VII. ПРОБЛЕМА ВМЕШАТЕЛЬСТВА И КОНКУРЕНЦИИ В БАЗЫ ДАННЫХ

7.1. Транзакции и свойства АСИД 81
7.2. Аномалии помех 83
7.3. Оптимистичные и пессимистичные блокировки 85

Библиография 87

ПРИЛОЖЕНИЯ 88

ВВЕДЕНИЕ

Фундаментальный курс «Безопасность баз данных» включен в учебный план для студентов третьего курса специальности «*Прикладная информатика*», специализация «Кибербезопасность». Он ориентирован на обучение студентов, заинтересованных в проектировании, создании, защите и управлении базами данных с использованием систем управления базами данных.

Учебно-педагогическая деятельность по дисциплине «Безопасность баз данных» организована в виде теоретических и лабораторных занятий. На лабораторных занятиях студенты выполняют практические задания, составленные преподавателем на основе тех тем, которые были затронуты на теоретических занятиях. Задачи формулируются в письменной форме, учащиеся выполняют их на компьютере, а затем объясняют, как они были решены. Эти задачи играют роль углубления знаний и стимулирования учебной деятельности.

На основе знаний, полученных на теоретических и лабораторных занятиях, каждый студент должен выполнить индивидуальную работу – создать базу данных, в которой были бы реализованы методы и практики безопасности баз данных. Студент должен спроектировать базу данных на основе плана безопасности, составленного индивидуально на основе знаний, полученных в аудитории, и представить ее в качестве финального проекта, включающего управление пользователями, внутреннюю безопасность данных, аудит базы данных, разработку плана резервного копирования и восстановления базы данных и предотвращение SQL Injection. Студент должен представить работу публично – коллегам и преподавателю. Практическая работа предполагает использование MS SQL Server и операционной системы.

Курс структурирован и состоит из 7 следующих единиц обучения:

1. Основные понятия безопасности баз данных. В этом разделе студент знакомится с СУБД и основными элементами, связанными с безопасностью баз данных, такими как: конфиденциальность, целостность и доступность данных, угрозы, уязвимости и риски безопасности.

2. Управление доступом к базе данных. Модуль предполагает, что студент в результате будет знать, как можно реализовывать политики администрирования и создание паролей. Также узнает, как могут быть созданы идентификаторы, пользователи и роли. Базы данных, а также как пользователям могут быть назначены определенные привилегии и роли.

3. Внутренняя безопасность данных. После изучения этого модуля студент сможет обеспечивать защиту данных, применяя ограничения целостности, создавая представления, а также шифруя личные данные.

4. Методы аудита базы данных. Модуль позволяет студенту узнать,

как можно реализовывать решения аудита для проверки правильности применения методов безопасности.

5. Уязвимости базы данных. В рамках модуля студенты узнают об уязвимостях баз данных, доступных в Интернете и о методах их предотвращения.

6. Методы защиты данных. Риски и процедуры восстановления данных. В этом разделе студент знакомится с различными типами планов резервного копирования для создания резервных копий и восстановления баз данных. Эти знания позволят студенту решать проблемы с повреждением данных.

7. Проблема вмешательства и конкуренции в базы данных. В результате изучения данного раздела студент сможет правильно управлять транзакциями, ознакомится с возможными аномалиями одновременного выполнения транзакций и будет в состоянии использовать методы контроля конкуренции.

Курс позволяет студенту узнать, как правильно и безопасно хранить, управлять и передавать данные. Важным аспектом является знание и использование методов прикладной математики и системного программного обеспечения для решения проблем, которые могут возникнуть в процессе защиты баз данных, а также применение методов и программного обеспечения для шифрования и защиты данных. Студент сможет анализировать, обнаруживать и устранять возможные уязвимости в процессе передачи данных в сети, а также аргументировать эффективность внедрения решений безопасности в экономических единицах, в которых он работает, путем создания безопасных приложений и компьютерных систем в различных типах компьютерных сетей.

Компетенции, формируемые в ходе курса:

- знание архитектуры компьютера и операционных систем;
- применение компьютерных сетей, системного программного обеспечения, персональных компьютеров в сфере профессиональной деятельности;
- использование методов прикладной математики и инструментального программного обеспечения для решения задач автоматизации управления предприятием;
- обеспечение информационных коммуникаций внутри предприятия через компьютерные сети;
- эффективное управление распределенными операционными системами, компьютерными сетями, базами данных в пределах экономических единиц;
- разработка WEB приложений.

Эти навыки можно укрепить, консультируясь с источниками, рекомендуемыми библиографическими материалами в конце каждого раздела.

I. ОСНОВНЫЕ ПОНЯТИЯ БЕЗОПАСНОСТИ БАЗ ДАННЫХ

1.1. Системы управления базами данных. Введение в безопасность баз данных

История баз данных начинается в 1960 году, когда была разработана Система Управления Файлами (СУФ). К первому поколению относятся IMS, СУБД разработки IBM, SOCRATE разработки CII и др.

Определение 1.1. *База данных* (БД) — это структурированный набор элементов данных и логических связей между ними, описывающих реальную или концептуальную вселенную (предприятие, организацию, страховую компанию, событие и т. д.), доступ к которой могут осуществлять одновременно несколько пользователей.

Определение 1.2. *Система управления базами данных* (СУБД) представляет собой интегрированный набор пакетов приложений (программ), оборудования и процедур, которые создают, хранят, модифицируют, обновляют и защищают БД предприятия.

Второе поколение СУБД, появившееся после 1970 г., основано на описанной Е. Коддом реляционной модели, направленной на упрощение понимания, оптимизацию структур данных и оптимизацию доступа к данным для пользователей. Эта модель данных имеет эффективный способ структурирования, основанный на теории нормализации, и определяет наилучший план доступа к данным, которые мы хотим получить. Это поколение, теоретическая база которого основана на реляционной теории, развивалось быстрее всего. Первые СУБД этого поколения появились на рынке после 1980 года, такие как DB2, Access, MS SQL, Oracle. Появляется и язык SQL, который позволяет запрашивать данные и манипулировать ими.

Определение 1.3. *Реляционная база данных* представляет собой совокупность данных, организованных в виде таблиц, в которых столбцы называются полями, а строки — записями. Между полями таблицы существуют взаимозависимости, называемые отношениями.

Третье поколение, которое ожидалось после 1990 года, — это объектно-ориентированные БД. Системы управления объектными базами данных (СУБДОО) были разработаны для управления сложными данными. Из-за неоднородности данных их обработка была затруднена и такой тип СУБД не получил широкого распространения.

Со временем СУБД развивалась на основе не реляционной модели данных, данные хранятся не в табличном виде, а в виде документов, например в формате JSON.

Реляционные СУБД снабжены возможностью создания связей между двумя и более таблицами. Эти связи позволяют осуществлять одновременный доступ к данным, находящимся в определенной связи из разных таблиц. Таким образом, программист освобождается от задачи поиска во всех таблицах тех данных, которые соответствуют запросу, причем эта операция выполняется СУБД автоматически. Программисту нужно в начале определить отношения между таблицами, и обнаружение определенной информации в БД приводит к автоматическому поиску соответствующей информации в других связанных таблицах. СУБД состоит из модулей, каждый из которых выполняет определенные задачи в системе.

СУБД — это набор программ, управляющих БД и обеспечивающих выполнение следующих операций:

- определение структуры БД;
- загрузка данных в БД;
- доступ к данным;
- обслуживание БД;
- реорганизация БД;
- обеспечение безопасности данных;
- сохранение целостности данных.

СУБД представляет собой сложную систему программ, приложений и языка программирования, которая обеспечивает интерфейс между БД и ее пользователями.

В курсе будет подробно описан ряд целей, которые ставит перед собой СУБД:

Реализация безопасности данных от несанкционированного доступа к данным. Администратор БД может предусмотреть, чтобы доступ к БД осуществлялся только по соответствующим каналам, и определить уровни авторизации для доступа к определенным данным.

Обеспечение целостности и безопасности данных от преднамеренного или непреднамеренного удаления с помощью процедур проверки, параллельных протоколов управления и процедур восстановления БД после инцидентов.

Если при проектировании БД безопасность рассматривается как важный аспект, а не как дополнение к БД, управление безопасностью будет минимальным, иначе затраты на обслуживание сильно возрастут. Фактически дизайн БД является наиболее важным шагом для успешной разработки и обслуживания приложения.

Проектирование решения по безопасности предполагает понимание организационных потребностей предприятия и для эффективного внедрения безопасности анализируется:

Характер данных. БД может содержать совершенно разные исключения безопасности в зависимости от специфики хранимых в нем данных, таких как: персональные данные, заработная плата, договорные обязательства, технологические тайны и т.д.

Существующие требования к аудиту. Анализ требований определяет, какие события должны быть проверены в приоритетном порядке. Определяется, какое требование аудита учитывается при проектировании системы и какое событие не может быть инициировано, если дизайн станет слишком сложным или административные расходы станут слишком высокими. Важно четко понимать, какие события необходимо проверять, прежде чем можно будет продолжить проектирование системы безопасности. После разработки и внедрения БД реализовывать стратегии аудита может быть гораздо сложнее, поскольку многие из них могут повлиять на общую концепцию БД, включающую регистрацию (как часто, в какое время, количество сбоев), изменение данных (особенно несанкционированное), просмотр отчетов.

Администратор сервера БД. Определение того, кто отвечает за управление сервером БД, имеет решающее значение для построения инфраструктуры безопасности сервера. Администратор – это тот, кто создает учетные записи, предоставляет привилегии, отзывает привилегии и назначает различные уровни безопасности. Рекомендуется свести к минимуму количество системных администраторов до одного. По мере уменьшения числа системных администраторов растет и индивидуальная ответственность за действия. Важна индивидуальная документация людей, которые управляют каждым сервером, и документация должна быть распределена среди всех, кто имеют доступ к БД, чтобы каждый знал, кто отвечает за администрирование. В результате повышается эффективность и снижается риск неправильных операции.

Механизм безопасности СУБД включает в себя способы ограничения доступа к системе, что предполагает реализацию путем создания учетных записей пользователей и паролей, так СУБД контролирует процесс входа в систему.

Безопасность БД предполагает защиту данных от несанкционированного использования, в частности от изменений, нежелательного уничтожения и несанкционированного чтения данных. Общепризнано, что на самом деле абсолютно безопасной защиты не бывает, есть только более эффективные или менее эффективные меры защиты и безопасности.

Незащищенные БД – мечта киберпреступников. БД, содержащие ценные бизнес-данные, могут стать легкой мишенью для атаки. Однако есть ряд рекомендаций, которым можно следовать, чтобы защитить БД бизнеса, и в то же время, свою репутацию.

Безопасность БД является важным компонентом общего плана обеспечения безопасности информационных систем любого предприятия. Разработчики БД несут ответственность за защиту конфиденциальности своих данных. Конфиденциальность – это право отдельных лиц иметь некоторый контроль над своей собственной информацией. Ряд законов разработаны для защиты конфиденциальности, и любой бизнес, который собирает и хранит информацию о физических лицах, по закону обязан принимать политики безопасности, соответствующие местным законам о конфиденциальности. Дизайн БД должен отражать приверженность компании к защите прав людей на неприкосновенность частной жизни. Информационная безопасность обычно следует модели КЦД – конфиденциальность (только авторизованные пользователи имеют доступ к информации для обеспечения конфиденциальности), целостность (только авторизованные пользователи могут изменить данные для обеспечения согласованности и надежности данных) и доступность (информация доступна для пользователей при необходимости). С ростом социальных сетей и онлайн-бизнеса благодаря Интернету поддержание конфиденциальности требует использования соответствующих методов шифрования, а также процедур авторизации, идентификации и аутентификации пользователей. Несоблюдение предполагает создание неверных данных, которые могут нанести вред отдельным лицам и компаниям. Атаки на безопасность компании могут привести к недоступности услуг компании, что приведет к нарушению торговых соглашений.

Безопасность обычно ассоциируется со следующими ситуациями:

- мошеннический доступ к данным;
- потеря конфиденциальности данных;
- потеря частного характера данных;
- потеря целостность данных;
- потеря доступности данных.

Формы преднамеренного и злонамеренного доступа к данным БД:

- несанкционированное чтение данных;
- несанкционированное изменение данных;
- уничтожение данных.

Безопасность БД – это широкая область, охватывающая множество тем, в том числе:

- этические и юридические вопросы, касающиеся права доступа к определенной информации;

- политические вопросы на уровне правительственной, институциональной или корпоративной политики, связанные с типом информации которая не должна быть доступна для общественности;

- проблемы с системой и уровни системы, которые обрабатывают различные функции безопасности.

Для защиты БД могут быть приняты многоуровневые меры безопасности:

- *на физическом уровне* – помещения, в которых расположены серверы БД, должны быть защищены от доступа посторонних лиц;

- *на человеческом уровне* – желательно очень осторожно выдавать разрешения на доступ и вести строгий учет уполномоченных лиц;

- *уровень операционной системы (ОС)* – недостатки в защите ОС должны быть устранены или компенсированы другими мерами;

- *на уровне СУБД* – система предоставляет определенные средства, поддерживающие защиту данных.

Поддержание безопасности на всех этих уровнях должно усилить безопасность БД. Слабости низких уровней безопасности (физических или человеческих) могут обойти строгие меры безопасности на более высоких уровнях (база данных). Безопасность операционной системы применяется на различных уровнях, от паролей для доступа к системе до изоляции запущенных в ней конкурирующих процессов. Файловая система также предлагает определенный уровень защиты.

Механизмы безопасности могут быть нацелены на политику управления доступом, основанная на идентификации пользователя, или политику ограничивающии доступ к информации, классифицируемой уполномоченным персоналом как конфиденциальная.

Можно говорить о двух типах механизмов безопасности БД:

- *дискреционные механизмы безопасности* для предоставления привилегий пользователям, включая доступ к файлам, записям или определенным полям данных определенным образом;

- *обязательные механизмы безопасности* для группировки данных на нескольких уровнях и пользователей по нескольким классам (или уровням) с последующей реализацией надлежащей политики безопасности предприятия.

Поскольку доступ к данным может быть осуществлен средствами, отличными от СУБД (например, путем прямого считывания с магнитного носителя), защита может быть обеспечена за счет хранения данных в закодированном виде на магнитном носителе. Расшифровка данных возможна только после идентификации пользователя. Таким образом, метод обеспечения безопасности также представляет собой шифрование данных, используемый для защиты конфиденциальных данных, передаваемых по сети связи. Шифрование обеспечивает дополнительную защиту конфиденциальных данных БД. Данные намеренно шифруются алгоритмом. Неавто-

ризованный пользователь, имеющий доступ к шифрованным данным, попытается расшифровать их, но только авторизованный пользователь будет иметь алгоритм шифрования или расшифровки (ключ) для этой цели. Шифрование используется для хранения важной информации в незащищенных местах и передачи ее по незащищенным каналам связи.

Существуют два типа шифрования:

- *симметричный*, для шифрования и расшифрования используется один и тот же ключ. Алгоритм и ключ заранее выбраны и известны обеим сторонам. Таким образом, шанс получить ключ выше, потому что его распространение может быть перехвачено нежелательными людьми. Хранение ключа в секрете является важной задачей для установления и поддержания безопасного канала связи.

- *асимметричный*, есть пара ключей, один публичный для шифрования информации, а другой, называемый частный, для расшифровки информации (он есть только у того, кто читает/расшифровывает информацию).

Симметричное шифрование намного более уязвимо, чем асимметричное шифрование из-за использования одного ключа, с другой стороны, симметричное шифрование намного быстрее, чем асимметричное шифрование по той же причине.

1.2. Конфиденциальность и целостность данных

В Европейском сообществе существует серьезная озабоченность по поводу обновления законодательства в соответствии с новыми потребностями, вызванными интенсивным использованием компьютеров. Основное внимание уделяется принятию законов для защиты компаний и защиты частных данных (чтобы они не были доступны широкой публике или даже относительно небольшой группе, если это нанесет ущерб владельцу этой информации). Можно перечислить широкий спектр сфер, которые работают с информацией частного характера, которую необходимо сохранять: банковские, медицинские, административно-финансовые записи, сфера производства в большинстве фирменных компаний и т. д. Руководство компании, владеющей БД, должно принять меры безопасности, снижающие риск потери или уничтожения информации.

Конфиденциальность данных — одна из функций СУБД, которая обеспечивается блокировкой доступа отдельных категорий пользователей к данным, представляющим информацию в которой они не нуждаются в своей деятельности, что минимизирует риск случайного уничтожения данных неправильными операциями. Политика безопасности БД включает опреде-

ление учетных записей использования и назначение прав доступа пользователям. Каждому пользователю индивидуально могут быть предоставлены права доступа, изменения, удаления данных.

Дискреционный доступ – это способ ограничить доступ к информации, основанный на привилегиях. Можно определить два уровня привилегий:

- *на уровне учетной записи* администратор указывает специальные привилегии, которые есть у каждого пользователя БД над таблицей (CREATE TABLE, CREATE VIEW, ALTER, MODIFY, SELECT);

- *на уровне отношений,* привилегии доступа к каждой таблице БД назначаются владельцу учетной записи, который имеет все привилегии и отвечает за их предоставление другим учетным записям.

Политику управления доступом к БД можно разделить на 2 группы:

- *закрытый*, допускается только явная авторизация доступа;

- *открытый*, доступ разрешен по умолчанию.

Важным вопросом управления БД является обеспечение того, чтобы данные БД соответствовали заранее определенному набору правил. Эти правила должны соответствовать бизнес-операциям системы. Целостность данных включает в себя определение правил, которые ограничивают допустимые значения для столбца таблицы.

Целостность БД относится к корректности информации и включает обнаружение, исправление и предотвращение различных ошибок, которые могут повлиять на данные, введенные в БД. Когда говорим о целостности данных, понимаем, что данные соответствуют всем вышеперечисленным ограничениям, и в результате данные считаются достоверными.

Ограничения целостности определены для таблиц, поэтому все представления подчиняются ограничениям целостности. Если оператор манипулирования данными (DML) пытается выполнить действие, нарушающее ограничение целостности, генерируется ошибка и выполняется откат транзакции. Ограничения целостности данных — это средства, используемые для предотвращения ввода неверных данных в таблицы БД, определенные для обеспечения соблюдения правил торговли, которые связанны с информацией БД. Ограничения целостности можно разделить на 2 группы:

1. Структурные ограничения выражаются функциональными зависимостями, многозначностями, соединениями или объявлением полей с уникальными значениями (чаще всего эти поля являются ключевыми). Они требуют соблюдения следующих правил целостности:

– первичный ключ таблицы должен быть уникальным и минимальным;

– для каждой строки первичного ключа в таблице должно быть присвоено значение;

– в таблице А, которая ссылается на таблицу В, значения внешнего ключа должны быть среди значений первичного ключа в В или принимать значение *null*.

2. Поведенческие ограничения определяют поведение данных и учитывают их значения. Поведенческие ограничения, будучи очень общими, управляются либо во время описания данных, либо вне модели во время выполнения приложения:

- *ограничение домена* связано с доменом, соответствующим атрибуту в таблице, который должен находиться между определенными значениями;

- *ограничения по времени*, значения атрибутов сравниваются с некоторыми временными значениями (результатами вычислений, такими как средняя зарплата, минимальное/максимальное рабочее время и т.д.).

1.3. Угрозы, уязвимости и риски безопасности

БД является одним из важнейших компонентов любой информационной системы, в которой хранятся и обрабатываются различные данные, которые могут быть системами документооборота и учета, системами управления производственными процессами и т.д. Поскольку БД содержит всю ценную информацию о компании, ее клиентах, финансовой деятельности, она является одним из критических факторов в ее структуре, что определяет повышенные требования к конфиденциальности, целостности и доступу к данным.

Риски, которые могут возникнуть в этом контексте, могут исходить как из внешней, так и из внутренней среды. Также необходимо учитывать надежность систем, СУБД, технических средств, которые могут подвергаться воздействию таких факторов, как пожары, стихийные бедствия и т.п., таких как:

1. Потеря консистенции BD:

- избыточная структура таблиц БД (группы полей, которые повторяются в нескольких таблицах);

- наличие нулевых записей для полей, идентифицирующих принадлежность экономического явления (дата операции, руководство, заказчик, поставщик, раздел, товарная группа);

2. Потеря целостности БД:

- отсутствие первичных ключей для полей, требующих ограничения ссылочной целостности (серийный номер счета-фактуры, номер счета-фактуры, инвентарный номер, регистрационный номер);

- отсутствие ограничений ссылочной целостности (отношения между первичным ключом и внешним ключом);

3. Несанкционированный доступ к БД (доступ к БД может получить любой пользователь с правами на изменение, удаление, копирование и т.д.).

4. Умышленная (мошенническая) или случайная (ошибочная) потеря БД из-за отсутствия резервных копий БД (бэкапов).

5. Недоступность сервера СУБД:

- отсутствие защиты от отключения электроэнергии;
- блокировка протоколов связи;
- сбой операционной системы.

Случайные угрозы безопасности включают в себя:

- пользователь может непреднамеренно запросить объект или операцию, для которых он не должен быть авторизован, а запрос может быть принят из-за некорректной авторизации или ошибки в СУБД или операционной системе;
- пользователь может случайно получить сообщение, которое должно быть адресовано другому пользователю, что приведет к несанкционированному раскрытию содержимого БД;
- ошибка системы связи может подключить пользователя к сеансу, принадлежащему другому пользователю с другими правами доступа;
- операционная система может случайно перезаписать и уничтожить часть БД, могут быть загружены некорректные файлы, которые затем случайно будут отправлены неавторизованному пользователю.

Преднамеренные угрозы безопасности возникают, когда пользователь умышленно получает несанкционированный доступ и/или выполняет несанкционированные операции в БД. Неудовлетворенный сотрудник, знающий ИТ-систему компании, представляет собой огромную угрозу безопасности. Привилегированные пользователи, такие как администраторы БД, которые получают доступ к данным конечных пользователей, доступ к которым им не должен быть разрешен, угрожают безопасности. Существует несколько способов преднамеренного тестирования безопасности:

- соединение линий связи для перехвата сообщений в БД и из нее;
- электронное прослушивание для приема сигналов с рабочих станций, принтеров или других устройств внутри здания;
- чтение экранов дисплея и чтение или копирование неконтролируемых распечаток авторизованными пользователями;
- выдачя себя за авторизованного пользователя или пользователя с более широким доступом, используя его логин и пароль;
- написание нелегально закодированных системных программ для об-

хода СУБД и ее механизма авторизации и доступа к данным БД напрямую через операционную систему;

- написание прикладных программ с кодом, допускающим несанкционированные операции;

- получение информации о скрытых данных путем интеллектуального запроса к БД;

- изменение запросов к БД с помощью SQL injection для получения несанкционированного доступа к данным или для злонамеренного изменения или удаления данных;

- удаление физических устройств хранения данных с компьютера;

- создание физических копий хранимых файлов без прохождения через СУБД, что позволяет обойти ее механизмы защиты;

- подкуп, шантаж или иное вовлечение авторизованных пользователей с целью использования их в качестве агентов для получения информации или повреждения БД;

- использование системных привилегий, чтобы разрешить доступ пользователю, который не должен получить этот доступ.

Уязвимость — это слабость, которая позволяет злоумышленнику снизить безопасность данных БД. Уязвимость представляет собой пересечение трех элементов: дефекта, доступа злоумышленника к дефекту и способности злоумышленника использовать дефект. Чтобы воспользоваться уязвимостью, злоумышленник должен иметь по крайней мере один применимый инструмент или технику для подключения к уязвимости БД.

Чтобы избежать уязвимости, нужно оценить конфигурацию БД, которая включает проверку того, как установлены БД и операционная система (проверка файла группы привилегий, журналов БД и транзакций). Также необходимо проверить, не запущены ли версии БД, содержащие известные уязвимости.

1.4. Аудит

Термин происходит от латинского Auditum (слушание). Выслушивая, затем исследуя и, наконец, предлагая решения, аудит позволяет внести вклад в мотивированное и независимое рассуждение. В качестве проверки для определения свойств представительства аудит первоначально применяется к финансовым представительствам.

Определение 1.4. *Аудит* — это процесс, посредством которого компетентные независимые люди собирают и оценивают доказательства, чтобы сформировать мнение о степени соответствия между наблюдаемыми и определенными заранее установленными критериями безопасности.

Аудит должен удостовериться, что выполняются необходимые условия для поддержания баланса, контроля беспорядка, адаптации к изменениям и оценки безопасности и риска. Аудит осуществляется после применения всех настроек и элементов управления. Он включает в себя самооценку и мониторинг рекомендаций аудита для проверки отклонений от цели (безопасности).

Мониторинг активности БД в режиме реального времени необходим для ограничения воздействия путем обнаружения вторжений и злоупотреблений. Предупреждение о необычном доступе указывает на наличие атаки SQL injection, несанкционированного изменения данных, изменения привилегий и изменения конфигурации учетной записи путем запуска сценария SQL. Очень важно отслеживать привилегированных пользователей, которые должны соблюдать такие правила, как SOX[1] и правила конфиденциальности. Мониторинг позволяет обнаруживать наиболее частые атаки которые осуществляются пользователями с более высокими привилегиями.

Большинство предприятий сегодня используют ту или иную форму аудита или приложения СУБД, но эти приложения часто отключаются по следующим причинам:

- сложность;
- высокие эксплуатационные расходы;
- проблемы с производительностью;
- отсутствие распределения обязанностей;
- потребность в дополнительном пространстве для хранения информации.

Решения должны разрабатываться с минимальным влиянием на производительность и стоимость (деньги, время).

Библиография к главе 1

1. H. Afyouni, *Database Security and Auditing: Protecting Data Integrity and Accessibility,* Cengage Learning, 2006.
2. A. Basta, M. Zgola, *Database Security*, Cengage Learning, 2011.
3. T. Kyte, D. Kuhn, *Oracle Database Transactions and Locking Revealed*, Apress, 2014.

[1] Федеральный закон появился в США в 2002 году в результате финансовых скандалов внутри таких корпораций, как Enron, Tyco International, WordCom, etc и др. Название происходит от имен сенатора Пола Сарбейнса и Майкла Оксли.

II. УПРАВЛЕНИЕ ДОСТУПОМ К БАЗЕ ДАННЫХ

Чтобы ограничить доступ к ресурсам компании, таким как данные о сотрудниках и клиентах, необходимо определить методы управления доступом. Управление доступом является ключевым компонентом обеспечения конфиденциальности и целостности.

План управления доступом должен начинаться с мер физической безопасности самого здания с особыми мерами предосторожности для компьютерного оборудования. Они начинаются у входной двери, где все сотрудники должны быть визуально идентифицированы охранниками или с помощью бейджей, отпечатков пальцев или других механизмов. Для доступа к компьютерным средствам требуется дополнительная идентификация. Меры физической безопасности следует распространить на любое место, где хранятся данные, например резервные копии. Наряду с физической защитой пользователи могут иметь другие формы идентификации, такие как смарт-карты, которые проходят через считыватель электронных карт для доступа к зданиям и комнатам, в которых размещены БД, а также к другим общим рабочим местам. Биометрия может обеспечить более безопасную форму идентификации, особенно в строго конфиденциальных приложениях. Биометрия может включать в себя отпечатки пальцев, распознавание лиц, распознавание голоса и сканирование сетчатки глаза.

С другой стороны, на уровне БД пользователям назначаются разные роли (определяемые администратором БД), поэтому они имеют доступ к определенным данным и могут выполнять только предопределенный набор операций. СУБД также содержат некоторые предопределенные роли, в том числе *sysadmin* или *securityadmin*, которые будут изучены в этом модуле.

2.1. Политика управления

При подключении к SQL Server происходит ряд вещей, которые на первый взгляд не выделяются и безопасность проверяется на трех уровнях:

1. На уровне операционной системы.
2. На уровне SQL Server (в виде идентификатора подключения к SQL Server).
3. На уровне отдельной БД (в виде имени пользователя БД).

Тот факт, что имеется идентификатор входа в систему, ничего не говорит о БД, к которым можно получить доступ, для этого требуется имя пользователя БД.

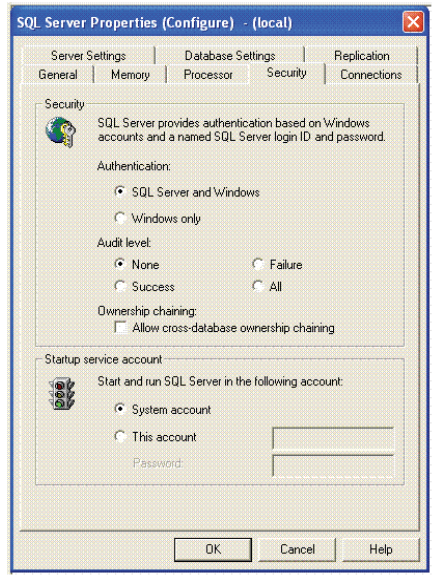
Аутентификация ОС Windows. При попытке подключения с клиентского компьютера к компьютеру Windows, на котором работает Microsoft SQL Server, Windows может потребовать проверки сетевого подключения.

Аутентификация SQL Server. Чтобы иметь возможность подключиться к SQL Server, нужно указать действительное имя пользователя и пароль SQL Server. Если учетные данные действительны, подключение к SQL Server выполнено успешно. В противном случае в доступе будет отказано, даже если проверка подлинности в сети Windows прошла успешно.

Имя пользователя БД SQL Server. Для того, чтобы иметь возможность работать с каждой БД в системе, необходимо явно получить доступ к БД. Доступ к БД можно получить несколькими способами, которые будут рассмотрены позже. Если нет пользователя, созданного для БД доступ к ней запрещен.

Права доступа. После подключения к SQL Server и доступа к БД предоставляется явное право на доступ к объектам БД (только для чтения или для ввода и вывода на экран).

Чтобы установить режим безопасности используемого сервером, *Windows Integrated Mode* или *Mixed Mode,* используется средство управления SQL Server под названием SQL Server Enterprise Manager (меню «Свойства» – «Безопасность»). Чтобы изменить атрибут безопасности, отметьте нужные опции и перезапустите службу SQL Server.



Можно включить функцию мониторинга (аудита) сервера, по умолчанию она отключена.

Доступны следующие варианты:

- *Failure* (неудачные попытки подключения к SQL Server);
- *Success* (удачные соединения);
- *All* (оба).

Результаты процесса мониторинга можно просмотреть в журнале ошибок SQL Server.

Информация об идентификаторах хранится в SQL Server (в БД *master*, в системной таблице *sysxlogins*).

```
select substring (name,1,25) AS name,
substring (password,1,30) AS password
FROM sysxlogins
```

	name	password
1	Admin	0x0100817C372775B10826FB72FC2561C5DD1117934C715D03A1FFAC4CE1CB
2	BUILTIN\Administrators	NULL
3	cezar	0x0100C1459C7C766546638BAFAAAD8CFF52A5ACODF3C415051B3876654663
4	sa	0x0100D37AFF5420AC89ACEACFB03B9D6A950B7B3367421DAA901BF2E94A56
5	teste1	0x0100D67CDA6ACA12FA3CABCD1907AAB492F81617E95E7F0E1B24B768C0AA
6	teste2	0x0100177D92327A0485173428281F007644DC33392FBF5CAD062189A4F448
7	teste3	0x0100487D5C2829BAC277CEB9328A991B5801501ABF4DDA15EBB8FD92C403

Авторизация требует определения, кто имеет доступ к системе, и конкретных данных, к которым им разрешен доступ. Большинство СУБД предназначены для нескольких пользователей и имеют собственные подсистемы безопасности, обеспечивающие авторизацию пользователей – метод, с помощью которого пользователям предоставляются права на использование объектов БД. Администратор БД определяет права пользователей через правила авторизации, определяющие пользователей, имеющих доступ к данным и операциям с этими данными. Механизм авторизации предназначен для защиты БД путем предотвращения чтения, обновления или уничтожения контента БД неавторизованным пользователям. Эти ограничения являются дополнением к механизмам безопасности, предоставляемым операционной системой.

Аутентификация – это процесс проверки личности пользователя, чтобы убедиться, что пользователь является тем, за кого себя выдает. Аутентификация первоначально реализуется на уровне операционной системы. Когда пользователь входит в систему, вводится идентификатор пользователя, который проверяется на достоверность. В некоторых компаниях используется многофакторная аутентификация, при которой пользователи должны предоставить две или более формы аутентификации. При многофакторном подходе пользователь может предоставить идентификатор пользователя и пароль, а также смарт-карту, бейдж или биометрическую форму. Процедура аутентификации может также состоять из ответов на ряд вопросов, которые могут занять больше времени и сама аутентификация сложнее чем воспроизведение одного пароля.

Еще одна мера безопасности – **блокировка** пользователя, не допускается к учетной записи после нескольких неудачных попыток входа. Политика блокировки не позволяет хакерам угадать пароль пользователя с помощью Брутфорса.

Последним компонентом управления доступом к информационной системе является ответственность. **Ответственность** относится к необходимости собирать и поддерживать файлы журналов, которые можно использовать для отслеживания при возникновении инцидентов безопасности. Например, операционная система хранит информацию для входа в систему о пользователях, а также файлы и БД, к которым они обращаются. В файлах журнала хранится информация о сетевом трафике, которая может использоваться для отслеживания удаленного доступа к системе. СУБД поддерживают файлы журналов как часть системы восстановления БД, записывают информацию о доступе пользователей, возможность обновления, введения и удаления данных.

2.2. Создание, изменение и удаление идентификаторов и пользователей базы данных

2.2.1. Управление идентификаторами

Первым шагом в настройке сервера БД для доступа является создание идентификаторов. Идентификаторы могут быть добавлены с помощью хранимой системной процедуры *sp_addlogin*:

sp_addlogin [@loginame =] 'identificator'
[,[@passwd =] 'parola'
[,[@defdb=] 'baza_date'
[,[@deflanguage =] 'limba']]]

В *sp_addlogin*:

- *identificator* – это имя, с которым пользователь подключается. Это имя должно быть действительным идентификатором SQL Server (начинается с буквы или одного из символов #, @ или _, содержит любые из этих символов, буквы и цифры – до 128 символов Unicode);

- *parola* – это пароль, соответствующий этому идентификатору, и равен *null* если он не выбран во время создания идентификатора;

- *baza_date* – это БД, к которой пользователь получает доступ по умолчанию при входе в систему. Если этот параметр не указан, БД по умолчанию будет master;

- *limba* – это язык, который будет использоваться для этого пользователя по умолчанию при подключении к системе.

Чтобы добавить идентификатор на сервер, откройте SQL Server Management Studio (средство управления и работы с SQL Server) и после подключения выполните следующую команду Transact-SQL (T-SQL):

Пример 2.2.1. *EXEC sp_addlogin 'numele', 'parola'*

Создается идентификатор *numele* с паролем *parola*, который на данном этапе не дает доступа к БД и требует назначения привилегий и ролей.

Тот же результат получаем с помощью оператора SQL:

CREATE LOGIN *identificator* WITH PASSWORD = *'parola'*

Чтобы изменить пароль, можно использовать хранимую системную процедуру *sp_password:*

sp_password [[@old =] 'veche',
[[@new =] 'noua',
[,[@loginame =] 'identificator'

В предыдущем синтаксисе:

- *veche* – старый пароль;
- *noua* – новый пароль;

Если войти в систему с ролью sysadmin или securityadmin, можно изменить любой пароль, не зная старого пароля просто объявив его NULL.

Пример 2.2.2. *EXEC sp_password 'parolaveche', 'parolanoua', 'numele'*

Члены роли *securityadmin* могут изменять любой пароль, кроме паролей членов роли *sysadmin*.

Тот же результат получаем с помощью оператора SQL:

ALTER LOGIN *numele*
WITH PASSWORD = *'parolanoua'*
[OLD_PASSWORD = *'parolaveche'*]

Используется только для текущего пользователя, вошедшего в систему. Если хотим изменить пароль для другого пользователя БД, последовательность [OLD_PASSWORD = *'parolaveche'*] выполнять не нужно.

Регулярная смена пароля — отличная идея. Microsoft SQL Server не предоставляет каких-либо методов для наложения ограничений на использование пароля и других мер безопасности. Это одна из причин, почему желательна реализация интегрированной безопасности. В операционной системе Windows можно указать минимальную длину пароля, частоту смены и правила минимальной сложности пароля.

Две другие хранимые системные процедуры, используемые для управления идентификаторами пользователей это: ***sp_helplogins*** и ***sp_droplogins***.

Хранимая процедура *sp_helplogins* предоставляет отчет с созданными на сервере идентификаторами. Хранимая системная процедура *sp_droplogins* удаляет идентификатор из таблицы *sysxlogins*. После удаления идентификатора ***sp_droplogin*** *identificator* этот пользователь не может подключиться к SQL Server.

Тот же результат получаем с помощью оператора SQL:

DROP LOGIN numele

2.2.2. Windows Authentication Mode

Соединение, которое установлено с помощью проверки подлинности Windows, называется доверенным соединением. В режиме проверки подлинности Windows после установления сетевого подключения к SQL Server должны отображаться атрибуты безопасности в Windows. Эти атрибуты создаются во время процесса подключения к сети Windows. Они автоматически проверяются без отображения сообщения. Доступ к SQL Server может

быть предоставлен напрямую через учетные записи безопасности Windows или косвенно через группы Windows.

Самым большим преимуществом Windows Authentication Mode является то, что пользователям не нужно беспокоиться о отдельном подключении к SQL Server. Это соответствует концепции, согласно которой пользователям необходимо пройти однократную сетевую процедуру и запомнить один пароль. Кроме того, можно воспользоваться тем фактом, что большинство сетевых пользователей имеют учетные записи Windows, что упрощает управление учетными записями SQL Server.

Первый шаг в настройке безопасности Windows Authentication Mode не имеет ничего общего с SQL Server — он включает в себя сначала создание групп пользователей Windows, создание учетных записей пользователей (если они еще не существуют) и добавление их во вновь созданные группы, а затем предоставление прав на подключение к SQL Server.

После определения пользователей и групп этим группам необходимо предоставить доступ к SQL Server. Для этого используются хранимые системные процедуры *sp_grantlogin*, *sp_revokelogin* и *sp_denylogin*. Первоначально права на подключение предоставляются группам Windows, а затем, по мере необходимости, отдельным пользователям. При использовании этого метода количество требуемых команд и усилия по администрированию сводятся к минимуму в условиях индивидуального контроля над привилегиями соединения.

Чтобы предоставить право на подключение, используйте хранимую системную процедуру *sp_grantlogin*:

sp_grantlogin [@loginame =] 'identificator'

В этом синтаксисе:

- *identificator* — это имя группы или пользователя Windows, которому предоставлено право подключения к SQL Server.

Таким образом, любой пользователь Windows, являющийся членом группы, которой предоставлено право доступа к SQL Server, может успешно подключиться благодаря принадлежности к этой группе. Если используете этот метод подключения, заметите, что, хотя группа является сущностью, которой были предоставлены права на подключение к SQL Server, пользователь появится в строке состояния. Этот тип подключения обеспечит гораздо лучший журнал аудита и последующий контроль над привилегиями.

Право пользователя или группы Windows на подключение к SQL Server может быть снято с помощью хранимой системной процедуры *sp_revokelogin*:

sp_revokelogin [@loginame =] 'identificator'

В этой процедуре:

- *identificator* – это имя группы или пользователя Windows, у которого отзывается право на подключение к SQL Server;

- *sp_revokelogin* – удаляет/отменяет предоставленное ранее право.

Для того чтобы отказать в праве подключения к SQL Server определенному пользователю или группе, используйте хранимую системную процедуру *sp_denylogin*:

sp_denylogin [@loginame =] 'identificator'

В этой процедуре:

- *identificator* – это имя группы или пользователя Windows, право на подключение к SQL Server которого запрещено.

Запрет прав всегда имеет приоритет над *предоставленными* правами.

2.2.3. Управление пользователями базы данных

SQL Server содержит два типа идентификаторов:

- *идентификаторы Windows*, для групп или отдельных пользователей;

- *идентификаторы SQL Server*, хранящиеся в системной таблице *sysxlogins* в БД master.

После настройки безопасности идентификаторов и определения идентификаторов можно переходить к настройке доступа к БД. Наличие идентификатора для SQL Server не дает доступа ко всем БД на сервере. Для этого требуется создать имя пользователя БД на основе идентификатора.

2.2.3.1. Создание пользователя

Каждая БД имеет отдельный путь, расположенный в системной таблице *sysusers* этой БД. По сути, идентификаторы связаны с именем пользователя в каждой БД, к которой пользователь должен получить доступ.

Можно создать пользователя БД с помощью хранимой системной процедуры *sp_grantdbaccess*:

sp_grantdbaccess [@loginame =] 'identificator'
[, [@name_in_db =] 'nume_din_bd]'

В данном синтаксисе:

- *identificator* идентификатор, который был ранее создан (пользователь SQL Server или пользователь/группа Windows);

- *nume_din_bd* это имя, которое можно использовать при работе с БД (username). Если не указан тогда используется идентификатор.

Пример 2.2.3. *EXEC sp_grantdbaccess 'numele', 'nume_bd'*

Тот же результат получаем с помощью оператора SQL:

CREATE USER *nume_bd* FOR LOGIN *numele*

Рекомендуется, чтобы имя пользователя по возможности совпадало с идентификатором, для безопасности БД чтобы упростить отслеживание/запоминание. Если имя остается прежним, путаница исчезает.

Если группа была добавлена, каждый участник может получить доступ к БД, но только у группы есть имя пользователя БД, то есть нет имени для каждого пользователя в группе.

2.2.3.2. Создание псевдонима (алиас)

Псевдоним имени пользователя БД можно добавить с помощью хранимой системной процедуры *sp_addalias*. С этим псевдонимом идентификатор может заменить другого пользователя БД вместо того, чтобы быть связанным с его собственным именем пользователя или именем группы.

sp_addalias *identificator, nume_utilizator*

В синтаксисе команды sp_addalias:

- *identificator* это идентификатор пользователя, которого хотите связать;
- *nume_utilizator* это пользователь БД, от имени которого входите в систему.

Техника псевдонимов была создана для упрощения процесса обеспечения безопасности БД. Можно связать столько угодно идентификаторов с одним и тем же именем пользователя в БД. Если идентификатор уже связан с именем пользователя в БД, он не может быть связан с другим именем в той же БД. Роли обеспечивают гораздо более точную и лучшую модель безопасности.

2.2.3.3. Отмена права доступа к базе данных

Чтобы отменить/отозвать право доступа к БД, запустите хранимую системную процедуру *sp_revokedbaccess*:

sp_revokedbaccess [@name in bd =] 'nume_din_bd']

В данной хранимой процедуре:

- *nume_din_bd* — имя пользователя БД, которого нужно отменить. Эти хранимые процедуры могут выполнять только члены ролей db_accessadmin и db_owner (или фиксированной роли сервера sysadmin).

Пример 2.2.4. *EXEC sp_revokedbaccess nume_bd*

Тот же результат получаем с помощью оператора SQL, для выполнения которого должна быть активна БД *nume_bd*:

DROP USER [IF EXISTS] *nume_bd*

Чтобы узнать, какие пользователи обращаются к той или иной БД и какой у каждого из них идентификатор, можно запустить хранимую системную процедуру *sp_helpuser*:

sp_helpuser [[@name in bd =] 'nume utilizator']

- *nume_utilizator* является необязательным и представляет либо имя пользователя, либо имя роли.

Если имя пользователя не указано, отчет будет создан со всеми пользователями, а в противном случае будут получены все роли только для указанного пользователя или роли.

Пример 2.2.5. *EXEC sp_helpuser 'nume_bd'*

- отображаются пользователи из БД *'nume_bd'*.

Пример 2.2.6. *EXEC sp_helpuser*

- отображение пользователей в текущей БД:

	UserName	RoleName	LoginName	DefDBName	DefSchemaName	UserID	SID
1	dbo	db_owner	NULL	NULL	dbo	1	0x0105000000000005150000000026A908E27DEE9530023D7E...
2	guest	public	NULL	NULL	guest	2	0x00
3	INFORMATION_SCHEMA	public	NULL	NULL	NULL	3	NULL
4	nume_bd	public	numele	master	nume_bd	5	0x477C4F8F32E3074E82B060717F80734E
5	sys	public	NULL	NULL	NULL	4	NULL

Пример 2.2.7. *EXEC sp_helpuser 'db_securityadmin';*

- отображает пользователей, которые являются частью роли *db_securityadmin.*

При создании БД у нее уже есть пользователь *dbo* (от имени владельца БД). Пользователь dbo по умолчанию связан с его идентификатором входа в систему. При установке SQL Server его идентификатор считается владельцем всех БД. Если кто-то другой с другим идентификатором создаст БД, ее владельцем будет идентификатор.

Владелец БД может делать абсолютно все внутри этой БД. У него такая же свобода, как и у членов роли системного администратора. Но только члены фиксированной серверной роли sysadmin имеют определенные привилегии *sysadmin*.

Если хотите удалить псевдоним, используйте системную хранимую процедуру *sp_dropalias*:

sp_dropalias *identificator*

- *identificator* — идентификатор пользователя.

2.2.3.4. Изменение владельца базы данных

Можно изменить владельца БД, если хотите взять на себя ответственность за конкретного администратора БД. В этом случае, идентификатор не должен существовать в БД как имя пользователя.

Чтобы произвести смену владельца, запустите системную хранимую процедуру *sp_changedbowner*:

sp_changedbowner [@loginame =]'identificator'

[,[@map =] indicator_reasociere_ alias]

В этом синтаксисе:

- *identificator* – идентификатор SQL Server;

- *indicator_reasociere_alias* – необязательный параметр, который если не указан приводит к удалению всех пользователей связанных с *dbo* при смене владельца БД. Если указан параметр TRUE, все пользователи, связанные со старым владельцем, будут связаны с новым владельцем.

Владелец БД также может быть изменен путем редактирования свойств БД в настройках SQL Server Management Studio.

2.2.3.5. Пользователь guest

Ранее было указано, что идентификатор не разрешает доступ к БД, если ему явно не предоставлено право, но есть одно исключение – имя пользователя **guest**.

Guest особый пользователь, который не соблюдает обычные правила. Когда БД содержит пользователя *guest*, любой кто запрашивает доступ к БД и не имеет имени пользователя в БД может получить доступ через эту учетную запись. Поэтому в процессе установки в каждой БД по умолчанию на сервере создается учетная запись *guest*. Имя пользователя *guest* можно удалить со всех существующих дисков БД, кроме БД *master*.

Если подключение выполняется с идентификатором, для которого в БД не создано конкретное имя пользователя и не существует учетная запись *guest*, будет получено сообщение об ошибке. Сообщение об ошибке указывает, что идентификатор неправильно связан с пользователем в БД, и подразумевает что учетная запись *guest* не существует. Чтобы решить эту проблему, либо (предпочтительно) добавьте имя пользователя, соответствующее идентификатору в БД, либо добавьте учетную запись *guest*, выполнив команду:

Пример 2.2.8. *EXCEC sp_grantdbaccess 'guest'*

Если БД master содержит учетную запись *guest*, каждая вновь созданая БД будет содержит учетную запись *guest*. Итак, мы рассмотрели два способа доступа к БД: задав имя пользователя, связанное с идентификатором, или через учетную запись *guest*.

Итак, в качестве разных способов доступа к BD после подключения к SQL Server мы имеем:

1) sa – администратор (или любой другой член роли сервера *sysadmin*) всегда может получить доступ к БД и отображаться как *dbo*, даже если владелец БД был изменен с помощью процедуры *sp_changedbowner*.

2) имя пользователя БД – «обычный» способ доступа к БД – использо-

вание имени пользователя, связанного с идентификатором SQL Server (имя пользователя или имя группы Windows). Это также относится к группам Windows, которым предоставлены права доступа.

3) alias – идентификатор ассоциирован с действительным пользователем БД, таким образом получаются права и привилегии этого пользователя.

4) guest – если все другие методы не работают, SQL Server проверяет наличие гостевой учетной записи в системной таблице БД sysusers. Если это так, guest доступ разрешен, в противном случае доступ запрещен.

2.3. Предоставление и отзыв привилегий

Привилегии – это право на выполнение определенного оператора SQL. Администратор БД – это пользователь высокого уровня, который имеет возможность предоставлять пользователям права доступа к БД и ее объектам.

Привилегии могут включать право на:

- подключиться к БД;
- создавать и запрашивать таблицы;
- выполнять хранимые процедуры, представления и т. д.

Пользователям нужны привилегии для манипулирования содержимым объектов в БД, это позволяет им выполнять определенные действия над конкретным объектом (привилегия на удаление строк из определенной таблицы, добавление или изменение записи).

Каждый объект имеет определенный набор привилегий, которые могут быть ему присвоены.

Привилегии объекта	***Таблица***	***Представление***	***Процедура***
ALTER	*		
DELETE	*	*	
EXECUTE			*
INSERT	*	*	
SELECT	*	*	
UPDATE	*	*	

Можно увидеть, что привилегии варьируют от одного объекта к другому.

2.3.1. Предоставление привилегий

Пользователям предоставляются привилегии, чтобы они могли получать доступ к данным в БД и изменять их. Пользователь может получить привилегии двумя способами:

- явный, прямой к пользователю;

-. назначается роль, а затем она может быть связана с одним или несколькими пользователями.

GRANT privilegiu [privilegiu,…] ON
[OBJECT ::] object_name [,...n])]
TO USER [user / rol, PUBLIC...];

Пример 2.3.1. *GRANT SELECT, INSERT, UPDATE, DELETE ON Student TO nume_bd*

Пользователь *nume_bd* может делать выборки, вставки, изменения и удаления в таблице *Student*.

Пример 2.3.2. *GRANT SELECT, INSERT ON Facultate TO nume_bd*

Пользователь *nume_bd* может делать выборки и вставки в таблице *Facultate*. Так что будет правильно выполнить:

INSERT INTO Facultate(name) values('Informatica')

*SELECT * FROM Facultate*

Такая операция, как *UPDATE Facultate set name = 'test'* не разрешена.

2.3.2. Отзыв привилегий

Привилегии пользователя могут быть отозваны с помощью оператора REVOKE:

REVOKE <privilegiu> [,...n] ON
[OBJECT ::] object_name [,...n])]
{ FROM | TO } <database_principal> [,...n]

- *<privilegiu> [,...n]* одна или несколько привилегий;

- *[OBJECT ::] object_name [,...n])]* один или несколько привилегированных объектов;

- *<database_principal> [,...n]* один или несколько пользователей, у которых отзывается привилегия.

Пример 2.3.3. *REVOKE UPDATE ON Student TO nume_bd*

Это означает, что право на редактирование таблицы *Student* будет отозвано.

2.4. Типы ролей. Создание, назначение и отзыв ролей

Роль пользователя – это набор привилегий, которые можно создавать и назначать пользователям. Роли являются связующим звеном между всеми аспектами SQL Server. Роли SQL Server позволяют группировать имена пользователей БД. Неважно, представляет ли ваше имя пользователя группы Windows, пользователей Windows или идентификаторы SQL Server. Роли могут даже содержать другие роли в качестве членов.

2.4.1. Роль public

В каждой БД есть предопределенная роль, называемая *public*. Все пользователи, все группы и все роли являются членами роли *public* и не могут быть удалены. Удобно ссылаться на всех пользователей, не называя их явно.

2.4.2. Роли на уровне сервера

Идентификатор **sa** всемогущ, он может делать что угодно с экземпляром SQL Server. Это потому, что он является членом роли *sysadmin* на уровне сервера. SQL Server имеет восемь ролей уровня сервера. Идентификатор может быть членом одной или нескольких ролей, но нельзя удалить или добавить роли на уровне сервера.

Список, описывающий полный набор ролей на уровне сервера, доступных в SQL Server:

1) **sysadmin** – члены этой роли могут делать что угодно в SQL Server и указаны как владельцы (dbo) БД (даже если они ими не являются). Роль выходит за рамки всех прав и систем безопасности;

2) **serveradmin** – члены этой роли могут задавать параметры конфигурации с помощью хранимой системной процедуры *sp_configure* и выключать сервер. Они также могут настроить сервис Full-Text. Операторы серверов - подходящие кандидаты на эту роль;

3) **setupadmin** – члены этой роли могут устанавливать и настраивать связанные серверы и отметить хранимую процедуру для выполнения при запуске;

4) **securityadmin** – члены этой роли могут создавать и контролировать идентификаторы подключения к серверу, а также разрешения на создание БД. Могут настраивать атрибуты безопасности связанных серверов и переопределять пароли идентификаторов SQL Server (за исключением членов роли *sysadmin*). Скорее всего, участниками этой роли будут операторы серверов и вспомогательный персонал;

5) **processadmin** – члены этой роли могут управлять процессами, выполняемыми на сервере БД. Эти процессы предполагают «уничтожение» проблемных запросов, и персоналу службы поддержки может понадобиться это право;

6) **dbcreator** – члены этой роли могут создавать, изменять и удалять БД с сервера, а также и восстановить БД на сервер. Администраторы БД могут быть членами этой роли;

7) **bulkadmin** – члены этой роли могут выполнять оператор BULK INSERT (возможность загружать данные непосредственно из файлов в таблицы);

8) **diskadmin** – участники этой роли могут управлять файлами на сервере и создавать их.

2.4.3. Роли на уровне базы данных

Каждая БД содержит роли, некоторые из них фиксированы. Также можно добавить свои собственные роли (что невозможно на уровне сервера). Имейте в виду, что эти роли специфичны для БД, поэтому роль не может влиять на несколько БД одновременно, но одни и те же роли могут быть созданы во всех БД.

Каждая БД содержит набор предопределенных ролей, которым можно присвоить имя пользователя. Таких ролей 9 — фиксированное число навсегда (ни одну из них нельзя удалить). Как и роли на уровне сервера, каждая роль уровня БД предоставляет пользователям определенные права и привилегии, например:

1) **db_owner** - участники этой роли могут делать все что угодно - но только в пределах своей БД. Пользователь, являющийся членом роли *db_owner*, получает практически все права и привилегии пользователя *dbo* (владельца БД). Исключение составляет команда *restore*.

2) **db_accessadmin** - члены этой роли могут предоставлять или отзывать права доступа пользователей к БД (например, запустив хранимую системную процедуру *sp_grantdbaccess*).

3) **db_securityadmin** - члены этой роли могут контролировать все привилегий, роли, членство в ролях и владельцев объектов в БД.

4) **db_ddladmin** - члены этой роли могут создавать, изменять и удалять все объекты в БД, но не могут выполнять команды, связанные с безопасностью (такие как *grant, revoke, deny*).

5) **db_backupoperator** - члены этой роли могут выполнять команды *checkpoint* и *backup*.

6) **db_datareader** - членам этой роли разрешено выполнить выборку данных из любой таблицы, представления или функции в БД.

7) **db_datawriter** - члены этой роли имеют право вводить, обновлять или удалять любую таблицу или представление в БД.

8) **db_denydatareader** - члены этой роли не могут выполнить выборку данных из любой таблицы, представления или функции в БД.

9) **db_denydatawriter** - члены этой роли не могут изменять какую-либо информацию с помощью команд *insert*, *update* или *delete* в любой таблице или представлении БД.

2.4.4. Пользовательские роли на уровне базы данных

В дополнение к предопределенным ролям, можно создавать роли, которые можно назначать пользователям или другим ролям. Причина создания ролей на уровне БД в SQL Server такая же, как и зачем создавать группы на

уровне операционной системы – для удобства группирования пользователей, выполняющих схожие функции. Число созданных ролей должно соответствовать потребностям. Нет никаких ограничений на количество ролей, которым может принадлежать пользователь или другая роль.

2.4.4.1. Создание ролей

Для создания роли используйте системную хранимую процедуру *sp_addrole*:

sp_addrole [@rolename =] 'rol' [,[@ownername =] 'proprietar']

В данном синтаксисе:

- *rol* – имя, которое будет присвоено новой роли;
- *proprietar* – имя пользователя SQL Server, который будет владеть ролью (каждый пользователь может иметь свои собственные роли).

Пример 2.4.1. *EXEC sp_addrole 'rol_student'*

Тот же результат получаем с помощью оператора SQL, для выполнения которого должна быть активна БД *nume_bd*:

CREATE ROLE *rol [* AUTHORIZATION *proprietar]*

Только члены роли *sysadmin* на уровне сервера или роли *db_owner* и *db_securiry* на уровне БД могут добавлять новую роль в БД. То же самое относится и к удалению ролей. Имя роли должно быть уникальным в пределах этой БД.

2.4.4.2. Отзыв ролей

Для удаления роли из БД запустите хранимую системную процедуру *sp_droprole*:

sp_droprole [@rolename =] 'rol'

- *rol* – имя роли, созданной пользователем, которую хотите удалить.

Пример 2.4.2. *EXEC sp_droprole 'rol_student'*

Роль не может быть удалена, если в ней есть пользователь или другие роли. Также нельзя удалить роли, если в них есть объекты.

Тот же результат получаем с помощью оператора SQL, для выполнения которого должна быть активна БД *nume_bd*:

DROP ROLE *rol [* AUTHORIZATION *proprietar]*

2.4.4.3. Назначение ролей

Для добавления пользователей к фиксированным или определяемым пользователем ролям используйте хранимую системную процедуру *sp_addrolemember*:

sp_addrolemember [@rolename] 'rol' , [@membername =] 'cont_baza_date'

В данном синтаксисе:

- *rol* – название роли, которая добавляется пользователю;

- *cont_baza_date* – имя пользователя, группы или роли, которую хотите добавить к этой роли.

Пример 2.4.3. *EXEC sp_addrolemember 'rol_student', 'nume_bd'*

Привилегии этой роли предоставляются следующим образом:

GRANT SELECT, UPDATE, INSERT, DELETE ON Examen TO rol_student

Тот же результат получаем с помощью оператора SQL, для выполнения которого должна быть активна БД *nume_bd*:

ALTER ROLE *rol* ADD MEMBER *cont_baza_date*

2.4.4.4. Удаление ролей

Чтобы удалить участника из роли, запустите системную хранимую процедуру *sp_droprolemember*:

sp_droprolemember [@rolename =] 'rol',[@membername=] 'cont_baza_date'

В данном синтаксисе:

- *rol* – название роли, из которой хотите удалить пользователя;

- *cont_baza_date* – имя пользователя, группы или роли, которую хотите удалить из этой роли.

Пример 2.4.5. *EXEC sp_droprolemember 'rol_student', 'nume_bd'*

Получаем те же результаты с помощью оператора SQL, для выполнения которой БД *nume_bd* должна быть активной:

ALTER ROLE *rol* DROP MEMBER *cont_baza_date*

Члены каждой роли сохраняются в комбинации системных таблиц *sysusers* и *sysmembers*. Существующие роли могут быть изучены с помощью хранимых процедур системы sp_helprole или sp_helprolemember. Обе процедуры получают только один параметр – имя роли между апострофами.

2.4.5. Роли на уровне приложений

Роли на уровне приложения — чрезвычайно полезная функция SQL Server. Хотя на первый взгляд они похожи на другие представленные роли, они выполняют другую функцию, отличаются тем, что могут быть «активированы» приложением. Как только приложение активирует роль на уровне приложения, все права пользователя приостанавливаются, и активны только права этой роли. Для активации роли необходим пароль.

Чтобы создать роль на уровне приложения, используйте хранимую системную процедуру *sp_addapprole*:

sp_addapprole [@rolename =] 'rol', [@password =]..'parola'

В данном синтаксисе

- *rol* – имя роли, которую хотите создать,

- *parola* – пароль, который приложение должно предоставить для активации роли.

Чтобы удалить роль на уровне приложения, запустите хранимую системную процедуру sp_dropapprole:

sp_dropapprole [@rolename =] 'rol'

- *rol* – имя роли, которую необходимо удалить.

Чтобы использовать роль на уровне приложения, запустите хранимую системную процедуру *sp_setapprole*:

sp_setapprole | 'parola' [,[@encrypt =] 'stil_criptare']

В данном синтаксисе:

- *rol* – имя роли, которая будет активирована;

- *parola* – пароль, указанный при выполнении процедуры *sp_addapprole*;

- *encrypt N 'parola'* требует, чтобы пароль был зашифрован при передаче по сети (если указать только пароль, он передается по незашифрованной сети);

- *stil_criptare* определяет используемый стиль шифрования: *none* или *odbc*. Укажите *odbc* (Open Database Connectivity) при использовании клиента на основе ODBC, для шифрования паролей будет использоваться стандартная функция шифрования ODBC.

2.5. Реализация политики паролей

СУБД позволяет пользователю получить доступ к БД, используя пароль. Пароли следует хранить в тайне и часто менять. Система никогда не должна отображать пароли при входе в систему, а сохраненные профили должны надежно храниться в зашифрованном виде. Пароли режима аутентификации SQL Server находятся в столбце паролей таблицы *sysxlogins* в БД master.

Необходимо помнить следующее:

✓ Столбец пароля может быть просмотрен только членом роли *sysadmin* (включая его/ее идентификатор).

✓ Пароли не хранятся в базе данных в открытой форме, вместо них в БД записывается результат применения одного криптографического алго-

ритма хеширования. Эти алгоритмы работают только в одном направлении: мы можем сгенерировать хэш любого контента, но процесс необратим, как бывает в алгоритмах шифрования. При входе в систему, предоставленный пароль проходит через тот же алгоритм хеширования, а затем сравнивается с хэшем пароля что хранится в таблице *sysxlogins*. Если эти два значения совпадают, доступ к серверу разрешен, в противном случае получается сообщение об ошибке *Login Failed* и подключение невозможно.

✓ Если используется Windows Integrated идентификатор безопасности, пароли не сохраняются в SQL Server.

Поскольку пароли являются ключами доступа к своим собственным значениям, их необходимо тщательно защищать, и при их выборе необходимо соблюдать несколько основных правил:

✓ пароль должен быть сложным: содержать цифры, прописные и строчные буквы, специальные символы и иметь длину 12 — 30 символов;

✓ измените стандартные пароли пользователей и администраторов;

✓ пароли необходимо менять не реже одного раза в полгода, а для особо важных данных и того чаще;

✓ общие пароли (используемые многими) должны быть изменены, как только человек покинет группу или отнимается право на ее использование;

✓ пароли будут изменены, как только возникнут подозрения в знании их посторонними лицами или если в силу форс-мажорных обстоятельств, их секрет пришлось раскрыть для восстановления временного нештатного состояния;

✓ если на компьютере хранятся дубликаты паролей, такие файлы необходимо защитить от несанкционированного доступа и создать резервную копию. Доступ к этому файлу должен предоставляться только уполномоченным лицам в соответствии с принципом «никогда не в одиночку». Списки паролей будут храниться в зашифрованном виде;

✓ пароли никогда не будут отображаться на оборудовании в конфигурации системы, а при их вводе поблизости не должно быть посторонних;

✓ чтобы заблокировать операций ввода тысячи повторных тестовых данных, компьютеры должны допускать ограниченное количество попыток ввода пароля, обычно три. Если лимит превышен пользователем, о намерении сообщается системному администратору или сотруднику службы безопасности. В течение этого времени терминал, с которого было предпринято слишком много неудачных попыток, должен быть заблокирован. В случае со специальными системами также рекомендуется блокировать комнату или место, откуда была предпринята попытка входа в систему повторным вводом ошибочных паролей, для идентификации соответствующего лица;

✓ после входа в систему пользователю не должно быть разрешено изменять личность, с которой он вошел в систему, и он не может входить в разделы, назначенные другим пользователям;

✓ если терминал работает в течение длительного времени, процесс аутентификации должен выполняться через регулярные промежутки времени, чтобы гарантировать, что никто другой не использует систему. Если терминал остается неиспользованным, но открытым, он должен автоматически закрыться через определенный интервал времени (например, через 10 минут);

✓ при открытии нового сеанса работы пользователь должен быть проинформирован о последнем входе в систему с соответствующим паролем, чтобы проверить, не использовал ли его кто-то еще.

Библиография к главе 2

1. B. Thuraisingham, *Database and Applications Security: Integrating Information Security and Data Management*, Auerbach Publications, 2005.
2. Microsoft SQL Docs, Permissions (Database Engine), 2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/relational-databases/security/permissions-database-engine?view=sql-server-ver15

III. ВНУТРЕННЯЯ БЕЗОПАСНОСТЬ ДАННЫХ

Важнейшей проблемой в управлении БД является обеспечение того, чтобы данные БД соответствовали заранее определенному набору правил. Эти правила должны соответствовать бизнес-операциям СУБД. Администратор БД и разработчик приложения (а также руководитель предприятия) несут ответственность за определение и создание этих правил.

Целостность данных включает в себя определение правил, которые ограничивают допустимые значения для столбца таблицы. Ограничения целостности определены для таблиц, поэтому все представления подчиняются ограничениям целостности. Если оператор языка манипулирования данными DML (Data Manipulation Language) пытается выполнить действие, нарушающее ограничение целостности, генерируется ошибка и выполняется откат транзакции.

3.1. Ограничения целостности

Для обеспечения целостности данных, содержащихся в БД, необходимо обеспечить следующее:

a) целостность домена, которая предполагает, что каждое отдельное значение соответствует определенному правилу;

Можно использовать различные способы проверки данных, введенных в БД:

- *столбец данных ограничен значениями, которые можно ввести в этот столбец* (это предотвратит ввод текстового описания в столбец с датами или даты в столбец зарплатами);

- *могут быть ограничены максимальная длина и минимальная длина для каждого заданного значения* (например, код студента может иметь длину не менее пяти и не более десяти символов);

- *могут быть ограничены данные, соответствующие определенному формату* (это может быть полезно для проверки почтовых индексов или телефонных номеров);

- *может использоваться ограничение допустимого диапазона значений* (вводимое значение может быть ограничено, например: день рождения студента, датами между 1990-01-01 и текущим днем), чтобы избежать возможных ошибок;

- *значение в столбце* (например, в поле учитель может быть записана одна из возможных ученых степеней: магистр, доцент, академик).

b) целостность объекта, которая предполагает, что каждый объект может быть однозначно идентифицирован.

Любой реальный объект должен легко идентифицироваться в БД. Сложно обращаться к клиенту: клиент, который живет в Кишиневе, имеет 3 детей и 100 сотрудников, ему 40 лет, его имя Ион, а номер телефона оканчивается на 345. Для людей эта информация может быть достаточной, чтобы просмотреть в памяти и идентифицировать клиента. Однако для компьютерной программы, такой как SQL Server, этот метод идентификации заставит ее последовательно применять разные условия, по одному условию для каждого атрибута. Вероятно, было бы проще идентифицировать каждого клиента по одному уникальному значению, например 256634, хранящемуся в столбце идентификации, таком как CustomerID. Таким образом, для поиска клиента SQL Server должен будет применить очень простое условие CustomerID = 25634.

Это особенно важно, если хотите установить связь между информацией и другими сущностями, потому что каждая ссылка должна основываться на простейших отношениях.

c) реляционная целостность, предполагает, что сообщаемая дата правильно связана.

Реляционные базы данных называются «реляционными», потому что единицы данных, хранящиеся в них, связаны между собой отношениями, такими как:

- у клиентов есть торговые представители, которые их обслуживают;
- клиенты размещают заказы;
- заказы имеют определенный порядок;
- каждый объект в заказе относится к одному товару;
- товары упорядочены по категориям;
- продукция поступает от производителей.

Необходимо убедиться, что все связи хорошо установлены, и что нет данных без ссылок на остальные данные. В некоторых случаях применения сложных правил целостности невозможно добиться с помощью стандартных реляционных структур, и в таких ситуациях будут создаваться хранимые процедуры, триггеры, пользовательские функции или внешние компоненты.

Ограничение целостности для таблицы устанавливается при создании или изменении таблицы. Для этой цели включим условие ограничения в команды *CREATE TABLE* или *ALTER TABLE* следующим образом:

CONSTRAINT nume_restrictie tip_restrictie

Существуют следующие типы ограничений целостности данных, которые рекомендуется применять во время проектирования БД, но в случае внедрения также будут учитываться такие факторы, как производительность

(время выполнения, объем данных, область действия) и масштабируемость (возможность расширения БД):

1. ***NULL***, по умолчанию все столбцы таблицы принимают нулевые значения. Установив ограничение целостности *NOT NULL*, мы заставляем указанный столбец иметь значение для каждой строки таблицы.

Пример 3.1.1.

```
CREATE TABLE test(
   id varchar(8) NOT NULL,
   name nvarchar(50) NOT NULL )
```

2. ***UNIQUE***, накладывает ограничение все значения в столбце различны, а столбец, определяемый ограничением уникальности, является столбцом с уникальным ключом. Не должно быть более одного кортежа с тем же значением, что и у ключа в таблице.

Пример 3.1.2.

```
CREATE TABLE test(
   id varchar(8) NOT NULL UNIQUE,
   name nvarchar(50) NOT NULL )
```

Пример 3.1.3.

```
CREATE TABLE test(
   id varchar(8) NOT NULL ,
   name nvarchar(50) NOT NULL,
CONSTRAINT c1 UNIQUE (id) )
```

Результат, полученный в примере 3.1.3, имеет тот же эффект, что и в примере 3.12. В этом случае будет правильным вставить следующие данные:

```
INSERT INTO test values(1,'Elena')
INSERT INTO test values(2,'Ion')
```

но не можем:

```
INSERT INTO test values(1,’Elena’)
INSERT INTO test values(1,’Ion’)
```

Пример 3.1.4.

```
CREATE TABLE test(
   id varchar(8) NOT NULL ,
   name nvarchar(50) NOT NULL,
CONSTRAINT c1 UNIQUE (id,name) )
```

Для этого случая будет правильным вставить данные:

```
INSERT INTO test values(1,'Elena')
INSERT INTO test values(1,'Ion')
```

Будет отображаться исключение проверки если попытаемся добавить:
INSERT INTO test values(1,'Elena')

Поскольку все нулевые значения считаются уникальными и отличными с точки зрения этого ограничения, рекомендуется поместить ограничение *NOT NULL* в столбцы с значениями *UNIQUE*, тем самым принудительно введя значение для этого столбца и убедившись, что значения отличаются.

3. ***Primary key*** определяет столбец или комбинацию столбцов в качестве первичного ключа таблицы, который однозначно идентифицирует каждую строку таблицы и неявно создает ограничения *NOT NULL* и *UNIQUE* для этого столбца. Хотя нет необходимости иметь первичный ключ для каждой таблицы, рекомендуется создать ключ, значения которого никогда не меняются.

Пример 3.1.5.

```
CREATE TABLE test(
   Id varchar(8) NOT NULL PRIMARY KEY,
   name nvarchar(50) NOT NULL )
```

или

```
CREATE TABLE test(
   id varchar(8) NOT NULL,
   name nvarchar(50) NOT NULL,
CONSTRAINT pk_test PRIMARY KEY (id) )
```

Также можно добавить два поля в качестве первичных ключей следующим образом:

CONSTRAINT pk_test PRIMARY KEY (id, name)

Каждая таблица будет иметь не более одного первичного ключа, который не может быть *long* или *long raw*.

4. Ограничение ***ссылочной целостности*** накладывает бизнес-правила, определяющие отношения между столбцами различных таблиц, и между таблицами создаются отношения родитель-потомок. Родительская таблица содержит ссылочный ключ, а дочерняя таблица содержит внешний ключ.

Определение 3.1. *Внешний/первичный ключ* называется столбец или набор столбцов, включенным в определение ограничения целостности посредством ссылки.

Определение 3.2. *Ссылочный ключ* называется уникальный ключ или первичный ключ той же таблицы или другой таблицы, на которую ссылается внешний ключ.

Определение 3.3. *Зависимой таблицей* называют таблицу, которая содержит внешний ключ.

Определение 3.4. *Ссылочной таблицей* называется таблица, на которую ссылается внешний ключ.

Пример 3.1.6. Пусть таблица *Curs* содержит поле *id*, которое является *PRIMARY KEY* для этой таблицы.

```
CREATE TABLE Curs(
   id varchar(8) NOT NULL,
   name nvarchar(50) NOT NULL,
   id_facult varchar(8) NOT NULL,
   nr_credite tinyint NOT NULL,
   nr_ore tinyint NOT NULL,
CONSTRAINT pk_Curs PRIMARY KEY(id) )
```

Будем ссылаться на это поле *id* в таблице *test*, где мы добавили *id_curs* как *FOREIGN KEY* по ссылке:

```
CREATE TABLE test(
   id varchar(8) NOT NULL,
   name nvarchar(50) NOT NULL,
   id_curs int NOT NULL,
CONSTRAINT pk_test PRIMARY KEY (id),
CONSTRAINT fk_test FOREIGN KEY(id_curs) REFERENCES Curs (id) )
```

5. *CHECK* явно определяет условие, которое должно быть истинным, и также будет оцениваться со значением *Unknown*, чтобы разрешить присутствие значения *NULL*, что означает, что каждая запись должна соответствовать этому условию. Чтобы удовлетворить этому ограничению, каждая строка таблицы должна соответствовать условию. Если предпринимается попытка выполнить инструкцию, которая привела бы к несоблюдению условия, то инструкция откатывается. Ограничение *check* не может содержать запросы.

Пример 3.1.7.

```
CONSTRAINT CK_Student_an CHECK ([an_nastere] between '1950-01-01' and GETDATE())
```

Пример 3.1.8.

```
CONSTRAINT CK_Student_sex CHECK ([sex] in ('F','M'))
```

3.2. Представления

Представления – это логические таблицы, похожие на окна, через которые можно просматривать одну или несколько таблиц или представлений. Представление выглядит как таблица со столбцами и строками, которые можно обновлять, вставлять или удалять, как если бы это была настоящая таблица (только если представление ссылается только на одну таблицу). Представление на самом деле представляет собой логическую таблицу, в ко-

торой не хранятся данные. Получает данные из таблиц или представлений, на которых они основаны. Представление создается с помощью запроса и поэтому может рассматриваться как виртуальная таблица или сохраненный запрос, поскольку представления выглядят и ведут себя как таблицы, их можно использовать как таблицы в большинстве ситуаций.

Представления являются динамическими и всегда отображают текущую информацию таблицы. При манипулировании таблицами (когда в таблицу вносятся изменения, они также отражаются в представлении) эти изменения мгновенно отражаются в представлении. Представления позволяют отображать данные в форме, отличной от той, в которой они хранятся в таблицах, и позволяют адаптировать представление данных к конкретным требованиям различных типов пользователей.

Представления создаются для следующих целей:

- Обеспечить более высокий уровень безопасности БД за счет ограничения доступа к заранее определенному количеству столбцов и строк таблицы. Это позволяет пользователям просматривать узкий набор данных.

- Для повышения безопасности БД путем сокрытия настоящих имен и местоположений базовых таблиц (данные могут быть из разных таблиц, из разных БД, с измененными именами столбцов, со столбцами содержащими данные в результате вычислений, измененное содержимое, например объединение данных в виде имени и фамилии, которые были в двух столбцах будут представлены в виде одного столбца).

- Чтобы упростить представление данных, скрыв структуры таблиц, которые лежат в основе представления (когда нужно просмотреть данные, содержащиеся в разных таблицах, потребуется, каждый раз выполнять сложный *SELECT* или *UNION*).

- Для отображения данных в представлении, отличном от представления базовых таблиц.

- Упростить работу по программированию за счет возможности запрашивать виртуальную таблицу вместо запросов к каждой из базовых таблиц.

3.2.1. Создание представлений

Чтобы создать представление, можно использовать команду SQL ***CREATE VIEW***. Представление можно определить с помощью любого запроса, который ссылается на таблицы или другие представления.

Пример 3.2.1.

```
CREATE VIEW vtest
AS SELECT name, surname, an_studii FROM Student
```

В этом примере код позволяет создать простое представление с *name, surname, an_studii* из таблицы *Student*.

Пример 3.2.2.

CREATE VIEW vtest

AS SELECT name, surname, an_studii FROM Student WHERE an_studii = 3

Код в примере позволяет создать простое представление с *name, surname, an_studii* из одной таблицы при условии, что *an_studii=3*.

Пример 3.2.3.

CREATE VIEW vtest1

AS SELECT s.name, s.surname, s.an_studii, c.name AS curs FROM Student s, Curs_Student cs, Curs c

WHERE s.id =cs.id_student AND cs.id_curs=c.id AND s.an_studii = 3

В этом случае код позволяет создать сложное представление *vtest1*, которое ссылается только на те записи, которые содержат *name*, *surname, an_studii* и *curs* при условии, что *an_studii = 3*.

Пример 3.2.4.

CREATE VIEW stud_fara_restante AS

SELECT s.name, s.surname

FROM Student s

INNER JOIN (SELECT id_stud FROM Examen GROUP BY id_stud

HAVING MIN(nota) > =5) e ON s.id = e.id_stud

В этом примере код позволяет создать более сложное представление *stud_fara_restante*, которое ссылается только на те записи, которые содержат *name* и *surname*, для которых минимальная оценка экзамена *nota* равна 5.

3.2.2. Изменение представлений

Определение вида может быть заменено двумя способами:

1. Представление можно уничтожить, а затем воссоздать с новым определением. При уничтожении представления все привилегии аннулируются и должны быть воссозданы для нового представления.

2. Представление можно воссоздать, переопределив его с помощью оператора CREATE VIEW с необязательным предложением REPLACE. Этот метод используется для замены текущего определения представления с сохранением всех текущих привилегий.

Пример 3.2.5.

CREATE or REPLACE VIEW vtest AS SELECT name, an_studii FROM Student

Замена представлений имеет следующие эффекты:

- обновляется определение представления и ни один из базовых объектов не затрагивается заменой представления;

- все ограничения, существовавшие в исходном представлении, не обнаруживаются в новом представлении, они уничтожаются.

На использование представлений распространяются следующие ограничения:

- представление нельзя использовать для выполнения операций *INSERT*, *UPDATE* или *DELETE*, когда запрос представления содержит операцию *JOIN*, операторы *SET* или *DISTINCT*, предложение *GROUP BY*;

- представление, определенное с помощью *WITH CHECK OPTION*, нельзя использовать для вставки или обновления базовых таблиц;

- записи не могут быть вставлены в базовую таблицу, если столбец *NOT NULL* в этой строке определен без *DEFAULT VALUE*;

- использование представлений требует дополнительного времени обработки перед чтением данных, что немного снижает скорость отклика сервера BD.

3.2.3. Перекомпиляция представлений

Для явной перекомпиляции представления будет использоваться команда:

ALTER VIEW nume_vedere COMPILE

Перекомпиляция позволяет обнаруживать любые ошибки, связанные с этим представлением, до того, как представление будет выполнено в реальной среде предприятия. Желательно явно перекомпилировать представление после любого изменения структуры любой из базовых таблиц представления.

Пример 3.2.6.

ALTER VIEW vtest COMPILE;

В примере перекомпилируется представление *vtest*.

Команда ALTER VIEW не изменяет определение представления или любого другого объекта, который может от него зависеть.

3.2.4. Уничтожение представления

Команда DROP VIEW используется для уничтожения представления.

Пример 3.2.7.

DROP VIEW vtest;

Код в примере позволяет уничтожить представление vtest.

Можем сделать следующие выводы:

Преимущества использования представлений:

1. *Упрощение запросов*, несколько запросов, примененных к нескольким БД, могут применяться только к одному представлению.

2. *Структурная простота*, представления могут создавать личное видение пользователя для интерпретации результатов.

3. *Безопасность*, ограничение доступа к данным для пользователей.

Недостатки использования представлений:

1. *Производительность*, запросы на просмотр могут превратиться в базовые запросы к таблицам, что отразится на производительности.

2. *Ограничения на обновления*, многие представления могут быть защищены только для чтения, в этом случае нельзя делать обновления.

3.3. Шифрование данных

Шифрование – это процесс, основанный на алгоритме, с помощью которого данные, подлежащие этому процессу, также называемые входными данными, изменят свою первоначальную форму так, что станут неразборчивыми без выполнения алгоритма расшифровки – процесс, выполняемый при шифровании, возобновляется в обратном порядке.

Наука, изучающая алгоритмы, лежащие в основе процесса шифрования и дешифрования информации, называется криптография и основана на сложных математических понятиях. Принцип шифрования заключается в том, что алгоритм, который будет обрабатывать набор данных/информацию, получит уникальный результат для исходной информации, никакой другой набор данных не будет иметь такого конечного результата. В информатике шифрование данных используется везде, где требуется безопасность передачи конфиденциальной информации с одного компьютера на другой или хранения данных с использованием пароля. Области применения варьируются от государственных и частных учреждений (таких как банки) до личных данных пользователей.

Для повышения уровня безопасности в процессе шифрования появилась изменяющаяся информация, но основанная на одном и том же алгоритме – так называемые ключи. Другими словами, алгоритм тот же, но будет использоваться другой ключ шифрования, что приведет к шифрованию с помощью шаблона шифрования, указанного ключом шифрования. Существует два типа шифрования: симметричное и асимметричное. Симметричное шифрование будет использовать один и тот же ключ как для шифрования, так и для расшифровки данных – тот, у кого есть ключ,

может одновременно шифровать и расшифровать информацию. Второй тип сложнее первого, потому что используются два ключа, один открытый – для шифрования, а другой закрытый – для расшифровки. Как следует из названия, открытый ключ виден всем и используется для шифрования данных. Обычно открытые ключи хранятся в открытом доступе и могут использоваться совместно, в то время как закрытые ключи генерируются с помощью пароля.

Проблемы безопасности, которые не решает шифрование.

Хотя есть много веских причин для шифрования данных, есть много причин не шифровать данные.

1. Шифрование не решает проблемы контроля доступа.

Известно, что при шифровании данных шифрование не должно мешать настройке управление доступом. Большинству предприятий необходимо ограничить доступ к данным для пользователей, которым необходимо видеть эти данные. Например, система управления персоналом может разрешить сотрудникам просматривать только записи о своей работе, позволяя руководителям сотрудников просматривать записи о подчиненных. Специалисты по кадрам также могут просматривать записи нескольких сотрудников.

Механизмы управление доступом обычно используются для реализации политик безопасности, которые ограничивают доступ к данным для тех, кому необходимо их видеть. Если управление доступом реализовано правильно, шифрование не добавит дополнительной безопасности БД. Пользователь, имеющий права доступа к данным в БД, не имеет больше или меньше прав из-за шифрования. Поэтому никогда не следует использовать шифрование для решения проблем контроля доступа.

2. Шифрование не защищает от злонамеренного администратора.

Некоторые предприятия обеспокоены тем, что злоумышленник пытается получить привилегии (например администратора БД). Правильным решением этой проблемы является защита учетной записи администратора БД и изменение паролей по умолчанию для других привилегированных учетных записей. Самый простой способ войти в БД — использовать пароль по умолчанию для привилегированной учетной записи, который администратор разрешил оставить без изменений.

Хотя есть много разрушительных действий, которые злонамеренный пользователь может сделать с БД после получения привилегии администратора БД, шифрование не защитит многие из них. Примеры включают повреждение или удаление данных, экспорт пользовательских данных в файловую систему, отправку электронных писем самому себе для запуска взломщика паролей и так далее.

Компании обеспокоены тем, что администраторы БД, которые обычно обладают всеми привилегиями, могут видеть все данные в БД. Эти компании считают, что администраторы БД должны управлять БД, но не должны видеть содержащиеся в нем данные. Сосредоточение привилегий на одном человеке кажется им несправедливым, и они предпочли бы разделить роль администратора БД или ввести правила доступа для двух человек.

Заманчиво полагать, что шифрование всех данных (или значительных объемов данных) решит эти проблемы, но есть лучшие способы защиты от угроз. Например, SQL Server поддерживает ограниченное совместное использование привилегий администратора БД *sysadmin* — они могут делать все что угодно в SQL Server и указаны как владельцы (dbo) БД (даже если они ими не являются) и могут обходить все разрешения и системы безопасности и *setupadmin* — могут устанавливать и настраивать связанные серверы и могут пометить хранимую процедуру для выполнения при запуске. Могут быть созданы меньшие роли, которые включают некоторые системные привилегии. Роль может включать не все системные привилегии, а только те, которые подходят администратору БД. Периодический аудит может помочь выявить неправомерные действия администратора.

Шифрование хранимых данных не должно мешать управлению БД, так как это может вызвать серьезные проблемы с безопасностью. Например, если данные повреждены при шифровании, возникает проблема безопасности, поскольку сами данные невозможно интерпретировать и восстановить. Шифрование можно использовать для ограничения возможности администратора БД или другого привилегированного пользователя просматривать данные в БД. Однако это не заменит надлежащее управление привилегиями администратора БД или управление использованием мощных системных привилегий. Если незаслуженные пользователи обладают значительными привилегии, то они могут представлять угрозу для бизнеса, некоторые из которых гораздо более серьезны, чем просмотр незашифрованных номеров кредитных карт.

3. Шифрование всего не обеспечивает безопасность данных.

Распространенной ошибкой является мнение, что если шифрование данных укрепляет безопасность, то зашифровав все, мы обезопасим все данные. Важно, чтобы шифрование не мешало обычному управлению доступом. Кроме того, шифрование всей рабочей БД означает, что все данные должны быть расшифрованы для чтения, обновления или удаления. Шифрование всех данных значительно повлияет на производительность.

Доступность является ключевым аспектом безопасности. Если шифрование данных делает данные недоступными или отрицательно влияет на доступность за счет снижения производительности, то полное шифрование

создаст новую проблему безопасности. На доступность также отрицательно влияет тот факт, что БД недоступна при изменении ключей шифрования, поскольку передовые методы обеспечения безопасности требуют регулярных изменений. Когда ключи необходимо сменить, БД недоступна, пока данные расшифровываются и набираются с помощью нового ключа или ключей.

Шифрование является преимуществом для данных, хранящихся offline. Например, компания может хранить резервные копии на срок от 6 месяцев до года в удаленном месте. Первая линия защиты заключается в обеспечении простоты хранения данных путем установления контроля физического доступа. Шифрование этих данных перед их сохранением может обеспечить дополнительные преимущества, поскольку доступ к ним не осуществляется в режиме онлайн, и производительность не должна учитываться (поэтому это никак не влияет на производительность). Есть провайдеры, которые предоставляют услуги шифрования. Прежде чем приступить к крупномасштабному шифрованию резервных копий данных, компании, рассматривающие такой подход, должны тщательно протестировать этот процесс. Важно убедиться, что данные, зашифрованные для хранения offline, могут быть успешно расшифрованы и повторно импортированы.

Библиография к главе 3

1. A. Watt, N. Eng, *Database Design*, 2014.
2. Beginner SQL Tutorial, SQL Integrity Constraints [citat 28.12.2021]. https://beginner-sql-tutorial.com/sql-integrity-constraints.htm

IV. МЕТОДЫ АУДИТА БАЗ ДАННЫХ

Аудит должен обеспечивать баланс, проверяя выполняются ли условия необходимые для его поддержания, контролируя беспорядок, адаптацию к переменам, оценивает степень безопасности и риски.

Определение 4.1. *Аудит* – это процесс, посредством которого компетентные и независимые лица собирают и оценивают доказательства, чтобы составить мнение о степени соответствия между наблюдаемыми и заранее установленными критериями.

Определение 4.2. *Триггер* — это особый тип хранимой процедуры, которая запускается автоматически при возникновении события на сервере базы данных.

4.1. Аудит входа в систему на уровне SQL Server

Этот тип аудита записывает событие для каждого входа в систему (logon). Для каждого события необходимо сохранить имя пользователя *loginname* и временную метку *timestamp*. Также может быть использована дополнительная информация, включающая в себя TCP/IP-адрес клиента (IP-адрес компьютера, с которого происходит подключение) и программа, используемая для инициации подключения (например, Management Studio).

Кроме того, будут фиксироваться все неудачные попытки подключения, которые иногда даже важнее успешных и на основании которых выдается оповещение. Чрезмерное количество неудачных попыток входа в систему — это интересный отчет о безопасности, который часто предполагает неудачные попытки подключения. В этом случае учитываются данные по следующим параметрам: *имя пользователя, IP-адрес клиента с которого не удалось установить соединение, исходная программа, время суток*.

Действия при входе в систему можно проверить:

- использовав возможности БД;
- или с помощью внешнего решения безопасности БД.

Все СУБД поддерживают эту базовую функцию аудита, и поскольку количество таких событий довольно мало (по крайней мере, по сравнению с количеством событий, которые можно получить при аудите текущих вызовов SQL), при выполнении этого аудита уровень штрафа за снижение производительности невелик.

Пример 4.1.1. Чтобы иметь возможность реализовать этот аудит, создается таблица *LogonAudit*, в которой хранится необходимая информация:

```
CREATE TABLE LogonAudit
(
```

```
AuditID INT NOT NULL CONSTRAINT PK_LogonAudit_AuditID
PRIMARY KEY CLUSTERED IDENTITY(1,1)
, UserName NVARCHAR(255)
, LogonDate DATETIME
, spid INT NOT NULL --SP ID numarul procesul interior SQL Server
, HostName varchar(255)
, ProgramName varchar(255)
, DbName varchar(255)
);
```

Затем создается триггер, который срабатывает при новом входе в систему, и в таблицу аудита LogonAudit записывается, кто вошел в систему (SUSER_SNAME()), когда произошел вход в систему (GETDATE()), идентификатор входа в SQL Server (@@SPID), имя сетевого компьютера (HOST_NAME()), с помощью какого приложения осуществлялось подключение (PROGRAM_NAME()) и к какой БД происходит подключение (DB_NAME()):

```
CREATE TRIGGER MyLogonTrigger ON ALL SERVER FOR LOGON
AS
BEGIN
  INSERT INTO Test.dbo.LogonAudit (UserName, LogonDate, spid,
HostName, ProgramName, DbName)
VALUES (SUSER_SNAME(), GETDATE(), @@SPID, HOST_NAME(),
PROGRAM_NAME(), DB_NAME());
END;
```

4.2. Аудит манипулирования данными DML (Data Manipulation Language)

Аудит действий DML (*SELECT, INSERT, UPDATE, DELETE*) является обязательным требованием, особенно когда речь идет о точности финансовой информации. Журналы аудита обработки данных распространены почти во всех аудиторских инициативах. Связанным с этим требованием аудита (хотя и не столь распространенным, как необходимость контроля действий DML) является полная запись старого и нового значения для каждого действия DML.

Пример 4.2.1. Журнал аудита может быть создан для столбца таблицы сотрудников, в котором хранятся годовые бонусы. В этом случае могут быть созданы два разных требования. Первая может содержать полную запись любого обновления этих значений, и каждая запись содержит информацию

о пользователе, выполнившем обновление, используемом приложении, времени выполнения обновления и текущем состоянии оператора SQL. Вторым требованием может быть запись всей вышеуказанной информации, а также запись значения до обновления и значения после обновления. Это нужно делать, потому что я могу дать себе (вношу изменения в свою пользу) 50% бонус, используя:

*UPDATE EMP SET BONUS = BONUS * 1.5*

Журналы аудита DML, которые включают запись старых и новых значений, являются важным типом аудита, который следует включать в большинстве случаев. Однако необходимо с осторожностью работать с этой категорией и проводить выборочный аудит, так как в некоторых случаях для простоты активируется аудит для каждой операции DML. Хотя это технически возможно, объем создаваемых данных может быть большим, и необходимо обеспечить, чтобы инфраструктура аудита могла справиться с задачей, особенно при включении старых и новых значений.

Пример 4.2.2. Предположим, что выполняются 1 миллион транзакций DML в день, и каждая транзакция обновляет одно значение, в БД 100 таблиц, в каждой по 10 значений, которые можно обновить, и аудит начинается с БД, содержащей 10 000 записей в каждой таблице. Хотя расчет прост и неточен, если записывать старые и новые значения, через год аудиторская БД будет более чем в 35 раз выше, чем сама БД.

Поэтому при обсуждении следует тщательно выбирать какие элементы и какие команды являются частью предмета аудита DML. Например, можете создать контрольные журналы для подмножества таблиц БД, для подмножества записей или учетных записей и т. д. Еще более избирательным является выбор таблиц и столбцов, которые должны поддерживать старые и новые значения.

Аудит DML поддерживается тремя основными методами, но ежедневное (или периодическое) сравнение в данном случае невозможно:

1. ***Возможности БД*** – это внутренние системные инструменты, с помощью которых мы можем создавать контрольные журналы.

2. ***Система внешнего аудита*** предполагает использование систем аудита БД, которые поддерживают аудиты DML на основе любых критериев фильтрации, включая объект БД, пользователя, приложение и т.д. Они также помогают собирать и сжимать эту информацию и делать ее доступной для структур отчетности, даже когда объем данных огромен.

3. ***Триггер,*** используются настраиваемые триггеры. Если нужно создать контрольный журнал DML для нескольких объектов, добавление триггеров,

отображающих информацию в специальной контрольной таблице, может быть самым простым и быстрым способом.

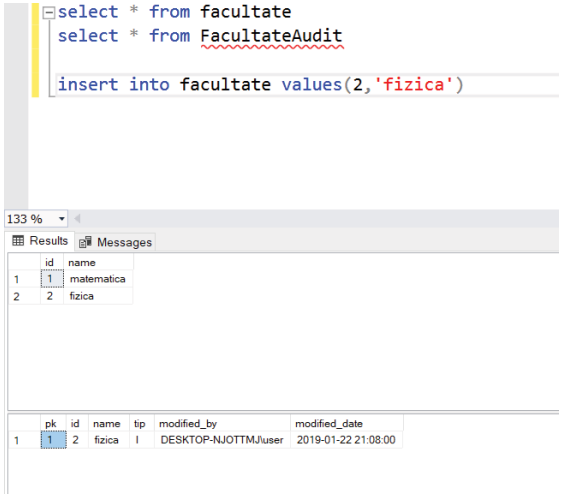
Пример 4.2.3. Для таблицы *Facultate*, которая содержит поля *id* и *name*, создается таблица *FacultateAudit*:

```
CREATE TABLE [dbo].[FacultateAudit](
    [pk] [int] IDENTITY(1,1) NOT NULL,
    [id] [int] NULL,
    [name] [nvarchar](50) NULL,
    [tip] [char](1) NOT NULL,
    [modified_by] [varchar](50) NOT NULL,
    [modified_date] [smalldatetime] NULL,
 CONSTRAINT [PK_FacultateAudit] PRIMARY KEY
([pk] ASC))
```

Для этой таблицы запускается триггер *INSERT UPDATE:*

```
CREATE TRIGGER [dbo].[trg_Facultate_IU] ON [dbo].[Facultate]
FOR INSERT, UPDATE
AS
BEGIN
    DECLARE @tip_op char(1) = 'I'
IF EXISTS(SELECT * FROM Deleted) SET @tip_op = 'U'
    INSERT INTO FacultateAudit([id], [name], [tip], [modified_by])
    SELECT [id], [name], @tip_op, SUSER_SNAME()
    FROM inserted;
 END
```

Таким образом, если в таблице *Facultate* была запись *matematica*, то после добавления записи *fizica*, в таблице *FacultateAudit* появляется новая запись, которая соответствует записи *(2, fizica)* из таблицы *Facultate*.



Если добавить в таблицу еще один факультет, *chimie* и изменить *fizica* на *Fizica*, тогда получим:

```
select * from facultate
select * from FacultateAudit

insert into facultate values(2,'fizica')
insert into facultate values(3,'chimie')
update facultate set name = 'Fizica' WHERE id = 2
```

133 %

Results | Messages

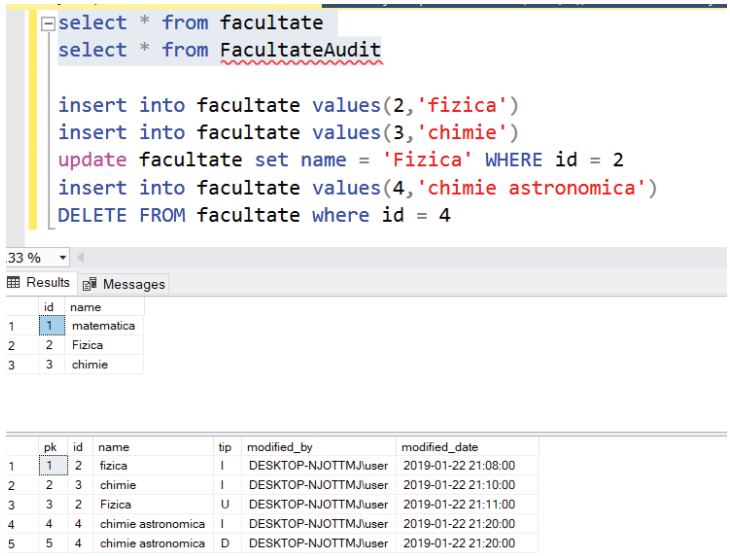
	id	name
1	1	matematica
2	2	Fizica
3	3	chimie

	pk	id	name	tip	modified_by	modified_date
1	1	2	fizica	I	DESKTOP-NJOTTMJ\user	2019-01-22 21:08:00
2	2	3	chimie	I	DESKTOP-NJOTTMJ\user	2019-01-22 21:10:00
3	3	2	Fizica	U	DESKTOP-NJOTTMJ\user	2019-01-22 21:11:00

Триггер *INSERT UPDATE DELETE* может срабатывать для той же таблицы:

```
CREATE TRIGGER [dbo].[trg_Facultate_IU]
   ON [dbo].[Facultate]
FOR INSERT, UPDATE, DELETE
AS
BEGIN
   DECLARE @tip_op char(1) = 'D'
   IF EXISTS(SELECT * FROM inserted)
   BEGIN
         SET @tip_op = 'I'
         IF EXISTS(SELECT * FROM Deleted)
               SET @tip_op = 'U'
      INSERT INTO FacultateAudit([id], [name], [tip], [modified_by])
         SELECT [id], [name], @tip_op, SUSER_SNAME()
         FROM inserted;
   END
   ELSE BEGIN
      INSERT INTO FacultateAudit([id], [name], [tip], [modified_by])
         SELECT [id], [name], @tip_op, SUSER_SNAME()
         FROM deleted;
   END
END
```

что приводит к такому результату, как:



```
select * from facultate
select * from FacultateAudit

insert into facultate values(2,'fizica')
insert into facultate values(3,'chimie')
update facultate set name = 'Fizica' WHERE id = 2
insert into facultate values(4,'chimie astronomica')
DELETE FROM facultate where id = 4
```

33 %

Results | Messages

	id	name
1	1	matematica
2	2	Fizica
3	3	chimie

	pk	id	name	tip	modified_by	modified_date
1	1	2	fizica	I	DESKTOP-NJOTTMJ\user	2019-01-22 21:08:00
2	2	3	chimie	I	DESKTOP-NJOTTMJ\user	2019-01-22 21:10:00
3	3	2	Fizica	U	DESKTOP-NJOTTMJ\user	2019-01-22 21:11:00
4	4	4	chimie astronomica	I	DESKTOP-NJOTTMJ\user	2019-01-22 21:20:00
5	5	4	chimie astronomica	D	DESKTOP-NJOTTMJ\user	2019-01-22 21:20:00

где I – *INSERT*, U – *UPDATE*, D – - *DELETE*. Первая таблица – это таблица, в которую вносятся изменения, а вторая – это таблица аудита. В каждом из представленных примеров можно проследить активность DML над элементами таблицы *Facultatea*, результаты которой регистрируются в таблице аудита *FacultateAudit*. В таблице аудита, помимо измененных данных, записывается информация о типе изменения, кем и когда оно было произведено.

4.3. Аудит действий DDL (Data Definition Language)

Аудит активности DDL всегда был важен и стал одним из наиболее часто применяемых журналов аудита, поскольку аудит изменений схемы важен с точки зрения безопасности, соответствия требованиям и управления конфигурацией (добавление/изменение столбца, добавление/изменение условие целостности). С точки зрения безопасности, DDL-команды потенциально являются наиболее опасными из существующих команд и, безусловно, могут быть использованы злоумышленником для компрометации любой системы. Даже кража информации часто может быть связана с DDL-командами (например, путем создания дополнительной таблицы, в которую можно скопировать данные перед извлечением). С точки зрения соответствия, многие правила требуют аудита любых изменений в структурах данных, таких как таблицы и представления.

Изменения схемы/структуры БД необходимо проверять и сохранять для дальнейшего использования, чтобы быстро выявлять и устранять ошибки, которые могут поставить под угрозу переносимость данных или вызвать их повреждение. Аудит деятельности DDL также реализован с целью устранения ошибками, которые могут быть допущены разработчиками и администраторами БД и которые могут иметь катастрофические последствия.

Пример 4.3.1. У клиента может быть время простоя из-за изменения, внесенного разработчиком — изменения, которое, по мнению разработчика, он делает на сервере разработки, но ошибочно внесено на рабочем сервере. Строгий контроль процесса управления конфигурацией важен и является одним из основных направлений аудита DDL.

Большинство СУБД позволяют проверять активность DDL с помощью различных механизмов аудита, таких как триггеры DDL, которые запускаются при изменении структуры схемы системы.

Пример 4.3.2. Создается таблица аудита *DDLEvents*:

```
CREATE TABLE dbo.DDLEvents
(
EventDate DATETIME NOT NULL DEFAULT CURRENT_
TIMESTAMP,
EventType NVARCHAR(64),
EventDDL NVARCHAR(MAX),
EventXML XML,
DatabaseName NVARCHAR(255),
SchemaName NVARCHAR(255),
ObjectName NVARCHAR(255),
HostName VARCHAR(64),
IPAddress VARCHAR(48),
ProgramName NVARCHAR(255),
LoginName NVARCHAR(255)
);
```

Создается триггер, позволяющий записывать информацию о типе события (Create Table, Alter Table, Drop Table и т. д.) выполненном оператором SQL: когда произошло событие, идентификатор сеанса входа в SQL Server, вся информация о событии хранящаяся в формате XML, БД над которой происходит событие, схема над которой происходит событие, над каким элементом схемы происходит событие (столбец, таблица), имя сетевого компьютера, IP-адрес, приложение которое использовалось и имя пользователя:

CREATE TRIGGER DDLTrigger_SchemaChange

```
ON DATABASE
  FOR CREATE_PROCEDURE, ALTER_PROCEDURE, DROP_
PROCEDURE,
CREATE_TABLE, ALTER_TABLE, DROP_TABLE,
CREATE_FUNCTION, ALTER_FUNCTION, DROP_FUNCTION,
CREATE_INDEX, ALTER_INDEX, DROP_INDEX,
CREATE_VIEW, ALTER_VIEW, DROP_VIEW,
CREATE_TRIGGER, ALTER_TRIGGER, DROP_TRIGGER
AS
BEGIN
SET NOCOUNT ON;
DECLARE
@EventData XML = EVENTDATA();
DECLARE @ip varchar(48) = CONVERT(varchar(48),
CONNECTIONPROPERTY('client_net_address'));
INSERT dbo.DDLEvents
(
EventType, EventDDL, EventXML, DatabaseName, SchemaName,
ObjectName, HostName, IPAddress, ProgramName, LoginName
)
SELECT
@EventData.value('(/EVENT_INSTANCE/EventType)[1]',
'NVARCHAR(100)'),
@EventData.value('(/EVENT_INSTANCE/TSQLCommand)[1]',
'NVARCHAR(MAX)'),
@EventData,
DB_NAME(),
@EventData.value('(/EVENT_INSTANCE/SchemaName)[1]',
'NVARCHAR(255)'),
@EventData.value('(/EVENT_INSTANCE/ObjectName)[1]',
'NVARCHAR(255)'),
HOST_NAME(),
@ip,
PROGRAM_NAME(),
SUSER_SNAME();
END
```

4.4. Аудит изменения привилегий, пользователей (user)/ данных входа в систему (login) и других атрибутов безопасности

Эта категория является обязательной для аудита БД и включает в себя сохранение всех изменений в модели безопасности и привилегиях БД. БД управляет сложной схемой безопасности, разрешениями и изменениями, но правило номер один в области безопасности заключается в том, что изменения в модели безопасности должны подвергаться аудиту. Следует рассмотреть возможность проведения аудита следующих изменений:

– добавление и удаление пользователей, идентификаторов и ролей;

– изменения отношений между идентификаторами и пользователями или ролями;

– изменение привилегий через пользователя или роль;

– смена пароля;

– изменение атрибутов безопасности на уровне сервера, БД или объекта (таблица, столбец, представление).

Поскольку модель безопасности БД является шлюзом к БД, любые изменения привилегий необходимо проверять. Злоумышленники попытаются повысить свой уровень привилегий, и часто совершаются ошибки, когда привилегии предоставляются неадекватно. Аудит любых изменений, которые могут повлиять на безопасность БД, может быть связан с размещением камеры наблюдения, которая отслеживает входную дверь здания, место изменения кода входа и место выдачи бейджей.

Поскольку изменения прав безопасности могут быть опасны для БД (в случае сценария атаки), они не должны быть основаны на типе сравнения, сделанном за определенный период времени (один раз в день/неделю). О незапланированных изменениях в производственной среде должны быть созданы уведомления своевременно, в реальном времени. Это означает использование внешней БД безопасности и аудита или создание предупреждений в реальном времени на основе журналов аудита.

Если этот аудит будет реализован, необходимо зафиксировать соответствующие события, а затем создать систему оповещения. Если используется внешняя система, поддерживающая как аудит в реальном времени, так и оповещения, в политику оповещений можно добавить правила, которые будут информировать об использовании процедур или команд безопасности.

Пример 4.4.1. Для реализации аудита DDL создается таблица *DDLUserAudit*:

```
CREATE TABLE dbo.DDLUserAudit
(
```

```
 Id INT IDENTITY(1,1) NOT NULL,
 CreateDate DATETIME NULL,
 LoginName SYSNAME NULL,
 ComputerName SYSNAME NULL,
 ProgramName NVARCHAR(255),
 DBName SYSNAME NOT NULL,
 SQLEvent SYSNAME NOT NULL,
 SchemaName SYSNAME NULL,
 ObjectName SYSNAME NULL,
 SQLCmd NVARCHAR(max) NULL
);
```

Для этой таблицы будет создан триггер:

```
CREATE TRIGGER [DDLTrigger_UserAudit] ON ALL SERVER
 FOR CREATE_LOGIN, ALTER_LOGIN, DROP_LOGIN,
 CREATE_USER, ALTER_USER, DROP_USER,
 CREATE_ROLE, ALTER_ROLE, DROP_ROLE,
 ADD_ROLE_MEMBER, DROP_ROLE_MEMBER
AS
BEGIN
 DECLARE @EventDataXML XML
 SET @EventDataXML = EVENTDATA();
 INSERT univer.dbo.DDLUserAudit
 (
 CreateDate, LoginName, ComputerName,
ProgramName, DBName, SQLEvent, SchemaName,
 ObjectName, SQLCmd
 )
 SELECT
GETDATE(),
SUSER_NAME(),
HOST_NAME(),
PROGRAM_NAME(),
DB_NAME(),
@EventDataXML.value('(/EVENT_INSTANCE/EventType)[1]',
'SYSNAME'),
@EventDataXML.value('(/EVENT_INSTANCE/SchemaName)[1]',
'SYSNAME'),
```

@EventDataXML.value('(/EVENT_INSTANCE/ObjectName)[1]', 'SYSNAME'),

@EventDataXML.value('(/EVENT_INSTANCE/TSQLCommand)[1]', 'NVARCHAR(MAX)');

END;

Структура таблицы аудита, полученная после создания двух логинов и одного пользователя, выглядит следующим образом:

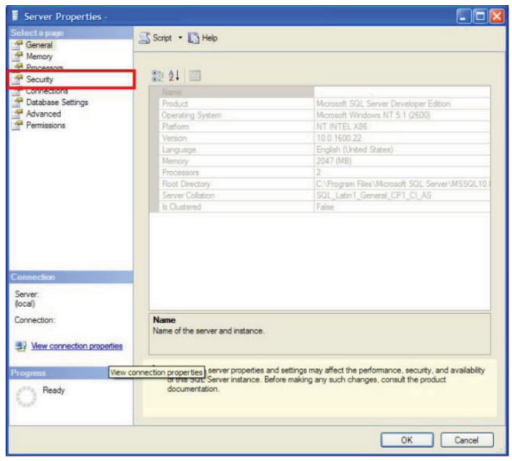
	Id	CreateDate	LoginName	ComputerName	ProgramName	DBName	SQLEvent	SchemaName	ObjectName	SQLCmd
1	4	2020-01-20 18:57:32.603	DESKTOP-NJOTTMJ\user	DESKTOP-NJOTTMJ	Microsoft SQL Server Management Studio - Query	master	CREATE_LOGIN	NULL	i2	create login i2 with PASSWORD = ********
2	5	2020-01-20 18:58:02.600	DESKTOP-NJOTTMJ\user	DESKTOP-NJOTTMJ	Microsoft SQL Server Management Studio - Query	master	CREATE_USER	NULL	i2	create user i2 for login i2
3	3	2020-01-20 18:21:59.050	DESKTOP-NJOTTMJ\user	DESKTOP-NJOTTMJ	Microsoft SQL Server Management Studio - Query	master	CREATE_LOGIN	NULL	i1	CREATE LOGIN i1 WITH PASSWORD = ********

4.5. Аудит ошибок в базе данных и SQL Server

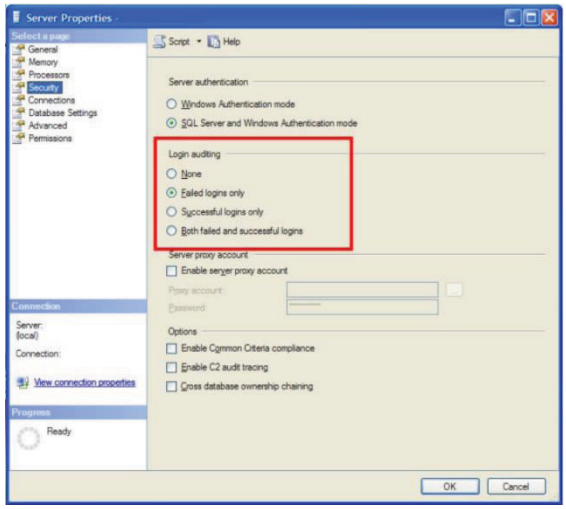
Аудит ошибок, возвращаемых БД, имеет большое значение с точки зрения безопасности и является аудитом, который необходимо внедрить. Когда дело доходит до атак SQL Injection, во многих случаях злоумышленники будут стараться, пока не добьются успеха. В случае атаки на основе UNION, в которой злоумышленники должны угадать правильное количество столбцов, БД будет постоянно возвращать код ошибки с сообщением: *столбцы, выбранные двумя операторами SELECT, не совпадают*. После регистрации всех ошибок эту ситуацию можно идентифицировать и отреагировать соответствующим образом. Неудачное подключение на сервер — еще один пример ошибки, которую необходимо регистрировать и отслеживать даже если аудит БД не проводится. Любая неудачная попытка поднять привилегии является явным признаком возможной атаки.

Аудит ошибок также важен с точки зрения качества. Производственные приложения, которые вызывают ошибки, связанные с ошибками приложений, должны быть идентифицированы и исправлены. В большинстве сред некоторые приложения генерируют базовые ошибки даже в рабочей среде. Однако ошибки приложений повторяются: одни и те же ошибки возникают примерно в одном и том же месте, потому что ошибки обычно возникают из-за ошибок кода – и они не меняются. Если вдруг возникают ошибки откуда-то еще или обнаруживается совсем другой код ошибки, то следует расследовать происходящее.

SQL Server позволяет проверять как успешные, так и неудачные соединения, в зависимости от ситуации существуют разные способы включения этого аудита с помощью SQL Server Management Studio. После подключения к SQL Server в Object Explorer щелкните правой кнопкой мыши на SQL Server и выберите *Properties* во всплывающем меню. Свойства сервера можно увидеть следующим образом:



Перейдите на страницу *Security*, после чего можно будет настроить аудит аутентификации:



Доступны 4 варианта:

- ***None*** — ни успешные, ни неудачные аутентификации не будут проверяться.

- ***Failed logins only*** — неудачные аутентификации будут проверены, а успешные аутентификации будут проигнорированы.

- ***Successful logins only*** — успешные аутентификации будут проверены, а неудачные аутентификации будут проигнорированы.

- ***Both failed and successful logins*** — аутентификация будет проверяться независимо от успеха или неудачи.

После выбора опции нажмите кнопку *OK*. Параметр для аудита аутентификации на самом деле является записью в журнал, которая различается в зависимости от версии SQL Server и имени экземпляра SQL Server. SQL

Server считывает этот параметр только при запуске SQL Server. Чтобы этот параметр вступил в силу, необходимо перезапустить SQL Server.

Для просмотра содержимого журнала SQL Server с помощью T-SQL существует расширенная хранимая процедура *xp_readerrorlog*.

Пример 4.5.1. *EXEC xp_readerrorlog*

Для поиска в текущем журнале ошибок и возврата неудачных аутентификаций вы можете использовать команду *xp_readerrorlog* с двумя параметрами: первый указывает текущий журнал ошибок (0 = текущий), второй параметр указывает тип журнала (1 = журнал ошибок SQL) и третий параметр указывает сообщение для поиска.

Пример 4.5.2. *EXEC sp_readerrorlog 0, 1, 'Login failed'*

Пример результата:

	LogDate	ProcessInfo	Text
1	2020-01-21 20:31:43.720	Logon	Login failed for user 'octav'. Reason: Password did not match that for the login provided. [CLIENT: <local machine>]
2	2020-01-21 20:31:51.660	Logon	Login failed for user '9i8u7'. Reason: Could not find a login matching the name provided. [CLIENT: <local machine>]
3	2020-01-21 20:32:07.860	Logon	Login failed for user 'numele'. Reason: Password did not match that for the login provided. [CLIENT: <local machine>]
4	2020-01-21 20:32:18.750	Logon	Login failed for user 'numele'. Reason: Password did not match that for the login provided. [CLIENT: <local machine>]

В качестве рекомендации следует проверить как минимум неудачные соединения в производственных системах. Могут быть случаи, когда успешную аутентификацию необходимо проверять, но следует понимать, что может быть сгенерировано множество событий, которые будет трудно проанализировать. Очень важно иметь этот уровень контроля над высокочувствительными системами. Однако в менее критичных системах может применяться только неудачная проверка аутентификации.

4.6. Аудит команд SELECT для конфиденциальных данных

В прошлом команды SELECT не были в центре внимания, но недавнее внимание к конфиденциальности данных изменило все. Если необходимо обеспечить соблюдение требований конфиденциальности или клиенты, партнеры и сотрудники должны быть уверены, что их конфиденциальная информация не просачивается из БД, то команды SELECT должны быть проверены. В частности, аудит должен показывать откуда поступил оператор SELECT (IP-адрес, приложение), кто выбрал данные (имя пользователя) и какие данные были выбраны. Как и в случае с журналами аудита DML, аудит команд SELECT нецелесообразен для всей БД и должен быть сосредоточен на подгруппах, которые являются значимыми и необходимыми. Первым шагом в аудите SELECT является классификация важных данных.

Пример 4.6.1. Фамилия не является конфиденциальной, но имя президента, а также номер водительского удостоверения и номер социального страхования являются конфиденциальными.

На этапе классификации необходимо определить, где находится конфиденциальная информация (какие имена объектов и какие имена столбцов) и какая комбинация данных является конфиденциальной. Таким образом, *набор конфиденциальности* представляет собой набор из двух кортежей, каждый из которых состоит из имени объекта и имени столбца.

4.7. Аудит использования базы данных в нерабочее время

Аудит действий, выполняемых в нерабочее время (обычно выполняемые только в отношении таблиц с конфиденциальными данными), являются интуитивным требованием и часто требуются бизнесом с точки зрения соответствия. Требование аудита использования БД в нерабочее время необходимо, поскольку действия в нерабочее время часто вызывают подозрения и могут быть результатом попытки неавторизованного пользователя получить доступ или изменить данные. Конечно, хороший хакер попытается взломать БД в «замаскированный» период.

При проведении аудита обычно недостаточно отслеживать только данные о подключении и записи, которые происходят в нерабочее время. Как правило, также желательно фиксировать выполняемые действия – обычно на уровне SQL. Если такое подключение вызывает подозрения, то важно понимать, что обычно делается в БД. Наличие полного журнала всех действий, выполняемых любым пользователем в нерабочее время, является хорошим критерием для реализации и отвечает многим внутренним нормативным требованиям.

На техническом уровне должно быть понятно, так как большинство СУБД работают круглосуточно и без выходных нежелательно генерировать много ложных срабатываний при управлении данными в нерабочее время (например, ночью часто выполняются тяжелые расчеты данных, это предполагает, что происходит контролируемая обработка данных и не требует сигналов тревоги). Таким образом, ключ к надлежащему проведению этого типа аудита не должен включать действия, которые запланированы для выполнения в нерабочее время.

Если обнаруживается, что какой-то тип активности происходит каждую ночь, то аудит должен исключать любую активность, выполняемую этими приложениями с использованием определенных имен для входа и с ранее известных IP-адресов (или, локальным хостом). Только аудит того, что отличается от основной деятельности СУБД уменьшает размер журна-

лов аудита, которые следует проверять поскольку регистрируются только те действия, которые происходят вне правил.

4.8. Аудит изменений исходного кода хранимых процедур и триггеров

Поскольку в хранимых процедурах и триггерах БД используются гибкие языки программирования, полные процедурных функций, можно легко скрыть преднамеренный вредоносный код, который в противном случае нельзя было бы обнаружить. Следовательно, надо применять надлежащую практику для рассмотрения всех модификаций этих конструкций.

Эта категория может быть проверена несколькими способами:

- Самый простой способ основан на проверке структуры кода и может быть реализован, периодически (например, ежедневно) извлекая код из БД и сравнения его с кодом, извлеченным из предыдущего периода. Этот метод относительно просто реализовать с помощью набора инструментов и скриптов (есть разные репозитории, код с более старой версией можно сравнивать с более новой).

- Другой вариант – использовать внешнюю систему безопасности и аудита, которая может предупреждать о любых командах для создания или изменения процедур и триггеров в режиме реального времени и может создавать набор отчетов с подробным описанием изменений.

Библиография к главе 4

1. H. Afyouni, Database Security and Auditing: Protecting Data Integrity and Accessibility, Cengage Learning, 2006.
2. R. Natan, Implementing Database Security and Auditing, Elsevier, 2005.
3. Microsoft SQL Docs, SQL Server Audit (Database Engine) 2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/relational-databases/security/auditing/sql-server-audit-database-engine?view=sql-server-ver15
4. Microsoft SQL Docs, Create a server audit and database audit specification, 2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/relational-databases/security/auditing/ create-a-server-audit-and-database-audit-specification?view=sql-server-2017
5. CREATE TRIGGER (Transact-SQL), 2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/t-sql/statements/ create-trigger-transact-sql?view=sql-server-ver15

V. УЯЗВИМОСТЬ БАЗЫ ДАННЫХ

5.1. Уязвимость базы данных в интернете

Если не используется программное обеспечение безопасности, все сообщения, отправляемые через Интернет в виде обычного текста, могут быть обнаружены злоумышленниками с помощью программного обеспечения для перехвата пакетов. Как отправители, так и получатели сообщений должны быть убеждены, что их разговоры (сообщения) остаются конфиденциальными. Очевидно, что клиенты, которые хотят покупать товары в Интернете, должны убедиться, что данные их кредитной карты защищены при отправке через Интернет. Предприятия, которые разрешают веб-соединения со своими внутренними сетями для доступа к своей БД, должны иметь возможность защитить ее от атак. Получатели сообщений должны иметь гарантию, что эти сообщения являются подлинными и заслуживающими доверия и не были подделаны. Отправители не должны иметь возможность отвергать их, отрицая что они их послали. Для решения этих проблем используется несколько методов.

5.1.1. Прокси – сервера

Прокси-сервер – это компьютер или программа, которая действует как посредник между клиентом и другим сервером, обрабатывая сообщения в обоих направлениях. Когда клиент запрашивает услугу, такую как соединение или веб-страницу, прокси-сервер оценивает ее и определяет, может ли он сам выполнить запрос. В противном случае он фильтрует запрос, может даже вносить изменения, а затем запрашивает услугу с сервера или другого ресурса. Он может кэшировать (сохранять копию) ответа сервера, чтобы можно было удовлетворить новый запрос, используя сохраненное содержимое, без повторного обращения к серверу. Прокси-сервер можно использовать для обеспечения безопасности путем сокрытия реального IP-адреса сервера, для повышения производительности кэширования, для предотвращения доступа к сайтам, которые компания хочет заблокировать для своих участников, для защиты сервера от вредоносных программ и для защиты данных путем сканирования исходящих данных.

5.1.2. Межсетевой экран – Firewall

Межсетевой экран – это аппаратный и/или программный барьер, который используется для защиты внутренней сети компании (интранета) от несанкционированного доступа. Для обеспечения того, чтобы входящие и исходящие сообщения из интрасети соответствовали стандартам компании,

используются различные методы. Например, прокси-сервер можно использовать для сокрытия фактического сетевого адреса. Другим методом является пакетный фильтр, который проверяет каждый пакет информации перед его входом или выходом из интрасети, чтобы убедиться, что он соответствует набору правил. Можно применять несколько механизмов безопасности приложений или соединений.

5.1.3. Цифровые подписи

Цифровые подписи – это криптографические алгоритмы с открытым ключом, которые используются для демонстрации подлинности сообщения и источника, а также для обеспечения целостности сообщений и неотказуемости источника. Фактически цифровая подпись является результатом шифрования *сущности сообщения* используя закрытый ключ отправителя. Эта сущность сообщения представляет собой значение, полученное в результате применения криптографического алгоритма хеширования к подписываемому сообщению. Для проверки подписи получатель сравнивает хеш-значение, полученное двумя способами: один из них вычисляется тем же способом, что и отправитель перед подписанием, с использованием того же хеш-алгоритма, второе значение будет получено путем расшифрования подписи используя открытый ключ отправителя. Подпись считается валидной, только в том случае, если эти два значения равны.

5.1.4. Центры сертификации

Клиенты, желающие приобрести товары на сайте электронной коммерции, должны делать это с уверенностью в том, что сайт, с которым они общаются, является подлинным и что информация об их заказе передается конфиденциально. Широко используемый метод проверки подлинности сайта – через центры сертификации (например Verisign). В процессе используется шифрование с открытым ключом. Сайт начинает процесс сертификации, создавая открытый и закрытый ключи и отправляя запрос в центр сертификации вместе с открытым ключом сайта. Центр сертификации генерирует шифрованный сертификат сайта, который сохраняется для последующего использования. Когда клиент хочет разместить заказ, используя защищенное соединение с сайтом, браузер запрашивает у сайта сертификат, и получает его в шифрованном виде. Браузер расшифровывает сертификат, используя открытый ключ центра сертификации, и проверяет, действительно ли это сертификат центра сертификации и правильность URL-адреса сайта. Сертификат также содержит открытый ключ сайта. Браузер создает сеансовый ключ, которого шифрует с помощью общедоступного ключа

сайта в сертификате, и отправляет сеансовый ключ на сайт. Во время сеанса ключ шифруется открытым ключом сайта. Только настоящий сайт может расшифровать его, используя свой закрытый ключ. И браузер, и сайт являются единственными владельцами сеансового ключа и могут обмениваться шифрованными сообщениями с сеансовым ключом, используя протокол. Описанный процесс используется в протоколе Secure Sockets Layer (SSL) и обычно используется для сообщений клиенту и от клиента в процессе заказа. Для передачи номера кредитной карты обычно используется дополнительная мера безопасности. В то время как пользователь браузера отправляет на сайт продавца дополнительную информацию о заказе, шифрованную его открытым ключом, когда клиент готов отправить информацию о кредитной карте в конце заказа, эта информация вместе с суммой, подлежащей списанию, отправляется непосредственно на веб-сайт компании карты для авторизации и утверждения.

Kerberos – это сетевой протокол проверки подлинности, обеспечивающий взаимную проверку подлинности, при которой и клиент, и сервер могут подтвердить подлинность. В качестве центра сертификации используется доверенный сервер Kerberos. Он имеет центр распространения ключей, в котором хранятся секретные ключи каждого клиента и сервера в сети и которые используются в качестве входных данных для создания билетов с отметками времени, когда клиент запрашивает услугу. Затем билет используется для демонстрации серверу того, что клиент одобрен для службы. Сообщения могут быть шифрованы с использованием любого симметричного ключа или протокола с открытым ключом.

5.2. SQL Injection

SQL Injection – очень распространенная уязвимость в веб-приложениях, использующих БД. Многие программисты не знают, что SQL-запросы могут быть изменены извне, и считают эти команды безопасными. Это означает, что любая атака SQL переопределит любые существующие меры контроля и безопасности на сайте, а также любой используемый метод аутентификации.

Direct SQL Command Injection – это метод, с помощью которого злоумышленник создает или меняет существующий SQL-запрос для получения конфиденциальной информации, перезаписи информации или даже для создания новых пользователей. SQL-запросы выполняются путем ввода SQL-команд во входные данные сайта для достижения намеченной цели. Большинство веб-приложений хранят данные в БД, поскольку это позволяет создавать динамические страницы. Веб-приложение получает от пользо-

вателей данные, которые используются приложением/скриптом для формирования запроса к БД.

Определение 5.1. *SQL Injection* – это уязвимость, возникающая, когда пользователи получают данные, которые не обрабатываются должным образом. Следовательно, потенциальный преступник может изменить запрос БД, тем самым похитив личные данные.

Приемы, которые используются преступниками в процессе эксплуатации уязвимости SQL Injection, демонстрируют, как легко получить контент БД, приватные данные, выполнить DoS-атаку, получить максимальные привилегии и т.д. SQL Injection использует тот факт, что SQL-запросы могут быть динамически встроены в код приложения.

Пример 5.2.1. Веб-форма, которая позволяет студенту ввести свой идентификатор и пароль в переменные *user_id* и *password*. Затем веб-сайт использует эту информацию для извлечения конфиденциальной информации учащегося из таблицы *Student*. В коде приложения запрос может быть построен динамически следующим образом:

*Student_query = “SELECT * FROM Student*
WHERE user_id = ' ” + user_id + “ ‘ AND password = ‘ ” + password +” ‘ ; ”

Символ + является оператором конкатенации строк. Если *user_id* содержит значение *Andrei*, а пароль содержит значение *pass123*, то *Student_query* будет содержать следующее:

*SELECT * FROM Student*
WHERE user_id = 'Andrei’ AND password = ‘pass123’

Student_query отправляется в БД для извлечения информации. Этот запрос работает по назначению только в том случае, если введенное значение для *user_id* и/или *password* не содержит ни одного символа в кавычках. Например, если пользователь случайно введет *pass'123*, запрос станет таким:

*SELECT * FROM Student*
WHERE user_id = 'Andrei' AND password = 'pass'123'

Кавычки в пароле заставят SQL генерировать сообщение об ошибке *'pass'123'; is invalid syntax in the SQL parse*. Злоумышленник может использовать этот метод, чтобы выяснить, уязвима ли БД для атаки SQL Injection, где сообщение об ошибке синтаксиса указывает злоумышленнику на то, что запрос создается динамически без какой-либо проверки входных данных. Злонамеренный запрос дает злоумышленнику доступ к информации *Andrei*, а также к информации всех других студентов.

Если предположить, что злоумышленник не знает никакого действительного *user_id* или *password*, и вводит значение *XXX* в качестве *user_id* и *YYY' OR '1'='1* в качестве *password*, запрос становится:

*SELECT * FROM Student*
WHERE user_id = 'XXX' AND password = 'YYY' OR '1'='1';

Даже при неправильном *user_id* и *password* этот запрос всегда оценивается как истина, потому что условие всегда будет выполняться. Запрос оценивается как следующий запрос, который возвращает информацию обо всех студентах:

*SELECT * FROM Student WHERE '1'='1';*

Пример 5.2.2. Поскольку некоторые СУБД допускают выполнение нескольких операторов SQL разделенных точкой с запятой в одном запросе, злоумышленники также могут воспользоваться этой уязвимостью наряду с простыми кавычками во входных данных для ввода дополнительных вредоносных операторов. Для иллюстрации этого типа атаки предположим, что злоумышленник вводит значение *XXX* в качестве *user_id* и значение *YYY' OR '1'='1'; DELETE * FROM Student; --* в качестве *password*. В этом случае запрос становится:

*SELECT * FROM Student*
WHERE user_id = 'XXX' AND password = 'YYY' OR '1'='1';
*DELETE * FROM Student;*
--';

Запрос извлекает всю информацию о студенте, а затем удаляет всю информацию из таблицы *Student*. Символы комментария « -- » используются в конце строки пароля чтобы любые другие символы были закомментированы вне выполнения запроса и избежать синтаксической ошибки.

Пример 5.2.3. Запрос, который извлекает данные из БД, может иметь форму:

*SELECT * FROM* Student *WHERE name = '$name'*

В этом случае значения в поле «*name*» сравниваются со значением «*$name*». Если значение получено из параметров URL, поля *imput* формы или файла cookie и не обработано для специальных символов, то запрос в БД уязвим и *злоумышленник может изменить запрос*. Если с веб-страницы (приложения) требуется определить данные о студенте по определенному *id,* который вводится пользователем. Для этого объявляется строковая переменная *string*, содержащая оператор SELECT:

*sql = 'SELECT * FROM student WHERE id =' + Text1.Value*

Если злоумышленник добавляет в поле ввода рядом с *id* код типа *1;SELECT name FROM sys.tables*, то на сервер приходит запрос на выполнение последовательности:

*SELECT * FROM Student WHERE id = 1;SELECT name FROM sys.tables*

в результате предоставляются данные студента, но также отображается список таблиц БД.

Пример 5.2.4. В случае, если злоумышленник добавит в поле ввода рядом с *id* и код *1;Select * FROM Facultate* то на сервер приходит запрос на выполнение последовательности:

*SELECT * FROM Student WHERE id* =1;Select * FROM Facultate

в результате будут предоставлены данные о студентах и отобразятся данные из таблицы *Facultate*.

Пример 5.2.5. В случае, если злоумышленник добавит в поле ввода рядом с *id* и код *1;DELETE FROM Facultate*, то на сервер придет запрос на выполнение последовательности:

*SELECT * FROM Student WHERE id = 1;DELETE FROM Facultate*

в результате предоставляются данные о студентах, а данные из таблицы *Facultate* удаляются.

Пример 5.2.6. Если с веб-страницы требуется определить данные о студенте по *nume*, где имя вводится пользователем, объявляется строковая переменная *string*, содержащая оператор SELECT:

*sql = 'SELECT * FROM Student WHERE name = ' ' + Text2.Value + ' ' '*

в этом случае злоумышленник, добавив в поле ввода рядом с *id* и код tipul *Ion' ';SELECT name FROM sys.tables WHERE ' '1' '=' '1* на сервер приходит запрос на выполнение последовательности:

*SELECT * FROM Student WHERE name = ' Ion' ;SELECT name*
FROM sys.tables WHERE '1'='1'

в результате предоставляются данные студентов, но также отображается список таблиц БД. В этом случае у нас есть SQL injection в поле *string*. Применяя те же манипуляции, мы можем получать, изменять или удалять данные из таблиц.

Процедуры тестирования веб-приложений с SQL injection сводятся к созданию списка параметров, с которыми работает приложение (как GET, так и POST), включая параметры cookie. Затем эти параметры проверяются по отдельности при обработке специальных символов или ключевых слов (таких как WHERE), которые помогают использовать уязвимость.

Действия по атаке веб-приложения с целью получения несанкционированной информации из БД:

1) выявление уязвимых параметров, в случае если веб-приложение настроено таким образом, что текст ошибки SQL и возможно часть запроса появляются в браузере;

2) определение типа атрибутов и количества атрибутов (столбцов атакуемых таблиц);

3) сама атака с применением известного метода.

Так как злоумышленник находит уязвимый параметр для использования уязвимости, он должен примерно знать тип SQL запроса, в котором пробуется вредоносный код. Чаще всего в веб-приложениях используются 4 типа SQL запросов: *SELECT*, *INSERT*, *UPDATE*, *DELETE*. Из логического и семантического анализа уязвимого скрипта можно определить тип запроса, применяемого в конкретном случае.

5.2.1. Методы атаки с помощью SQL injection

- Если скрипт выводит данные, соответствующие определенному идентификатору, то скорее всего запрос *SELECT*.

- Если скрипт добавляет какие-то данные в БД, например добавление комментария или сообщения на форуме —запрос *INSERT*.

- Если скрипт меняет информацию, например смена пароля, редактирование сообщения на форуме — запрос относится к типу *UPDATE*.

- Если информация удаляется, например, аннулирование учетной записи, запрос может быть либо *DELETE*, либо *UPDATE*.

Для выполнения SQL injection можно использовать следующие методы:

- оператор SQL Union позволяет объединить два запроса в один результат, используемый в операторе SELECT;

- *boolean*, использование логических условий чтобы проверить какие условия являются истинными или ложными;

- *ошибки*, метод заставляет БД генерировать ошибку, которая может предоставить злоумышленнику дополнительную информацию для уточнения запроса в момент атаки;

- *время*, использование команд БД для удерживания ответов в запросах.

5.2.2. Рекомендации по предотвращению SQL injection

✓ Тщательно защитить данные, введенные пользователем. Любые данные, полученные от пользователя, должны считаться небезопасными. Устранение всех одинарных и двойных кавычек может предотвратить большинство типов этой атаки.

✓ Пишите все SQL запросы, используя параметризованные запросы, чтобы запрос был предопределен, а введенные данные проверялись и передавались позже.

✓ Ограничьте привилегии пользователя, которого используете для подключения, до минимума.

✓ Необходимо проверить, соответствуют ли входные данные от пользователей тому, что ожидаете. Используйте регулярные выражения (regexp).

✓ Любые данные, введенные пользователем, нужно преобразовывать в ожидаемый вид.

✓ Количество символов в пользовательском вводе должно быть ограничено, хотя это не предотвратит все атаки SQL, но остановит небольшую их часть.

✓ Не рекомендуется вывод на экран сообщения об ошибках SQL (есть риск того, что злоумышленник узнает структуру БД из этих сообщений).

✓ Одним из способов предотвращения инъекций является избавление от символов, имеющих особое значение в SQL.

- в большинстве случаев код SQL вставляется как в одинарные, так и в двойные кавычки (‘,“, `);
- с помощью специальных символов комментария СУБД (/ *, -) часть запроса можно опустить;
- символы, разделяющие операторы SQL (;) позволяют формировать несколько запросов к БД;
- данные введенные пользователем должны быть проверены на наличие других символов, таких как _,%, *,.

Следует отметить, что уязвимости SQL Injection очень распространены, поскольку позволяют хакеру узнать/извлечь информацию о сервере или даже провести DoS-атаку.

Во избежание наличия данной уязвимости необходимо обрабатывать абсолютно все данные, поступающие от пользователей со специальными символами. В эту категорию входят параметры GET, POST и даже cookie. Каждая СУБД имеет свои нюансы, которые потенциальный преступник может использовать в своих интересах.

Библиография к главе 5

1. A. Watt, N. Eng, *Database Design*, 2014.
2. Microsoft SQL Docs, Vulnerability assessment for SQL Server2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/relational-databases/security/sql-vulnerability-assessment?view=sql-server-2017

VI. МЕТОДЫ БЕЗОПАСНОСТИ ДАННЫХ. РИСКИ И ПРОЦЕДУРЫ ВОССТАНОВЛЕНИЯ ДАННЫХ

Одной из проблем, с которыми сталкиваются компании, является увеличение объема данных, которые необходимо сохранять и/или извлекать в случае инцидента, вызванного внутренними или внешними причинами. Хорошо известно, что планирование непрерывности бизнеса в случае стихийного бедствия является жизненно важной деятельностью. Перед созданием плана обеспечения непрерывности необходимо рассмотреть возможные последствия стихийных бедствий и связанные с ними риски. Сам план должен поддерживаться, тестироваться и проверяться, чтобы гарантировать, что он соответствует потребностям бизнеса. Доступность данных должна быть максимально высокой и обеспечивать непрерывность бизнеса даже в случае повреждения или потери производственных данных.

Для любого плана восстановления в случае инцидента необходимо указать два эталонных значения:

1. RPO - Recovery Point Objective

– представляет собой максимальное количество информации, которое допустимо к потере, выраженное в единицах времени (при копировании БД в 10:00 каждые два часа допускается потеря информации, накопленной в течение 2 часов);

2. RTO - Recovery Time Objective

– представляет собой максимальное время, прошедшее с момента возникновения инцидента до момента, когда система снова станет работоспособной (с момента падения БД до момента ее подъема);

– при превышении этой цели деятельность начинает сильно страдать.

6.1. Резервные копии базы данных

Важную роль в деятельности администратора БД играет разработка и реализация стратегии, позволяющей максимально эффективно сохранять, архивировать и восстанавливать данные.

Хорошая стратегия резервного копирования и восстановления БД может предотвратить:

– потерю данных при невозможности доступа к БД из-за повреждения файла или группы файлов;

– потерю некоторых специфических функции при переходе на новые версии приложений с использованием БД или новой версии БД;

– потерю данных при следующих операциях: импорт данных, появление новых функций в приложении, ошибки в работе пользователей;

– потеря БД в случае недоступности сервера в связи с определенными

внешними факторами: перебои в подаче электроэнергии, пожары, наводнения.

6.1.1. Разработка плана резервного копирования и восстановление базы данных

При создании плана резервного копирования учитываются:

1. Тип БД. Системные и пользовательские БД обычно имеют разные требования как к резервному копированию так и к восстановлению. Например, БД master необходима для выполнения всех операций на SQL Server. Если она выйдет из строя или станет недоступна сервер не будет работать. Однако нет необходимости выполнять резервное копирование этой БД каждый час или полчаса, как в случае с БД, которая обрабатывает сложные транзакции клиентов в режиме реального времени. Основное резервное копирование БД выполняется (обычно раз в неделю – полное резервное копирование) только при создании новой БД, создании новых учетных записей пользователей или выполнении операций, вызывающих серьезные изменения в БД.

2. Важность БД. Важность БД обуславливает выбор типа резервного копирования.

3. Периодичность внесения изменений в БД. Доступная для просмотра БД требует меньше резервных копий, чем редактируемая БД.

4. Скорость восстановления данных. Резервные копии больших БД (с массовыми изменениями) будут создаваться инкрементально за более короткий промежуток времени. Например, каждый час, чтобы в случае сбоя нужно было восстанавливать меньший объем информации и, соответственно, быстрее восстанавливать БД.

5. Наличие адекватного резервного оборудования. Аппаратное обеспечение, необходимое для резервного копирования, может включать: оптические накопители, внешние накопители, HDD, SSD.

6. Планирование резервного копирования. Резервное копирование обычно выполняется, когда нагрузка на систему минимальна, что увеличивает скорость резервного копирования.

7. Хранение резервных копий в местах, отличных от производственных. Такая политика желательна во избежание потери данных в случае стихийных бедствий (серверы данных БД могут храниться в другом городе, стране).

Резервная копия – это копия данных в БД, которая может быть использована для восстановления этих данных. Резервные копии можно разделить на:

- ***физические резервные копии*** – это резервные копии физических файлов, используемых для хранения и восстановления БД, таких как файлы данных, управляющие файлы и архивные журналы, которые хранят информацию БД в другом месте, на диске, offline;

- ***логические резервные копии*** содержат логические данные (например, сохраненные таблицы или процедуры), экспортированные из БД с помощью утилиты экспорта и сохраненные в двоичном файле.

Физическое восстановление является основой любой стратегии копирования и восстановления данных. Логические резервные копии во многих случаях являются полезным дополнением к физическим резервным копиям, но их недостаточно для защиты от потери данных без физического резервного копирования. Чтобы сделать резервную копию части или всей БД необходимо сделать физическую копию.

Технологии, которые необходимо учитывать при обосновании стратегии резервного копирования:

- ***On-Premise*** (локальное резервное копирование данных) выполняется на локальном или удаленном (вне площадки) оборудовании. Необходимо учитывать затраты, связанные с увеличением объемов закупок/управления, масштабируемостью оборудования/систем, программного/аппаратного обеспечения и затраты на электроэнергию.

- ***Cloud Backup*** (облачное резервное копирование) сохранение и архивирование выполняются непосредственно в облаке автоматически и по расписанию. Эти сервисы должны работать в фоновом режиме, при этом облачное хранилище может быть расширением локального хранилища, и они будут управляться вместе. Зачастую это экономически более выгодно, чем локальное резервное копирование.

Для восстановления данных, в зависимости от типа хранилища и его местоположения, многие компании хранят их вне офиса или в облаке по той простой причине, что восстановление занимает гораздо меньше времени, чем локальное резервное копирование. Чем короче время восстановления данных и/или возврата к работе, тем выше затраты.

Применение стратегии на практике и ее адаптация к бизнес-требованиям (например, автоматизация процесса резервного копирования или процесса восстановления данных с использованием резервного центра обработки данных) требует тестирования и проверки правильности процессов резервного копирования. Чтобы иметь 100% безопасную резервную копию, доступную в любое время, она должна быть автоматически проверена, чтобы гарантировать, что любой сохраненный блок данных является точной копией оригинала, даже если это связано с большими затратами. Тестирование процесса включает в себя моделирование ситуаций простоя, даже если во

время тестирования данные подвержены определенной степени риска. Имея чисто теоретическую стратегию, не проверенную в реальных условиях, в момент инцидента компания может столкнуться с уникальной ситуацией, а способ реагирования на инцидент иногда может быть неадекватным.

6.1.2. Типы резервных копий

Резервную копию БД можно сделать как при включенной БД, так и при выключенной. Каждая ситуация имеет свои преимущества и недостатки.

1. Резервное копирование с включенной БД:

– рекомендуется для использования в производственной среде, обеспечивает минимальную потерю данных;

– не требует остановки БД, на пользователей эта операция не влияет;

– необходимо контролировать доступное место на диске, если места, зарезервированного для операции резервного копирования, недостаточно, его заполнение может привести к отключению сервера.

2. Резервное копирование с отключенной БД:

– БД необходимо остановить;

– пользователи не могут подключиться к БД;

– во время этой операции прерываются все активные соединения.

Существует широкий спектр методов резервного копирования, таких как полное, инкрементное, дифференциальное (и опционально полное синтетическое, постоянное инкрементное или инвертированное инкрементное), все со своими преимуществами и недостатками. Самыми известными методами защиты данных остаются защита в реальном времени и репликация в реальном времени. Однако, несмотря на множество инновационных решений, до сих пор широко используется решение, при котором данные просто копируются через определенные промежутки времени в места, отличные от основного (можно сделать зеркало БД в реальном времени на другом диске).

Полное резервное копирование — это базовый метод резервного копирования данных, при котором они копируются полностью, включая объекты БД, системные таблицы и данные, включая части журнала транзакций, необходимые для процесса восстановления, в указанный набор данных (в виде файла). Это режим резервного копирования, который требует больше всего времени для хранения данных, но также требует большой емкости хранилища. Преимуществом этой методики является быстрое и простое восстановление данных из полной резервной копии.

Инкрементное резервное копирование предполагает наличие полного резервного копирования, выполняемого, например один раз в неделю, тогда

как инкрементный режим копирует только измененные данные по истечении времени полного резервного копирования. Преимуществом является малое потребление места и времени, но есть недостаток сложность восстановления сохраненных данных. В случае необходимости, все инкрементные копии должны быть объединены с существующей полной копией.

Пример 6.1.1. В воскресенье создается полная резервная копия. В понедельник сохраняются изменения, сделанные в понедельник, во вторник сохраняются изменения только вторника и так далее. Это дает нам файлы для каждого дня. Еще одна полная резервная копия делается в следующее воскресенье, а те файлы, что были созданы в течение недели, удаляются. Для восстановления поврежденной БД в среду, пока в среду не была создана инкрементная резервная копия, восстанавливается полная резервная копия, затем инкремент понедельника и потом вторника.

Дифференциальное резервное копирование похоже на инкрементное резервное копирование, при котором ежедневно создаются копии данных, измененных после дня создания полного резервного копирования. Отличие от инкрементного в том, что данные обновляются ежедневно. На случай необходимости выполняется полное резервное копирование и одно дифференциальное резервное копирование. Преимуществом является скорость, с которой может быть выполнена, но время и рабочая нагрузка могут быть увеличены по сравнению с инкрементным режимом.

Пример 6.1.2. В воскресенье создается полная резервная копия. В понедельник сохраняются изменения понедельника, во вторник к изменениям понедельника добавляются изменения вторника и так далее, и получается один файл. В воскресенье делается еще одно полное резервное копирование и дифференциальное резервное копирование, полученное в течение недели, удаляется. Для восстановления БД, поврежденного в среду до создания дифференциальной резервной копии, восстанавливается полная резервная копия, а затем дифференциальная резервная копия, полученная во вторник.

Резервное копирование журнала транзакций. Журнал транзакций представляет собой последовательную запись всех изменений, внесенных в БД, и используется при выполнении операций восстановления данных для закрытия или отмены транзакции.

Журналы транзакций необходимы при восстановлении БД в SQL Server. В отличие от полных или дифференциальных резервных копий, резервные копии журнала транзакций обычно являются добавочными, что означает, что каждая копия журнала транзакций имеет журнал транзакций только в определенном интервале времени транзакции. Журналы транзакций всегда применяются последовательно после окончания последней полной или

дифференциальной резервной копии предыдущей транзакции. Кроме того, если БД подлежит восстановлению, каждая последовательность должна использоваться после момента сбоя.

Пример 6.1.3. Если полная резервная копия сделана в 1:00, а БД выходит из строя в 1:46, то БД восстанавливается на основе последней полной резервной копии, после чего используются резервные копии журналов транзакций созданные в 1:15, 1:30 и 1:45. Так, если бы не использовались последовательные резервные копии журналов транзакций, все транзакции, совершенные после 1:00, когда была сделана полная резервная копия БД, были бы потеряны. При восстановлении БД SQL Server предоставляет некоторые параметры, предотвращающие последовательное использование журналов транзакций. Время окончания последней полной или дифференциальной резервной копии отмечает начало использования последовательности копирования журналов транзакций. Если использование резервного журнала транзакций признано недействительным, необходимо создать полную или дифференциальную копию БД, чтобы начать новую последовательность.

Резервное копирование файлов и групп файлов. В этом случае копируются только файлы БД, а не вся БД. Вариант полезен при работе с очень большими БД. Однако в этом случае необходимо сделать резервную копию журнала транзакций (если данные большие, то их можно хранить сгруппированными по годам или месяцам, и в этом случае резервные копии делаются только для данных за определенный период).

6.1.3. Типы шаблонов восстановления базы данных

1. Простой. Предназначен для восстановления БД с момента последней резервной копии. Стратегия резервного копирования для этой модели включает создание полных и дифференциальных резервных копий. Резервные копии журнала транзакций не создаются.

2. Полный. Рекомендуется для БД, которые необходимо восстановить с момента последнего сбоя или с определенного момента времени. Все операции регистрируются, в том числе операции загрузки данных. Стратегия включает создание полных резервных копий, дифференциальных резервных копий и резервных копий журнала транзакций или только журнала транзакций.

3. Регистрируются данные о загрузке. Этот шаблон уменьшает пространство, используемое журналом. Операции загрузки данных протоколируются до минимума, и требуется возобновление операций загрузки если БД дает сбой до выполнения полного или дифференциального резервного копирования.

У каждой БД может быть своя модель восстановления. По умолчанию БД master, msdb и tempdb используют *простую* модель, а БД model (которая является шаблоном для всех новых БД) использует модель *полного* восстановления.

6.2. Повреждение базы данных

Во избежание повреждения БД администратор БД должен обеспечить ее оптимальную работу, что влечет за собой выполнение следующих задач:

Обеспечить ее надлежащее функционирование, а именно:

– оптимальная загрузка ресурсов сервера: процессор не должен быть занят 80 - 90% времени, память не должна быть занята более 90%, должно быть достаточно места для увеличения БД не менее чем на 25%;

– максимально возможное время отклика БД на текущую деятельность: введение документов, финансовых отчетов, балансов и т.п.;

– диагностика и устранение инцидентов в БД в кратчайшие сроки, чтобы пользователи как можно меньше пострадали.

Создание и внедрение надежной стратегии резервного копирования:

– восстановить работоспособность в кратчайшие сроки и с минимальными потерями БД в случае повреждения файлов, случайного удаления данных, утери БД и т.д.;

– планировать выполнение заданий резервного копирования и обслуживания БД таким образом, чтобы их выполнение как можно меньше влияло на пользователей;

– периодически проверить если резервная копия БД может быть восстановлена без ошибок в тестовой среде;

– убедиться, что носитель данных защищен, обеспечена избыточность данных и имеется достаточно места.

Периодически проверить, чтобы в логах БД не было инцидентов и ошибок:

– если позволяет инфраструктура, создавать уведомления по электронной почте в случае возникновения ошибок на уровне БД;

– периодически проверять журнал БД (какие события произошли, например: диск недоступен);

– в случае появления сообщений об ошибках отправить эти инциденты службе поддержки и следовать полученным инструкциям.

Постоянно контролировать показатели работоспособности БД:

– отслеживать полученные оповещения (например, относительно свободного места для выполнения резервных копий);

– максимально предвидеть рост БД и не допускать ситуаций, при которых на работе сервера может повлиять нехватка ресурсов: места на диске, выделенной памяти, загрузки ЦП или влияние других приложений на БД;

– создавать и постоянно контролировать выполнение заданий обслуживания БД, таких как задание обновления статистики, дефрагментация БД, степень загрузки файлов БД.

Во избежание повреждения БД могут быть приняты следующие меры предосторожности:

1. Отслеживание активности на БД и степени ее загрузки, выявление в режиме реального времени блокировок, которые могут возникнуть и устранение причин.

2. Предугадывать рост БД, выделить вовремя необходимое пространство, чтобы не было блокировок на уровне БД по этой причине.

3. Статистические данные обновляются только в том случае, если объем данных очень быстро увеличивается за очень короткое время или если они уже не актуальны. Ежедневное обновление статистики для БД при умеренном увеличении объема данных не оправдано. В этом случае рекомендуется обновлять их еженедельно.

4. Создание индексов в соответствии со спецификой каждой БД. Теоретически для БД хорошо иметь индекс для каждого внешнего ключа. Это не общее правило, индексы нужно вводить постепенно и проверять их влияние на БД.

5. Разделение данных в соответствии с их важностью и способом доступа к ним. Если БД использует очень большую таблицу, было бы идеально хранить эту таблицу на RAID (см. Приложение 1) отдельно от других данных. Теоретически индексирование на отдельном носителе обеспечивает лучшее время отклика БД.

6.3. Методы восстановления базы данных

Разработка стратегии резервного копирования и/или аварийного восстановления требует тщательного анализа затрат и выгод для бизнеса, нуждающегося в таких услугах. Восстановление БД — это процесс возврата БД в правильное состояние после возникновения сбоя. Причинами сбоев являются: системные сбои (hard или soft), экологические сбои (разрушение среды хранения данных), ошибки программных приложений, стихийные бедствия, халатность и саботаж.

Вне зависимости от причины, есть два основных следствия:

1) потеря основной памяти, в том числе буферов BD;

2) потеря дисковой копии BD.

Возможны две ситуации, которые зависят от степени повреждения БД:

- если БД была физически повреждена (упал диск, на котором хранится БД), необходимо восстановить последнюю резервную копию и повторно

применить операции обновления транзакций, выполненные с использованием журнала транзакций. Рекомендуется хранить журнал на отдельном диске, чтобы снизить риск повреждения одновременно с повреждением БД;

- если БД не была физически повреждена, но стала не консистентной из-за сбоя системы во время транзакции, резервное копирование не требуется, но можно восстановить, используя предыдущие и последующие образы в журнале транзакций.

Техники восстановления базы данных:

1. Техника восстановления с помощью отложенного обновления. При использовании этого протокола обновления не записываются в БД до тех пор, пока транзакция не достигнет точки завершения. Если транзакция терпит неудачу до достижения этой точки, она не меняет БД, и, следовательно, не требуется никаких отмен или изменений. Может потребоваться только возобновление обновлений транзакций, так как их действие могло не дойти до БД. Для этого используется журнал транзакций.

2. Техника восстановления с помощью немедленного обновления. При использовании этого протокола обновления применяются к БД, как только они происходят, не дожидаясь завершения транзакции. Помимо возобновления обновлений, теперь может потребоваться отмена эффектов транзакций, которые не были совершены в момент сбоя. Для этого используется журнал транзакций. Важно, чтобы записи журнала были написаны до соответствующего описания в БД. Эта операция известна как протокол записи журнала.

3. Затенение пагинации. Это альтернатива методам, использующим журнал транзакций. В этом методе во время транзакции сохраняются две таблицы страниц: одна для текущей страницы и одна для теневой страницы. В начале транзакции две страницы идентичны. Теневая страница никогда не меняется во время транзакции, в то время как текущая страница отражает все обновления, внесенные в БД этой транзакцией. В конце транзакции текущая таблица страниц становится теневой таблицей страниц.

Использование методов восстановления БД зависит от параметров, использованных для создания резервной копии и состояния БД.

Библиография к главе 6

1. A. Basta, M. Zgola, *Database Security*, Cengage Learning, 2011.
2. Microsoft SQL Docs, Restore a Database Backup Using SSMS 2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/restore-a-database-backup-using-ssms?view=sql-server-ver15

VII. ПРОБЛЕМА ВМЕШАТЕЛЬСТВА И КОНКУРЕНЦИИ В БАЗЫ ДАННЫХ

7.1. Транзакции и свойства АСИД

Основная цель СУБД — предоставить множеству пользователей одновременный доступ к общим данным. Но в системах, где доступ к БД осуществляется одновременно несколькими пользователями, возникают конфликтные ситуации из-за одновременного доступа к данным, являющимся общим ресурсом. Способ разрешения конфликта зависит от характера запросов на доступ к данным:

1. Если запросы на доступ относятся к типу просмотра (извлечения), то достаточно последовательного доступа к носителю данных и никаких дополнительных мер предосторожности не требуется. Пока никто не вносит изменения в данные, порядок в котором пользователь имеет доступ к данным, не имеет значения. Поэтому в этом случае разрешение конфликтных ситуаций, вызванных одновременными операциями чтения, можно оставить на операционную систему, которая устанавливает порядок удовлетворения запросов по критерию обеспечения максимальной операционной конкуренции, минимизируя общее время удовлетворения этих запросов.

2. Если некоторые запросы относятся к типу обновления (данные изменяются), необходимо применять соответствующие стратегии обработки запросов на доступ. Типичными примерами являются системы бронирования отелей или авиабилетов, в которых несколько агентов постоянно вносят изменения. В этом случае существует опасность, что два процесса, которые считывают и меняют значение одного и того же объекта, могут взаимодействовать при одновременном выполнении. Таким образом, необходимо наложить ограничения на одновременное выполнение процессов, выполняющих операции чтения и изменения данных. Немедленным и простым решением будет блокировка БД на время обработки запроса. Но это равносильно блокированию конкуренции и снижению производительности системы, иногда делая ее непригодной для использования.

Определение 7.1. *Транзакция* — это один или несколько операторов SQL, составляющих единую логическую единицу работы.

Определение 7.2. *Транзакция* — это логическая единица неделимой (атомарной) обработки данных БД, обеспечивающая ее непротиворечивость.

Транзакция может быть завершена двумя способами:

✓ *транзакция завершилась успешно* (тогда мы говорим, что транзакция была зафиксирована) и в этом случае эффекты транзакции становятся постоянными в БД;

✓ *транзакция не была успешно завершена*, тогда последствия транзакции удаляются из БД (в этом случае транзакция откатывает то, что предполагается откатить) и транзакция завершается. Устранение последствий транзакции называется отменой транзакции.

Инструкции BEGIN TRANSACTION, COMMIT и ROLLBACK доступны для разграничения транзакций в большинстве языков обработки данных. Если пользователь не использует эти ограничения, вся программа обычно считается одной транзакцией, и СУБД автоматически выполняет оператор COMMIT, если программа завершена правильно, или ROLLBACK в противном случае. Таким образом, у нас есть:

- *BEGIN TRANSACTION* – начать новую транзакцию;

- *COMMIT* – сохраняет любые изменения и завершает текущую транзакцию;

- ROLLBACK — отменяет любые изменения, сделанные в ходе текущей транзакции, и закрывает/завершает транзакцию.

Если используется BEGIN TRANSACTION, то любое обновление данных не будет выполняться мгновенно, а только после COMMIT или ROLLBACK для завершения транзакции.

Построение транзакций должно учитывать свойства АСИД.

1. **Атомарность** — *это свойство «все или ничего»*. Транзакция представляет собой неделимую единицу, которая выполняется полностью или не выполняется вообще. Если транзакция прерывается по какой-либо причине, то СУБД обеспечит либо завершение и проверку правильности транзакции, либо отказ от транзакции и отмену всех последствий действий, выполненных транзакцией до прерывания. Это произойдет после устранения причин, прервавших выполнение транзакции.

2. **Согласованность** *подразумевает, что транзакция должна преобразовать БД из одной согласованной формы в другую согласованную форму*. В промежуточных состояниях, через которые проходит БД во время транзакции, могут быть некоторые несоответствия, но конечное состояние, в которое БД переходит после выполнения транзакции, должно быть согласованным. Статус БД является согласованным, если он соответствует всем неявным или явным ограничениям целостности. СУБД не проверяет несогласованность БД после каждой операции в транзакции, но задача разработчиков приложений БД состоит в том, чтобы операции, предусмотренные в транзакциях, создавали согласованное состояние БД. Например, столбец содержит true или false (0 или 1), если пользователь пытается ввести другое значение, то правило согласованности не разрешает этого и возвращается в исходное состояние.

3. **Изолированность** *предполагает, что транзакция выполняется независимо от любой другой, т. е. частичные последствия незавершенной*

транзакции не должны влиять на другую транзакцию. Если выполняются несколько параллельных транзакций одновременно, то они не «видят» частичных изменений, сделанных транзакцией, начавшей изменения до момента валидации транзакции (до *comit*). Свойство изолированности транзакций очень важна, потому что она устраняет явление каскадных откатов транзакций. Если частичные результаты транзакции видны другим транзакциям до ее проверки и если эта транзакция отменена и откатывается (*rollback*), то все транзакции, которые обращались к ее частичным результатам, необходимо будет отменить; эти операции отмены могут, в свою очередь, привести к другим отменам и так далее.

4. **Долговечность** *подразумевает, что результаты успешно завершенной транзакции окончательно записываются в БД*. Это свойство, благодаря которому после проверки транзакции сделанные ею изменения в БД больше не могут быть потеряны в результате последующих сбоев системы. Свойство долговечности обеспечивается методами восстановления (recovery) СУБД (даже если выключится свет, эту транзакцию уже нельзя отменить).

Для восстановления БД, где обычно могут возникать несоответствия, большинство БД периодически копируются на магнитные носители, которые хранятся в надежном месте. Также ведется запись обо всех изменениях, внесенных в БД с момента создания последней копии. Файл, содержащий эти изменения, называется журналом. Каждая запись журнала содержит идентификатор программы, в которой было выполнено изменение, старое значение и новое значение, введенное для элемента. Журнал также фиксирует различные важные моменты из разработки программ (начало, завершение операций и т.д.). Файл журнала хранится на диске, и на него не влияют различные ошибки выполнения, за исключением катастрофического сбоя диска. Кроме того, файл журнала периодически сохраняется на вспомогательном носителе в качестве меры предосторожности для таких катастрофических дефектов. По завершении транзакции, независимо от того, была ли она завершена нормально или по ошибке, БД должна иметь ту же степень целостности, что и при выполнении транзакции.

7.2. Аномалии помех

Неконтролируемое взаимодействие двух и более транзакций может привести к несогласованным состояниям БД и ошибочным результатам. Две транзакции Ti и Tj подвержены (влиянию) помехам, если результат их одновременного выполнения может отличаться от результата последовательного выполнения. Неконтролируемое взаимодействие двух и более транзакций может привести к несогласованным состояниям БД и ошибочным результа-

там. Две транзакции Ti и Tj подвержены влиянию помех, если результат их одновременного выполнения отличается от результата последовательного выполнения. Помехи могут возникать между двумя транзакциями, если они выполняют операции над общими данными. Если эти две транзакции выполняются конкурентно, то мы говорим, что они конфликтуют. Итак, две транзакции Ti и Tj являются конфликтующими, если они параллельны и подвержены помехам.

В зависимости от характера операций, которые выполняются с общими данными, между двумя транзакциями может быть несколько типов помех, вызывающих аномалии помех:

1) аномалия потерянных обновлений (lost updates), соответствует конфликту записи-записи и заключается в том, что результат обновления, выполненного одной транзакцией, теряется в результате обновления тех же данных другой транзакцией, без влияния на обновление результата первого обновление;

2) неповторяющаяся аномалия чтения (nonrepeatable read), соответствует конфликту чтения-записи и возникает, когда транзакция фиксирует временно несогласованное состояние БД. Эта аномалия может возникнуть, когда одна из транзакций была заброшена, а другая конкурирующая транзакция использовала измененные данные до того, как данные вернулись в исходное состояние;

3) аномалия неповторимого чтения (dity read), соответствует конфликту чтения-записи и возникает, когда одна и та же транзакция находит разные значения при повторном чтении одних и тех же данных. Это может произойти, если транзакция T считывает элемент дважды, и между двумя чтениями этот элемент был изменен другой транзакцией T1. В этой ситуации транзакция T получает два разных значения одного и того же элемента;

4) аномалия фантомного чтения (fantom read) возникает, когда транзакция обрабатывает набор записей в результате запроса; если во время этой обработки другая транзакция вставила или удалила запись, которая удовлетворяет условию этого запроса, то записи могут появиться или исчезнуть из набора первоначально результирующих записей, поведение аналогично появлению или исчезновению призраков.

Разница между фантомным чтением и неповторимого чтения заключается в том, что фантомное чтение происходит, если конкурирующие транзакции изменяют количество записей, используемых текущей транзакцией (через INSERT или DELETE), а неповторимое чтение происходит, если конкурирующие транзакции изменяют значения в существующих и обрабатываемых записей текущей транзакции (через UPDATE). Такие аномалии, ко-

торые могут возникать при неконтролируемом одновременном выполнении двух или более транзакций, подчеркивают необходимость контроля конкуренции в БД. Согласованность данных, хранящихся в БД, должна быть обеспечена даже в условиях, в которых транзакция не может быть успешно завершена из-за возникновения неисправности.

SET TRANSACTION задает свойства транзакций и поддерживает несколько параметров для настройки управления транзакциями.

Уровень изолированности транзакций (*isolation level*) с возможными значениями:

- READ UNCOMMITTED;
- READ COMMITTED;
- REPETABLE READS;
- SERIALIZABLE;
- SNAPSHOT.

Уровни изолированности определяют, как СУБД вводит различные механизмы контроля конкуренции (чаще всего блокировки с несколькими состояниями). Например, на уровне READ COMMITTED общие блокировки предоставляются для всех элементов чтения, что предотвращает неправильное чтение, но эти блокировки снимаются до завершения транзакции и могут привести к неповторимым и фантомным чтениям. На любом уровне изолированности, включая самый слабый (READ UNCOMMITTED), используются механизмы контроля транзакций для предотвращения потери обновлений. Такие аномалии очень серьезны, БД не отражает операции, которые были выполнены над данными и нет возможности восстановить эти потери. Поэтому не предусмотрен уровень изолированности, который позволит потерю данных.

7.3. Оптимистичные и пессимистичные блокировки

Контроль конкурентного выполнения нескольких транзакций необходим для обеспечения свойств атомарности, изолированности и долговечности транзакций и, следовательно, согласованности хранимых данных. Контроль над конкуренцией может быть достигнут с помощью протоколов (набора правил), налагаемых на транзакции, так что, если они соблюдаются каждой транзакцией, любое планирование, в котором участвуют такие транзакции, является сериализуемой и, следовательно, правильной.

СУБД используют механизмы (алгоритмы), которые обеспечивают целостность транзакций и поддерживают непротиворечивость данных, когда к одним и тем же данным обращаются несколько пользователей одновременно:

1) механизмы управления *блокировкой*;

2) механизмы управления *маркировкой*.

Протоколы управления конкуренцией реализуются СУБД, поэтому разработчики приложений не оперируют блокировками или маркерами явным образом, а устанавливают параметры, с помощью которых СУБД принимает определенные методы управления конкуренцией.

Существуют два основных типа реализации контроля за конкуренцией:

- *методы с оптимистическим контролем над конкуренцией*, когда одна транзакция изменяет данные и в то же время другая транзакция изменяет те же данные, то первая транзакция откатывается назад;
- *пессимистические методы управления*, транзакция, которая изменяет данные блокирует эти данные и другие транзакции не могут выполнять действия над данными, пока транзакция не будет завершена.

Блокировка может предотвратить изменение данных пользователями таким образом, что это повлияет на других пользователей, и обеспечить целостность БД, особенно в отношении запросов, которые в противном случае могли бы привести к неожиданным результатам.

Библиография к главе 7

1. K. Delaney, SQL Server Concurrency: Locking, Blocking and Row Versioning, RedGate, Simple Talk Publishing, 2012.
2. Microsoft SQL Docs, Transaction Isolation Levels, 2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/t-sql/language-elements/transaction-isolation-levels?view=sql-server-ver15

Библиография

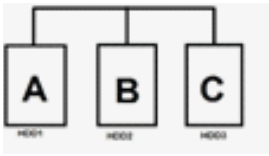
a) H. Afyouni, *Database Security and Auditing: Protecting Data Integrity and Accessibility*, Cengage Learning, 2006.
b) A. Basta, M. Zgola, *Database Security*, Cengage Learning, 2011.
c) K. Delaney, *SQL Server Concurrency: Locking, Blocking and Row Versioning*, RedGate, Simple Talk Publishing, 2012.
d) T. Kyte, Darl Kuhn, *Oracle Database Transactions and Locking Revealed*, Apress, 2014.
e) R. Natan, *Implementing Database Security and Auditing*, Elsevier, 2005.
f) B. Thuraisingham, *Database and Applications Security: Integrating Information Security and Data Management*, Auerbach Publications, 2005.
g) A. Watt, N. Eng, *Database Design*, 2014.
h) Microsoft SQL Docs, Permissions (Database Engine), 2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/relational-databases/security/permissions-database-engine?view=sql-server-ver15
i) Microsoft SQL Docs, Create a server audit and database audit specification, 2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/relational-databases/security/auditing/ create-a-server-audit-and-database-audit-specification?view=sql-server-2017
j) Microsoft SQL Docs, Transaction Isolation Levels, 2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/t-sql/language-elements/transaction-isolation-levels?view=sql-server-ver15
k) Microsoft SQL Docs, Restore a Database Backup Using SSMS 2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/restore-a-database-backup-using-ssms?view=sql-server-ver15
l) Beginner SQL Tutorial, SQL Integrity Constraints [citat 28.12.2021]. https://beginner-sql-tutorial.com/sql-integrity-constraints.htm
m) Microsoft SQL Docs, SQL Server Audit (Database Engine) 2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/relational-databases/security/auditing/sql-server-audit-database-engine?view=sql-server-ver15
n) Microsoft SQL Docs, Vulnerability assessment for SQL Server2021 [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/relational-databases/security/sql-vulnerability-assessment?view=sql-server-2017
o) CREATE TRIGGER (Transact-SQL) [citat 28.12.2021]. https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver15

ПРИЛОЖЕНИЯ

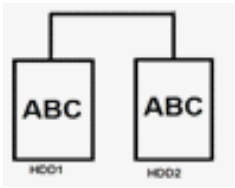
Приложение 1. Типы RAID

RAID (Redundant Array of Independent Disks или Redundant Array of Inexpensive Disks) — комбинация нескольких HDD, целью которой является либо увеличение скорости записи/чтения, либо защита данных, либо комбинация этих двух. Существует несколько типов RAID. Наиболее распространенные и известные являются RAID0, RAID1, RAID5 и RAID10.

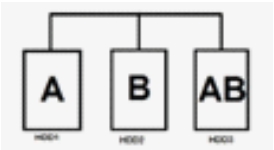
RAID0 используется для повышения производительности и скорости копирования и чтения данных. Пусть конфигурация RAID0 с 3 HDD, и хотим сохранить файл, состоящий из 3 частей (A, B и C). Сохранение происходит следующим образом: часть A файла сохраняется на первый HDD, часть B на второй HDD и часть C на третий HDD. Запись/чтение на несколько жестких дисков осуществляется одновременно, что повышает общую производительность системы. RAID0 — самый эффективный и быстрый способ копирования, хранения и чтения данных, но выход из строя одного из 3-х жестких дисков приводит к полной потере сохраненной информации.



RAID1 используется, когда хотим большей безопасности важных данных. Для конфигурации RAID1 используются 2 жестких диска. Сохранение происходит следующим образом: части A, B и C файла сохраняются на каждый HDD отдельно. Другими словами, файл дублируется на обоих HDD в одинаковой форме. Если один из двух жестких дисков выходит из строя, данные остаются в безопасности, поскольку файл существует на обоих жестких дисках. Недостаток этой конфигурации в том, что объем хранилища составляет половину емкости жестких дисков.

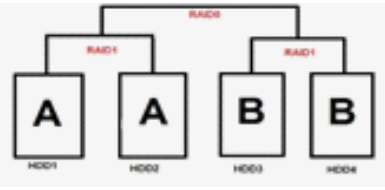


RAID5 Эта конфигурация включает в себя как высокую производительность, так и безопасность данных в случае повреждения жесткого диска. Чтобы объединить жесткие диски в конфигурацию RAID5, потребуется как минимум 3 жестких диска. Если хотим сохранить файл из 2 частей (A и B) тогда часть A файла сохраняется на первом жестком диске, часть B файла сохраняется на втором жестком диске, а весь файл сохраняется на третьем жестком диске. С этой конфигурацией имеем высокую производительность и скорость, потому что запись/чтение на несколько HDD происходит одновременно. В этом случае есть безопасность,



потому что любой из жестких дисков в случае потери данных остается доступным.

RAID10 Эта конфигурация представляет собой комбинацию RAID0 и RAID1. Нам нужно как минимум 4 HDD. RAID10 обеспечивает чтение данных в соответствии с моделью RAID 0, а хранение осуществляется как в RAID 1. Таким образом, мы выигрываем от скорости RAID0 и безопасности RAID1. Недостатком является то, что мы можем использовать только половину емкости 4 жестких дисков.



Если хотим повысить скорость работы системы, и возможная потеря данных не является серьезной проблемой, можем использовать RAID0. Если повышенная безопасность данных без увеличения скорости чтения/записи и урезанного вдвое объема хранилища не является препятствием, то рекомендуется RAID1. Для пользователей, которым требуется повышенная безопасность, но также и повышенная производительность, рекомендуется RAID 5, а для самых требовательных пользователей с большим бюджетом рекомендуется RAID 10. Можно создавать комбинации из нескольких RAID друг с другом с упором либо на безопасность, либо на скорость.

Таблица 2.1 DDL (Data Definition Language): определение данных

Оператор	Значение
CREATE	позволяет создавать таблицы
ALTER	позволяет изменить таблицы
TRUNCATE	позволяет удалить содержимое таблицы
DROP	позволяет удалять таблицы
RENAME	позволяет изменить имя таблицы

Таблица 2.2 DML (Data Manipulation Language): манипулирование данными

Оператор	Значение
INSERT	позволяет добавлять новые записи в таблицу
UPDATE	позволяет обновлять значения записей в таблице
DELETE	позволяет удалять записи из таблицы

Таблица 2.3 TCL (Transaction Control Language): обработка транзакции

Оператор	Значение
SAVEPOINT	позволяет определить точку сохранения, к которой можно вернуться, чтобы отменить изменения, сделанные после этой точки в БД
COMMIT	позволяет сделать изменения, внесенные в БД, постоянными
ROLLBACK	позволяет сбрасывать изменения в БД

Таблица 2.4 DQL (Data Query Language): запрос данных

Оператор	Значение
SELECT	позволяет извлекать записи, хранящиеся в таблицах

Таблица 2.5 DCL (Data Control Language): управление данными

Оператор	Значение
GRANT	позволяет предоставлять привилегии
REVOKE	позволяет отозвать привилегии

Таблица 3.1. Хранимые процедуры безопасности базы данных

Процедура	Значение
sp_addlogin	создание идентификаторов
sp_password	изменение пароля
sp_helplogins	предоставляет отчет с созданными на сервере идентификаторами
sp_droplogins	позволяет удалить идентификатор
sp_grantlogin	предоставляет разрешения на подключение
sp_revokelogin	позволяет снять права на подключение к SQL Server для пользователя или группы
sp_denylogin	позволяет запретить пользователю или группе право подключаться к SQL Server
sp_grantdbaccess	позволяет ассоциировать идентификатор с пользователем БД
sp_addalias	позволяет создать псевдоним
sp_revokedbaccess	отменяет право доступа к БД
sp_helpuser	предоставляет отчет с пользователями, созданными в БД
sp_dropalias	позволяет удалить псевдоним
sp_changedbowner	позволяет изменить владельца БД
sp_addrole	позволяет создать роль
sp_droprole	позволяет удалить роль из БД
sp_addrolemember	позволяет добавлять пользователей к фиксированным или определяемым пользователем ролям
sp_droprolemember	позволяет удалить члена из роли

Татьяна ПАША

БЕЗОПАСНОСТЬ БАЗ ДАННЫХ

курс лекций

Asistenţă computerizată – *Maria Bondari*

Bun de tipar 2022. Formatul 70x100 $^{1}/_{12}$.
Coli de tipar 5,75. Coli editoriale 4,5.
Comanda 7. Tirajul 50 ex.

Centrul Editorial-Poligrafic al USM
str. Al. Mateevici, 60, Chişinău, MD 2009