Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

_____Інститут електроенергетики_____
(інститут)
_____факультет інформаційних технологій_____
(факультет)
Кафедра__інформаційних систем та технологій та комп'ютерної інженерії___
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня ____бакалавра___
(бакалавра, спеціаліста, магістра)

Студента____Станішевського_Олексія Ігоровича_____
(ПІБ)
академічної групи_____126-17-1_____
(шифр)
спеціальності _____126 « Інформаційні системи та технології»____
(код і назва спеціальності)
за освітньо-професійною програмою
_____«Інформаційні системи та технології»_____

(офіційна назва)
на тему__«Розробка системи захисту бази даних від несанкціонованого доступу»_____

(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	Проф.Гнатушенко В.В			
розділів:				

Рецензент				

Нормоконтролер				

Дніпро

2021

ЗАТВЕРДЖЕНО:
завідувач кафедри

___інформаційних технологій___
___та комп'ютерної інженерії____
(повна назва)
_____ __Гнатушенко В.В.__
(підпис) (прізвище, ініціали)

«_____»_____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня ____бакалавр _____
(бакалавра, спеціаліста, магістра)

студенту __Станішевському О.І._____ академічної групи ____126-17-1___
(прізвище та ініціали) (шифр)

спеціальності ___126 « Інформаційні системи та технології »_____

за освітньою-професійною програмою _____

_____« Інформаційні системи та технології»_____
на тему___«Розробка системи захисту бази даних від несанкціонованого доступу»
_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 07.06.2021р. № 317-с

Розділ	**Зміст**	**Термін виконання**

Завдання видано _____ _____
(підпис керівника) (прізвище, ініціали)

Дата видачі _____

Дата подання до екзаменаційної комісії ___24.06.2021 р._____

Прийнято до виконання _____ __Станішевський О.І._____
(підпис студента) (прізвище, ініціали)

Реферат

Пояснювальна записка містить 99 сторінок , 2 малюнки і 14 джерел.

Об’єкт розробки: Розробка системи захисту бази даних від несанкціонованого доступу

Мета дипломної роботи:Показати як працює захист від несанкціонованого входу у базу даних

У вступі йде мова про початок роботи над захистом баз даних від несанкціонованого доступу

У пояснювальній записці наведені основні поняття , можливі типи несанкціонованого входу , можливості уникнути їх, методи захисту інформації з бази даних , методи відновлення інформації в базі даних , та було розглянуто систему захисту СУБД Microsoft Access .

Практична значимість даної роботи полягає у тому, щоб розкрити методи захисту бази даних .

Abstract

The explanatory note contains 99 pages, 2 figures and 14 sources.

Object of development: Development of a database protection system against unauthorized access

The purpose of the thesis: To show how protection against unauthorized entry into the database

The introduction talks about starting work on protecting databases from unauthorized access

The explanatory note presents the basic concepts, possible types of unauthorized entry, the ability to avoid them, methods of protecting information from the database, methods of restoring information in the database, and considered the security system of Microsoft Access.

The practical significance of this work is to reveal the methods of database protection.

Зміст

Список використаних скорочень

ACID	Atomicity, Consistency, Isolation, Durability (атомарность, з-гласованность, ізольованість, довговічність)
CDI	Constrained Data Items (контрольовані дані)
DAC	Discretionary Access Control (виборчий контроль доступу)
DRI	Declarative Referential Integrity (декларативна посилальна цілісність)
IVP	Integrity Verification Procedure (процедура перевірки цілосності)
MAC	Mandatory Access Control (обов'язковий контроль доступу)
MS	Microsoft
NRD	No Read Down (Не читай нижче)
NRU	No Read Up (Не читай вище)
NWD	No Write Down (Не пиши нижче)
NWU	No Write Up (Не пиши вище)
RBAC	Role-Based Access Control (заснований на застосуванні ролей контроль доступу)
SID	Security Identifier (ідентифікаційний номер облікової записи)
SQL	Structured Query Language (структурована мова запитів)
TCB	Trusted Computing Base (довірена обчислювальна база)
TP	Transformation Procedure (процедура перетворення)
T-SQL	Transact-SQL
UDI	Unconstrained Data Items (неконтрольовані дані)
URL	Universal Resource Locator (універсальний локатор ресурсу)
WAL	Write Ahead Log (пиши спочатку в журнал)
БД	база даних
ДВБ	довірена обчислювальна база
КС	комп'ютерна система
НСД	несанкціонований доступ
ПК	персональний комп'ютер
СУБД	система управління базами даних

ЕС	експертна система

Вступ

У сучасних умовах діяльність будь-якої організації пов'язана з оперуванням великим об'ємом інформації, доступ до якої має широке коло осіб. Наслідком зрослого останнім часом значення інформації стали високі вимоги до конфіденційності, цілосності і доступності даних. В таких умовах зловмисні або просто некомпетентні дії всього лише одного із співробітників організації здатні завдати непоправної шкоди організації в цілому. Мова навіть може не йти про розкрадання цінної інформації. Досить просто заблокувати будь-якими засобами доступ до важливого інформаційного ресурсу на досить тривалий час.

Системи управління базами даних (СКБД), особливо реляціонні СУБД і експертні системи (ЕС), стали домінуючим инструментом в області зберігання, обробки і представлення даних. Будь-який збій в роботі СУБД (ЕС), що супроводжується втратою, хоч і часткового, доступу до даних, негайно відбивається на конкурентній здатності підприємства. Тому захист даних від несанкціонованого доступу, від несанкціонованої модифікації або просто від їх руйнування є одним із пріоритетних завдань при проектуванні будь-якої інформаційної системи. Ця проблема (проблема захисту даних) охоплює як фізичний захист даних і системних програм, так і захист від несанкціонованого доступу до даних, що передаються по лініях зв'язку і знаходиться на накопичувачах, що є результатом діяльності як сторонніх осіб, так і спеціальних програм-вірусів. Якщо взяти до уваги, що ядром інформаційної системи є СУБД, то забезпечення інформаційної безпеки набуває вирішальне значення при виборі конкретних засобів забезпечення необхідного рівня безпеки організації в цілому.

Дані в комп'ютерній формі зосереджують в локальному і невеликому обсязі (наприклад, на флеш-карті типу MicroSD) величезні масиви інформації, несанкціонований доступ до якої або її руйнування можуть призводити часом до катастрофічних наслідків і збитку. Можливість швидкого (і в окремих випадках навіть без слідів) копіювання, модифікації або видалення величезних масивів даних, що знаходяться в комп'ютерній формі, в тому числі і видалених але розташованих додатково провокує зловмисників на несанкціонований доступ до інформації, її модифікацію або руйнування.

Разом з тим теоретична проробка питань забезпечення безпеки інформації та їх практична реалізація довгий час відставала від рівня розвитку програмної індустрії СУБД, і у комерційних продуктах засоби забезпечення безпеки даних стали лише в 90-х роках минулого століття.

Перші дослідження теорії і практики забезпечення безпеки даних в комп'ютерних системах були обумовлені, перш за все, потребами

військової сфери, де проблема безпеки в цілому, і компьютерної безпеки зокрема, стоять особливо гостро. Початок цим процесам було покладено дослідженнями питань захисту комп'ютерної інформації, проведеними в кінці 70-х - початку 80-х років національним центром комп'ютерної безпеки (NCSC - National Computer Security Center) Міністерства оборони США. Результатом цих досліджень стало видання Міністерством оборони США в 1983 р документа під назвою «Критерії оцінки надійних комп'ютерних систем», згодом за кольором обкладинки отримав назву «Помаранчевої книги». Даний документ став фактично першим стандартом в області створення захищених комп'ютерних систем і згодом основою організації системи сертифікації комп'ютерних систем за критеріями захисту інформації. Підходи до побудови та аналізу захищених систем, представлені в «Помаранчевої книзі», слугували теоретичною та методологічною базою для подальших обстежень в цій сфері. У 1991 р NCSC був виданий новий документ - Інтерпретація «Критеріїв оцінки надійних комп'ютерних систем» в застосуванні до поняття надійної системи управління базою даних, відомий під скороченою назвою TDI або «Рожевої книги», що конкретизує і розвиває основні положення «Помаранчевої книги »з питань створення та оцінки захищених СУБД.

В сучасних БД і ЕС досить успішно вирішуються завдання захисту конфіденційних даних від несанкціонованого доступу, забезпечення цілісності і доступності даних. Забезпечення доступності даних на фізичному рівні досягається шляхом використання отказостійких пристроїв зберігання даних, наприклад, кількох жорстких дисків, об'єднаних в масив RAID. Періодичне створення резервних копій і зберігання результатів усіх операцій у файлі журналу дозволяють відновити дані практично після будь-якого збою. Цілісність даних забезпечується досить широким набором процедур, що забезпечує достовірність і несуперечливість даних. Сучасні СУБД забезпечують логічну цілісність даних вже на етапі опису моделі даних. Оператори контролю доступу входять в стандарт мови запитів SQL і реалізовані в багатьох СУБД. Але все ж, незважаючи на широкий діапазон засобів контролю і захисту, загальний рівень захищеності і СУБД і ЕС визначається можливостями операційної системи, оскільки ці два компоненти працюють в тісному зв'язку між собою.

1. Специфіка захисту в базах даних

При побудові захисту баз даних необхідно враховувати ряд загроз безпеки інформації, пов'язаних з концентрацією в базах даних великої кількості різноманітної інформації, а також з можливістю використання складних запитів обробки даних. До таких загроз належать:

- инференции і агрегування (Logical Inference and Aggregation);
- комбінація дозволених запитів для отримання закритих дан них (Browsing);
- організація прихованих каналів передачі інформації (*Covert Channel*s);
- SQL ін'єкції (SQL Injection);
- програмні закладки, оцінний код (Backdoors, Trapdoors)
- троянські коні (Trojan Horses).

1.1.Інференції і агрегування

Під інференцією розуміється отримання конфіденційної інформації з відомостей з меншим ступенем конфіденційності шляхом висновків. Якщо враховувати, що в базах даних зберігається інформація, отримана з різних джерел в різний час, що відрізняється ступенем узагальненості, то аналітик може отримати конфіденційні відомості шляхом порівняння, доповнення та фільтрації даних, до яких він допущений. Крім того, він обробляє інформацію, отриману з відкритих баз даних, засобів масової інформації, а також використовує прорахунки осіб, що визначають ступінь важливості і конфіденційності окремих явищ, процесів, фактів, отриманих результатів. Такий спосіб отримання конфіденційних відомостей, наприклад, по материалам засобів масової інформації, використовується давно, і показав свою ефективність.

Близьким до інференції є інший спосіб добування конфіденціальних відомостей - агрегування. Під агрегуванням розуміється спосіб отримання більш важливих відомостей у порівнянні з важливістю тих окремо взятих даних, на основі яких і виходять ці відомості.

Якщо інференції і агрегування є способами добування інформації, які застосовуються не тільки по відношенню до баз даних,то спосіб спеціального комбінування запитів використовується тільки при роботі з базами даних. Використання складних, а також послідовно простих логічно пов'язаних запитів дозволяє отримувати дані, до яких доступ користувачеві закритий. Така можливість є, перш за все, в базах даних, що дозволяють отримувати статистичні дані. При цьому окремі записи, поля, (індивідуальні дані) є закритими. В результаті запиту, в якому

можуть використовуватися підсумкові операції, користувач може отримати такі величини, як кількість записів, сума, максимальне або мінімальне значення.

Протидія подібним загрозам здійснюється наступними методами:

- блокування відповіді при неправильному числі запитів;
- спотворення відповідним шляхом округлення та іншої навмисної корекції даних;
- поділ баз даних;
- випадковий вибір запису для обробки;
- контекстно-орієнтований захист;
- контроль запитів, що надходять.

Метод блокування відповіді при неправильному числі запитів - відмова у виконанні запиту, якщо в ньому міститься більше виділеного числа співпадаючих записів з попередніх запитів. Таким чином, даний метод забезпечує виконання принципу минимального взаємозв'язку запитів. Цей метод складний в реалізації, так як необхідно запам'ятовувати і порівнювати всі попередні запити.

Метод корекції полягає в незначній зміні точної відповіді на запит користувача. Для того, щоб зберегти прийнятну точність статистичної інформації, застосовується так званий свап даних. Суть його полягає у взаємному обміні значень полів запису, в результаті чого всі статистики і-го порядку, включаючи і атрибутів, виявляються захищеними для всіх і, менших або рівних певному числу. При свапі результат виконання статистичної операції не змінюється, але якщо зловмисник і зможе виявити деякі узагальнені дані, то він не зможе визначити, до якого конкретно запису вони відносяться.

Застосовується також метод поділу баз даних на групи. У кожну групу може бути включено не більше певної кількості записів. Запити дозволені до будь-якої безлічі груп, але забороняються до підмножини записів з однієї групи. Застосування цього методу обмежує можливості виділення даних зловмисником на рівні не нижче групи записів. Метод поділу баз даних не знайшов широ кого застосування через складність отримання статистичних даних, оновлення та реструктуризації даних.

Ефективним методом протидії дослідженню баз даних є метод випадкового вибору записів для статистичної обробки. Така організація вибору записів не дозволяє зловмисникові простежити безліч запитів.

Сутність контекстно-орієнтованої захисту полягає в назначені атрибутів доступу (читання, вставка, видалення, оновлення, управління) Елементів бази даних (записів, полях, групам полів) в залежності від попередніх запитів користувача.

Одним з найбільш ефективних методів захисту інформації в базах даних є контроль запитів, що надходять на наявність «підозрілих» запитів або комбінації запитів. Аналіз дозволяє виявити можливі канали отримання несанкціонованого доступу до закритих даних.

1.2.Приховані канали передачі інформації

Неформально під прихованим каналом (Covert channel) передачі інформації розуміють будь-який канал зв'язку, спочатку для передачі інформації призначений. У більш загальному вигляді під прихованим, інакше непрямим каналом, порушення конфіденційності мається на увазі механізм, за допомогою якого суб'єкт, який має високий рівень допуску, може надати певні аспекти конфіденційної інформації суб'єктам, ступінь допуску яких нижче рівня конфіденційності цієї інформації.

Приховані канали практично неможливо усунути, і наші зусилля повинні бути спрямовані на мінімізацію пропускної здатності цих каналів. Напевно, єдиний спосіб усунення прихованих каналів полягає в повній ліквідації всіх загальних ресурсів і всіх ком- комунікація.

Виділяють такі типи прихованих каналів:

1. Приховані канали по пам'яті, в яких інформація передається через доступ відправника на запис і одержувача на читання до одних і тих же ресурсів або об'єктів;
2. Приховані канали по часу, які характеризуються доступом відправника і одержувача до одного і того ж процесу або вимірюваного в часі атрибуту.

Наведемо приклади прихованих каналів передачі інформації. Розглянемо систему, в якій є два рівні секретності: Високий і Низький. Передача інформації з рівня Низький на рівень Високий дозволена, а в зворотному напрямку - заборонена. Мета порушника полягає в тому, щоб організувати прихований канал для передачі інформації від програмно-апаратного агента, який функціонує в середовищі Високий, до іншого програмно-апаратного агенту, що функціонує в середовищі Низький. Суб'єкт, що функціонує в середовищі Високий, може здійснювати абсолютно нешкідливі дії, наприклад, змінювати налаштування параметрів безпеки елементів файлової системи, доступні для спостереження в середовищі Низький. В цьому випадку зловмисник може закодувати передану інформацію в значеннях параметрів без небезпеки тих чи інших елементів файлової системи.

Приклад прихованого каналу за часом. В даному випадку між рівнями Високий і Низький немає загальних ресурсів, за винятком деякої системної бібліотеки або програми, доступ до якої можливий тільки на читання. Для організації прихованого каналу передачі інформації суб'єкт

в середовищі Високий може модулювати певним образом інтервали зайнятості бібліотеки, а суб'єкт в середовищі Низький - сканувати час зайнятості бібліотеки, здійснюючи запити до неї з заданою періодичністю.

Підходи до вирішення завдання виявлення прихованих каналів передачі інформації в даний час активно вивчаються і вдосконалюються. На сьогоднішній день найбільш поширеним є такі методи:

1. Метод поділюваних ресурсів Кемерера, який складається в наступном: для кожного ресурсу, що в системі будується матриця, рядки якої відповідають всіляким атрибутам розділеного ресурсу, а стовпці - обробки, яка виконується в системі. Значення в комірках матриці відповідають діям, які здійснюються при виконанні тих чи інших операцій в відношенні атрибутів поділюваних ресурсів. Отримана в результаті матриця дозволяє відстежити інформаційні потоки, що існують в системі.
2. Сигнатурний аналіз вихідних текстів програмного забезпечення. Даний метод передбачає проведення аналізу вихідних текстів програм з метою виявлення конструкцій, характерних для прихованих каналів передачі інформації. Необхідність проведення аналізу автоматизованих систем в ході проведення сертифікаційних випробувань регламентується відповідними стандартами і зазвичай є необхідним для високо довірених систем.

1.3.SQL ін'єкції

Ін'єкцією SQL (SQL Injection) називають технологію злому, котра додає в параметр динамічного SQL НЕ необхідне значення, а деякий код SQL. Ця технологія особливо небезпечна тим, що будь-який, хто має доступ до БД або Web-сайту організації, що звертається до БД, і здатний вводити дані в текстові поля, потенційно може стати джерелом атак за допомогою ін'єкцій SQL. SQL-ін'єкція може возникнути в наступних випадках:

- передача шкідливого коду в параметри динамічного запиту (прикріплюється код і змінені пропозиції WHERE);
- неправильна обробка типів;
- пошук уразливості в СУБД;
- умовні помилки.

Існує безліч шкідливих прийомів, які використовують при закріпленні коду і змінені пропозиції WHERE.

Прикріплення шкідливого коду.Додавання термінатора інструкції,

інший інструкції SQL і вмісту введення дозволяє зломщикові передати програмний код в рядок виконання. Наприклад, додаток зчитує параметр customer з поля введення, розміщеного на формі, і динамічно формує рядок SQL виду

query = "SELECT * FROM Customers WHERE CustomerName = '" +
customer + "'"

Якщо користувач замість введення імені клієнта вводить наступний рядок:

User1 '; Delete FROM Orders--

то в динамічну рядок SQL буде додана інструкція DDL DELETE, яка і буде виконана в пакеті з основною:

SELECT * FROM Customers WHERE CustomerlD = 'User1'; De- lete
FROM Orders-- '

Термінатор інструкції (;) завершує закладену програмістом операцію, після чого СУБД розглядає отриманий текст як таку інструкцію в пакеті. Завершальна лапки могла б привести до помилки виконання, однак ця проблема легко вирішується шляхом додавання маркера коментаря (- для MS SQL Server). В результаті отримаємо порожню таблицю замовлень (Orders).

Серед інших популярних прикріплюються кодів можна відзначити запуск команди xp_commandshell (в MS SQL Server) і установку пароля для користувача sa.

Прикріплення Or 1 = 1. Ще одним методом "ін'єкцій" SQL являє- ся модифікація пропозиції WHERE для вибірки більшого коліче- ства рядків, ніж було спочатку передбачено. Якщо користувач вво- дить в текстове поле рядок

123 'Or 1 = 1 -

то умова 1 = 1 (яке завжди істинно) впроваджується в пропозицію WHERE:

SELECT * FROM Customers WHERE CustomerlD = '123' or 1 = 1 - '

Так як в інструкції відбираються всі рядки, далі все залежить толь до від того, як система обробляє безлічі рядків. Однак незалежні Сімо від цього, програма зробить зовсім не те, що повинна була зробити.

Або введемо в поле введення пароля наступне

secreta net 'Or 1 = 1 - отримаємо:

SELECT * FROM users WHERE username = 'User2' AND password

= '123456' OR 1 = 1-- '

Значення 1 = 1 завжди істинно, таким чином, значення пароля вже не матиме значення.

Коментування коду. Ще однією розповсюдженою "ін'єкцією" коду SQL є коментування решти коду, що підлягає виконанню. Якщо користувач вводить в Web-форму реєстрації на сайті:

UserName: Alex

Password: qwerty

то результуючу інструкцію SQL можна буде прочитати наступним чином:

SELECT UserlD FROM Users WHERE UserName = 'Alex' - 'AND

Password =' qwerty '

Включення маркера коментаря в поле імені користувача призводить до того, що подальша частина пропозиції WHERE, включаючи усови для пароля, ігнорується.

Захист від SQL ін'єкцій. Запобігти проникненню "ін'екцій" коду можна кількома шляхами.

1. Натомість динамічного формування інструкцій використовувати підготовлені інструкції, наприклад:

   PreparedStatement pstmt = conn.prepareStatement ("select balance from account where account_number =?"); Pstmt.setString (1, accnt_number);

   pstmt.execute ();

2. Виробляти перевірку типу даних, що вводяться і обмежувати число символів, що вводять.
3. Використовувати збережені процедури і функції.
4. Застосовувати функції, автоматично усувають потенційно небезпечні символи або послідовності символів, що містять термінатори інструкцій, коментарі, одиничні лапки і символи xp_ (для SQL Server) з даних, що вводяться. Наприклад: функція блокування mysql_real_escape_string () в MySQL блокує введення одиночних лапок, термінаторів інструкцій та інше.
5. Ретельно визначати права доступу, щоб інструкції не мали дозволів на запуск DDL інструкцій.
6. Відключати видачу необроблених помилок.

Проблеми ін'єкцій коду SQL слід приділяти підвищену увагу. Якщо

ваш додаток передбачає введення даних з Інтернету, і ви не передбачите дії, спрямовані проти потенційних ін'єкцій, руйнування бази даних стане тільки питанням часу.

1.4.Управління доступом до даних

У будь-якій організації діють певні правила накопичення і використання відомостей, що обмежують доступ до інформаційних ресурсів. Конкретні правила визначаються інформаційної політикою керівництва підприємства, однак в будь-якому випадку розумні ограничення доступу засновані на наступних принципах:

- підприємство є власником усієї службової інформації, отриманої його підрозділами або службовцями;
- підрозділ або службовець є власником отриманої ним інформації і може використовувати її в інтересах підприємства без обмежень;
- службовець має право доступу до тих і тільки тих відомостей, ко торие необхідні для виконання його службових обов'язків;
- службовець має право виконувати ті, і тільки ті маніпуляції доступними відомостями, які обумовлені його службовими обов'язками.

Ці принципи реалізуються у вигляді системи правил, що обмежують права доступу співробітників до інформаційних ресурсів підприємства. Тимчасові СУБД мають добре розвинені засоби підтримки подібних правил - підсистеми адміністрування даних. Метою подсістеми адміністрування є забезпечення санкціонованого доступа службовців підприємства до збережених даних

За надання їм доступ до комп'ютерної системи зазвичай відповідає системний адміністратор, в обов'язки якого входить створення облікових записів користувачів. Кожному користувачеві присвоюється унікальний ідентифікатор, який використовується операційною системою для того, щоб визначити, хто є хто. З кожним ідентифікатором зв'язується пароль, який обирається користувачем. При реєстрації користувач повинен надавати системі свій пароль для аутентифікації. Подібна процедура дозволяє організувати контрольований доступ до комп'ютерної системи, але не обов'язково надає право доступу до СУБД чи іншої прикладної програми. Для отримання користувачем права доступа до СУБД може використовуватися окрема подібна процедура. Відповідальністьственность за надання прав доступу до СУБД зазвичай несе адміністратор бази даних, в обов'язки якого входить створення індивідуальних ідентифікаторів користувачів, на цей раз вже в середовищі самій СУБД. Кожен з ідентифікаторів користувачів СУБД також связується з паролем, який повинен бути відомий тільки даному

користувачеві.

Деякі СУБД підтримують списки дозволених ідентифікаторів користувачів і паролів, що відрізняються від аналогічного списку, підтримуваного ОС. Інші типи СУБД підтримують списки, елементи яких приведені у відповідність існуючим списками користувачів операційної системи і виконують реєстрацію, виходячи з поточного ідентифікатора користувача, зазначеного їм при реєстрації в системі.

Виходячи з вищесказаного, сформулюємо термін «управління доступом» більш детально. Управління доступом- є метод захисту інформації шляхом регулювання використання ресурсів системи (еле ментів БД, програмних і технічних засобів). Включає наступні функції захисту:

- ідентифікація користувачів і ресурсів системи;
- встановлення аутентичності об'єкта або суб'єкта по пред'явленному їм ідентифікатору (аутентифікація);
- розмежування і перевірку повноважень (авторизація), створення умов роботи в межах встановленого регламенту;
- реєстрація звернень до ресурсів, що захищаються ;
- реагування при спробах несанкціонованого доступу.

1.5.Ідентифікація та аутентифікація користувачів

Для того щоб отримати доступ до БД, користувач повинен вказати свій ідентифікатор (ID) і підтвердити право на його використання. Пізнавши користувача, система готова виконувати операції над даними від його імені. Залежно від необхідного ступеня захищеності си стеми можуть використовуватися різні процедури встановлення особисто-сті користувача. Наведемо найпростіші і найбільш часто використовувані програмні процедури аутентифікації, не пов'язані з аналізом «почерк» користувача (наприклад, за швидкістю натискання клавіш клавіатури і тривалості пауз між окремими натисканнями):

Парольний захист. У найпростішому варіанті з ID зв'язується пароль

- відомий тільки власнику ID і системі набір символів (секрет). Мінімальний набір повноважень власника ID в цьому варіанті вклю- ет права входу в систему і зміни свого пароля. Жоден користувач (включаючи адміністратора системи) не має права перегляду паролей. Паролі зберігаються в вигляді хеш-коду або в зашифрованому вигляді.

Реалізуючи парольний захист, адміністратор повинен визначити ал-фавіт паролів і їх довжину, а також встановити максимальне число спроб введення пароля і / або час реакції користувача на запит пароля. Слід мати на увазі, що чим багатше алфавіт і довше пароль, тим важче

зламати захист шляхом його підбору. Однак, з іншого боку, довгий і витончений пароль важче ввести без помилок. Для підняття надійності захисту може бути встановлений термін дії пароля або максимальне число підключень з цим паролем. Додатково вводять затримку при введенні неправильного пароля, відбраковування паролів по словнику і журналу історії паролів. У ряді випадків вводять заборону на вибір пароля користувачем і виробляють автоматичну генерацію пароля.

Захист "питання-відповідь". Більш складний варіант захисту входу може бути таким. Власник ID вводить при реєстрації серію питань разом з відповідями. Питання мають особистий характер, тому правильність ні відповіді на них неможливо вгадати. Наприклад: "На яку шкільну кличку відгукується Ваш племінник?", "Де народилася Ваша бабуся?" і т.п. Для встановлення особи користувача при спробі входу система пред'являє йому випадково вибране запитання (и) з цієї серії.

Зумовлений алгоритм. Якщо потенційний зловмисник може підключитися до комунікаційної лінії, що зв'язує комп'ютер-клієнт з сервером, то для захисту входу можна використовувати певний алгоритм, відомий власнику ID і системі. При спробі входу система пред'являє користувачу випадкове число. Користувач повинен виконати необхідні перетворення і ввести результат. Сторонній спостерігач на лінії може побачити тільки вихідне і кінцеве числа. Такого виду захист набула широкого поширення в Інтернет системах доступу до банківського рахунку. Власнику рахунку видається спеціальний портативний пристрій, оснащений зчитувачем інформації з чіпа банківської смарт-карти. При встановленні з'єднання власник повинен спочатку вставити карту в зчитувач, ввести пін-код, а вже потім ввести випадковим чином сгенероване на сторінці доступу число. Відповідь даного пристрою посилається назад на сервер.

При з'єднанні по мережі аутентифікацію повинні пройти обидва з'єднаних об'єкта. Після встановлення з'єднання необхідно виконати вимоги захисту при обміні повідомленнями:

- одержувач повинен бути впевнений в достовірності джерела даних;
- одержувач повинен бути впевнений в достовірності переданих даних;
- відправник повинен бути впевнений в доставці даних одержувачу;
- відправник повинен бути впевнений в достовірності доставлених даних.

Виконати цю вимогу захисту можна за допомогою так званої цифрового підпису. Якщо всі ці чотири вимоги реалізовані в КС, то забезпечується функція підтвердження (незаперечності передачі).

Відправник не може заперечувати ні факту посилки повідомлення, ні його змісту, а одержувач не може заперечувати ні факту отримання повідомлення, ні автентичності його змісту.

Інформація про зареєстрованих користувачів БД зберігається в її системному каталозі. Сучасні СУБД не мають загального синтаксису SQL-пропозиції з'єднання з базою даних, так як їх власний синтаксис склався раніше, ніж стандарт ISO. Найбільш часто употреб- ляется пропозицію CONNECT. Наприклад, для IBM DB2:

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

З'єднання з системою не ідентифікованих користувачів і користувачів, у яких перевірка справжності пред'явленого іден- тіфікатора при аутентифікації не підтвердилася, виключається. В про- процесі сеансу роботи користувача (від вдалого проходження ідентифікації і аутентифікації до від'єднання від системи) все його дії безпосередньо зв'язуються з результатом ідентифікації. Від'єднання користувача може бути як нормальним, так і насильницьким (походить від користувача-адміністратора, наприклад в разі видалення користувача або при аварійному обриві каналу зв'язку клієнта і сервера). У другому випадку користувач повинен бути проінформований про це, і всі його дії анулюються до останньої фіксації, зроблених ним в таблицях бази даних.

Авторизація користувачів

Авторизація користувачів (суб'єктів доступу) полягає в наданні певних прав (або привілеїв), що дозволяють їх власнику мати законний доступ як до самої системи, так і до її від- діловою об'єктів. Всі суб'єкти доступу можуть бути розділені для си- стеми на ряд категорій, наприклад: CONNECT, RESOURCE і DBA. Набір таких категорій визначається виробником СУБД. Наростання можливостей (повноважень) для кожного окремого виду підключення відбувається в зазначеному порядку:

CONNECT - кінцеві користувачі. За замовчуванням їм дозволено тільки з'єднання з базою даних і виконання запитів до даних, всі їхні дії регламентовані виданими їм привілеями;

RESOURCE - привілейовані користувачі, які мають право створення власних об'єктів в базі даних (таблиць, уявлень, збережених процедур і тригерів).

DBA - категорія адміністраторів бази даних. Включає можли- ності обох попередніх категорій, а також можливість регістріро- вать суб'єкти захисту або змінювати їх категорію.

Слід особливо відзначити, що в деяких СУБД адміністративні дії також розділені, що обумовлює наявність додаткових категорій. Так, в

Oracle користувач з ім'ям DBA є адмін і- мусить зареєструвати сервера баз даних, а не однієї-єдиної бази даних. У IBM DB2 існує ряд категорій адміністраторів: SYSADM (найвищий рівень; системний адміністратор, що володіє всіма привілеями); DBADM (адміністратор бази даних, що володіє всім набором привілеїв в рамках конкретної бази даних). Прівілегії управління сервером баз даних є у користувачів з іменами SYSCTRL (найвищий рівень повноважень управління системою, який застосовується тільки до операцій, що впливає на системні ресурси; безпосередній доступ до даних заборонений, дозволені опера ції створення, модифікації, видалення бази даних, створення і видалення табличних просторів). Для кожної адміністративної операції в IBM DB2 визначено необхідний набір адміністративних категорій, до ко-менту, котрим повинен належати користувач, який виконує той чи інший запит адміністрування. Так, виконувати операції призначення привілегій користувачам може SYSADM або DBADM, а для того щоб здати об'єкт даних, користувач повинен володіти привілеєм CREATETAB.

Адміністратор кожного БД займається створенням списку можливих користувачів БД і розмежуванням повноважень цих користувачів.

Дані про розмежуваннях розташовуються в системному каталозі БД. Очевидно, що дана інформація може бути використана для несанкціонованого доступу і тому також підлягає захисту. Захист цих даних здійснюється засобами самої СУБД.

В даний час найбільш часто використовується три основних під- ходу до розмежування доступу: виборчий, обов'язковий і рольової. Виборчий підхід описується дискреционной моделлю розмежування доступу, Discretionary Access Control (DAC), обов'язковий або повноважний - мандатної моделлю розмежування доступу, Mandatory Access Control (MAC). Обидва забезпечують створення системи безпеки як для БД в цілому, так і для окремих її об'єктів - таблиць, переставлений, кортежів і т.д. аж до конкретного значення деякого атрибуту в певному кортежі певного ставлення. Рольова модель розмежування доступу, Role Based Access Control (RBAC) в деякому сенсі комбінацією вищезазначених моделей. Слід згадати також і модель Китайської стіни,

Оператори SQL надання та скасування привілеїв. У стандартах SQL визначені два оператора GRANT і REVOKE для надавати пріоритет лення і скасування привілеїв відповідно.

Оператор надання привілеїв має наступний формат:

GRANT {<список дій> | ALL PRIVILEGES} ON <ім'я об'єкта> TO
{<список користувачів> | PUBLIC} [WITH GRANT OPTION]

де <список дій> визначає набір дій з доступного списку дій над об'єктом даного типу (параметр ALL PRIVILEGES вказує, що дозволені всі дії, допустимі для об'єктів даного типу), <ім'я об'єкта> визначає ім'я об'єкта захисту: таблиці, перед- уявлення , збереженої процедури або тригера, <список користувачів> визначає список ідентифікаторів користувачів, кому предоставля- ються дані привілеї. Замість списку ідентифікаторів можна вос- користуватися параметром PUBLIC. Параметр WITH GRANT OPTION яв- ляется необов'язковим і визначає режим, при якому передаються не тільки права на зазначені дії, але і право передавати ці права іншим користувачам. Передавати права в цьому випадку користувач може тільки в рамках дозволених йому дій. У загальному випадку набір привілеїв залежить від реалізації СУБД (визначається Вироб- ник). Виділяються привілеї маніпулювання даними:

SELECT - переглядати дані;

INSERT [(<список полів>)] - додавати дані;

UPDATE [(<список полів>)] - оновлювати дані;

DELETE - видаляти дані;

REFERENCES [(<список полів>)] - посилатися на зазначені поля при визначенні посилальної цілісності;

USAGE - використовувати домени і обмежувачі цілісності;

EXECUTE - виконувати збережені процедури і функції.

Серед привілеї створення / зміни об'єктів БД виділимо наибо- леї часто використовувані:

CREATE <тип об'єкта> - створення об'єкта деякого типу;

ALTER <тип об'єкта> - зміна структури об'єкта; DROP <тип об'єкта> - видалення об'єкта;

ALL - всі можливі дії над об'єктом.

У реалізаціях можуть бути присутні й інші різновиди пріві- Легій, наприклад:

CONTROL (IBM DB2) - комплексна привілей управління струк- турою таблиці;

RUNSTAT - виконання збору статистичної інформації по таблиці.

Розглянемо приклад: нехай у нас існують три користувача з уні Кальне іменами User1, User2 і User3. Всі вони є пользовате- лями однієї БД. User1 створив об'єкт Tab1, він є власником цього об'єкта і може передати права на роботу з цим об'єктом іншим поль зователем. Припустимо, що користувач User2 є оператором, який повинен вводити дані в Tab1 (наприклад, таблицю нових козака-поклик), а користувач User3 є менеджером відділу, який повинен регулярно переглядати введені дані. Тоді логічно буде ви- конати такі призначення:

GRANT INSERT ON Tab1 TO User2

(Або GRANT INSERT, DELETE, UPDATE ON Tab1 TO
User2) GRANT SELECT ON Tab1 TO User3

Ці призначення означають, що користувач User2 має право тільки вводити нові рядки в ставлення Tab1, а користувач User3 має право переглядати всі рядки в таблиці Tab1. При призначенні прав доступу на операцію модифікації можна уточнити, значення яких по- лей може змінювати користувач. Припустимо, що менеджер відділу має право змінювати ціну на послуги, що надаються, що задається в поле COST відносини Tab1. Тоді операція призначення привілеїв користувачу User3 може змінитися і виглядати наступним чином:

GRANT SELECT, UPDATE (COST) ON Tab1 TO User3

Якщо користувач User1 передбачає, що користувач User4 може його заміщати у разі його відсутності, то він може надати цьому користувачеві всі права по роботі зі створеною таблицею Tab1.

GRANT ALL PRIVILEGES ON Tab1 TO User4 WITH GRANT OPTION

У цьому випадку користувач User4 може сам призначати привілеї по роботі з таблицею Tab1 за відсутності власника об'єкта користувача User1. Тому в разі появи нового оператора користувача User5 він може призначити йому права на введення нових рядків у таблицю командою

GRANT INSERT ON Tab1 TO User5

Якщо при передачі повноважень набір операцій над об'єктом ограничен, то користувач, якому передані ці повноваження, може передати іншому користувачеві тільки ті повноваження, які є у нього, або частина цих повноважень. Тому якщо користувачеві User4 були делеговані такі повноваження:

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

то користувач User4 не зможе передати повноваження на введення дан них користувачу User5, тому що ця операція не входить в список дозволених для нього самого.

Для скасування раніше призначених привілеїв в стандарті SQL визна льон оператор REVOKE. Оператор скасування привілеїв має наступний синтаксис:

REVOKE {<список дій> | ALL PRIVILEGES} ON <ім'я об'єкта>
FROM {<список користувачів> | PUBLIC} {CASCADE |
RESTRICT}

Параметри CASCADE або RESTRICT визначають, яким чином повинна проводитися скасування привілеїв. Параметр CASCADE відміняє привілеї не тільки користувача, який безпосередньо згадувався в операторі GRANT при наданні йому привілеїв, але і всім користувачам, яким цей користувач присвоїв привілеї, скориставшись параметром WITH GRANT OPTION. Наприклад, при використанні операції:

REVOKE ALL PRIVILEGES ON Tab1 FROM User4 CASCADE

будуть скасовані привілеї і користувача User5, якому користувач User4 встиг привласнити привілеї. Параметр RESTRICT обмежив кількість скасування привілеїв тільки користувачеві, безпосередньо згаданому в операторі REVOKE. Але при наявності делегованих привілеїв при виконанні інструкції REVOKE виникне помилка. Так, наприклад, операція:

REVOKE ALL PRIVILEGES ON Tab1 FROM User4 RESTRICT

нічого очікувати виконано, тому що користувач User4 передав частину своїх повноважень користувачеві User5.

За допомогою оператора REVOKE можна відібрати все або тільки деякі з раніше присвоєних привілеїв по роботі з конкретним об'єк єктом. При цьому з опису синтаксису оператора скасування привілеїв видно, що можна відібрати привілеї одним оператором відразу у декількох користувачів або у цілої групи (параметр PUBLIC). Тому коректним буде таке використання оператора REVOKE:

REVOKE INSERT ON Tab1 FROM User2, User4 CASCADE

При роботі з іншими об'єктами (наприклад, з збереженими процедурами) змінюється список операцій, які використовуються в операторах GRANT і REVOKE. За замовчуванням, дія, відповідне запуску (виконання) збереженої процедури, призначається всім членам групи PUBLIC. Якщо потрібно змінити цю умову, то після створення хранімой процедури необхідно записати оператор REVOKE

REVOKE EXECUTE ON COUNT_EX FROM PUBLIC CASCADE

а потім призначити нові права певним користувачам

GRANT EXECUTE ON COUNT_EX FROM User4

Об'єктні привілеї призначає адміністратор або власник БД. Наприклад, (в SQL Server):

GRANT CREATE DATABASE ON SERVER_0 TO main_user

За принципом ієрархії користувач main_user, створивши свою БД, тепер може надати права на створення або зміна будь-яких об'єк тів в цій БД іншим користувачам, наприклад:

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

User1

У цьому випадку користувач User1 може створювати, змінювати або видаляти таблиці в БД MyDB, проте він не може дозволити створювати або змінювати таблиці в цій БД іншим користувачам, тому що йому дано дозвіл без права делегування своїх можливостей.

Переваги та недоліки дискреційного розмежування доступу. До переваг дискреційного розмежування доступу відносяться відноси тельно проста реалізація (перевірка прав доступу суб'єкта до об'єкта проводиться в момент відкриття цього об'єкта в процесі суб'єкта), хороша вивченість, універсальність, наочність і гнучкість. Однак дискреционная захист є досить слабкою, так як привілеї існують окремо від даних і доступ обмежується тільки до імені об'єктів, а не власне до даних, що зберігаються. У разі ре- ляціонному БД об'єктом буде, наприклад, іменоване відношення (таб- особи). У цьому випадку не можна в повному обсязі обмежити доступ тільки до частини інформації, що зберігається в таблиці. Це пов'язано з тим, що навіть якщо ввести окремий атрибут, який буде зберігати інформацію про мітку конфіденційності документа, то засобами SQL можна буде отримати вибірку даних без урахування атрибута даної мітки. Фактично це означає, що або сам сервер баз даних повинен надати більш високий рівень захисту інформації, або доведеться реалізувати дан ний рівень захисту інформації за допомогою жорсткого обмеження операцій, які користувач може виконати за допомогою SQL. На деякому рівні таке розмежування можна реалізувати за допомогою збережених процедур, але не повністю - в тому сенсі, що саме ядро СУБД дозволяє розірвати зв'язок «захищається - мітка конфі- денціальності». Дискреційне розмежування доступу має ряд і інших недоліків. Перерахуємо всі недоліки дискреційної моделі у вигляді списку: що або сам сервер баз даних повинен надати більш високий рівень захисту інформації, або доведеться реалізувати дан ний рівень захисту інформації за допомогою жорсткого обмеження операцій, які користувач може виконати за допомогою SQL. На деякому рівні таке розмежування можна реалізувати за допомогою збережених процедур, але не повністю - в тому сенсі, що саме ядро СУБД дозволяє розірвати зв'язок «захищається - мітка конфіденціальності». Дискреційне розмежування доступу має ряд і інших недоліків. Перерахуємо всі недоліки дискреційної моделі у вигляді

списку: що або сам сервер баз даних повинен надати більш високий рівень захисту інформації, або доведеться реалізувати дан ний рівень захисту інформації за допомогою жорсткого обмеження операцій, які користувач може виконати за допомогою SQL. На деякому рівні таке розмежування можна реалізувати за допомогою збережених процедур, але не повністю - в тому сенсі, що саме ядро СУБД дозволяє розірвати зв'язок «захищається - мітка конфі- денціальності». Дискреційне розмежування доступу має ряд і інших недоліків. Перерахуємо всі недоліки дискреційної моделі у вигляді списку: На деякому рівні таке розмежування можна реалізувати за допомогою збережених процедур, але не повністю - в тому сенсі, що саме ядро СУБД дозволяє розірвати зв'язок «захищається - мітка конфі- денціальності». Дискреційне розмежування доступу має ряд і інших недоліків. Перерахуємо всі недоліки дискреційної моделі у вигляді списку: На деякому рівні таке розмежування можна реалізувати за допомогою збережених процедур, але не повністю - в тому сенсі, що саме ядро СУБД дозволяє розірвати зв'язок «захищається - мітка конфі- денціальності». Дискреційне розмежування доступу має ряд і інших недоліків. Перерахуємо всі недоліки дискреційної моделі у вигляді списку:

1. Зберігання привілеїв доступу окремо від даних;
2. Обмеження доступу проводиться на рівні іменованих об'єк тів, а не самих збережених даних;
3. Уразливість по відношенню до шкідливих програм виду тро янських коней. Дискреційна модель дозволяє користувачам без обмежень передавати свої права іншим користувачам (що і використовується троянськими кіньми). Немає різниці між користувачем і суб'єктом, тобто між людиною, кому, в конеч- ном рахунку, були призначені певні права доступу до об'єктивним там і процесам, породженим даними користувачем. Це також дозволяє троянським коням, запущеним від імені авторізован- них користувачів, отримувати вільний доступ до даних.
4. Статичність розмежування доступу - права доступу до вже від- критого об'єкту в подальшому не змінюються незалежно від з- трансформаційних змін стану комп'ютерної системи;
5. Відсутність засобів захисту від витоку конфіденційної інфор мації. Інакше кажучи, діскреціонное розмежування доступу не забезпечує можливість перевірки, чи не призведе дозвіл доступу до об'єкту для деякого суб'єкта до порушення безпеки інформації в комп'ютерній системі;
6. Засоби захисту не дозволяють відстежити передачу секретних матеріалів;

7. Можливість множинного призначення і відкликання привілеїв доступу до одного і того ж об'єкту може привести до неконтрольованому доступу до даних. Припустимо, суб'єкт s1 надав певні права доступу до об'єкта o1 суб'єкту s2. Потім суб'єкт s3 надав ті ж привілеї до o1 все того ж суб'єкту s2, будучи не повідомленим, що це вже було зроблено суб'єктом s1. Пізніше суб'єкт s3 змінив свою думку і відкликав надані їм привілеї. Але його дей ствие не викликавши бажаний ефект, оскільки відкликані їм привілеї, як і раніше залишаються в матриці доступу, оскільки вони були раніше призначені суб'єктом s1;
8. При великій кількості користувачів важко відстежити всі пу- ти доступу.

Зазначених недоліків багато в чому позбавлене мандатний разграніче- ня доступу, що розглядається в наступному пункті.

Дискреційна модель є дуже популярною у розробників СУБД. Вона реалізована в практично всіх SQL-сумісних СУБД.

Оператори SQL GRANT, REVOKE, DENY, що реалізують діскреціон- ву модель розмежування доступу, визначені в стандарті мови SQL.

2. Забезпечення цілісності даних

Під цілісністю даних розуміють відповідність інформаційної моделі предметної області, тобто даних, що зберігаються в базі даних, об'єк єктам реального світу і їх взаємозв'язкам в кожен момент часу. Лю бій зміна в предметної області, значуще для побудованої моделі, має відображатися в базі даних, і при цьому повинна зберігатися одно- значная інтерпретація інформаційної моделі в термінах предметної області. Цілісність БД не гарантує достовірності що міститься в ній інформації, але забезпечує принаймні правдоподібність цієї інформації, відкидаючи свідомо неймовірні, неможливі значен ня. Таким чином, не слід плутати цілісність БД з достовірністю даних. Достовірність (або істинність) є відповідність фактів, хра- нящіхся в БД, реального світу.

2.1.Принципи забезпечення цілісності даних

Поняття «неналежне зміна» введено Д. Кларком і Д. Вільсоном: жодному користувачеві КС, в тому числі і авторизованому, не повинні бути дозволені такі зміни даних, які спричинять за собою їх руйнування або втрату. У роботах Кларка і Вільсона визна лени дев'ять абстрактних теоретичних принципів, виконання яких дозволить забезпечити цілісність даних:

- коректність транзакцій;
- авторизація користувачів;
- мінімізація привілеїв;
- розмежування функціональних обов'язків;
- аудит подій, що відбулися;
- об'єктивний контроль;
- управління передачею привілеїв;
- ефективне застосування механізмів захисту;
- простота використання захисних механізмів.

За першим принципом дані можуть змінюватися тільки за допомогою «Коректних» транзакцій. Пряме (довільним чином) зміна даних не допускається. У свою чергу коректність транзакцій повинна бути деяким способом доведена. Другий принцип говорить, що ви- ня даних може здійснюватися тільки авторизованими пользова- ками, що мають певні привілеї. Мінімальність прівіле- гий має на увазі, що користувачі (в кінцевому рахунку, суб'єкти) повинні бути наділені тими і тільки тими привілеями, які ми- мінімальними необхідні для виконання тих чи інших дій. Аудит подій, що відбулися (включаючи можливість відновлення

повної картини того, що сталося) є превентивним заходом щодо по-потенційних порушників і дозволяє відновити дані в разі їх пошкодження. Розмежування функціональних обов'язків подразуме-кість організацію роботи з даними таким чином, що в кожній з ключових стадій, що складають єдиний критично важливий з точки зору цілісності процес, необхідна участь різних пользовате- лей. Цим гарантується, що один користувач не може виконати весь процес цілком (або навіть дві його стадії) з тим, щоб порушити цілісність даних. Принцип об'єктивного контролю також є одним з наріжних каменів політики контролю цілісності. Суть даного принципу полягає в тому, що контроль цілісності даних має сенс лише тоді, коли ці дані відображають реальний положе-ня речей. Очевидно, що немає сенсу дбати про цілісність даних, пов'язаних з розміщенням бойового арсеналу, який вже відправлений на переплавку. У зв'язку з цим Кларк і Вільсон вказують на необхід ність регулярних перевірок, метою яких є виявлення віз мужніх невідповідностей між захищеними даними і об'єктивною реальністю, яку вони відображають. Управління передачею привілеїв необхідно для ефективної роботи всієї політики безпеки. Якщо схема призначення привілеїв неадекватно відображає організаційну структуру підприємства або не дозволяє адміністраторам безпеки гнучко маніпулювати нею для забезпечення ефективності виробниц-жавної діяльності, захист стає важким тягарем і провакує спроби обійти її там, де вона заважає «нормальної» роботі. В основу восьмого принципу контролю цілісності закладено низку ідей, покликаних забезпечити ефективне застосування наявних механізмів забезпечення безпеки. На практиці часто виявляється, що передбачені в системі механізми безпеки або некоректно яку вони відображають. Управління передачею привілеїв необхідно для ефективної роботи всієї політики безпеки. Якщо схема призначення привілеїв неадекватно відображає організаційну структуру підприємства або не дозволяє адміністраторам безпеки гнучко маніпулювати нею для забезпечення ефективності виробницької діяльності, захист стає важким тягарем і провакує спроби обійти її там, де вона заважає «нормальної» роботі. В основу восьмого принципу контролю цілісності закладено низку ідей, покликаних забезпечити ефективне застосування наявних механізмів забезпечення безпеки. На практиці часто виявляється, що передбачені в системі механізми безпеки або некоректно яку вони відображають. Управління передачею привілеїв необхідно для ефективної роботи всієї політики безпеки. Якщо схема призначення привілеїв неадекватно відображає організаційну структуру підприємства або не дозволяє адміністраторам безпеки гнучко маніпулювати нею для забезпечення ефективності виробниц-жавної діяльності, захист стає важким тягарем і провакує спроби обійти її там, де вона заважає

«нормальної» роботі. В основу восьмого принципу контролю цілісності закладено низку ідей, припокликаних забезпечити ефективне застосування наявних механізмів забезпечення безпеки. На практиці часто виявляється, що передбачені в системі механізми безпеки або некоректно Якщо схема призначення привілеїв неадекватно відображає організаційну структуру підприємства або не дозволяє адміністраторам безпеки гнучко маніпулювати нею для забезпечення ефективності виробниц-жавної діяльності, захист стає важким тягарем і провоці- рует спроби обійти її там, де вона заважає «нормальної» роботі. В основу восьмого принципу контролю цілісності закладено низку ідей, при-покликаних забезпечити ефективне застосування наявних механізмів забезпечення безпеки. На практиці часто виявляється, що передбачені в системі механізми безпеки або некоректно Якщо схема призначення привілеїв неадекватно відображає організаційну структуру підприємства або не дозволяє адміністраторам безпеки гнучко маніпулювати нею для забезпечення ефективності виробницької діяльності, захист стає важким тягарем і провакує спроби обійти її там, де вона заважає «нормальної» роботі. В основу восьмого принципу контролю цілісності закладено низку ідей, покликаних забезпечити ефективне застосування наявних механізмів забезпечення безпеки. На практиці часто виявляється, що передбачені в системі механізми безпеки або некоректно захист стає важким тягарем і провакує спроби обійти її там, де вона заважає «нормальної» роботі. В основу восьмого принципу контролю цілісності закладено низку ідей, при- покликаних забезпечити ефективне застосування наявних механізмів забезпечення безпеки. На практиці часто виявляється, що передбачені в системі механізми безпеки або некоректно захист стає важким тягарем і провакує спроби обійти її там, де вона заважає «нормальної» роботі. В основу восьмого принципу контролю цілісності закладено низку ідей, покликаних забезпечити ефективне застосування наявних механізмів забезпечення безпеки. На практиці часто виявляється, що передбачені в системі механізми безпеки або некоректно

використовуються, або повністю ігноруються системними адміністраторами. Простота використання захисних механізмів передбачає, що найбезпечніший шлях експлуатації системи буде також найбільш простим, і навпаки, найпростіший - найбільш захищеним.

На практиці найбільш часто використовуються дві моделі забезпечення цілісності даних, модель цілісності Кларка-Вільсона і модель Біба.

2.2.Модель Кларка-Вільсона

Модель цілісності Кларка-Вільсона була запропонована в 1987 р як результат аналізу практики паперового документообігу в комерческьких компаніях, ефективної з точки зору забезпечення цілісності інформації. Модель Кларка-Вільсона є описовою і не тримає яких би то ні було строгих математичних конструкцій - скоріше її доцільно розглядати як сукупність практичних рекомендацій з побудови системи забезпечення цілісності в КС. Основні поняття даної моделі - це коректність транзакцій і розмежування функціональних обов'язків. Модель задає правила функціонування КС і визначає дві категорії даних і два класи операцій над ними. Все, що містилося в системі дані підрозділяються на контрольовані і на неконтрольовані елементи даних. Цілісність перших підтримується, а цілісність других ніяк не контролюється. Введемо наступні позначення:

- S - безліч суб'єктів;
- D - безліч даних в автоматизованій системі (безліч об'єктів);
- CDI (Constrained Data Items) - дані, цілісність яких контролюється;
- UDI (Unconstrained Data Items) - дані, цілісність яких не контролюється;

При цьому D = CDI □ UDI, CDI □ UDI = Ø.

- TP (Transformation Procedure) - процедура перетворення, тобто компонент, який може ініціювати транзакцію - послідовність операцій, що переводять систему з одного стану в інший;
- IVP (Integrity Verification Procedure) - процедура перевірки целостності CDI.

Правила моделі Кларка-Вільсона:

1. В системі повинні бути IVP, здатні підтвердити цілісність будь-якого CDI. Прикладом IVP може служити механізм підрахунку контрольних сум;
2. Застосування будь-TP до будь-якого CDI має зберігати цілісність цього CDI;
3. Тільки TP можуть вносити зміни в CDI;
4. Суб'єкти можуть ініціювати тільки певні TP над певними CDI. Дана вимога означає, що система повинна підтримувати відносини виду (s, t, d), де s □ S, t □ TP, d □ CDI. Якщо відношення визначено, то суб'єкт s може примінити перетворення t до об'єкта d;
5. Повинна бути забезпечена політика поділу обов'язків суб'єктів - т. е. суб'єкти не повинні змінювати CDI без залучення в операцію інших суб'єктів системи;

6. Спеціальні ТР можуть перетворювати UDI в CDI;
7. Кожне застосування ТР має реєструватися в спеціальному CDI. При цьому:
   - даний CDI повинен бути доступний тільки для додавання інформації;
   - в даний CDI необхідно записувати інформацію, достатньо для відновлення повної картини функціонування системи;
8. Система повинна розпізнавати суб'єкти, які намагаються ініціювати ТР;
9. Система повинна дозволяти проводити зміни в списках авторізації тільки спеціальним суб'єктам (наприклад, адмін мусить зареєструвати безпеки). Дана вимога означає, що трійки (s, t, d) можуть модифікувати тільки певні суб'єкти;

Безумовними достоїнствами моделі Кларка-Вільсона є її простота і легкість спільного використання з іншими моделями безпеки.

2.3.Модель Біба

Модель Біба була розроблена в 1977 році як модифікація моделі Белла-ЛаПадули, орієнтована на забезпечення цілісності даних. Аналогічно моделі Белла-ЛаПадули, модель Біба використовує грати класів безпеки, що трактували в ній як грати класів цілісності.

Базові правила моделі Біба формулюються наступним чином:

1. Просте правило цілісності (Simple Integrity, SI). Суб'єкт з рівнем цілісності XS може читати інформацію з об'єкта з

   рівнем цілісності XO тоді і тільки тоді, коли XO переважання дає над XS.
2. * - властивість (star-integrity). Суб'єкт з рівнем цілісності XS може писати інформацію в об'єкт з рівнем цілісності XO тоді і тільки тоді, коли XS переважає над XO.

Для першого правила існує мнемонічне позначення No Read Down (NRD) - доступ на читання дається, якщо рівень цілісності (безпеки) об'єкта не нижче (а також включає в себе) рівень цілісності (безпеки) суб'єкта, а для другого No Write Up (NWU) - доступ на запис дається, якщо рівень цілісності (безпеки) суб'єкта не вище (а також включає в себе) рівня цілісності (безпеки) об'єкта. Отже, стан системи буде цілісним тоді і тільки тоді, коли воно безпечне з читання та запису.

Окремого коментарю заслуговує питання, що саме розуміється в моделі Біба під рівнями цілісності. Дійсно, в большості додатків цілісність даних розглядається як якась властивість, яка або зберігається, або не зберігається - і введення іерархіческім рівнів цілісності може

представлятися зайвим. У дей ствительности рівні цілісності в моделі Біба варто розглядати як рівні достовірності, а відповідні інформаційні потоки
- як передачу інформації з більш достовірної сукупності даних в менш достовірну і навпаки. Тобто, модель Біба грунтується на наступних припущеннях: чим вище рівень безпеки об'єкта, тим вище його достовірність і чим вище рівень безпеки суб'єкта, тим більш достовірну інформацію він може вносити в систему.

Формальний опис моделі Біба повністю аналогічно опису моделі Белла-ЛаПадули. До переваг моделі Біба слід віднести її простоту, а також використання добре вивченого математичного апарату. У той же час модель зберігає всі недоліки, властиві моделі Белла-ЛаПадули.

2.4.Спільне використання моделей безпеки

У реальних автоматизованих системах рідко зустрічаються системи захисту, орієнтовані виключно на забезпечення конфіденціальності або виключно на забезпечення цілісності інформації. Як правило, система захисту повинна поєднувати обидва механізму - а значить, при побудові та аналізі цієї системи буде необхідне спільне використання декількох формальних моделей безопасно- сті. Розглянемо як приклад можливі варіанти спільного використання моделей Белл-ЛаПадули і Біба.

1. Дві моделі можуть бути реалізовані в системі незалежно один від одного. В цьому випадку суб'єктам і об'єктам незалежно присваіваються рівні секретності і рівні цілісності;
2. Можливо логічне об'єднання моделей за рахунок виділення загальних компонентів. У разі моделей Біба і Белла-ЛаПадули таким загальним компонентом є порядок розмежування доступа в межах одного рівня секретності;
3. Можливе використання однієї і тієї ж решітки рівнів як для секретності, так і для цілісності. При цьому суб'єкти та об'єкти з високим рівнем цілісності будуть розташовуватися на низьких рівнях секретності, а суб'єкти та об'єкти з низьким рівнем цілісності - на високих рівнях секретності.

Якщо брати в розрахунок КС, то остання реалізація дозволяє, наприклад, розмістити системні файли на нижньому рівні ієрархії, що забезпечить їх максимальну цілісність, не акцентуючи увагу на зайвої в даному випадку секретності.

2.5.Оператори мови SQL завдання обмежень цілісності

Як ми вже знаємо з курсу «Моделі даних і СУБД», існують такі види обмежень:

1. Обмежувачі значень (domain constraints);
2. Обмежувачі ключів (key constraints);
3. Обмежувачі записів (entity constraints);
4. Посилальна цілісність (Declarative Referential Integrity, DRI).

Обмежувачі ключів задаються за допомогою заборони невизначених Null значень для ключових полів (по невизначеному значенню неможливо зробити ідентифікацію записи) і вимогою унікальності значення ключового поля. При порівнянні невизначених значен ний не діють стандартні правила порівняння: одне невизначене значення ніколи не вважається рівним іншому невизначеному значен ня. У SQL у фразі WHERE для виявлення рівності значення деякого атрибута невизначеному значенню застосовують спеціальні пре- дікати:

<Ім'я атрибута> IS NULL і <ім'я атрибута> IS NOT NULL

Унікальність значень будь-якого поля задається за допомогою оператора UNIQUE. Для оголошення унікальності сукупності полів приміняється конструкція виду:

UNIQUE (<поле1>, <поле2>, ...)Для оголошення первинного ключа використовується оператор PRIMARY KEY. Додатково, за допомогою специфічних для кожної СУБД оператора, може встановлюватися автонумерация ключового поля.

Обмежувачі записи також задаються за допомогою заборони невизначеним ленних значень, але вже не для ключових, але важливих в семантичному сенсі полів. Тоді, принаймні, щодо, які мають хоча б одна обов'язкова до заповнення поле, не буде повністю порожніх кор- тежів.

Посилальна цілісність забезпечує підтримку несуперечливого стану БД (узгодженого стану зовнішніх ключів) в процесі модифікації даних, а також при виконанні операцій додавання або видалення записів. У SQL для цього вводиться оператор FOREIGN KEY

FOREIGN KEY (<список полів>) REFERENCES <ім'я таблі- ци> (<список полів>) ON DELETE CASCADE ON UPDATE CASCADE.

Наприклад: FOREIGN KEY (client_id) REFERENCES clients (id) ON DELETE CASCADE

Обмежувачі ключів, записи і довідкова цілісність визначають правила роботи СУБД з реляційними структурами даних. Але з другого боку, ці аспекти ніяк не стосуються змісту бази даних. Для визначення деяких обмежень, які пов'язані зі зсмістом бази даних, вводяться обмежувачі значень. Обмежувачі значень задаються шляхом визначення типу поля (домену), завдання умови на значення, завдання варіанти вибору і

визначення значення за замовчуванням. Значення за замовчуванням задається за допомогою оператора

DEFAULT (<значення або вираз>) {FOR <ім'я поля>},

а обмежувачі значень вводяться за допомогою оператора CHECK (<умова>). Усередині CHECK можуть використовуватися оператори порівняння, функції IN, BETWEEN, LIKE і інші функції SQL. наприклад:

DEFAULT (GetDate ()) for date CHECK
(price between 0 and 100000) CHECK
(passport like 'MP% 20')

CHECK (місто in ('Мінськ', 'Москва', 'Київ')) або
CHECK (місто in (select capital from capitals))

Всі розглянуті вище обмежувачі цілісності можна напряму задати в інструкції визначення таблиці CREATE TABLE, наприклад (для MS SQL Server):

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

CREATE TABLE Books (Book_ID INTEGER IDENTITY (1,1) NOT NULL PRIMARY KEY, Name VARCHAR (100) NOT NULL, ISBN VARCHAR (14) UNIQUE NULL, PublisherID INTEGER NULL FOREIGN KEY REFER-

ENCES Publishers (Publisher_ID), Publ_Year SMALLINT DE- FAULT (Year (GetDate ()) CHECK (Publ_Year> = 1960 AND Publ_Year <= YEAR (GetDate ())), Pages SMALLINT NULL CHECK (Pages BETWEEN 5 AND 10000))

У наведених інструкціях оператор IDENTITY (1,1) визначає авто нумерацію записів. Перший його параметр дозволяє задати початкове значення, другий - приріст.

Для аналізу помилок доцільно іменувати все обмеження, особливо якщо таблиця містить кілька обмежень одного типу, так як ім'я обмеження виводиться в повідомленні порушення заданого обмеження. Для іменування обмежень використовується ключове слово CONSTRAINT, після якого слід унікальне ім'я обмеження, потім тип обмеження і його тіло, наприклад:

CONSTRAINT PK_BOOKS PRIMARY KEY (Book_ID)

CONSTRAINT DEF_PYEAR DEFAULT (Year (GetDate ()) FOR
Publ_Year CONSTRANT CH_PAGES CHECK (PAGES> = 5 AND
PAGES <= 10000)

Іменовані обмеження можна задати безпосередньо в інструкції CREATE TABLE (через кому, після визначення полів) або з допомогою оператора зміни таблиці:

ALTER TABLE <ім'я таблиці> ADD CONSTRANT <ім'я обмеження>

<Тіло обмеження>

Видалити іменоване обмеження можна за допомогою оператора:

ALTER TABLE <ім'я таблиці> DROP CONSTRAINT <ім'я обмеження> {CASCADE | RESTRICT}

3.Відновлення цілісного стану БД

Можливість відновлення забезпечується за рахунок накопичення з надлишкового інформації. По-перше, періодично виконується резерв- ве копіювання бази даних. По-друге, інформація про транзакції, що виконувалися в проміжку часу між двома послідовними копіюванням, зберігається в спеціальному файлі-журналі змін бази даних. По-третє, створюються контрольні точки, необхідні для синхронізації процесу відновлення і зменшення обсягу позову в файлі журналу. Ця інформація використовується для відновлення узгодженого стану БД після будь-якого збою. Оскільки транзакція є ключовим поняттям при організації відновлення цілістного стану БД, спочатку коротко розглянемо основні властивості транзакцій

3.1.Поняття транзакції

Під транзакцією розуміється виконувана від імені певного користувача або процесу послідовність операцій маніпулювання даними, яка переводить БД з цілісного початкового стану в цілісно кінцевий стан. При цьому проміжні стану БД не повинні бути обов'язково цілісними. Транзакції притаманні такі властивості, відомі як вимоги ACID (Atomicity, Consistency, Isolation і Durability):

Атомарність і узгодженість. Транзакція як одиниця роботи з даними має властивість атомарности (або виконуються всі передбачені транзакцією поновлення даних, або не виконується жодна) і узгодженості (цілісне початковий стан БД преобра- -зується в цілісне кінцевий стан). Транзакція не може бути виконана частково. Якщо в процесі обробки транзакції виникає якийй-небудь збій, то все виконане до цього моменту поновлення дан них повинно бути скасовані.

Довговічність. Транзакція має властивість довговічності: пролення даних, вироблені успішно завершилася транзакцією, стають постійними і не можуть бути загублені або скасовані ні за яких обставин. Кажуть, що транзакція завершилася успішно, якщо в процесі виконання не виникла якась аварійна ситуація, і всі перевірки обмежень цілісності дали позитивний резуль- тат. В цьому випадку всі вироблені поновлення даних зберігаються (фіксуються) в постійній пам'яті. Система гарантує фіксацію оновлень успішно завершилася транзакції навіть в тому випадку, якщо

в наступний момент часу відбудеться системний збій. У разі неуспішного завершення транзакції по будь-якої причини система гарантує відсутність будь-яких слідів її роботи у зовнішній пам'яті.

Ізольованість. Розрахована на багато користувачів система, як правило, одночасно підтримує сеанси роботи декількох користувачів.

Деякі з них можуть намагатися отримати доступ до одних і тих же даних в одному і тому ж інтервалі часу. У зв'язку з цим виникає проблема управління одночасним доступом до БД. Повинна бути вибрана така стратегія управління, яка забезпечувала б взаємну ізоляцію користувачів. Два одночасно працюючих користувача не будуть помічати один одного, якщо на результати роботи будь-якої транзакції кожного з них не впливатимуть дії паралельно виконуючих транзакції іншого. Кажуть, що транзакція має властивість ізольованості, якщо їй недоступні проміжні результати других, паралельно виконуються транзакції і її проміжні результати недоступні іншим транзакцій. Іншими словами, таким чином, поняття транзакції можна сформулювати як виконуваної від імені одного ідентифікатора суб'єкта неподільну, отже операцій над даними, що володіє властивостями атомарності, узгодженості, ізольованості і довговічності. У стандартах SQL визначені наступні оператори управління транзакціями: BEGIN TRANSACTION - повідомляє диспетчеру транзакцій про явне початку нової транзакції (неявно будь-яка інструкція поновлення даних
запускається в рамках транзакції);

COMMIT TRANSACTION - повідомляє диспетчеру транзакцій про успішне завершення транзакції і вимогу фіксації результатів у зовнішній пам'яті;

ROLLBACK TRANSACTION - повідомляє диспетчеру транзакцій про виникнення помилки і вимогу відкату проведених змін. Підтримка узгодженості забезпечується механізмом перевірки обмежень цілісності. Можна виділити два види обмежень цілостності: негайно перевіряються і відкладаються. Негайно перевіряючи обмеження цілісності, що зачіпають один об'єкт (відно шення або домен). Наприклад, не має сенсу відкладати перевірку обмеження цілісності сутності або цілісності значень при виконанні операції вставки кортежу в відношення. Негайно проверяються обмеження цілісності відповідають рівню окремих ін-трукцій поновлення. При їх порушеннях відкат транзакції не потрібен.

Досить відкинути відповідну інструкцію. Відкладаються обмеження цілісності - це обмеження, що зачіпають кілька відносин, або кортежів відносини. Їх називають обмеженнями на базу даних. Прикладом може служити обмеження посилальної цілістності або обмеження, що зачіпає кілька кортежів. Подібні обмеження повинні (і можуть) бути перевірені лише після виконання деякої логічно замкнутій групи операцій оновлення. Таким чином, відкладені обмеження відповідають рівню транзакції. Їх порушення викликає автоматичне скасування оновлень, тобто заміну COMMIT на ROLLBACK.

Підтримка ізольованості - одне з пріоритетних завдань диспетчера транзакцій. Ізольованість досягається, якщо об'єкт, обробляти ваемий транзакцією, не змінюється непередбачувано іншими транзакціями, виконаними в тому ж інтервалі часу. Існує два основних методу управління паралельною, що дозволяють організувати безпечне одночасне виконання транзакцій: метод блокування і метод тимчасових міток. Обидва ці методи відносяться до песимістичному підході управління транзакціями, коли відкладається виконання транзакцій, здатних в майбутньому увійти в конфлікт з іншими транзакціями. Оптимістичні методи ґрунтуються на припущенні, що ймовірність конфлікту невисока і тому вони допускають асинхронне виконання транзакцій. Перевірка конфліктів проводиться на етапі фіксації транзакцій.

3.2.Принципи відновлення даних

В основі відновлення даних лежать наступні два принципи:

1. Результати зафіксованих транзакцій повинні бути збережені у відновленому стані бази даних;

2. Результати незафіксованих транзакцій повинні бути відсутніми у відновленому стані бази даних.

Це, власне, і означає, що відновлюється останнім за часом узгоджене стан бази даних. Процедури восстанов- лення залежать від типу збою.

Перерахуємо ситуації, в результаті яких база даних може виявляється в неузгоджену стані:

локальний збій - це аварійне припинення транзакції. Причи- ною може бути, наприклад, спроба поділу на нуль або порушення обмежень цілісності, або тупикова ситуація. В цей же ряд слід поставити явне завершення транзакції оператором ROLLBACK. Ес чи транзакція виконувала оновлення даних, то стан БД в момент

локального збою виявиться неузгодженим. Для відновлення цілосності даних необхідно усунути зміни даних, вироблені перерваної транзакцією - зробити індивідуальний відкат транзакції.

м'який збій системи може статися, наприклад, внаслідок аварійного відключення живлення або при виникненні непереборного збою процесора і т.п. В цьому випадку втрачається вміст оперативної пам'яті. Аварійно перериваються всі існуючі транзакції. Можуть виявитися не зафіксованими в БД результати транзакцій, що завершився шихся оператором COMMIT. Після перезавантаження системи повинен бути виконаний відкат всіх не завершилися до моменту збою транзакцій. Закінчені, але незафіксовані транзакції повинні бути автоматично виконані повторно.

жорсткий збій - це фізичне руйнування бази даних. Ситуація дуже малоймовірна, але її наслідки для організації власника даних завжди катастрофічні. Тому система повинна бути в стані відновити базу даних навіть у цьому випадку.

У будь-якому випадку для відновлення узгодженого стану БД необхідна деяка інформація. Розглянемо, яка саме інформація необхідна для відновлення і як вона повинна використовуватися. Всі фрагменти БД, які обробляються транзакціями, зчитуються системою в робочі буфери бази даних в оперативній пам'яті. Транзакції виконують всі оновлення даних в цих буферах. Таким чином, стан робочих буферів відображає поточний стан тієї частини бази даних, яка доступна для діючих транзакцій.

Якщо відбувається локальний збій (або транзакція завершується оператором ROLLBACK), то необхідно відновити стан робочих буферів на момент початку транзакції. Для скасування оновлень достатньо точно мати повну інформацію про операції поновлення, виконаних транзакцією, а саме: тип операції, об'єкт поновлення, деталі оновлення. Тоді, виконавши в зворотному порядку зворотні за змістом операції, можна відновити стан змінювалися транзакцією буферів БД.

Таким чином, для відновлення узгодженого стану БД, при програмному відкат транзакції або після локального збою повинна зберігатися повна інформація про всі операції поновлення, виконаних транзакцією.

Ця інформація зберігається в спеціальному системному файлі - журналі реєстрації транзакцій. У журнал для кожної транзакції записуються її унікальний ідентифікатор і деталі всіх операцій оновлення данних, виконаних транзакцією. Крім того, кожної транзакції, завершеною штатно (тобто операторами COMMIT або ROLLBACK), представляється запис про закінчення транзакції.

Але в разі м'якого збою цього мало, так як м'який збій призводить до втрати вмісту робочих буферів БД. Потрібно ще мати у своєму розпорядженні з- стоянням БД на якийсь момент часу t0 і знати, які транзакції в цей момент існували в системі. Тоді, якщо в момент t1> t0 про- ізойдёт м'який збій, то для відновлення системи достатньо буде завантажити в оперативну пам'ять зафіксоване в момент t0 стан робочих буферів БД, проаналізувати записи журналу транзакцій, зроблених в інтервалі [t0, t1], і відновити успішно завершені або відкотити незавершені транзакції. З метою мінімізації звернень до пристроїв зовнішньої пам'яті програми формують записи файлів (ло- ня) в своїх робочих буферах ОЗУ. Скидання записів у зовнішнє пам'яті відбувається автоматично, коли буфер заповнюється.

Отже, для забезпечення можливості відновлення системи після

м'якого збою стан робочих буферів БД має періодично фіксуються у зовнішній пам'яті, і кожен запис журналу транзакцій повинен мати тимчасову мітку. Однак і цього недостатньо. Потрібно ще, щоб всі записи журналу від транзакцій, завершених до моменту збою, перебували у зовнішній пам'яті. Буфер журналу має розмір, достатній для розміщення записів від багатьох транзакцій. Велика вероґідність того, що до моменту м'якого збою він не буде скинутий у зовнішню пам'ять автоматично, і частина записів журналу буде загублена, по-цьому буфер журналу повинен примусово виштовхувати у зовнішню пам'ять при завершенні транзакції. Тільки після цього транзакція вважається закінченою. Оновлення, скоєні транзакцією, не можуть потрапити в зовнішню пам'ять раніше, ніж відповідна їй запис журналу. Це правило становить суть протоколу попереднім записом в журнал.

Відповідний протокол журналізації (і управління буферіза- цією) називається Write Ahead Log (WAL) - «пиши спочатку в журнал» і полягає в тому, що якщо потрібно записати на зовнішній пам'ять ізменен- ний об'єкт бази даних, то перед цим потрібно гарантувати запис у зовнішню пам'ять журналу транзакцій записи про його зміну. Іншими словами, якщо у зовнішній пам'яті бази даних знаходиться деякий об'єкт бази даних, по відношенню до якого виконана операція мо діфікації, то у зовнішній пам'яті журналу обов'язково знаходиться запис, відповідна цієї операції. Зворотне невірно, тобто якщо у зовнішньої пам'яті журналі міститься запис про деяку операції з-трансформаційних змін об'єкта бази даних, то сам змінений об'єкт може бути відсутній у зовнішній пам'яті бази даних.

Додаткова умова на виштовхування буферів накладається тим вимогою, що кожна успішно завершилася транзакція долж- на бути реально зафіксована у зовнішній пам'яті. Який би збій не сталося, система повинна бути в змозі відновити стан бази даних, що містить результати всіх зафіксованих до моменту збою транзакцій. Простим рішенням було б виштовхування буфера журналу, за яким слід масове виштовхування буферів сторінок бази даних, змінювалися даної транзакцією. Досить часто так і роблять, але це викликає суттєві накладні витрати при виконан неніі операції фіксації транзакції. Виявляється, що мінімальною вимогою, що гарантує можливість відновлення останнього узгодженого стану бази даних, є виштовхування при фісації транзакції в зовнішню пам'ять журналу всіх записів про зміни ванні бази даних цієї транзакцією. При цьому останнім записом в журнал, виробленої від імені даної транзакції, є спеціальний запис про кінець транзакції. Слід зазначити, що швидкість запису в файл журналу може виявитися одним з найважливіших факторів, визна ляющих загальну продуктивність роботи з БД.

Подивимося тепер, як можна відновлювати стан бази дан них, якщо в системі підтримується журнал транзакцій і протокол попереднім записом.

Індивідуальний відкат транзакції. Індивідуальний відкат проводиться або по явно заданому оператору ROLLBACK, або як слідство локального збою. Для здійснення відкату створюється список записів цього журналу від даної транзакції. Елементи списку розміщені в порядку, зворотному хронологічному. Список послідовно просматрюється і для кожного запису виконується протилежна за змістом операція, що відновлює попередній стан об'єкта бази дан них. Початковим станом для процедури відкоту є стан буферів БД в момент припинення транзакції. З точки зору системи ця процедура є транзакцією. Тому зворотні операції також реєструються в журналі. Це "перестраховка" на випадок м'якого збою системи в процесі відкоту. Маючи ці записи, система зможе «доотка- тить» транзакцію, відкат якої був перерваний.

Відновлення після м'якого збою.Робочі буфери бази даних, на відміну від буфера журналу, які не виштовхуються в зовнішню пам'ять при кожному успішному завершенні транзакції. Реально виконання опера-

тора COMMIT зводиться до того, що стає неможливою скасування проблених транзакцій змін. Сам же змінений об'єкт може ще довго існувати тільки в робочому буфері БД. Може трапитися так, що в системі відбудеться збій після успішного виконання COMMIT, але до того, як оновлений транзакцією об'єкт потрапить у зовнішню пам'ять. Відповідно до принципу довговічності транзакції система повинна при перезавантаженні встановити ці оновлення в БД, незважаючи на те, що їх вже немає в буфері оперативної пам'яті.

Крім того, система може тимчасово поміщати в БД проміжні результати транзакцій, якщо необхідно звільнити місце в буферах. При нормальній роботі ці неузгоджені зміни по завершенні внесла їх транзакції відкидаються. Оскільки м'який збій перериває всі транзакції, проміжні результати виявляються "зафіксіро- ванними". При перезавантаженні система повинна відкотити все перервані транзакції і виконати повторно все успішно завершилися, але не зафіксовані в БД. При цьому транзакції, перервані або завер шівшіеся оператором ROLLBACK до м'якого збою, в процесі відновлення не беруть участь.

Поточний стан БД в момент збою не може бути використано як опорна для відновлення, оскільки невідомо, до якого моменту часу воно відноситься. Для того щоб мати таке опорне перебування БД, використовується механізм створення контрольних точок. Протягом деякого інтервалу часу СУБД чекає завершення чергових операцій

оновлення у всіх транзакціях і не допускає запуску нових операцій. Виконання діючих транзакцій призупиняється. У момент прийняття контрольної точки, коли всі транзакції пріостанов- лени, в журналі створюється спеціальний запис контрольної точки, з- тримає список всіх транзакцій, що існують в системі. Потім в зовнішню пам'ять примусово виштовхується вміст буфера журналу і робочих буферів БД. По завершенні цього процесу ісполне- ня призупинених транзакцій триває. Відзначимо,

В процесі відновлення виконуються наступні дії.

- Створюється два списки транзакцій: відміняються (UNDO) і виняемое повторно (REDO). У список UNDO включаються всі тран-ЗАКЦ, зазначені в запису контрольної точки. Список REDO залишається порожнім.
- Виконується аналіз записів журналу реєстрації, починаючи з записи контрольної точки.
- Якщо виявлена запис про початок транзакції Т, то ця транзакція буде додано до списку UNDO.
- Якщо виявлена запис про завершення транзакції Т оператором COMMIT, то ця транзакція буде додано до списку REDO.
- Журнал реєстрації проглядається від кінця до запису контрольної точки, і скасовуються транзакції зі списку UNDO.
- Журнал реєстрації проглядається від запису контрольної точки до кінця, і виконуються повторно транзакції зі списку REDO.

Після закінчення цієї процедури система готова до роботи.

Відновлення після жорсткого збою. При жорсткому збої фізична база даних виявляється зруйнованою. Тому для відновлення необхідно мати її резервну копію. Зазвичай резервне копіювання БД виконується за фактом переповнення журналу транзакцій. Для цього в файлі журналу встановлюється так звана «жовта зона», по до- сягнення якої запуск нових транзакцій не проводиться. Система чекає закінчення всіх існуючих транзакцій. Після цього робочі буфери журналу і бази даних виштовхуються в зовнішню пам'ять. Створене таким чином стан БД копіюється на резервний носій, а файл журналу очищується. Може бути також створена резервна копія журналу.

При відновленні після жорсткого збою відновлюється БД на момент останнього копіювання, а потім за поточним журналу реєстрації повторно виконуються всі транзакції, успішно завер шівшіеся до моменту збою. Оскільки жорсткий збій може не супроводжуватися втратою буферів, після відновлення можна навіть продовжити виконання не завершилися до моменту збою транзакцій. Проте, всі незавершені

транзакції зазвичай відкочуються.

3.3.Методи відновлення

Розглянемо два методи відновлення, які можуть бути примі- нени в разі, тобто коли БД не була повністю зруйнована, але лише втратила узгоджене стан. Метод відкладеного оновлення та метод негайного поновлення відрізняються один від одного способом внесення оновлень у зовнішню пам'ять.

Метод відкладеного оновлення. При використанні цього методу оновлення не заносяться в БД до тих пір, поки не почнеться фаза фіксацію цієї транзакції. Якщо виконання транзакції буде припинено до досягнення цієї точки, ніяких змін в БД виконано не буде, поки

цього не буде потрібно і їх скасування. Однак в такому випадку може бути потрібний повторний прогін вже завершених транзакцій, оскільки їх результати могли ще не досягти зовнішньої пам'яті. Протокол відновлення буде складатися з наступних дій.

При запуску транзакції в протоколі можна відслідковувати запис: Початок транзакції. При виконанні операції вставки або поновлення даних в файл журналу записуються нові значення всіх змінених елементів даних (відзначимо, запис значень елементів даних до оновлення не потрібно). Реально запис змін в саму БД не проводиться. Коли транзакція досягає своєї кінцевої точки, в протоколі можна відслідковувати запис: Транзакція завершена. Всі записи журналу по даній транзакції виводяться в зовнішню пам'ять, після чого виконується фіксація внесених транзакцією змін. Для внесення дійсних зраді ний в БД використовується інформація, вміщена в файл журналу.

У разі скасування виконання транзакції, записи журналу по даній транзакції ігноруються, і зроблені зміни не фіксуються у зовнішній пам'яті. Відповідно до принципу WAL записи журналу про виконану транзакції зберігаються в зовнішній пам'яті до того, як результати транзакції будуть зафіксовані. Тому, якщо відмова БД відбудеться повторно в процесі дійсного виконання обнов- лений в БД, розміщені в журналі відомості будуть збережені і вима- ються поновлення можна буде виконати пізніше. У разі відмови файл журналу аналізується з метою виявлення транзакцій, які знаходь- лись в процесі виконання в момент відмови, починаючи з останньої будів ки. Файл журналу проглядається в зворотному напрямку аж до запису про останньої виконаної контрольній точці. Усі транзакції, для яких у файлі журналу присутні записи Початок транзакції і Транзакція завершена, повинні бути виконані повторно. Процедура повторного прогону транзакцій виконує всі операції запису в БД, використовуючи інформацію про стан

елементів даних після обновлення, що міститься в записах журналу по даній транзакції, причому в тому порядку, в якому вони були записані в файлі журналу. Якщо опера ції записи вже були успішно завершені до виникнення відмови, це не матиме ніякого впливу на стан елементів даних, оскільки вони не можуть бути зіпсовані, якщо будуть записуватися ще раз. Слідів- вательно, даний метод гарантує, що будуть оновлені будь еле менти даних, які не були коректно оновлені до моменту від- за. Процедура повторного прогону транзакцій виконує всі операції запису в БД, використовуючи інформацію про стан елементів даних після обновлення, що міститься в записах журналу по даній транзакції, причому в тому порядку, в якому вони були записані в файлі журналу. Якщо опера ції записи вже були успішно завершені до виникнення відмови, це не матиме ніякого впливу на стан елементів даних, оскільки вони не можуть бути зіпсовані, якщо будуть записуватися ще раз. Отже, даний метод гарантує, що будуть оновлені будь еле менти даних, які не були коректно оновлені до моменту від- за. Процедура повторного прогону транзакцій виконує всі операції запису в БД, використовуючи інформацію про стан елементів даних після обновлення, що міститься в записах журналу по даній транзакції, причому в тому порядку, в якому вони були записані в файлі журналу. Якщо операції записи вже були успішно завершені до виникнення відмови, це не матиме ніякого впливу на стан елементів даних, оскільки вони не можуть бути зіпсовані, якщо будуть записуватися ще раз. Отже, даний метод гарантує, що будуть оновлені будь-які елементи даних, які не були коректно оновлені до моменту від- за.

Будь-яка транзакція, для якої в файлі журналу присутні записи Початок транзакції і Скасування транзакції, просто ігнорується, оскільки ніяких реальних оновлень інформації в БД по ній не виконуваних, а значить, не потрібно і реального виконання їх відкату. Ес чи в процесі відновлення виникне інший системний збій, записи файлу журналу можуть бути використані для відновлення БД ще раз. В такому випадку не має значення, скільки разів кожна з рядків жур- налу була використана для повторного внесення змін до БД.

Метод негайного поновлення. При використанні цього прото кола всі зміни вносяться в БД відразу ж після їх виконання в тран- ЗАКЦ, перш ніж вона закінчиться. Крім необхідності повторного прогону змін, виконаних транзакціями, що закінчилися до появи збою, в даному випадку може знадобитися виконати відкат змін, внесених транзакціями, які не були завершені до моменту.

При застосуванні даного методу файл журналу використовується з

метою відновлення в такий спосіб. При запуску транзакції в протоколі можна відслідковувати запис: Початок транзакції. При виконанні будь-опера ції поновлення БД проводиться запис в файл журналу всіх вихідних і кінцевих значень всіх змінених елементів даних. Після виконан ня записи в файл журналу виробляються виконання оновлень в буфері БД. Безпосередньо в БД зміни будуть внесені при очеред- ної вивантаженні буферів БД в зовнішню пам'ять. Коли транзакція завершує своє виконання, в файл журналу заноситься запис: Транзакція за-вершена.

В даному методі дуже важливо, щоб в файл журналу все записи по-міщені до внесення відповідних змін до БД. Якщо зраді ня спочатку будуть внесені в БД і збій в системі виникне до размеще ня інформації про це в файл журналу, то менеджер відновлення не буде мати можливості скасувати (або повторити) дану опера цію. При використанні протоколу попереднім записом журналу менеджер відновлення завжди зможе безпечно припустити, що якщо для певної транзакції в файлі журналу відсутній запис Транзакція завершена, значить, ця транзакція була активна в момент виникнення відмови і, отже, повинна бути скасована. Якщо ви-конання транзакції було припинено, то для скасування виконаних нею змін використовуються відомості про вихідних значеннях всіх вимірюв- наних елементів даних, збережених у файлі журналу. Оскільки транзакція може виконати кілька змін одного і того ж елемента, скасування оновлень виконується в зворотному порядку. Незалежно від того, чи були результати виконання транзакції внесені в саму БД, наявність в записах журналу вихідних значень полів гарантірует, що БД буде приведена в стан, що відповідає початку скасованої транзакції.

На випадок повторної відмови системи процедурою відновлення передбачено використання файлу журналу для повторного прогону або відкату транзакцій. Для будь-якої транзакції Т, для якої в файлі журналу присутні записи Початок транзакції і Транзакція завер шена, слід виконати її повторний прогін, використовуючи для внесення в БД нових значень всіх оновлених полів. Для будь-якої транзакції, для якої в файлі журналу присутній запис Початок транзакції, але немає запису Транзакція завершена, необхідно виконати відкат внесен- них нею змін. На цей раз із записів файлу журналу витягується інформація про значення змінених полів до їх зміни, що дозволяє привести базу даних в стояння, яке вона мала до початку дан ної транзакції. Операції скасування виконуються в порядку, зворотному порядку їх запису в файл журналу.

3.4.Організація відновлення даних в СУБД MS SQL Server

SQL Server пропонує на вибір три моделі відновлення, в основ- ном відрізняються використанням журналу транзакцій, і п'ять варіанта тов резервного копіювання. Моделі відновлення наступні:

1. Проста модель. Журнал транзакцій резервується (НЕ создает- ся його резервна копія у зовнішній пам'яті);
2. Модель з неповним протоколированием. Масові операції не записується в журнал транзакцій. Журнал транзакцій резервіруется;
3. Повна модель. Усі транзакції заносяться в журнал. Журнал тран- ЗАКЦ резервується.

Резервне копіювання можливо наступних видів:

- Полное. Резервуються всі дані;
- Диференційоване. Проводиться резервування всіх сторінок даних, змінених з моменту останнього повного резервного копіювання;
- Журналу транзакцій. Проводиться резервування всіх транзак- ций в журналі.
- Файлу або файлової групи. Проводиться резервування всіх даних, що містяться у файлі або файлової групі.
- Файлове диференційоване. Проводиться резервування всіх сторінок даних, модифікованих з моменту останнього резервного копіювання файлу або файлової групи.

Відновлення завжди починається з використання повної резервної копії. Після цього з архівів диференційованого і транзакційного резервування відновлюються всі транзакції, виконані з мо- мента створення повної резервної копії. Модель відновлення конфі- гурірует настройки бази даних SQL Server таким чином, щоб забезпе-чити той тип відновлення, який необхідний базі даних. Ключові відмінності між різними моделями відновлення пов'язані з тим, в якій мірі в них задіяний журнал транзакцій і які дані в ньому реєструються.

Проста модель відновлення ідеально підходить тим баз дан них, яким потрібне забезпечення атомарности транзакцій, але не обов'яз- ково підтримка їх живучості. Проста модель форсує очищення сервером баз даних журналу в контрольних точках. При цьому журнал буде зберігати транзакції, запис яких в базу даних ще не підтверджене; простір ж, відведений для зберігання всіх інших транзакцій, звільняється для повторного використання. Так як журнал транзакцій в цій моделі є тільки тимчасовим місцем хране- ня, відпадає потреба в його резервування. Ця

модель восстанов- лення має ряд переваг. По-перше, журнал транзакцій має ма ленькі розміри, однак розплачуватися за це доведеться втратою всіх транзакцій, виконаних з моменту останнього повного або дифференцарювати резервування. План відновлення,

стій моделі, дозволяє виконувати повне резервування раз в тиждень, а диференційоване - в кінці кожного дня тижня. Повна і диффе- ренціpованние резервні копії замінюються, коли буде виконано наступне повне резервне копіювання.

Відповідно, відновлення в простій моделі передбачає дві операції.

1. Відновлення з останньої повної резервної копії;
2. Відновлення з останньої (не обов'язково) однієї дифференцарювати резервної копії.

Повна модель відновлення пропонує найбільш грубий і надіжак- ний план відновлення. У цій моделі все транзакції, в тому числі і масові операції, записуються в журналі. Будь-яка системна функція, така як створення індексу, також протоколюється. Основною перевагою цієї моделі є те, що всі транзакції, виполнен- ні в базі даних, можуть бути відновлені, аж до моменту, що передував системному збою. До недоліків слід віднести повільне виконання масових операцій, дуже великий розмір файлу журналу транзакцій і більш тривалий час виконання операцій резервіpованія і відновлення журналу транзакцій.

Повна модель відновлення може використовувати всі п'ять типів резервування бази даних. Найчастіше повна модель відновлення передбачає виконання повного резервування двічі в тиждень, а диференційованого - в кінці кожного дня. Резервування журналу транзакцій виконується регулярно протягом усього дня; при цьому проміжки часу між окремими сесіями можуть коливатися від декількох годин до декількох хвилин. Відновлення бази даних в цій моделі передбачає виконання таких операцій.

- Відновлення з останньої повної резервної копії;
- Відновлення з останньої диференційованої резервної ко пії, створеної після виконання останнього повного резервіро- вання (якщо така є);
- Відновлення всіх резервних копій журналу транзакцій, з- будівлі з моменту останнього повного або диффеpенціpованно- го резервного копіювання.

Якщо останній створеної резервної копією була повна, то її восстановлення буде досить. Якщо останній створеної резервної ко пією була диференційована, то перед її відновленням слід виконати відновлення з останньої повної резервної копії.

Модель відновлення з неповним протоколированием аналогічна повної моделі, за винятком того, що в журнал транзакцій зано- сятся такі операції:

- масові вставки;
- інструкції DML SELECT * INTO;
- операції над особливо великими об'єктами WRITETEXT і UPDATETEXT;
- інструкції CREATE INDEX (включаючи індексовані перед- дання).

Компроміс в продуктивності операцій в цій моделі восста- лення досягається за рахунок того, що масові операції НЕ розглядають ваются як транзакції. Оскільки масові операції в даній моделі відновлення не реєструються, якщо збій стався після масової операції, але до створення архіву транзакцій, масова операція буде загублена і у відновленні зможе брати участь тільки попередня резервна копія журналу транзакцій. Виходячи з цього, якщо використовується модель відновлення з неповним протоколированием, за кожною мас- совою операцією повинно негайно слідувати створення резервної ко пії журналу транзакцій.

Ця модель відновлення може виявитися корисною тільки в тому випадку, якщо в базі даних виконується велика кількість масових операцій і дуже важливо підвищити їх продуктивність. Якщо ж про- тивність виконання масових операцій в базі даних достатньо точно висока, то краще зупинити свій вибір на повній моделі відновлення.

Модель відновлення, встановлена в системній базі model, при- змінюється до всіх створюваним баз даних. Для установки моделі відновлення використовується інструкція DDL ALTER DATABASE:

ALTER DATABASE імя_бази_данних SET Recovery параметр

Допустимими параметрами в цій інструкції є наступні: Full, Bulk-Logged і Simple. У наступному прикладі модель відновленя навчальної бази даних Test змінюється на повну:

USE Test

ALTER DATABASE Test SET Recovery Full

Настійно рекомендується в програмному коді, що створює базу даних, явно вказувати модель відновлення. Поточну модель вос- становлення, встановлену в базі даних, можна визначити з допомогою на гою подання каталогу sys.database:

SELECT name, recovery_model_desc FROM sysdatabases

Незважаючи на те що в базі даних зазвичай встановлюється лише одна модель відновлення, ніхто не забороняє переключатися між різними моделями в залежності від складу виконуваних операцій з метою оптимізації продуктивності і задоволення поточних по- потреб.

Для виконання операції резервування бази даних є уті- літа backup.

Якщо не брати до уваги безліч додаткових параметрів, то загальний вигляд команди резервного копіювання буде наступним:

BACKUP DATABASE
імя_бази_данних TO DISK =
'шлях_до_файлу'

WITH NAME = 'імя_архіва'

Наступна інструкція резервує базу даних Test в файл на дис- ке і привласнює архіву ім'я TestBackup:

BACKUP DATABASE Test

TO DISK = 'e: \ TestBackup.bak'
WITH NAME = 'TestBackup'

Журнал транзакцій також підлягає резервуванню. Віртуально журнал можна уявити собі як послідовний список транзак- ций, відсортованого за датою створення. У той же час на фізичному рівні SQL Server записує різні частини фізичного журналу в віртуальні блоки без будь-якого певного порядку. Всі транзакції в журналі можна розділити на дві групи: активні транзакції (ті, які ще не підтверджені і не записані в файл бази даних), і неактивні транзакції, які передують самій ранній активної транзакції. Активна частина журналу не обов'язково містить тільки непідтверджені транзакції, вона містить всі транзакції з моменту початку найстарішою непідтвердженою транзакції. Всього одна дуже стара непідтверджена транзакція може зробити активної дуже велику частину журналу.

Виконання резервування журналу транзакцій практично не відличается від повного або диференційованого резервного копіювання. Інструкція T-SQL має наступний вигляд:

BACKUP LOG Test

TO DISK = 'e: \ TestBackupLog.bak'
WITH NAME = 'TestBackupLog'

При резервуванні журналу транзакцій використовуються ті ж параметри, які використовуються при резервному копіюванні бази даних. Журнал транзакцій не може бути зарезервований, якщо в базі даних

використовується проста модель відновлення, або в базі даних використовується модель відновлення з неповним протоколированием, або були пошкоджені файли бази даних. У будь-якому з цих випадків слід виконати повне резервування бази даних.

Зазвичай резервне копіювання бази даних виконується по строго визначеним графіком, однак неможливо заздалегідь передбачити, скільки диференційованих і резервних копій і копій транзакцій має бути відновлено. Так як кожен файл резервної копії вимагає окремої інструкції RESTORE, неможливо створити коррект- ний код, не включивши в нього обстеження бази даних msdb і створення правильної послідовності відновлення. Інструкція RESTORE може виконувати відтворення бази з повною, диференційованою і транзакционной резервної копії. Її загальний синтаксис наведено нижче.

```
RESTORE DATABASE | LOG
імя_бази_данних FROM
устройство_резервірованія

WITH FILE = номер_файла, PASSWORD = пароль,
NORECOVERY | RECOVERY
```

Для відтворення повної або диференціальної резервної копії використовується інструкція RESTORE DATABASE; якщо відтворюється журнал транзакцій, використовується інструкція RESTORE LOG. Параметр WITH FILE дозволяє задати номер відновлюваної резервної копії у файлі або на пристрої резервування. Параметри RECOVERY / NORECOVERY є життєво важливими в інструкції відтворення. При кожному запуску SQL Server автоматично перевіряється журнал транзакцій. При цьому відкочуються все непідтверджені транзакції і доводяться до кінця все підтверджені. Цей процес получіл назву відтворення (recovery) бази даних. Таким чином, якщо в інструкції відтворення вказано параметр NORECOVERY, SQL Server відтворить журнал, що не обробивши при цьому жодної транзакції; якщо ж в інструкції буде вказано параметр RECOVERY, транзакції будуть проведені. наприклад,

```
Use Master

RESTORE DATABASE Test

FROM DISK = 'e: \ TestBackup.bak'
WITH FILE = 1, NORECOVERY
```

Продовжуємо відновлення:
RESTORE LOG TestBackupLog

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

3.5Створення відмовостійких систем

Для створення відмовостійких систем вищеперелічених засобів відновлення недостатньо, оскільки потрібно забезпечити безперервне ний режим роботи сервера. Для підвищення надійності та (або) виводительности дискової підсистеми жорсткі диски об'єднуються в RAID масиви. Найбільш часто використовуваними є RAID масиви рівнів 0, 1, 5, 6 і 10.

RAID 0(Stripe). Режим, при використанні якого досягається максимальна продуктивність, але не надійність. Дані рівномірний але розподіляються по дисках масиву, диски об'єднуються в один, ко торий може бути розмічений на кілька. Розподілені операції читання і запису дозволяють значно збільшити швидкість роботи, оскільки кілька дисків одночасно читають / записують свою порцію даних. Користувачеві доступний весь обсяг дисків, але це знижує надійність зберігання даних, оскільки при відмові одного з дисків масив зазвичай руйнується і відновити дані практично невоз- можна.

RAID 1(Mirror). Кілька дисків (зазвичай 2), що працюють синхроно на запис, тобто повністю дублюють один одного. Вище-ня продуктивності відбувається тільки при читанні. Самий надійний спосіб захистити інформацію від збою одного з дисків. Через високу вартість зазвичай використовується при зберіганні дуже важливих даних. Висока вартість обумовлена тим, що лише половина від загальної ємності дисків доступна для користувача.

RAID 10. Іноді також називається RAID 1 + 0, так як є комбінації двох перших варіантів (масив RAID 0 з масивів RAID 1). Має всі швидкісні переваги RAID 0 і перевага надежно- сти RAID 1, зберігаючи недолік - високу вартість дискового масиву, так як ефективна ємність масиву дорівнює половині ємності використана в ньому дисків. Для створення такого масиву потрібно ми- німум 4 диска (їх число повинне бути парним).

RAID 5.Масив, також використовує розподілене зберігання даних аналогічно RAID 0 (і об'єднання в один великий логічний диск) плюс розподілене зберігання кодів парності для відновлення даних при збої. Можливо як одночасне читання, так і запис. Плюсом цього варіанту є те, що доступна для користувача ємність масиву зменшується на ємність лише одного диска, хоча надїжак- ність зберігання даних нижче, ніж у

RAID 1. По суті, є комісії між RAID 0 і RAID 1, забезпечуючи досить високу ско- кість роботи при непоганий надійності зберігання даних. Мінімальна

кількість дисків для такого масиву - 3. У разі відмови одного диска з масиву дані можуть бути відновлені без втрат в автоматичному режимі, хоча сам процес відновлення відбувається з повною нагруз- кой на робочі вінчестери, що при перегріванні може привести до повно- му виходу томи з ладу.

RAID 6. RAID 6 відрізняється від RAID 5 тим, що в кожному ряду дан них (по-англійськи stripe) має не один, а два блоки контрольних сум. Контрольні суми - «багатовимірні», тобто незалежні один від одного, тому навіть відмова двох дисків в масиві дозволяє зберегти результат- ні дані. Обчислення контрольних сум за методом Ріда-Соломона вимагає більш інтенсивних порівняно з RAID 5 обчислень, поет- му раніше шостий рівень практично не використовувався. Зараз він підтримується багатьма продуктами, так як в них стали встановлювати спеціалізовані мікросхеми, які виконують всі необхідні ма- тематичні операції. Згідно з деякими дослідженнями, відновлено ня цілісності після відмови одного диска на томі RAID 5, состав- ленном з дисків великого обсягу (500 гігабайт і вище), в 5% випадків закінчується втратою даних. Іншими словами, в одному випадку з два- дцати під час регенерації масиву RAID 5 на диск резерву можливий вихід з ладу другого диска. RAID 6 і був покликаний вирішити вказаний недолік RAID 5.

Для запобігання простою сервера під час відновлення дан них використовується технологія створення резервних серверів (Database Mirroring). Вона дозволяє захиститися від незапланованого простою, викликаного відмовою сервера або збоєм бази даних і, як випливає з її назви, забезпечує відмовостійкість на рівні бази даних. Дану технологію можна використовувати для однієї або декількох баз даних знаходяться на одному і тому ж примірнику SQL Server. При її активації резервна база даних буде завжди оновлюватися поточними транзакціями, виробленими на основному сервері баз даних. При цьому, що дуже важливо, Database Mirroring із мінімальним впливом на продуктивність.

Database Mirroring забезпечує захист від збоїв і дискової підсистеми, так як і основний і резервний сервери мають свої власні дискові підсистеми. Database Mirroring реалізована за допомогою трьох систем: основний сервер, резервний сервер і сервер-свідок.

Основний сервер (primary server) забезпечує сервіс баз даних в даний час. За замовчуванням, всі клієнтські підключення відбуваються саме до основного сервера. Завдання резервного сервера (secondary server) полягає у підтримці резервної копії основного сервера. свиде-

тель (witness), діючий незалежно, відповідає за те, який з серверів зараз є основним.

Віддзеркалення баз даних працює наступним чином. Коли клі- ентское додаток записує транзакцію на основний сервер, то перед зміною файлу даних відбувається запис змін в журнал транзакцій. Потім ці записи журналу транзакцій відправляються на резервний сервер в журнал транзакцій дзеркальної бази даних. Після того як резервний сервер зафіксує ці записи в журналі транзакцій він відправляє підтвердження основного сервера. Це дозволяє обом системам знати, що запис була прийнята і обидва журналу транзакцій мають одні і ті ж дані. У разі операції фіксації, основний сервер чекає підтвердження від резервного сервера і тільки після цього він спрямовує відповідь клієнтського додатку, кажучи про те, що дана опе- рація завершена. Якщо додаток втрачає підключення до основного сервера, система зробить ще одну спробу підключення до основного сервера. Якщо підключення не відбудеться, наступна спроба підключення буде проведена вже до резервного сервера.

4. Захист даних за допомогою уявлень, зберігання процедур, функцій і тригерів

Збережені процедури, функції і тригери є фрагментами коду на високорівневої мовою програмування, являє собою розширення мови SQL, що зберігається і виконуються на сервері. Реалізація всіх правил «бізнес логіки» за допомогою збережених процедур, функцій і тригерів надає значно більше високий рівень захисту даних, ніж їх реалізація в клієнтських про грамах.

4.1.визначення уявлень

Уявлення (VIEW), являють собою тимчасові, (інакше вір- туальной) таблиці і є об'єктами бази даних, інформація в яких не зберігається постійно, як в базових таблицях, а формується динамічно при зверненні до них. По суті, уявлення - це іменний запит, що зберігається в базі даних. Подання не вимагає для свого зберігання дискової пам'яті, за винятком пам'яті, необхідної для зберігання визначення самого уявлення. Виставу не може існувати саме по собі, а визначається тільки в термінах однієї або декількох таблиць. Застосування уявлень дозволяє розробнику Чику бази даних створити інтерфейс програми, що не залежить на всі 100% від реально існуючих таблиць, а також надати кожному користувачу або групі користувачів обмежений набір даних, приховуючи поля і записи з конфіденційною інформацією. Уявлення дозволяють обмежити доступ до даних на рівні записів, доповнення, таким чином, дискреционную модель розмежування доступу. Поведінка і зовнішній вигляд уявлення не відрізняється від базової табліці. У користувача створюється враження, що він працює зі справжньою, реальною таблицею.

Оператори створення, зміни і видалення уявлень в стані мови та реалізації в MS SQL Server збігаються і представлені надають командами:

```
{CREATE | ALTER} VIEW імя_уявлення
    [(ім'я_стовпця [... n])] [WITH ENCRYPTION] AS
    SELECT ... [WITH CHECK OPTION]

DROPVIEWімя_уявлення [... n]
```

Розглянемо призначення основних параметрів. За замовчуванням імена полів поданні відповідають іменам полів у вихідних таблицях. Явна вказівку імені поля потрібно для обчислюваних полів або при об'єднанні кількох таблиць, що мають поля з однаковими іменами. Імена полів перераховуються через кому, відповідно до порядку їх слідування в

поданні.

Наприклад, відобразимо в поданні клієнтів з Києва:

```
CREATE VIEW Києвляни AS

SELECT КодКліента, Прізвище, ГородКліента
FROM Клієнт WHERE ГородКліента = 'Київ'
```

Звернення до подання здійснюється за допомогою оператора SELECT, як і до звичайної таблиці:

```
SELECT * FROM Києвляни
```

Подання можна використовувати так само, як і будь-яку іншу таблицю. До подання можна будувати запит, оновлювати дані (якщо виконуються певні вимоги), з'єднувати з іншими таблицями. Зміст уявлення не фіксоване і оновлюється кожного разу, коли на нього посилаються в команді. Уявлення значно розширюють можливості управління даними. Зокрема, це чудовий спосіб дозволити доступ тільки до певної інформації в таблиці, приховавши частину даних. Так, в наведеному вище прикладі Києвляні обмежує доступ користувача до даних таблиці Клієнт, дозволяючи бачити тільки частину значень.

Параметр WITH ENCRYPTION наказує сервера шифрувати SQL-код запиту, що гарантує неможливість його несанкціонованого перегляду і використання. Якщо при визначенні уявлення необхідно приховати імена вихідних таблиць і полів, а також алгоритм об'єднання даних, необхідно застосувати цей аргумент.

Параметр WITH CHECK OPTION наказує сервера виконувати перевірку змін, вироблених через уявлення,на відмінність умові, визначеній у фразі WHERE оператора SELECT. Це означає, що не допускається виконання змін, які призведуть до зникнення рядка з вистави. Використання аргумента WITH CHECK OPTION гарантує, що будь-які зроблені зміни будуть відображені в поданні. Якщо користувач намагається виконати зміни, що призводять до виключення рядка з уявлення, при заданому аргументі WITH CHECK OPTION сервер видасть повідомлення про помилку і всі зміни будуть відхилені. Наприклад, виконаємо команду:

```
INSERTINTOКиєвляниVALUES (12, 'Іванов', 'Львів')
```

Це допустима команда в поданні, і рядок буде додана за допомогою уявлення Києвляни в таблицю Клієнт. Однак, коли інформація буде додана, рядок зникне з подання, оскільки назва міста відмінно від Києва. Для запобігання подібних моментів додамо параметр WITH CHECK OPTION у визначенні уявлення Києвляни:

```
ALTER VIEW Києвляни

SELECT КодКліента, Прізвище, ГородКліента FROM Клієнт
WHERE ГородКліента = 'Київ' WITH CHECK OPTION
```

Для зміненого уявлення вищезгадана вставка запису буде відхилена системою.

Не усі уявлення в SQL можуть бути поновлюваними. Оновлюване уявлення визначається наступними критеріями:

- грунтується тільки на одній базовій таблиці;
- містить первинний ключ цієї таблиці;
- не містить DISTINCT в своєму визначенні;
- не використовує GROUP BY в своєму визначенні;
- по можливості не застосовує в своїй ухвалі підзапити;
- включає кожен стовпець таблиці, має атрибут NOTNULL;
- оператор SELECT перегляду не використовує функції, з'єднання таблиць, збережені процедури і функції, визначені користувачем;
- грунтується на одиночному запиті, тому об'єднання UNION заборонено.

Якщо уявлення задовольняє цим умовам, до нього можуть застосовуватися змінюватися оператори INSERT INTO, UPDATE, DELETE. Відмінності між оновлюваними уявленнями і уявленнями, призначеними тільки для читання, не випадкові. З оновлюваними уявленнями в основному обходяться точно так же, як і з базовими таблицями. Фактично, користувачі не можуть навіть зрозуміти, чи є об'єкт, який вони запитують, базової таблицею або поданням, тобто перш за все це засіб захисту для приховування конфіденційних або не відносяться до потреб даного користувача частин таблиці. Уявлення в режимі «тільки для читання» дозволяють отримувати дані більш раціонально, об'єднуючи у запиті безліч базових таблиць. Отримані уявлення можуть використовуватися в інших запитах, що дозволить уникнути складних предикатів і знизити ймовірність помилкових дій. Наприклад, уявлення з розрахунком підсумкових значень за типом товару і потім будемо використовувати його в запиті вибірки:

```
CREATE VIEW Замовлення (Код, Клієнт, Товар, Кількість, Ціна, Код_доставкі) AS
SELECT КодЗамовлення, Прізвище + "" + Ім'я, Товар, Кількість, Ціна
* Кількість, КодДоставки FROM Клієнти INNER JOIN Замовлення
ON Клієнти.КодКліента = Замовлення.КодКліента INNER JOIN
```

товари ON Замовлення.КодТовару = Товари.КодТовару ORDER BY Товар

CREATE VIEW Доставка_Ітоги (Вид_доставки, Кількість_товара, Сума) AS SELECT ТіпДоставки, Count (Кількість_товара), SUM (Ціна)

FROM Доставка INNER JOIN Замовлення ON Доставка.Код_доставкі = Замовлення.Код_доставкі GROUP BY ТипДоставкі ORDER BY ТипДоставкі

4.2.Переваги та недоліки уявлень

Механізм використання уявлень - потужний засіб СУБД, що дозволяє приховати реальну структуру БД від деяких користувачив за рахунок визначення уявлень. Розглянемо основні переваги застосування уявлень в подібному середовищі.

1. Незалежність від даних. За допомогою уявлень можна зробити узгоджену, незмінну картину структури бази дан них, яка буде залишатися стабільною навіть в разі зміни вихідних таблиць (наприклад, додавання або видалення полів, зміни зв'язків, поділу таблиць, їх реструктуризації або перейменування). Якщо в таблицю додаються або з неї видаляються які не використовуються в поданні поля, то змінювати визна ня цього подання не буде потрібно. Якщо структура результату таблиці переупорядочівать або таблиця розділяється, можливо створити уявлення, що дозволяє працювати з віртуальною таблицею колишнього формату. За рівної кількості вихідної таблиці, колишній формат може бути віртуально відтворений з допомогою уявлення, побудованого на основі поєднання новостворених таблиць - звичайно, якщо це виявиться можливим.
2. Підвищення захищеності даних. Права доступу до даних можуть бути надані виключно через обмежений набір уявлень, що містять тільки ту підмножину даних, яке необхідно користувачу. Подібний підхід дозволяє істотно посилити контроль доступу окремих категорій користувачів до інформації в базі даних.
3. Зниження вартості. Уявлення дозволяють спростити структуру запитів за рахунок об'єднання даних з декількох таблиць в єдину віртуальну таблицю. В результаті багатоособисті запити

зводяться до простих запитах до одного уявлення.

4. Додаткові зручності. Створення уявлень може забезпечувати користувачів додатковими зручностями - наприклад, можливістю роботи тільки з дійсно потрібної частиною даних. В результаті можна домогтися максимального спрощення тієї моделі даних, яка знадобиться кожному кінцевому користувачеві.
5. Можливість настройки. Уявлення є зручним засобом настройки індивідуального образу бази даних. В результаті одні і ті ж самі таблиці можуть бути пред'явлені користувачем в абсолютно різному вигляді.
6. Забезпечення цілісності даних. Якщо в операторі CREATE VIEW буде вказана фраза WITH CHECK OPTION, то СУБД здійснює контроль за тим, щоб в вихідні таблиці бази даних не була введена жодна з рядків, що не задовільняють пропозицією WHERE у визначальному запиті. Цей механізм гарантує цілісність даних в поданні.

Однак використання уявлень в середовищі SQL не позбавлене недоліки:

1. Обмежені можливості оновлення. У більшості випадків уявлення не дозволяють вносити зміни в що містяться в них дані.
2. Структурні обмеження. Структура представлення устанавли-ється в момент його створення. Якщо визначає запит перед-ставлен в формі SELECT * FROM ..., то символ * посилається на всі стовпці, існуючі в початковій таблиці на момент з- будівлі уявлення. Якщо згодом у вихідну таблицю бази даних додадуться нові стовпці, то вони не з'являться в дан- ном поданні до тих пір, поки це подання не буде видалено і знову створено.
3. Зниження продуктивності. Використання уявлень пов'язано з певним зниженням продуктивності. В одних випадках вплив цього фактора зовсім небагато, тоді як в інших воно може послужити джерелом істотних проблем. Наприклад, подання, визначене за допомогою складного багатотабличного запиту, може зажадати значи- тільних витрат часу на обробку, оскільки при його разре- шеніі потрібно виконувати з'єднання таблиць щоразу, ко

   гда знадобиться доступ до даного подання. Виконання дозволу уявлень пов'язано з використанням додаткового тільних обчислювальних ресурсів.

   Поняття процедури,

Збережені процедури (Stored Procedure) представляють собою групи

пов'язаних між собою операторів SQL, що зберігаються в базі даних в відкомпілювався вигляді. При цьому одна процедура може бути викозовано в будь-якій кількості клієнтських додатків, що дозволяє су- нням заощадити трудовитрати на створення прикладного про- граммного забезпечення і ефективно застосовувати стратегію повторного використання коду. Так само, як і будь-які процедури в стандартних язиках програмування, збережені процедури можуть мати вхідні і ви- хідні параметри або не мати їх зовсім. Збережені процедури пишуться на спеціальному вбудованій мові програмування, вони можуть вклю- чати будь оператори SQL, а також включають певний набір опера торів, що керують ходом виконання програм, які багато в чому схожі з подібними операторами процедурно орієнтованих мов програмування. У комерційних СУБД для написання текстів хра- німих процедур використовуються власні мови програмування,так, в СУБД Oracle для цього використовується мова PL / SQL, а в MSSQLServer використовується мова TransactSQL. Виконання в базі даних хра- німих процедур замість окремих операторів SQL дає користувачеві наступні переваги:

- SQL інструкції пройшли етап синтаксичного аналізу і находят- ця в виконуваному форматі;
- збережені процедури підтримують модульне программірова- ня, так як дозволяють розбивати великі завдання на самостійні, більш дрібні і зручні в управлінні частини;
- збережені процедури можуть викликати інші процедури, що і функції;
- збережені процедури можуть бути викликані з прикладних програм інших типів;
- як правило, збережені процедури виконуються швидше, ніж поотже окремих операторів;
- збережені процедури простіше використовувати: вони можуть складатися з десятків і сотень команд, але для їх запуску достатньо вказати лише ім'я потрібної збереженої процедури. Це дозволяє зменшити розмір запиту, що посилається від клієнта на сервер, а значить, і навантаження на мережу;
- зберігання процедур в тому ж місці, де вони виконуються, обеспечивает зменшення обсягу переданих по мережі даних та повишает загальну продуктивність системи.

Застосування збережених процедур спрощує супровід програмних комплексів і внесення змін до них. Зазвичай все ограничення цілісності у вигляді правил і алгоритмів обробки даних реали- ся

на сервері баз даних і доступні кінцевому додатком у вигляді набору процедур, які і представляють інтерфейс обробки даних. Для забезпечення цілісності даних, а також з метою безпеки, додаток зазвичай не отримує прямого доступу до дан- ним - вся робота з ними ведеться шляхом виклику тих чи інших процедур. Подібний підхід робить вельми простий модифікацію алго- ритмів обробки даних, негайно ж стають доступними для всіх користувачів мережі, і забезпечує можливість розширення системи без внесення змін до сам додаток: досить змінити хра- німую процедуру на сервері баз даних. Розробнику не потрібно пере- компілювати додаток, створювати його копії, а також інструктіро- вать користувачів про необхідність роботи з новою версією. пользоваки взагалі можуть не підозрювати про те, що в систему внесені зраді ня.

Збережені процедури існують незалежно від таблиць або будь-яких інших об'єктів баз даних. Вони викликаються клієнтської про- грамою, інший збереженої процедурою або тригером. Розробник мо жет управляти правами доступу до інформації, що зберігається процедурою, дозволяючи або за- прещая її виконання. Змінювати код процедури, що дозволяється тільки її власнику або члену фіксованою ролі бази даних. При необхідності можна передати права володіння нею від одного пользова- теля до іншого.

4.3.Типи процедур

У SQL Server є кілька типів збережених процедур.

- системні збережені процедури;
- призначені для користувача процедури, що зберігаються;
- Тимчасові локальні і глобальні збережені процедури.

Системні збережені процедури призначені для виконання різних адміністративних дій. Практично всі дії з адміністрування сервера виконуються з їх допомогою. Можна ска- зать, що системні збережені процедури є інтерфейсом, забезпе чувати роботу з системними таблицями, яка, в кінцевому рахун- ті, зводиться до зміни, додаванню, видалення і вибірці даних з системних таблиць як для користувача, так і системних баз даних. Системні збережені процедури мають префікс sp_, зберігаються в системний базі даних і можуть бути викликані в контексті будь-якої іншої бази даних.

Призначені для користувача процедури, що реалізують ті чи інші дей ствия, визначені користувачем. Призначені для користувача процедури хра- нятся в конкретній базі даних. Виклик такої процедури можливий в контексті тієї бази даних, де знаходиться процедура.

Тимчасові збережені процедури існують лише деякий час, після чого автоматично знищуються сервером. Вони діляться на ло- Кальне і глобальні. Локальні тимчасові процедури, що мо- гут бути викликані тільки з того з'єднання, в якому створені. При з- будівлі такої процедури їй необхідно дати ім'я, що починається з одно- го символу #. Як і всі тимчасові об'єкти, збережені процедури цього типу автоматично видаляються при відключенні користувача, переза- пуску або зупинки сервера. Глобальні тимчасові збережені проце дурепи доступні для будь-яких з'єднань сервера, на якому є такая ж процедура. Для її визначення достатньо дати їй ім'я, починаючи- ющееся з символів ##. Видаляються ці процедури при перезапуску або зупинці сервера, а також при закритті з'єднання, в контексті кото рого вони були створені.

Створення нової та зміна наявної процедури, що здійс- нюється за допомогою наступної команди:

{CREATE | ALTER} [PROCEDURE] імя_процедури [; номер] [{@ імя_параметра тип_даних} [VARYING] [= default] [OUT- PUT]] [, ... n]

[WITH {RECOMPILE | ENCRYPTION}]

AS sql_оператор [... n]

Розглянемо параметри даної команди.

Використовуючи префікси sp_, #, ##, створювану процедуру можна ви- ділити як системної або тимчасовою. Як видно з синтаксису команди, не допускається вказувати ім'я власника, якому буде при- належних створювана процедура, а також ім'я бази даних, де вона повинна бути розміщена. Таким чином, щоб розмістити створювану збережену процедуру в конкретній базі даних, необхідно виконати команду CREATE PROCEDURE в контексті цієї бази даних. При про- рощення з тіла збереженої процедури до об'єктів тієї ж бази даних можна використовувати укорочені імена, т. Е. Без вказівки імені бази даних. Коли ж потрібно звернутися дооб'єктів, розташованих в інших базах даних, вказівка імені бази даних обов'язково.

Номер в імені - це ідентифікаційний номер зберігається процеду- ри, однозначно визначає її в групі процедур. Для зручності управління процедурами логічно однотипні процедури, можна групувати, привласнюючи їм однакові імена, але різні іден- тіфікаціонние номера.

Для передачі вхідних і вихідних даних в створюваній збереженій процедурі можуть використовуватися параметри, імена яких, як і імена локальних змінних, повинні починатися з символу @. В одній хра- Німой процедурі можна задати безліч параметрів, розділених за- п'ятими. У тілі

процедури не повинні застосовуватися локальні змінні, чиї імена збігаються з іменами параметрів цієї процедури.

Для визначення типу даних, який буде мати відповідним щий параметр збереженої процедури, годяться будь-які типи даних SQL, включаючи певні користувачем. Наявність ключового слова OUTPUT означає, що відповідний параметр призначений для повернення даних з збереженої процедури. Однак це зовсім не озна- чає, що параметр не підходить для передачі значень в збережену про- цедуру. Вказівка ключового слова OUTPUT наказує сервера при виході з процедури, що привласнити поточне значення параметра локальної змінної, яка була вказана при виклику процедури в якості значення параметра. Відзначимо, що при вказівці ключового слова OUTPUT значення відповідного параметра при виклику процедури може бути поставлено лише за допомогою локальної змінної. Не дозволяється використання будь-яких виразів або констант, допустимий для звичайних параметрів.

Ключове слово VARYING застосовується спільно з параметром OUTPUT, що має тип CURSOR. Воно визначає, що вихідним пара- метром буде результуюча безліч. = Default означає, що пара- метру можна привласнити значення за замовчуванням. Таким чином, при виклику процедури можна не вказувати явно значення відповідного параметра.

Так як сервер кешируєт план виконання запиту і компілірован- ний код, при наступному виклику процедури будуть використовуватися вже готові значення. Однак в деяких випадках все ж потрібно ви- няти перекомпіляцію коду процедури. Вказівка ключового слова RECOMPILE наказує системі створювати план виконання храни- мій процедури при кожному її виклику.

Ключове слово ENCRYPTION наказує сервера виконати шифрування коду збереженої процедури, що може забезпечити захист від використання авторських алгоритмів, що реалізують роботу збереженої процедури.

Ключове слово AS розміщується на початку власне тіла збереженої процедури, тобто набору команд SQL, за допомогою яких і буде реали- зовивать ту чи іншу дію. У тілі процедури можуть застосовуватися практично всі команди SQL, оголошуватися транзакції, встановлюватися блокування і викликатися інші процедури, що зберігаються. Вихід з храни- мій процедури може здійснюватися за допомогою команди RETURN.

Видалення збереженої процедури здійснюється командою:

DROP PROCEDURE {імя_процедури} [... n]Поняття функції користувача

При реалізації на мові SQL складних алгоритмів, які можуть знадобитися більш ніж один раз, відразу постає питання про збереження раз- працювати коду для подальшого застосування. Це завдання можна було б реалізувати за допомогою збережених процедур, однак їх архітектура не дозволяє використовувати процедури безпосередньо в виразах,тому вони вимагають проміжного присвоєння повернутого значення змінної, яка потім і вказується в вираженні. Природно, по-Добнєв метод застосування програмного коду не дуже зручний. Віз- вість створення призначених для користувача функцій була надана в середовищі MS SQL Server, починаючи з версії 2000. У інших реалізаціях SQL в розпорядженні користувача були лише вбудовані функції, забезпе чувати виконання найбільш поширених завдань: пошук максимальні або мінімального значення та ін.

Як і процедури, функції користувача є самостійні об'єкти бази даних, розташовуються в певній базі даних і доступні тільки в її контексті. У SQL Server є сліду- ющие класи функцій користувача:

- Scalar - функції повертають звичайне скалярний значення, каж дая може включати безліч команд, що об'єднуються в один блок за допомогою конструкції BEGIN ... END;
- Inline - функції містять всього одну команду SELECT і повер ють користувачеві набір даних у вигляді значення типу даних TABLE;
- Multi-statement - функції також повертають користувачеві значен ня типу даних TABLE, що містить набір даних, проте в те- ле функції знаходиться безліч команд SQL (INSERT, UPDATE і т.д.). Саме з їх допомогою і формується набір даних, кото-рий повинен бути повернутий після виконання функції.

На відміну від збережених процедур, призначені для користувача функції можуть застосовуватися в запитах так само, як і системні вбудовані функції. Призначені для користувача функції, які повертають таблиці, можуть стати аль- тернатівой уявленням. Уявлення обмежені одним виражений- ням SELECT, а призначені для користувача функції здатні включати допол- вальні вираження, що дозволяє створювати більш складні і мощ- ні конструкції.

Видалення будь-якої функції здійснюється командою:

DROP FUNCTION імя_функциі [... n]

Скалярні функції.Створення і зміна скалярної функції виконується за допомогою команди:

{CREATE | ALTER} FUNCTION імя_функциі

([{@ Імя_параметра скаляр_тіп_данних [= default]} [... n]])

RETURNS скаляр_тіп_данних

[WITH {ENCRYPTION | SCHEMABINDING} [... n]]
[AS] BEGIN

<Тело_функції>

RETURN
скаляр_вираженіе END

Розглянемо призначення параметрів команди.

Функція може містити один або кілька вхідних параметрів або не містити жодного. Кожен параметр повинен мати унікаль- ну в межах створюваної функції ім'я і починатися з символу "@". Після імені вказується тип даних параметра. Додатково можна вказати значення, яке буде автоматично присвоюватися параметру (DEFAULT), якщо користувач явно не вказав значення відповідним ного параметра при виклику функції.

За допомогою конструкції RETURNS скаляр_тіп_данних вказується, який тип даних буде мати повертається функцією значення.

Розширені можливості пошуку, з якими повинна бути створена функ- ція, можуть бути вказані за допомогою ключового слова WITH. Завдяки ключовим словом ENCRYPTION код команди, який використовується для ство-ня функції, буде зашифрований, і ніхто не зможе переглянути його. Ця можливість дозволяє приховати логіку роботи функції. Крім того, в тілі функції може виконуватися звернення до різних об'єктів бази даних, а тому зміна або видалення відповідних об'єктивним тов може привести до порушення роботи функції. Щоб уникнути це- го, потрібно заборонити внесення змін, вказавши при створенні цієї функції ключове слово SCHEMABINDING.

Між ключовими словами BEGIN ... END вказується набір команд, вони і будуть тілом функції.

Коли в ході виконання коду функції зустрічається ключове слово RETURN, виконання функції завершується і як результат її вичісле- ня повертається значення, вказане безпосередньо після слова RETURN. Відзначимо, що в тілі функції дозволяється використання безлічі команд RETURN, які можуть повертати різні зна- чення. Як повертається допускаються як звичайні константи, так і складні вирази. Єдина умова - тип даних, що повертається повинен збігатися з типом даних, зазначеним після ключового слова RETURNS.

Для прикладу створимо і застосуємо функцію скалярного типу для ви- числения сумарної кількості товару, що надійшов за певну дату:

```
CREATE FUNCTION sales (@data DATETIME) RETURNS
INT AS BEGIN

   DECLARE @c INT

   SET @ c = (SELECT SUM (кількість) FROM Угода
WHERE дата = @ data )

   RETURN
(@c) END
```

В якості вхідного параметра використовується дата. Функція повер щает значення цілого типу, отримане з оператора SELECT шляхом підсумовування кількості товару з таблиці Угода. Умовою відбору записів для підсумовування є рівність дати угоди значенням вхідного параметра функції.

Звернення дофункції наступне:

```
DECLARE @kol INT

SET @ kol = user1.sales ('02 .11.01 ')
SELECT @kol
```

Абопросто:SELECT user1.sales ('02 .11.01 ')

Функції Inline. Створення та зміна функції цього типу ви- вується за допомогою команди:

```
{CREATE | ALTER} FUNCTION імя_функциі

([{@ Імя_параметра скаляр_тіп_данних [= default]} [... n]])
RETURNSTABLE [WITH {ENCRYPTION | SCHEMABINDING}
[... n]] [AS] RETURN [(] SELECT_оператор [)]
```

Основна частина параметрів, використовуваних при створенні табличних функцій, аналогічна параметрам скалярної функції. Проте, з- будівлю табличних функцій має свою специфіку. Після ключового слова RETURNS завжди має зазначатися ключове слово TABLE. Таким чином, функція даного типу повинна строго повертати значен ня типу даних TABLE. Структура повертається типу TABLE не вказується явно при описі власне типу даних. Замість цього сервер буде автоматично використовувати для возвращае- мого значення TABLE структуру, що повертається запитом SELECT, ко торий є єдиною командою функції.

Особливість функції даного типу полягає в тому, що структура значення TABLE створюється автоматично під час виконання запиту, а

не вказується явно при визначенні типу після ключового слова RETURNS.

Що повертається функцією значення типу TABLE може бути використ зовано безпосередньо в запиті, тобто в розділі FROM.

Створимо і застосуємо функцію табличного типу для визначення двох найменувань товару з найбільшим залишком:

```
CREATE FUNCTION itog () RETURNS TABLE

AS RETURN (SELECT TOP 10 Товар.Названіе FROM Товар
INNER JOIN Склад ON Товар.КодТовара = Склад.КодТовара
ORDER BY Склад.Остаток DESC)
```

Використовувати функцію для отримання 10 найменувань товару з найбільшим залишком можна наступним чином:

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

Іншийприклад:

```
USE
MyDB
GO

CREATE FUNCTION dbo.fn_GetLastTask () RETURNS
TABLE AS RETURN (SELECT TOP 1 Заданіе.код_заданія
AS ID, Заданіе.время, Продукція.названіе

FROM Завдання INNER JOIN Продукція ON Заданіе.код_продукціі

= Продукція.код_продукціі ORDER BY ID DESC)
GO
```

4.4.Визначення тригера в стандарті мови SQL

Тригер являє собою спеціальний тип збережених процедур, що запускаються сервером автоматично при спробі зміни даних в таблицях, з якими тригери пов'язані. Тригери використовуються для перевірки цілісності даних, а також для відкату транзакцій. Таким про- разом, тригер - це відкомпільоване SQL-процедура, виконання до- торою обумовлено настанням певних подій всередині реляці- ційної бази даних. Застосування тригерів здебільшого вельми зручно для користувачів бази даних. І все ж їх використання часто пов'язано з додатковими витратами ресурсів на операції вво- / виводу. У тому випадку, коли тих же

результатів можна досягти за по- міццю збережених процедур або прикладних програм, застосування тригерів недоцільно. Право на створення тригера має тільки власник бази даних.

Найчастіше тригери застосовуються для підтримки цілісності даних в базі даних, так як за допомогою обмежень цілісності і значень за замовчуванням не завжди можна домогтися потрібного рівня функціональності. Часто потрібно реалізувати складні алгоритми перевірки даних, відстежувати зміни значень таблиці, щоб потрібним чином змінити пов'язані дані.

Кожен тригер прив'язується до конкретної таблиці і виконується в складі відповідної транзакції модифікації даних. У разі виявлення помилки або порушення цілісності даних в коді трігге- ра можна зробити відкат транзакції. Тим самим внесення змін (подія, яка викликала тригер) скасовується. Скасовуються також всі вимірюв- нання, вже зроблені тригером. На відміну від звичайної процедури,тригер виконується неявно в кожному випадку виникнення тригерній події.

Тригер не має аргументів. Основний формат команди CREATE TRIGGER показаний нижче:

CREATE TRIGGER імя_тріггера

BEFORE | AFTER <тріггерное_собитіе> ON <ім'я_таблиці>
[REFERENCING <спісок_псевдонімов>]

[FOR EACH {ROW | STATEMENT}]

[WHEN (условіе_тріггера)]

<Тело_тріггера>

Тригерні події складаються з вставки, видалення і відновлення рядків в таблиці. В останньому випадку для тригерній події можна вказати конкретні імена стовпців таблиці. Час запуску тригера визначається за допомогою ключових слів BEFORE (тригер запускається до виконання пов'язаних з ним подій) або AFTER (після їх виконання). Що Їх тригером дії задаються для кожного рядка (FOR EACH ROW), охопленої цією подією, або тільки один раз для кожної події (FOR EACH STATEMENT).

Позначення <спісок_псевдонімов> відноситься до таких компонен- там, як стара або нова рядок (OLD / NEW) або стара або нова таб- особи (OLD TABLE / NEW TABLE). Ясно, що старі значення не примі Німи для подій вставки, а нові - для подій видалення.

За умови правильного використання тригери можуть стати дуже потужним механізмом. Але їм притаманні і деякі недоліки:

- прихована функціональність: перенесення частини функцій в базу даних і збереження їх у вигляді одного або декількох тригерів

іноді призводить до приховування від користувача деяких функціональних них можливостей (тільки розробник знає все). Хоча це певною мірою спрощує його роботу, але, на жаль, мо жет стати причиною незапланованих, потенційно нежела- тільних і шкідливих побічних ефектів, оскільки в цьому випадку користувач не в змозі контролювати всі процеси, проис- ходять в базі даних;

- вплив на продуктивність: перед виконанням кожної команди по зміні стану бази даних СУБД повинна про- вірити тригерні умова з метою з'ясування необхідності за- пуску тригера для цієї команди. Виконання подібних вичіс- лений позначається на загальній продуктивності СУБД, а в мо- менти пікового навантаження її зниження може стати особливо за- Метн.
- Неправильно написані тригери можуть привести до серйозних проблем, таких, наприклад, як поява "мертвих" блокіро- вок. Тригери здатні тривалий час блокувати множе- ство ресурсів, тому слід звернути особливу увагу на све дення до мінімуму конфліктів доступу.

4.5.Реалізація тригерів в середовищі MS SQL Server

Уреалізації СУБД MS SQL Server використовується наступний опера тор створення або зміни тригера:

{CREATE | ALTER} TRIGGER імя_тріггера

ON {ім'я_таблиці | імя_представленія} [WITH ENCRYPTION] FOR

{AFTER | INSTEAD OF} {DELETE | INSERT | UPDATE | WITH APPEND}

AS sql_оператори

Тригер може бути створений тільки в поточній базі даних, але допус- кається звернення всередині тригера до інших баз даних, в тому числі і розташованим на віддаленому сервері. Ім'я тригера має бути уні Кальний в межах бази даних. Додатково можна вказати ім'я власника. При вказівці аргументу WITH ENCRYPTION сервер ви- няет шифрування коду тригера, щоб ніхто, включаючи адміністратора, не міг отримати до нього доступ і прочитати його.

типи тригерів. У SQL Server існує два параметра, визна ляющих поведінку тригерів:

- AFTER. Тригер виконується після успішного виконання ви- звавших його команд. Якщо ж команди з якої-небудь причини не можуть бути успішно завершені, тригер не виконується. Слід

зазначити, що зміни даних в результаті виконання запиту користувача і виконання тригера здійснюється в тілі однієї транзакції: якщо станеться відкат тригера, то будуть відхилені і призначені для користувача зміни. Можна визначити кілька AFTER-тригерів для кожної операції (INSERT, UPDATE, DELETE). Якщо для таблиці передбачено виконання несколь- ких AFTER-тригерів, то за допомогою системної збереженої проце дурепи sp_settriggerorder можна вказати, який з них буде виконуватися першим, а який останнім. За замовчуванням в SQL Server все тригери є AFTER-тригерами.

- INSTEAD OF. Тригер викликається замість виконання команд. На відміну від AFTER-тригера INSTEAD OF-тригер може бути визначений як для таблиці, так і для подання. Для кожної операції INSERT, UPDATE, DELETE можна визначити тільки один INSTEAD OF тригер.

Тригери розрізняють за типом подій, наякі вони реагують. Існує три типи тригерів:

- INSERT TRIGGER - запускається при спробі вставки даних за допомогою команди INSERT;
- UPDATE TRIGGER - запускається при спробі зміни даних за допомогою команди UPDATE;
- DELETE TRIGGER - запускається при спробі видалення даних за допомогою команди DELETE.

Допускається створення тригера, що реагує на дві або на все три команди. Аргумент WITH APPEND дозволяє створювати кілька тригерів кожного типу.

4.6. Система захисту Microsoft Access

Система захисту СУБД Microsoft Access базується на використанні таких засобів:

1. Шифрування / дешифрування;
2. Показ або приховування об'єктів у вікні бази даних;
3. Використання пароля БД;
4. Використання захисту на рівні користувача.

Шифрування бази даних - це найпростіший спосіб захисту. При шифруванні бази даних її файл стискається і стає недоступним для читання за допомогою службових програм або текстових редакторів. Шифрування незахищеною бази даних неефективно, оскільки кожен зможе відкрити таку базу даних і отримати повний доступ до всіх її об'єктів. Шифрування зазвичай застосовується при електронній передачі

бази даних або збереженні її на дискету, касету або ком- пакт-диск.

Іншим способом захисту об'єктів в базі даних від сторонніх користувачів є приховування об'єктів у вікні бази даних. Цей спо соб захисту є найменш надійним, оскільки щодо про- сто можна відобразити будь-які приховані об'єкти.

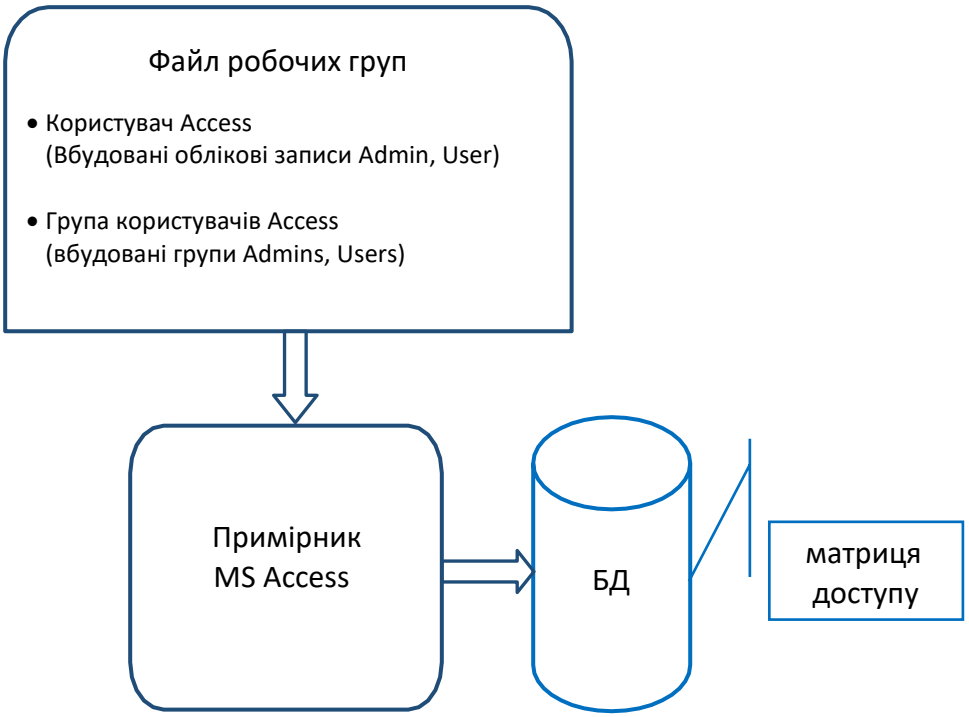
Іншим найпростішим способом захисту є установка пароля для відкриття бази даних (.mdb). Після установки пароля при кожному від- критті бази даних буде з'являтися діалогове вікно, в яке вима- ється ввести пароль. Тільки ті користувачі, які введуть правильний пароль, зможуть відкрити базу даних. Цей спосіб більш надійний, ніж попередній (Microsoft Access шифрує пароль, тому до нього немає до- ступа при безпосередньому читанні файлу бази даних), але він діє тільки при відкритті бази даних. Після відкриття бази даних всі об'єкти стають доступними для користувача (поки не визначені інші типи захисту, описані нижче в цьому розділі). Даний спосіб не є надійним способом захисту баз даних. Існує достаточ- ну кількість безкоштовних і платних утиліт, що відображають пароль. У тому числі доступні вихідні коди коду на VBA,

4.7.Використання захисту на рівні користувача

Найбільш гнучкий і поширений спосіб захисту бази даних називають захистом на рівні користувача, що дозволяє встановити різні рівні доступу до важливих даних і об'єктів у базі даних для різних користувачів. Даний вид захисту реалізований в версіях Microsoft Access по 2003 включно. Починаючи з Access 2007, Microsoft реалізує інший шлях захисту на основі цифрових підписів. Захист на рівні користувачів Microsoft Access реалізує вибірковий підхід розмежування доступу. При активізації даного виду захисту в базі даних Microsoft Access адміністратор бази даних або власник об'єк єкта надає певні дозволи окремим користувачам і групам користувачів на наступні об'єкти: таблиці, запити, форми, звіти і макроси. Облікові записи системи безпеки ви- деляют користувачів і групи осіб, які можуть до- ступ до об'єктів. Ця інформація, яку називають відомостями про робочу групу, зберігається в спеціальному файлі робочої групи. Файл робочої групи містить відомості про користувачів, що входять в робо чую групу. До таких відомостей належать імена облікових записів пользовате- лей, їх паролі і імена груп, в які входять користувачі. Для з- будівлі файлу робочих груп і підключення його до БД використовується спе соціальне додаток «Адміністратор робочих груп».

Після входу в систему аналізується файл робочої групи, в кото ром кожен користувач ідентифікується унікальним кодом. Груп- пам і

користувачам надаються дозволи, що визначають віз можность їх доступу до кожного об'єкту бази даних. Існують два типи дозволів на доступ: явні і неявні. Дозволи називаються явними, якщо вони безпосередньо присвоєні облікового запису пользова- теля; такі дозволи не впливають на вирішення інших користувачів. Неявними називаються дозволу на доступ, присвоєні облікової за- писи групи. Користувач, включений в таку групу, отримує всі дозволи, надані групі; видалення користувача з цієї групи позбавляє його всіх дозволів, привласнених даній групі. Раз рішення зберігаються в самій БД. Роздільне зберігання облікових записів і матриці доступу (дивисьМал. 2) Підвищує надійність системи захисту. Якщо користувач з даними ідентифікатором не виявляється в при- з'єднаному файлі робочих груп, доступ до об'єктів відхиляється.



Мал. 4.1 Схема безпеки СУБД MS Access

При спробі користувача виконати будь-яку операцію з защи- щенним об'єктом бази даних його поточні дозволу визначаються комбінацією явних і неявних дозволів на доступ. На рівні пользо- вателей завжди діють мінімальні обмеження з накладаються яв- ними дозволами для

користувача і для всіх груп, до яких при- належить даний користувач. Тому найпростішим способом управ ління робочою групою є створення нових груп і визначення дозволів на доступ для цих груп, а не для індивідуальних пользо- вателей. Після цього зміна дозволів для окремих користувачів здійснюється шляхом додавання користувачів в групи або уда- лення їх з груп. Крім того, при необхідності надати нові дозволи, вони надаються відразу всім членам групи в одній операції.

У Microsoft Access визначені дві стандартні групи: администра- тори (група «Admins») і користувачі (група «Users»), але допускається визначення додаткових груп. Члени групи «Users» можуть бути допущені тільки до перегляду даних в таблицях. члени групи
«Admins» мають дозволу на доступ до всіх об'єктів бази даних. Якщо для системи захисту досить групи адміністраторів і групи користувачів, то немає необхідності створювати інші групи. В цьому випадку необхідно присвоїти відповідні дозволи на доступ стандартній групі «Users» і додати додаткових адміністрато- рів в стандартну групу «Admins». Кожен новий користувач авто- матически додається в групу «Users». Типові дозволу на доступ для групи «Users» можуть включати «Читання даних» і «Оновлення даних» для таблиць і запитів та «Відкриття / запуск» для форм і звітів

У разі необхідності більш розгалуженої структури управління для різних груп користувачів, є можливість створення нових груп, присвоєння групам різних наборів дозволів на до- ступ і додавання нових користувачів до відповідних груп. Наприклад, можна організувати захист бази даних «Замовлення» з догою створення облікових записів «Керуючі» для керівництва фірми,
«Представники» для торгових представників і «Персонал» для адміні ністративного персоналу. Після цього слід присвоїти найбільшу кількість дозволів групі «Керуючі», проміжне кількість групі «Представники» і мінімальне групі «Персонал». При створенні облікового запису для нового співробітника цей запис буде добавкою ляться в одну з груп, і співробітник автоматично отримає дозволу, що належать цій групі.

Коли користувач вперше запускає Access після установки Microsoft Office, Access автоматично створює файл робочої групи, ко торий ідентифікується по зазначеними користувачем імені і назву організації. Відносне розташування файлу робочої групи записується в наступні параметри реєстру:

HKEY_CURRENT_USER \ Software \ Microsoft \ Office \ 10.0 \ Access \ Jet \ 4.0 \ Engines \ SystemDB і HKEY_USERS \ .DEFAULT \ Software \ Microsoft \ Office \ 10.0 \ Access \ Jet \ 4.0 \ Engines \ SystemDB

Так як отримання цієї інформації не становить особливих труднощів, існує ймовірність того, що хтось зможе створити іншу версію файлу робочої групи і придбати невід'ємні дозволу на до- ступ за допомогою стандартної облікового запису адміністратора, тому що про- грами установки автоматично додає в групу «Admins» стандартному обліковий запис користувача «Admin». Щоб уникнути цього, при активізації захисту на рівні користувача необхідно створити новий файл робочої групи і приєднати його до захищається БД. Со будівлі файл робочої групи використовується при відкритті не тільки захищатися, але будь-який інший бази даних, що викликає серйозні не- зручності. Якщо планується використовувати файл робочої групи для кон- конкретній БД, слід створити ярлик для відкриття цієї БД і в командному рядку якого вказати параметр / wrkgrp і повне ім'я файлу робочої групи. наприклад:

"C: \ Program Files \ MSOffice2003 \ OFFICE11 \ MSACCESS.EXE"

/ WrkGrp C: \ test \ gr.mdw

Призначення дозволів. Призначати і змінювати дозволу можуть наступні учасники:

- члени групи «Admins»;
- власник об'єкта;
- будь-який користувач, який отримав на цей об'єкт дозволу адіністратора.

Користувачі, які є членами групи «Admins» або володарями об'єктів і не мають дозволу на виконання якої-небудь дії, мають можливість призначити їх собі самі.

Користувач, який створив таблицю, запит, форму, звіт або макрос, є власником цього об'єкта. Крім того, користувачі, що входять до групи «Admins», можуть також змінити власника об'єкта або заново створити об'єкт, що є альтернативним способом зміни власника об'єкта. Для створення об'єкта достатньо імпортувати або експорті- ровать цей об'єкт в іншу базу даних або зробити копію об'єкта. Цей прийом є найпростішим способом зміни власника об'єктів, в тому числі і всієї бази даних.

Деякі дозволу на доступ автоматично надають іншого дозволу. Наприклад, дозвіл «оновлення даних» на табліцю автоматично надає дозволу «читання даних» і «читання макета», необхідні для зміни даних в таблиці. Дозволи

«Зміна макета» і «читання даних» автоматично надають дозвіл «читання макета». Для макросів дозвіл «читання макета» тягне надання дозволу «відкриття / запуск». При зміні об'єкта і його подальшому збереженні

дозволу на доступ до нього зберігаються. Однак якщо об'єкт зберігається під новим ім'ям, він стає новим об'єктом і, отже, отримує дозволи, встановлені за замовчуванням для даного типу об'єктів, а не дозволу вихідного об'єкта.

Установка і зняття захисту. Насправді, система захисту Microsoft Access завжди включена. До активізації захисту на рівні користувача Microsoft Access автоматично підключає всіх пользова- телей за допомогою вбудованої облікового запису користувача «Admin» з порожнім паролем. Microsoft Access неявно використовує цей обліковий запис як обліковий запис адміністратора робочої групи, а також як власника всіх створюваних баз даних і таблиць, форм, звітів і мак росів.

Щоб активувати захист на рівні користувача необхідно вве- сти пароль користувача при запуску Microsoft Access. Для активації появи вікна введення пароля досить призначити пароль користувачу «Admin» (спочатку він порожній). Відповідно, для відключення виведення діалогового вікна введення пароля слід зняти пароль записи «Admin». Оскільки облікові записи користувача «Admin» абсолютно однаковий ви для всіх екземплярів Microsoft Access, то першим кроком при нізації системи захисту є визначення облікових записів адміністратора і власника (або єдиної облікової записи, що є записом і адміністратора, і власника). Після цього слід видалити обліковий запис користувача «Admin» з групи «Admins». Якщо цього не сде- лать, будь-який користувач Microsoft Access зможе підключитися до робо чий групі за допомогою облікового запису «Admin» і отримати всі разреше- ня на доступ до таблиць, запитам, формам, звітам і макросам робо чий групи. Крім того, групі Users за замовчуванням присвоєні всі пра- ва доступу. Так як будь-який користувач Access входить в цю групу, то він неявно також отримує всі права доступу. Тому, другим кроком є мінімізація прав групи Users до мінімально необхідного рівня. Наступним важливим кроком є зміна власника всіх об'єктів БД з облікового запису «Admin» на новий обліковий запис власника шляхом створення нової бази даних та імпорту в неї всіх об'єктів з ис Ходнев бази даних. На закінчення проводиться настройка дозволів. До групи «Admins» дозволяється додавати довільне число облік- них записів, проте власником бази даних може бути тільки одна обліковий запис - та, що була активною при створенні бази даних, або та, що була активною при передачі права власника шляхом створення но- вої бази даних і імпорту в неї всіх об'єктів з вихідної бази даних. Однак облікові записи груп можуть бути власниками таблиць, запитів, форм, звітів і макросів бази даних.

Увійдіть до облікового запису групи неможливий.

Для зняття захисту на рівні користувача необхідно створити но- ву

БД, в ярлику прописати шлях до цієї БД, до файлу робочих груп защіщённой БД і при відкритті вказати ім'я та пароль власника. Потім необхідно імпортувати в створену БД всі об'єкти захищеної БД, після чого змінити власника всіх об'єктів на вбудовану обліковий запис Admin. На закінчення залишається відключити вікно появи запиту пароля, знявши пароль записи «Admin».

Захист Access є відносно ненадійною. Є спеціалізовані програми, що дозволяють дізнатися ім'я і пароль власника БД. При відсутності файлу робочих груп його теж можна відновити. Для цього буде потрібно дізнатися імена і ідентифікатори власників об'єктів БД. Оскільки ця інформація міститься в файлі бази даних, то і вона може бути залучена за допомогою таких програм як AOPR.

Іноді для злому БД Access зовсім не обов'язково використовувати програми, що дозволяють визначити пароль БД або користувача. Часто програмісти зовсім не дбають про приховування пароля в тексті программи. Запустивши програму, що працює з захищеної БД, можна від- крити в шістнадцятковому редакторі WinHex віртуальну пам'ять це- го додатки. Провівши пошук Unicode рядків виду 'User ID ='; 'Password ='; 'Database Password =' або 'pwd =' можна знайти ім'я пользова- теля, його пароль і пароль бази даних. Можна і зовсім проігнорувати наявність захисту. Існують програми, наприклад, програма восста- лення БД Access AccessRecovery, яка створює новий файл без за- щити і переносить в нього таблиці, запити, форми, макроси, звіти і код модулів.

4.8.Методи протидії злому захисту Access.

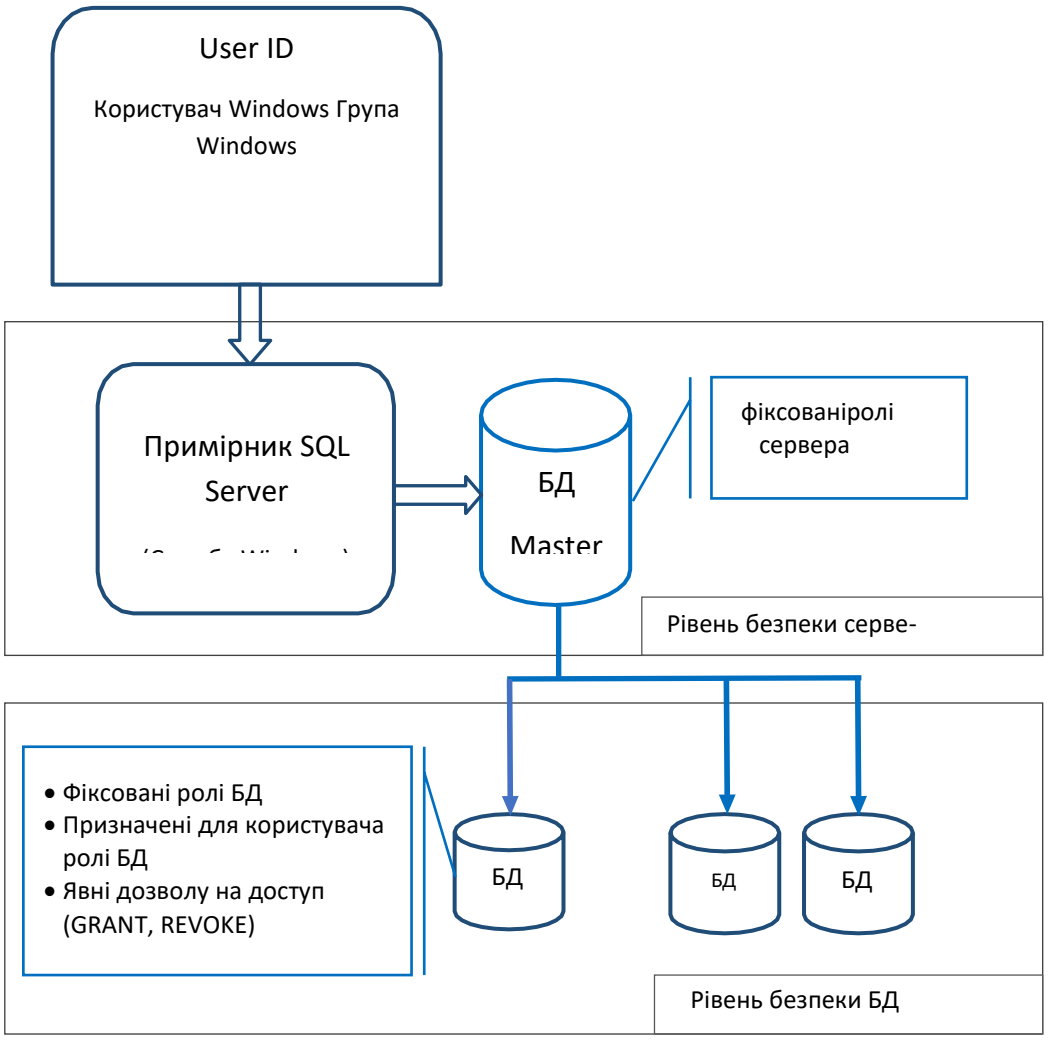
Захист за допомогою пароля БД, що містить нецензурні символи.В першу чергу цей спосіб націлений на протидію визначення паролів за допомогою спеціальних програм. Спосіб осно- ван на те, що пароль БД формату Access 2000 і 2002-2003 - текстовий рядок у форматі Unicode. При цьому, немає ніяких обмежень на її від- повідні дані. Стандартний спосіб установки і використання пароля БД має на увазі його введення з клавіатури в діалоговому вікні. Якщо рядок па- роля містить нецензурні символи, то вони не будуть коректно відобра- дружини програмою, що відкриває паролі БД. З іншого боку, цей па- роль не можна ввести в діалоговому вікні при відкритті БД в MS Access, що вимагає написання спеціальної програми завантажувача. У специфікації баз даних і в довідці по DAO 3.60 зазначено, що максимальне число символів в паролі - 14. Але насправді їх може бути 20.

Захист шляхом модифікації файлу БД і його розширення. Даний спосіб захисту заснований на модифікації перших байт файлу. Таким

про- разом, перед відкриттям БД за допомогою додаткової програми в її файл записується правильний заголовок, що зберігається в програмі доступа, а після закриття повертається неправильний. При спробі відкрити файл даних за допомогою Access з'являється повідомлення про помилку. Цей спосіб захисту добре поєднати з способом зміни роз- ренію файлу. Наприклад, можна взяти заголовок dbf файлу і записати його в початок mdb файлу. Далі змінити розширення файлу на dbf. Вказаним ний метод не досить ефективний, так як програму, що працює з БД, можна перервати штучно і на диску залишиться не захищена БД. Тому варто його використовувати тільки в поєднанні з іншими спо собами.

Захист зміною версії БД. Цей спосіб - подальший розвиток ідеї модифікації заголовка файлу з метою протидії програм- мам, які читають паролі. Метод заснований на тому, що для роботи з БД Access 97 і вище програми використовують різні алгоритми читання па- роля і при цьому намагаються самостійно визначити версію mdb фай- ла. Для визначення версії можна використовувати послідовність з 40 байт, починаючи з 122 байта від початку файлу. Якщо в БД Access 97 впі- сать цю послідовність від Access 2003 то MS Access, бібліотеки доступу ADO і DAO будуть нормально працювати з цим файлом, а біль-шинство зламували паролі програм немає. Аналогічний результат може бути отриманий при пересадці байтовой послідовності з Access 97 в Access 2003.8. Архітектура системи безпеки SQL Server

Система безпеки MS SQL Sever реалізує дискреційний і рольові підходи до управління доступом і має два рівні: рівень сер- віра і рівень бази даних (дивисьМал. 3). На рівні сервера дозволя- ється або відхиляється доступ користувачів до самого сервера. На рівні бази даних користувачі, які мають доступ на рівні сервера, одержу- ють доступ до об'єктів бази даних. Такий підхід дозволяє більш гнучко управляти доступом користувачів до баз даних.



Мал.4.2 Узагальнена схема безпеки СУБД MS SQL Server

На рівні сервера система безпеки оперує наступними по- поняттями:

- аутентифікація (authentication);
- обліковий запис (login);
- вбудовані ролі сервера (fixed server roles).

На рівні бази даних використовуються поняття:

- користувач бази даних (database user);
- фіксована роль бази даних (fixed database role);
- призначена для користувача роль бази даних (users database role);
- роль додатка (application role). Отже, можна виділити дві групи ролей:
- ролі сервера (server role);
- ролі бази даних (database role).

4.9.Система безпеки рівня сервера

SQL Server використовує двоетапну схему аутентифікації. Спочатку користувач аутентифицирующей сервером. Тільки після успішної аутентифікації на сервері користувачеві може бути надано до- ступ до однієї або декількох БД. SQL Server зберігає всю інформацію про реєстраційні записах в базі даних master.

SQL Server пропонує два режими аутентифікації користувачів:

- Режим аутентифікації засобами OS Windows. В цьому режимі SQL Server повністю довіряє операційній системі при аутен- тіфікації користувачів;
- Змішаний режим аутентифікації (Windows Authentication and SQL Server Authentication). В цьому режимі системи аутентіфіка- ції Windows і SQL Server існують незалежно один від одного.

При установці SQL Server одним з перших рішень, які сліду- ет прийняти, є вибір використовуваного методу аутентифікації. Встановлений при інсталяції режим аутентифікації можна ви- нитку на сторінці Security діалогового вікна SQL Server Properties в уті- літі Management Studio. У програмному коді встановлений режим аутентифікації можна перевірити за допомогою системної збереженої про- цедури xp_loginconfig:

```
EXEC xp_loginconfig 'login mode'
```

Результат виконання цієї процедури виглядає наступним обра- зом:

```
name            config_value
login mode Mixed
```

Відповідно, користувач повинен бути ідентифікований серве- ром одним з таких методів:

- за допомогою облікового запису користувача Windows (в загальному ви- чаї належить деякому домену або робочої групи);
- на основі членства в одній з груп користувачів Windows;
- за допомогою окремої реєстраційної записи SQL Server (якщо на сервері використовується змішана модель безпеки).

Не слід плутати реєстраційні записи SQL Server з обліковими записами Windows в цьому сервері баз даних. Ці два типи реєстрації абсолютно різні. Користувачам SQL Server не потрібен доступ до ка- талоге, де зберігаються файли бази даних, або до будь-яких інших файлів, так як реальний доступ до файлів здійснює процес SQL Server, а не сам користувач. У той же час права доступу до файлів потрібні са мому процесу SQL Server, отже, йому потрібна обліковий запис

Windows. Тут також доступні два варіанти:

- Обліковий запис локального адміністратора. SQL Server може використовувати цей обліковий запис для отримання доступу до компь- ютер, на якому встановлений. Цей варіант ідентичний установці на окремий сервер, так як в цьому випадку немає можливості підтримувати мережеву систему безпеки Windows, необхід мую для розподіленої обробки;
- Обліковий запис користувача домену (рекомендується). SQL Server може використовувати обліковий запис користувача домену Windows, створену спеціально для нього. Облікового запису поль зователя SQL Server можуть бути призначені адміністративні привілеї для локального сервера, і за допомогою нього він може отримати доступ до мережі для підтримки взаємодії з другі- ми серверами.

Набір ролей сервера строго обмежений. Ніхто, включаючи администра- тора сервера, не може створити нову або видалити існуючу роль сервера. Тому вони називаються фіксованими ролями (fixed server roles). Нижче наведено список деяких фіксованих ролей сервера з коротким описом кожної з них:

Sysadmin (System Adminis- trators)	Члени цієї ролі мають абсолютні права в SQL Server. Ніхто не має більших прав доступу, ніж члени даної ролі.
Setupadmin (Setup Adminis- trators)	Цією ролі надані права керування пов'язаними серверами, конфігурації збережених процедур, запуска- -ваних автоматично при старті SQL Server, а також право додавати облікові записи в роль setupadmin.
Serveradmin (Server Adminis- trators)	Зазвичай в цю роль включаються користувачі, які повинні виконувати адміністрування сервера. Мають пра- во на останов сервера (SHUTDOWN), змінювати параметри роботи служб (sp_conf igure), застосовувати зміни (RECONFIGURE), управляти повнотекстових пошуком (Sp_fulltext_service).
Securityadmin (Security Admin- istrators)	Члени цієї ролі мають можливість створювати нові облікові записи, яким вони можуть надавати права на створення баз даних і її об'єктів, а також керувати свя заннимі серверами, включати облікові записи в роль

	securityadmin і читати журнал помилок SQL Server.
Dbcreator (Database Crea-tors)	Члени цієї ролі можуть створювати нові бази даних, уда- лять і перейменовувати наявні, відновлювати ре-резервні копії бази даних і журналу транзакцій.

Щоб додати обліковий запис користувача в ту чи іншу фиксированную роль сервера можна скористатися збереженої процедурою sp_addsrvrolemember, що має наступний синтаксис:

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

Наприклад, розглянемо додавання облікового запису Windows з ім'ям Admin комп'ютера STORAGE в фіксовану роль сервера sysadmin:

EXEC sp_addsrvrolemember 'STORAGE \ Admin', 'sysadmin'

Хоча в загальному випадку потрібно, щоб на сервері існувала облікная запис, яку передбачається включити в одну з фіксованих ролей, є один виняток. Справа в тому, що користувач Windows може отримати доступ до SQL Server як член групи Windows, якій надано доступ до сервера. Такі облікові записи можна включати в ролі сервера без попереднього надання доступу безпосеред- ньо облікового запису. Для отримання інформації про те, яка облік- ная запис в яку фіксовану роль сервера включена, використовується наступна системна збережена процедура:

sp_helpsrvrolemember [[@srvrplename =] 'role']

Якщо процедура викликається без параметрів, то виводиться повний список облікових записів, включених в будь-яку з ролей сервера. Коли ж необхідно отримати список облікових записів, включених в конкретним роль сервера, потрібно вказати ім'я ролі. У наведеному нижче прикладі виводиться інформація про членів ролі sysadmin:

EXEC sp_helpsrvrolemember 'sysadmin'

Для видалення облікового запису з фіксованою ролі сервера предназначена системна збережена процедура sp_dropsrvroiemember, що має синтаксис:

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

З наведеного списку ролей рівня сервера видно, що включення користувачів в ці ролі дозволяє визначити тільки адміністраторів сервера, а не рядових користувачів БД. Шлях додавання звичайних користувачів БД залежить від обраного режиму аутентифікації

користувачів.

Аутентифікація Windows

Використання аутентифікації Windows означає, що користувачеві для доступу до SQL Server досить мати обліковий запис Windows. Ідентифікатор безпеки Windows передається з операційної си стеми на сервер баз даних. SQL Server передбачає, що процес регі- страції користувачів в мережі достатньо захищений, і тому не ви- няет ніяких додаткових перевірок. Користувач автоматично отримує відповідні права доступу до даних SQL Server відразу ж після реєстрації в домені. Такий метод надання доступу називаються ється встановленням довірчого з'єднання. Операційна систе- ма працює з обліковими записами (logins), які містять всі дані про користувача, включаючи його ім'я, пароль, членство в групах, каталог за замовчуванням і т. Д. Кожна обліковий запис має унікальний ідентифі- катор (Login ID) або ,

Ідентифікатор є довге (85-бітове) шестнадца- терічное число, яке генерується випадковим чином операційною системою під час створення облікового запису. Такий підхід дозволяє з- бігти підробки облікових записів користувачів. Якщо користувач був видалений з домену, то навіть повторне створення користувача з аналогич- ними характеристиками (ім'я облікового запису, пароль, членство в групі і т. Д.) Не дасть можливості отримати доступ до об'єктів, до яких мав доступ оригінальний користувач. Стосовно до SQL Server можна сказати, що якщо користувач домену мав певні права доступу, але був вилучений, то ніхто не зможе привласнити його права доступу.

Аутентифікація Windows передбачає збереження в системній базі даних Master тільки ідентифікаційного номера облікової записи користувача в домені (SID). Інформація про ім'я користувача, його пароль і т. Д. Зберігається в базі даних домену. Зміна імені пользо- Ватель або його пароля ніяк не відіб'ється на правах доступу до SQL Server. Інформація про реєстрацію користувача і його членство в групах Windows зчитується SQL Server з бази даних системи без- небезпеки домену під час підключення користувача. Якщо адміні- Стратор вніс якісь зміни в обліковий запис користувача, напри-мер, виключив його з деякою групи, то зміни позначаться тільки під час чергової реєстрації користувача в домені або в SQL Server в залежності від категорії зроблених змін.

Аутентифікація Windows дає певні переваги. На користувачів автоматично відбиваються всі правила політики без- небезпеки, встановлені в домені. Користувачеві не доводиться запо- міна ще один пароль. Це підвищує рівень загальної захищеності даних. Наприклад,

автоматично контролюється мінімальна довжина пароля і термін його дії. Операційна система вимагає від пользова- теля періодичної зміни пароля. Додатково можна заборонити користувачам установку паролів, вже вказував раніше. Крім то- го, Windows має вбудовані засоби захисту від підбору паролів. Аутентифікація Windows працює також з групами користувачів. Ко- гда ім'я групи Windows передається в SQL Server в якості реєстраційні ційної записи, будь-який член цієї групи може бути аутентифікований сервером баз даних.

Для створення облікового запису користувача або групи Windows в SQL Server в програмному коді використовується системна збережена про- цедура sp_grantlogin. Як аргумент їй необхідно передавати повне ім'я користувача, що включає ім'я домена, як в наступному прикладі:

EXEC sp_grantlogin 'RFE \ Sidorov'

Для видалення облікового запису або групи Windows з SQL Server програмним шляхом використовується системна збережена процедура sp_revokelogin:

EXEC sp_revokelogin 'RFE \ Sidorov'

Зрозуміло, ця операція не видаляє цей обліковий запис з опе- раціонної системи -проізойдет всього лише виняток користувача зі списку користувачів сервера баз даних.

Заборона облікового запису Windows можна зробити за допомогою си- стемной збереженої процедури sp_denylogin. Отже, доступ лю- бого користувача в SQL Server може бути примусово закритий. Це повністю забороняє доступ користувача до SQL Server, навіть якщо він бал відкритий за допомогою іншого методу. наприклад:

EXEC sp_denylogin 'RFE \ Sidorov'

Якщо користувач Windows був доданий в SQL Server, а потім був видалений з домену Windows, він продовжує існувати як користувач в півночі баз даних, але вважається вже осиротілим. Це означає, що, незважаючи на те, що користувач формально має доступ до сервера, він не має доступу до ресурсів мережі і, отже, до всього комплексу засобів SQL Server. Системна збережена процедура sp_validatelogins дозволяє знайти "осиротілих" користувачів і віз обертає їх ідентифікатори системи безпеки Windows NT і імена облікових записів. У наступному прикладі користувачеві RFE \ Joe був від- критий доступ до SQL Server, після чого його обліковий запис була видалена з Windows:

EXEC sp_validatelogins

SID NT Login

Ox0105000000000000515000000FCE31531A931 RFE \ Joe

Це не можна вважати проломом в системі безпеки. Не маючи облікового запису Windows з таким ідентифікатором (неможливо повторно з- здати обліковий запис і заданим SID), користувач не зможе підключати читься до SQL Server. Для вирішення проблеми "осиротілих" пользо- вателей можна використовувати наступний протокол:

1. Відкличте права доступу користувача до всіх баз даних з по-міццю збереженої процедури sp_revokedbaccess;
2. Відкличте право доступу даного користувача до сервера з допомогою на гою sp_revokelogin;
3. Створіть для користувача новий обліковий запис;
4. Зареєструйте обліковий запис на сервері.

Аутентифікація SQL Server

Цей тип аутентифікації реалізується на самому сервері SQL Server. В цьому випадку в системній базі Master зберігається повна інформація про користувачів включаючи ім'я користувача і його пароль. Для кожного користувача вказується ім'я облікового запису, унікальний ідентифікація тор SQL Server, пароль і інша інформація. При спробі користувача підключитися до сервера система безпеки потребує ввести ім'я облікового запису та її пароль. Потім система порівнює введені дані з інформацією, що зберігається в системних таблицях. Якщо дані збігатися дають, то доступ надається. В іншому випадку користувач по- лучает повідомлення про помилку, і з'єднання не встановлюється.

Наявність додаткових реєстраційних записів SQL Server доречно тоді, коли аутентифікація Windows недоступна або НЕ під- ходить для створюваної системи. Ця реєстрація має зворотну сумісного Стімость з попередніми версіями сервера, а також з додатками, в яких реєстраційний запис вбудована в програмний код. Аутенті- фикация SQL Server в основному застосовується клієнтами, для яких недоступна реєстрація в домені Windows. Наприклад, користувачами Novell NetWare, Unix і т. Д. При підключенні до SQL Server через Internet реєстрація в домені не виконується, тому в даному випадку також необхідно використовувати аутентифікацію SQL Server.

В процесі інсталяції SQL Server і виборі змішаного режиму аутентифікації майстер установки створює спеціальну обліковий запис sa (system administrator) з порожнім паролем, який є членом фикси- рова серверної ролі sysadmin і має всі права доступу до сер- віру. Найчастіше йому забувають привласнити пароль, і ці прямі ворота в системі безпеки відкривають шлях до сервера будь-якого зломщика. Як правило,

наявність такої проломи хакери перевіряють в першу чергу. Ис ходячи з цього, перш за все потрібно поставити пароль або просто відключити користувача sa і призначити адміністративної ролі sysadmin якогось іншого користувача (або створити додаткові ролі з администра- нормативними привілеями). Цей обліковий запис залишена для збереження зворотної сумісності з попередніми версіями SQL Server. Раніше обліковий запис була обов'язковою, мала абсолютні права по управле- нию сервером і не могла бути вилучена. Користувачеві sa в процесі установки завжди присвоюється однаковий на всі комп'ютерах іден- тіфікатор безпеки 0x01.

Ідентифікатор безпеки зберігається в стовпці sid таблиці sysxlogins системної бази даних Master. У ній зберігається інформація про облікові записи як SQL Server, так і Windows. Для облікових записів SQL Server максимальний розмір ідентифікатора безпеки состав- ляє 16 байт, для облікових записів Windows - 28 байт. Кожний рядок цієї таблиці відповідає один обліковий запис. Таким чином, в таблиці sysxlogins може міститися досить багато рядків з інформа- цією про облікові записи. Для більш зручної роботи з локальними облік- ними записами можна використовувати уявлення syslogins, включаю ний тільки ті рядки таблиці sysxlogins, які мають в стовпці srvid (ідентифікаційний номер сервера) значення NULL.

Для реєстрації користувача використовується системна збережена процедура sp_addlogin:

```
sp_addlogin 'ім'я', 'пароль', 'база_по_умолчанію', 'язик_по_умолчанію',
'ідентіфікатор_пользователя_сервера', 'параметр_шіфрованія'
```

Серед переданих процедурі аргументів обов'язковим є тільки ім'я реєстраційного запису. Так як в даному випадку потрібно налаштування користувача, а не просто вибір його зі списку, дана задача більш складна, ніж виконання процедури sp_grantlogin. Наприклад, наступний програмний код створює користувача SQL Server з ім'ям

Den і призначає йому в якості бази даних за замовчуванням навчальну базу test:

```
EXEC sp_addlogin 'Den', 'myoldpassword', 'test'
```

Параметр шифрування skip_encryption вказує сервера зберігати па- роль в системній таблиці sysxlogins без всякого шифрування. У той же час SQL Server очікує, що пароль обов'язково буде зашифрований, тому він не розпізнає створений таким чином пароль. Следова- тельно, слід уникати використання цього параметра.

Якщо деякий користувач створюється на двох серверах як один і той

же, то для другого сервера слід явно задати ідентифікатор SID, присвоєний першим сервером. Для отримання ідентифікатора SID ис користується уявлення sysserver_principals:

```
SELECT Name, SID FROM sysserver_principals WHERE Name =
'Den'
```

```
Name        SID
Den         0X1EFDC478DEB52 045B52D241B33B2CD7E
```

Пароль може бути змінений за допомогою системної збереженої проце дурепи sp_password, наприклад:

```
EXEC sp_password 'myoldpassword', 'mynewpassword', 'Den'
```

Якщо пароль порожній, то в збережену процедуру замість порожнього рядка ('') передається NULL. Якщо параметр @sid опущений або для нього зазначено зна чення NULL (також є значенням за замовчуванням для параметра @sid), то збережена процедура sp_addlogin самостійно згенерує ідентифікатор безпеки. Для прикладу створимо обліковий запис з конкретним ідентифікатором безпеки і паролем:

```
USE pubs

EXEC sp_addlogin 'Andrey', 'analitik', @sid =
0x0123456789ABCDEF0123456789ABCDEF
```

В принципі, після створення облікового запису можна змінити Ідент-фікатор безпеки за допомогою команди UPDATE, безпосередньо обра-тівшись до системної таблиці. При виборі значення для параметра @sid слід дотримуватися вимога унікальності ідентифікаторів без- ності. Тобто на поточному сервері до моменту реєстрації не повинно бути облікових записів з ідентифікатором безпеки, рівним ви- лайки значенням. Список використаних ідентифікаторів безопасно- сти локального сервера можна переглянути за допомогою наступного за-проса:

```
SELECT sid FROM syslogins
```

Для видалення реєстраційного запису SQL Server використовується си стемная збережена процедура sp_droplogin, наприклад:

```
EXEC sp_droplogin 'Den'
```

Щоб видалити обліковий запис призводить до видалення і всіх її налаштувань безпеки.

Система безпеки рівня бази даних

Для того щоб користувач мав можливість виконувати ті чи інші дії з об'єктами бази даних, він повинен попередньо по- лучити необхідні права доступу. Власник бази даних або конкретному ного об'єкта повинен надати користувачам бази даних можли ність звернення до об'єктів бази даних. Крім прав доступу, вида- ваемих користувачеві явно, існує клас неявних прав доступу. Не- явні права доступу надаються користувачеві через членство у фіксованій ролі сервера або бази даних. Наприклад, немає ніякої іншої можливості надати користувачеві право на створення баз даних, крім як включити його в роль сервера dbcreator. Іншим примі ром роботи неявних прав доступу є отримання власником об'єк єкта абсолютних прав на управління об'єктом володіння. Якщо пользова- тель створив об'єкт в базі даних,

Таким чином, на рівні бази даних користувачеві можуть бути надані привілеї за допомогою прикріплення до фіксованої ролі бази даних або шляхом явного призначення привілеїв за допомогою оператора SQL GRANT. Винятком є користувачі, облікові записи яких включені в фіксовану роль сервера sysadmin. Чле- ни даної ролі мають необмежені права в межах сервера, і, як наслідок, повний доступ до будь-якої базі даних, що є на сервері.

Перерахуємо всі фіксовані ролі бази даних:

db_securityadmin	Члени ролі можуть управляти правами доступу до об'єктів бази даних інших користувачів і членством їх в ролі
db_owner	Члени ролі мають права власника, т. Е. Можуть виконувать будь-які дії
db_denydatawriter	Членам цієї ролі заборонено зміна даних незаСімо від виданих їм дозволів
db_denydatareader	Членам цієї ролі заборонений перегляд даних незаСімо від виданих їм дозволів
db_ddladmin	Члени ролі можуть створювати, змінювати і видаляти об'єкти бази даних
db_datawriter	Користувачі, включені в цю роль, можуть змінювати дані в будь-якій таблиці або поданні бази даних

db_datareader	Користувачі, включені в цю роль, можуть читати дан ні з будь-яких таблиць і уявлень бази даних
db_backupoperator	Члени ролі виконують резервне копіювання бази даних
db accessadmin	Члени ролі мають право керувати користувачами бази даних: створювати, видаляти і змінювати

Призначені для користувача ролі - це додаткові ролі, службовці в якос- стве груп. Ролі може бути дозволений доступ до об'єкта бази даних, а користувачеві може бути призначена для користувача роль бази даних. Всі користувачі автоматично стають членами стандартної ролі бази даних public.

Включення користувачів в роль реалізує неявне призначення привілегій. Явні дозволу до об'єктів призначаються за допомогою ін- трукцій SQL GRANT, REVOKE, DENY і вони досить деталізіро- вани. Існують окремі дозволи для кожного з можливих дій (SELECT, INSERT, UPDATE, RUN і т.д.) над будь-яким об'єктивним тому. Заборона привілеї заміщає собою її надання, а надання привілеї заміщає собою її відгук. Користувачеві може бути надано безліч дозволів до об'єкта (індивідуальних, успадкованих від деякої ролі або забезпечених приналежністю до ролі public). Деякі фіксовані ролі бази даних також влі- яют на доступ до об'єкта, наприклад, керуючи можливістю зчитувати інформацію і записувати її в базу даних. Цілком можлива ситуація, коли користувач був розпізнаний в SQL Server, але у нього немає доступу ні до однієї з його баз даних. Також можливо і зворотне: користувачеві відкритий доступ до баз даних, але він не був розпізнаний сервером. Передання бази даних і її дозволів на інший сервер без паралельного переміщення реєстраційних записів сервера може привести до виникнення таких "осиротілих" користувачів.

У будь-якій базі даних автоматично створюються два користувача:

1. dbo (database owner). Це спеціальний користувач бази даних, що є її власником. Власник бази даних має абсолютного ні права з управління нею. Користувача dbo не можна видалити. За замовчуванням в користувача dbo відображається обліковий запис sa, ко торою тим самим надаються максимальні права в базі даних. Крім того, всі члени ролі бази даних dbowner також вважаються власниками бази даних. Користувач dbo включений в роль db_owner і не може бути вилучений з неї;
2. guest. Якщо облікового запису явно не надано доступ до бази

даних, то вона автоматично відображається сервером в пользователя guest. За допомогою цього користувача можна надавати дозволи на доступ до об'єктів бази даних, необхідні лю- бому користувачеві. Дозволивши доступ користувачеві guest, ви, тим самим, даєте аналогічні права доступу всіх облікових записів, сконфігурованим на SQL Server. Для підвищення безопасно- сти зберігається рекомендується видаляти користувача guest з бази даних.

Інформація про користувачів, створених в базі даних, зберігається в системній таблиці sysusers. Інформацію про користувачів поточної бази даних можна отримати за допомогою системної збереженої процедури sp_helpuser:

```
USE pubs

EXEC sp_helpuser
```

Створення нового користувача і зв'язування його з обліковим записом виконується за допомогою однієї з двох збережених процедур:

- sp_adduser. Дана процедура залишена для забезпечення сумісного Стімость з попередніми версіями SQL Server і оперує уста застарілих поняттями.
- sp_grantdbaccess. Ця збережена процедура повністю соответствует поняттям системи безпеки SQL Server і прийшла на зміну попередній процедурі, починаючи з версії SQL Server 7.0.

Процедура sp_grantdbaccess має наступний синтаксис:

```
sp_grantdbaccess [@loginame =] 'login' [, [@ name_in_db =]
'name_in_db' [OUTPUT]]
```

Правом виклику зазначеної процедури, що володіють члени фиксірованной ролі сервера sysadmin і члени фіксованих ролей бази даних db_owner і db_accessadmin. Параметр @loginame визначає ім'я облікового запису, якій передбачається надати доступ до поточної базі даних. Зазначена обліковий запис повинен існувати на сервері. За допомогою параметра @name_in_db вказується ім'я, яке буде присвоєно створюваному користувачеві. Це ім'я має бути унікаль- ним в межах бази даних. Наведений далі приклад ілюструє використання збереженої процедури sp_grantdbaccess. У базі даних pubs створюється новий користувач з ім'ям Admin, який связывает- ся з обліковим записом STORAGE \ Admin:

```
USE pubs

EXEC sp_grantdbaccess 'STORAGE \ Admin', 'Admin'
```

Видалення користувача виконується за допомогою системної збереженої процедури sp_revokedbaccess, що має синтаксис:

sp_revokedbaccess [@name_in_db =] 'name'

Правом виклику зазначеної процедури, що володіють члени фиксірованной ролі сервера sysadmin і члени фіксованих ролей бази даних db_owner і db_accessadmin. Єдиний параметр процедури визначає ім'я користувача, якого необхідно видалити. Однак, перш ніж зважитися на подібний крок, слід переконатися, що пользо- вателем не належить ніякої об'єкт бази даних. Якщо ж пользова- тель є власником одного з об'єктів бази даних, то слід або видалити цей об'єкт за допомогою команди DROP, або змінити власника об'єкта за допомогою системної збереженої процедури sp_changeobjectowner. Наприклад, для видалення користувача Admin до- статочно буде виконати команду:

EXEC sp_revokedbaccess 'Admin'

Щоб включити нового члена в фіксовану роль бази даних, необхідно викликати системну збережену процедуру sp_addrolemember, що має синтаксис:

sp_addrolemember [@rolename =] 'role', [@membername =]

'security_account'

За допомогою параметра @rolename вказується ім'я ролі, в яку потрібно додати нового члена. Зазначена роль повинна існувати в поточній базі даних. наприклад:

EXEC sp_addrolemember 'db_accessadmin', 'User1'

Для отримання інформаці про членство у всіх ролях поточної бази даних можна використовувати процедуру sp_helprolemember. наприклад:

USE pubs

EXEC sp_helprolemember

Виняток з фіксованою ролі виконується за допомогою проце- дури sp_droprolemember. Наприклад, для виключення з фіксованою ролі db_accessadmin користувача User1 необхідно виконати сліду- ющую команду:

USE pubs

EXEC sp_droprolemember 'db_accessadmin', 'User1'

Якщо фіксовані ролі призначені для наділення користувачів спеціальними правами в базі даних, то для користувача ролі служать

лише для угруповання користувачів з метою полегшення управ ління їх правами доступу до об'єктів. Створення користувальницької ролі виконується за допомогою системної збереженої процедури sp_addrole, ко торая має синтаксис:

sp_addrole [@rolename =] 'role' [, [@ownername =] 'owner']

або простоCREATE ROLE <rolename>

Правом виконання зазначеної процедури, що володіють члени фіксованою ролі сервера sysadmin і фіксованих ролей бази даних db_owner і db_securityadmin. За замовчуванням власником ролі стає власник бази даних (користувач dbo). Таким чином, користувач dbo набуває повний контроль над роллю. Якщо необ- ходимо привласнити права володіння роллю іншому користувачеві, то ім'я цього користувача має бути вказано за допомогою параметра @ownername. Указується користувач повинен бути в базі даних. Власником користувальницької ролі може також бути і роль додатка вання. Як приклад розглянемо створення користувальницької ролі AccessDBUser, власником якої буде користувач guest:

USE pubs

EXEC sp_addrole 'AccessDBUser', 'guest'

Видалення призначеної для користувача ролі здійснюється за допомогою храни- мій процедури sp_droprole, що має наступний синтаксис:

sp_droprole [rolename =] 'role'

За допомогою єдиного параметра процедури вказується ім'я користувача ролі, яку необхідно видалити з поточної бази даних. Однак перед подібною операцією потрібно видалити з неї всіх членів, для чого можна використовувати збережену процедуру sp_droprolemember. Якщо роль володіє одним або більше об'єктами бази даних, то знищити таку роль буде неможливо. Перш ніж при- ступити до видалення, необхідно або передати права володіння соответ- ствующими об'єктами за допомогою процедури, що sp_changeobjectowner, або видалити ці об'єкти. За допомогою наведених ного нижче прикладу можна видалити призначену для користувача роль

USE pubs

EXEC sp_droprole 'AccessDBUser'

Щоб переглянути список явних прав доступу можливий запуск систем-ної процедури, що sp_helprotect, що має синтаксис:

sp_helprotect [[@name =] 'object_statement'] [, [@username

=] 'Security_account'] [, [@grantorname =] 'grantor'] [,
[@permissionarea =] 'type']

наприклад:
sp_helprotect Clients (Для об'єкта Clients)

sp_helprotect @username = 'RFE \ Joe'(для користувача RFE \ Joe)

Дана процедура ефективна лише для перегляду явних прівіле- гий, призначених за допомогою оператора GRANT. Для перегляду всіх привілеїв, включаючи неявні, слід використовувати наступний SQL запит:

WHERE principal_type_desc <> 'DATABASE_ROLE'
UNION

--role members

SELECT rm.member_principal_name, rm.principal_type_desc,
p.class_desc, p.object_name, p.permission_name,
p.permission_state_desc, rm.role_name

FROM perms_cte p right outer JOIN (

select role_principal_id, dp.type_desc as principal_type_desc,
member_principal_id, user_name (member_principal_id) as
member_principal_name, user_name (role_principal_id) as role_name -
, *

from sys.database_role_members rm INNER
JOIN sys.database_principals dp

ON rm.member_principal_id = dp.principal_id

) rm

ON rm.role_principal_id = p.principal_id order by
1

Для переміщення БД на інший комп'ютер необхідно спочатку отсо-

единить БД. Для цього на комп'ютері джерелі спочатку слід кош- нитка контекст на системну БД Master, потім викликати збережену проце- дуру sp_detach_db має один обов'язковий параметр - ім'я отсо- единящей бази даних:

```
USE [master]
GO

sp_detach_db 'test' go
```

Після переміщення файлів БД в місце призначення проводиться про- ратна процедура. Якщо місце, куди скопійовані файли, відрізняється від каталогу за замовчуванням для сервера на комп'ютері призначення, то вима- ється призначити даного каталогу повні права доступу групи SQLServer2005MSSQLUser $ MyPC $ SQLEXPRESS (примітка: між символами долара знаходиться ім'я сервера). Якщо сервер запущений від імені деякого користувача, то права призначаються йому, а не вище- зазначеній групі. Підключення БД проводиться за допомогою храни- мій процедури sp_attach_db. Першим її аргументом йде ім'я БД, потім шлях до файлу БД з розширенням mdf, потім шлях до файлу БД з розширенням ldf, наприклад:

```
use [master] go

sp_attach_db 'test', 'D: \ Service \ Database files \ MSSQL.1 \
Data \ test.mdf', 'D: \ Service \ Database files \ MSSQL.1 \ Data
\ test_log.ldf'

go
```

Наступним кроком є відновлення осиротілих користувачів. Покажемо процедуру відновлення на прикладі користувача operator. Спочатку проводиться додавання облікових записів БД.

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

Після етапу додавання облікових записів проводиться восстановле- ня (генерація) їх системних ідентифікаторів. Для цього потрібні спеціальні повноваження, які дозволяють реконфігурацію налаштувань сервера. У сервері версії 2000 їх можна отримати, викликавши процедуру sp_configure 'allow update', 1 і потім викликати команду RECONFIGURE WITH OVERRIDE. Системний ідентифікатор отримують за допомогою функції SUSER_SID ('логін'). наприклад:

```
USE test
GO

sp_configure 'allow update', 1
RECONFIGURE WITH OVERRIDE

GO


USE Test

UPDATE sysusers SET sid = SUSER_SID ( 'operator') WHERE name =
'operator'

GO
```

На закінчення відновлюються вихідні настройки сервера

```
sp_configure 'allow update', 0
RECONFIGURE

GO
```

Для серверів версії 2005 і вище необхідно запустити сервер в монопольном режимі (ключ -m) і потім провести процедуру оновлення SID осиротілих користувачів.

На закінчення розглянемо приклад створення БД в SQL Server 2005 і виконання дій з адміністрування БД. Ці дії ви- ним в консолі, використовуючи клієнтську програму sqlcmd.

```
cmd

sqlcmd -S lab48-sk \ SQLEXPRESS
```

Після з'єднання всі команди виконуються в клієнті. Для початку увійдемо в систему з правами адміністратора. Отримаємо список пользова- телей, включених в фіксовану роль сервера sysadmin:

```
sp_helpsrvrolemember 'sysadmin'
```

Включимо користувача RFE \ Joe в роль sysadmin:

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

У разі змішаної аутентифікації сервера додамо непрівілегі- рованного користувача RFE \ User (база за замовчуванням test):

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

У разі аутентифікації Windows просто надамо доступ користувачам RFE \ User до сервера

sp_grantlogin 'RFE \ User'

Додамо користувача RFE \ User в список користувачів бази даних test (з ім'ям user)

USE test
go

sp_grantdbaccess 'RFE \ User', 'user'

Включимо RFE \ User в фіксовану роль БД
sp_addrolemember 'db_datareader', 'user' Додамо ще одного користувача Operator sp_grantlogin 'RFE \ Operator' sp_grantdbaccess 'RFE \ Operator', 'operator'
Надамо йому явні права доступу

GRANT SELECT, INSERT, UPDATE, DELETE ON Clients

TO operator

GRANT SELECT, INSERT, UPDATE, DELETE ON Orders TO operator GRANT SELECT, INSERT, UPDATE, DELETE ON Goods TO operator

DENY UPDATE (Goods.price) ON Goods TO operator
Переглянемо список призначених явних прав доступу
exec sp_helprotect @ username = 'operator'

Висновок

У дипломній роботі було розглянуто можливі види несанкціонованого входу у базу даних , можливі види боротьби с цими входами , було показано як захистити базу даних від несанкціонованого входу , були наведені приклади як ці входи можуть нашкодити базі даних і можливості порятунку втраченої інформації с бази даних .

Література

1. Смирнов, С. Безпека систем баз даних / С.М. Смирнов. - М .: Геліос-АРВ, 2007.
2. Цирль, В. Л. Основи інформаційної безпеки. Короткий курс / В. Л. Цирль. - Фенікс, 2008.
3. Герасименко В.А., Малюк А.А. Основи захисту інформації. М .: МІФІ, 2001 г.
4. Neerja Bhatnagar. Security in Relational databases / N. Bhatnagar // - In: Handbook of Information and Communication Security. ed. by P. Stavroulakis, M. Stamp.- Springer. - 2010. - pp.
5. Pernul, G. Database Security / G. Pernul // In: Advances in Computers, Vol. 38. ed. by MC Yovits. - Academic Press. - 1994. - pp.
6. Хорев П.Б. Програмно-апаратний захист інформації: навчальний посібник / П.Б. Хорев. - М .: Форум, 2013.
7. Бєлов О.Б. [та ін]. Основи інформаційної безпеки: учеб- ве посібник для студентів вузів, що навч. по спец. в обл. інформ. безпеки / Є.Б. Бєлов - М: Гаряча лінія - Телеком, 2006.
8. Романець Ю.В. Захист інформації в комп'ютерних системах і мережах / Ю.В. Романець, П.А. Тимофєєв, В.Ф. Шаньгина. - М: Радіо і зв'язок, 1999.
9. Голіков А.М. Основи інформаційної безпеки: навчальний посібник / А.М. Голіков. - Томськ: Томськ. держ. ун-т систем упр. і радіоелектроніки, 2007.
10. Захист інформації. Основні терміни та визначення. ГОСТ Р 50922 - 2006. - М: Стандартинформ, 2008.
11. Захист інформації. Забезпечення інформаційної безпеки в організації. Основні терміни та визначення. ГОСТ Р 53114- 2008. - М: Стандартинформ 2009.
12. Міхєєва, В. Microsoft Access 2003 / В. Міхєєва і І. Харитонова. - СПб: БХВ, 2004.
13. Нільсен, Пол. SQL Server 2005. Біблія користувача. Пер з англ. / Пол Нільсен. - Москва, СПб, Київ: Вільямс (Діалектика), 2008.
14. Мамаєв, Е. Microsoft SQL Server 2000. Найбільш повне керівництво / Е. Мамаєв - СПб: БХВ, 2005.

Додаток А

Додаток Б

Додаток В