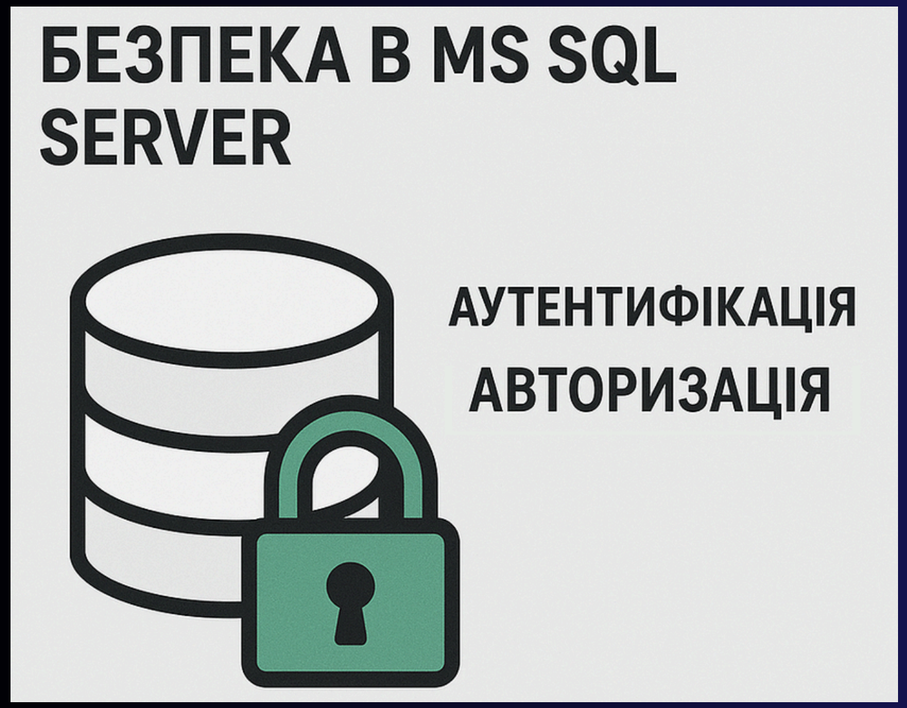
Лекція 8
Адміністрування інфраструктури баз даних

Аутентифікація підключень до SQL Server



Приклад створення SQL Server-логіна:

```
CREATE LOGIN student_login
WITH PASSWORD = 'StrongPassword123!';
```

Після створення логіна необхідно створити користувача всередині бази даних, який буде пов’язаний із цим логіном.

```
CREATE USER student_user
FOR LOGIN student_login;
```

Аутентифікація підключень до SQL Server

Перевірити, які користувачі бази даних існують на сервері, можна за допомогою системного представлення:

```
SELECT name, type_desc
FROM sys.database_principals;
```

Після встановлення з’єднання доцільно перевірити, під яким обліковим записом виконуються запити.

```
SELECT SYSTEM_USER;
SELECT USER_NAME();
```

Перший запит показує логін сервера, другий — користувача бази даних.

Аутентифікація підключень до SQL Server

Адміністратор має можливість контролювати активні підключення до сервера, використовуючи динамічні системні представлення.

```
SELECT
 login_name,
 host_name,
 program_name,
 connect_time
FROM sys.dm_exec_sessions
WHERE is_user_process = 1;
```

Авторизація облікових записів на доступ до баз даних

Доступ до бази даних може бути організований кількома способами. Найпростішим є включення користувача до фіксованих ролей бази даних, які вже містять визначені набори дозволів. Наприклад, роль db_datareader дозволяє читання всіх таблиць бази.

```
ALTER ROLE db_datareader
ADD MEMBER manager_user;
```

Адміністратор має можливість надавати доступ не до всієї бази, а лише до окремих об’єктів. Наприклад, дозволити читання тільки таблиці Employees.

```
GRANT SELECT ON Employees TO manager_user;
```

Авторизація облікових записів на доступ до баз даних

Для перевірки наданих дозволів використовуються системні представлення безпеки.

```
SELECT
 dp.name AS UserName,
 o.name AS ObjectName,
 p.permission_name,
 p.state_desc
FROM sys.database_permissions p
JOIN sys.database_principals dp
 ON p.grantee_principal_id = dp.principal_id
JOIN sys.objects o
 ON p.major_id = o.object_id;
```

Авторизація облікових записів на доступ до баз даних

Важливим аспектом авторизації є можливість обмеження доступу. Механізм DENY має вищий пріоритет, ніж GRANT, і використовується для явної заборони доступу.

```
DENY SELECT ON Employees TO manager_user;
```

Для повного видалення доступу до бази даних використовується оператор DROP USER.

```
DROP USER manager_user;
```

Авторизація облікових записів на доступ до баз даних

Для відновлення зв’язку використовується повторне призначення логіна.

```
ALTER USER manager_user
WITH LOGIN = manager_login;
```

Авторизація між серверами

Налаштування починається зі створення зв’язаного сервера на SERVER_A.

```
EXEC sp_addlinkedserver
 @server = 'SERVER_B',
 @srvproduct = '',
 @provider = 'SQLNCLI',
 @datasrc = 'SERVER_B';
```

Цей оператор реєструє віддалений сервер у системі, однак ще не визначає механізм автентифікації та авторизації. Для цього використовується процедура налаштування логінів.

```
EXEC sp_addlinkedsrvlogin
   @rmtsrvname = 'SERVER_B',
   @useself = 'false',
   @locallogin = NULL,
   @rmtuser = 'remote_login',
   @rmtpassword = 'RemotePass123!';
```

Авторизація між серверами

Після налаштування авторизації можна виконати тестовий запит до віддаленої таблиці.

```
SELECT * FROM [SERVER_B].TrainingDB.dbo.Employees;
```

Авторизація між серверами

Існує сценарій, коли кожен локальний логін має власну відповідність на віддаленому сервері. У цьому випадку налаштування виконується індивідуально.

```
EXEC sp_addlinkedsrvlogin
 @rmtsrvname = 'SERVER_B',
 @useself = 'false',
 @locallogin = 'analyst_login',
 @rmtuser = 'analyst_remote',
 @rmtpassword = 'AnalystRemotePass!';
```

Механізм OPENQUERY, дозволяє передавати запит безпосередньо на віддалений сервер.

```
SELECT * FROM OPENQUERY
 (SERVER_B, 'SELECT * FROM TrainingDB.dbo.Employees');
```

Авторизація між серверами

Адміністратор має можливість контролювати конфігурацію зв’язаних серверів через системні представлення.

```
SELECT *
FROM sys.servers
WHERE is_linked = 1;
```

Перевірка налаштувань логінів:

```
SELECT * FROM sys.linked_logins;
```

Призначення ролей на рівні сервера та бази даних

Робота з ролями на рівні сервера починається з аналізу наявних ролей і їх членів.

```
SELECT name, type_desc
FROM sys.server_principals
WHERE type = 'R';
```

Для перегляду членства користувачів у ролях використовується зв’язок між системними представленнями.

```
SELECT
 sp.name AS LoginName,
 sr.name AS ServerRole
FROM sys.server_role_members rm
JOIN sys.server_principals sp
 ON rm.member_principal_id = sp.principal_id
JOIN sys.server_principals sr
 ON rm.role_principal_id = sr.principal_id;
```

Результат дозволяє визначити, які логіни мають адміністративні повноваження на сервері.

Призначення ролей на рівні сервера та бази даних

Призначення ролей виконується на рівні логінів, оскільки саме вони представляють облікові записи сервера. Додавання логіна до ролі здійснюється через оператор ALTER SERVER ROLE.

```
ALTER SERVER ROLE securityadmin
ADD MEMBER student_login;
```

Видалення логіна з ролі виконується аналогічним чином.

```
ALTER SERVER ROLE securityadmin
DROP MEMBER student_login;
```

Призначення ролей на рівні сервера

У практичному адмініструванні часто виникає необхідність створення власних серверних ролей для делегування окремих функцій. Це дозволяє уникнути використання надмірних прав стандартних ролей і створити точну модель доступу.

```
CREATE SERVER ROLE audit_role;
```

Після створення ролі до неї додаються логіни.

```
ALTER SERVER ROLE audit_role
ADD MEMBER auditor_login;
```

Далі ролі надаються необхідні дозволи.

```
GRANT VIEW SERVER STATE TO audit_role;
```

Призначення ролей на рівні бази даних

Для визначення користувачів, що входять до цих ролей, використовується зв’язок системних представлень.

```
SELECT
 dp.name AS UserName,
 dr.name AS RoleName
FROM sys.database_role_members rm
JOIN sys.database_principals dp
 ON rm.member_principal_id = dp.principal_id
JOIN sys.database_principals dr
 ON rm.role_principal_id = dr.principal_id;
```

Результат показує, які користувачі мають доступ до функціональності конкретних ролей.

Призначення ролей на рівні бази даних

Надання ролі виконується через додавання користувача до її складу. Операція здійснюється на рівні бази даних, оскільки ролі належать саме їй.

```
ALTER ROLE db_datareader
ADD MEMBER student_user;
```

Після цього користувач отримує право читання даних з усіх таблиць бази.

Якщо необхідно надати повний контроль над базою, користувача додають до ролі db_owner.

```
ALTER ROLE db_owner
ADD MEMBER student_user;
```

Робота з ролями на рівні бази даних.

Адміністратор може створювати власні ролі для реалізації специфічної моделі доступу. Це особливо актуально в системах, де необхідно чітко розмежувати функціональні обов’язки різних груп користувачів.

```
CREATE ROLE sales_role;
```

Після створення ролі до неї додаються користувачі.

```
ALTER ROLE sales_role
ADD MEMBER student_user;
```

На цьому етапі роль існує, але не має дозволів. Далі їй призначаються права доступу до об’єктів бази.

```
GRANT SELECT ON Employees TO sales_role;
```

Авторизація користувачів на доступ до об'єктів.

GRANT використовується для надання прав доступу. Наприклад, дозволити користувачу читання даних з таблиці.

GRANT SELECT ON Employees TO student_user;

Оператор DENY застосовується для явної заборони доступу. Його особливість полягає в тому, що він має вищий пріоритет над дозволами, отриманими через ролі або інші механізми.

DENY SELECT ON Employees TO student_user;

REVOKE використовується для скасування раніше наданого дозволу або заборони.

REVOKE SELECT ON Employees FROM student_user;

Авторизація користувачів на доступ до об'єктів.

Авторизація може виконуватися безпосередньо для користувача або через ролі. У практичному адмініструванні перевага надається ролям, оскільки вони дозволяють централізовано керувати доступом.

GRANT SELECT ON Employees TO analyst_role;

У цьому випадку всі користувачі ролі analyst_role отримують доступ до таблиці. Контроль доступу може бути реалізований на різних рівнях деталізації. Наприклад, доступ до всієї таблиці або лише до окремих колонок.

GRANT SELECT ON Employees(FullName, Position) TO student_user;

Авторизація користувачів на виконання коду.

Приклад створення збереженої процедури:

```
CREATE PROCEDURE dbo.GetEmployees
AS
BEGIN
 SELECT FullName, Position
 FROM Employees;
END;
```

Для надання доступу іншим користувачам використовується оператор GRANT EXECUTE.

```
GRANT EXECUTE ON dbo.GetEmployees TO student_user;
```

Після цього користувач може запускати процедуру:

```
EXEC dbo.GetEmployees;
```

Авторизація користувачів на виконання коду.

Особливим механізмом є виконання коду від імені іншого користувача. У SQL Server це реалізується через контекст безпеки EXECUTE AS. Він дозволяє програмному модулю виконуватися з правами іншого облікового запису.

```
CREATE PROCEDURE dbo.AdminProcedure
WITH EXECUTE AS OWNER
AS
BEGIN
 SELECT * FROM Employees;
END;
```

Захист даних за допомогою шифрування та аудиту:

Опції аудиту доступу до даних у SQL Server.

Перед налаштуванням необхідно створити об’єкт аудиту на рівні сервера.

```
CREATE SERVER AUDIT Audit_Login
TO FILE (FILEPATH = 'C:\SQLAudit\');
GO
```

Після створення аудит необхідно увімкнути.

```
ALTER SERVER AUDIT Audit_Login
WITH (STATE = ON);
```

Захист даних за допомогою шифрування та аудиту:

Опції аудиту доступу до даних у SQL Server.

Аудит може бути налаштований і на рівні бази даних.

```
CREATE DATABASE AUDIT SPECIFICATION Audit_DataAccess
FOR SERVER AUDIT Audit_Login
ADD (SELECT ON dbo.Employees BY public),
ADD (INSERT ON dbo.Employees BY public),
ADD (UPDATE ON dbo.Employees BY public);
GO

ALTER DATABASE AUDIT SPECIFICATION Audit_DataAccess
WITH (STATE = ON);
```

Перегляд результатів аудиту виконується через системні функції.

```
SELECT * FROM sys.fn_get_audit_file('C:\SQLAudit\*', DEFAULT, DEFAULT);
```

Захист даних за допомогою шифрування та аудиту:
Застосування аудиту SQL Server.

Налаштування аудиту починається зі створення об’єкта аудиту на рівні сервера. У ньому визначається, куди записуватимуться дані, наприклад у файл.

```
CREATE SERVER AUDIT Audit_Main
TO FILE (FILEPATH = 'C:\SQLAudit\');
GO
```

Після створення аудит необхідно активувати.

```
ALTER SERVER AUDIT Audit_Main
WITH (STATE = ON);
```

Захист даних за допомогою шифрування та аудиту:

Керування аудитом SQL Server.

Перевірити наявні аудити на сервері можна через системний каталог.

```
 SELECT name, is_state_enabled
FROM sys.server_audits;
```

Для перегляду специфікацій аудиту використовується інше представлення.

```
 SELECT name, is_state_enabled
FROM sys.server_audit_specifications;
```

Увімкнути аудит мжна без видалення конфігурації.

```
 ALTER SERVER AUDIT Audit_Main
WITH (STATE = ON);
```

Вимкнення аудиту виконується аналогічною командою.

```
 ALTER SERVER AUDIT Audit_Main
WITH (STATE = OFF);
```

Захист даних за допомогою шифрування та аудиту:

Керування аудитом SQL Server.

Адміністратор може змінювати параметри аудиту. Наприклад, змінити місце збереження журналів або формат запису.

```
ALTER SERVER AUDIT Audit_Main
TO FILE (FILEPATH = 'D:\SQLAudit\');
```

Керування аудитом включає контроль специфікацій подій. У разі зміни політики безпеки можна додати нові події або вилучити існуючі.

```
ALTER SERVER AUDIT SPECIFICATION Audit_Logins
ADD (LOGOUT_GROUP);
```

Важливим аспектом керування є моніторинг результатів аудиту.

```
SELECT * FROM sys.fn_get_audit_file('D:\SQLAudit\*', DEFAULT, DEFAULT);
```

Захист даних за допомогою шифрування.

Шифрування — це процес перетворення даних у форму, непридатну для читання без наявності ключа дешифрування.

Створення головного ключа бази даних виконується після переходу до відповідної БД.

```
CREATE MASTER KEY
ENCRYPTION BY PASSWORD = 'StrongMasterKeyPass!';
```

Для налаштування Transparent Data Encryption необхідно створити сертифікат.

```
CREATE CERTIFICATE TDE_Cert
WITH SUBJECT = 'Certificate for TDE';
```

Після цього створюється ключ шифрування бази.

```
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE TDE_Cert;
```

Захист даних за допомогою шифрування.

Окремим механізмом є Always Encrypted, який дозволяє шифрувати дані на стороні клієнта ще до передачі в SQL Server.

Шифрування використовується і для захисту резервних копій баз даних. Це дозволяє запобігти витоку інформації у разі втрати носіїв або несанкціонованого доступу до архівів.

```
BACKUP DATABASE TrainingDB
TO DISK = 'D:\Backup\TrainingDB.bak'
WITH ENCRYPTION
(
 ALGORITHM = AES_256,
 SERVER CERTIFICATE = TDE_Cert
);
```

Моделі відновлення SQL Server: Стратегії резервного копіювання.

Модель Simple передбачає мінімальне ведення журналу транзакцій. Після завершення транзакції частина записів журналу очищується автоматично, що зменшує його розмір.

Модель Full забезпечує повне ведення журналу транзакцій. Усі операції записуються, що дозволяє виконувати відновлення до будь-якого моменту часу.

Модель Bulk-Logged є компромісною і використовується під час масових операцій, таких як імпорт великих обсягів даних або створення індексів. Вона зменшує обсяг журналювання, однак обмежує можливість точкового відновлення у періоди виконання масових операцій.

Моделі відновлення SQL Server: Стратегії резервного копіювання.

Перевірити поточну модель відновлення бази даних можна через системне представлення.

```
SELECT name, recovery_model_desc
FROM sys.databases;
```

Зміна моделі виконується адміністратором відповідно до політики резервного копіювання.

```
ALTER DATABASE TrainingDB
SET RECOVERY FULL;
```

Журнал транзакцій SQL Server.

Перевірити інформацію про журнал транзакцій можна через системні представлення.

```
DBCC SQLPERF(LOGSPACE);
```

Ця команда відображає відсоток використання журналу та його розмір.
Детальнішу інформацію можна отримати через:

```
SELECT name, recovery_model_desc, log_reuse_wait_desc
FROM sys.databases;
```

Параметр log_reuse_wait_desc показує причину, з якої журнал не може бути очищений, наприклад відсутність резервного копіювання або активні транзакції.

Планування стратегії резервного копіювання SQL Server.

Планування стратегії резервного копіювання — це процес визначення політики створення, зберігання та використання резервних копій для гарантування відновлення баз даних у разі збоїв.

Показник RPO визначає максимальний допустимий обсяг втрати даних у часі.
Показник RTO — максимально допустимий час відновлення системи після збою.

Резервне копіювання баз даних та журналів транзакцій.

Основним типом є повна резервна копія, що містить усі дані та об’єкти бази на момент створення.

```
BACKUP DATABASE TrainingDB
TO DISK = 'D:\Backup\TrainingDB_full.bak'
WITH INIT;
```

Для зменшення часу створення копій і обсягу даних застосовується диференціальне резервне копіювання. Воно зберігає лише зміни, внесені після останньої повної копії.

```
BACKUP DATABASE TrainingDB
TO DISK = 'D:\Backup\TrainingDB_diff.bak'
WITH DIFFERENTIAL;
```

Резервне копіювання баз даних та журналів транзакцій.

Резервне копіювання журналу транзакцій дозволяє відновити базу до конкретного моменту часу між повними резервними копіями.

```
BACKUP LOG TrainingDB
TO DISK = 'D:\Backup\TrainingDB_log.trn';
```

Процес відновлення бази даних виконується послідовно. Спочатку використовується повна резервна копія, після чого можуть застосовуватися диференціальні копії і журнал транзакцій.

```
RESTORE DATABASE TrainingDB
FROM DISK = 'D:\Backup\TrainingDB_full.bak'
WITH NORECOVERY;

RESTORE LOG TrainingDB
FROM DISK = 'D:\Backup\TrainingDB_log.trn'
WITH RECOVERY;
```

Керування резервними копіями баз даних.

Перевірка цілісності резервних копій.

```
RESTORE VERIFYONLY
FROM DISK = 'D:\Backup\TrainingDB_full.bak';
```

Старі копії архівуються або видаляються відповідно до регламенту.

```
 BACKUP DATABASE TrainingDB
 TO DISK = 'D:\Backup\TrainingDB_archive.bak'
 WITH INIT;
```

Параметр INIT дозволяє перезаписати існуючий файл резервної копії, що використовується для циклічного резервування.

Відновлення баз даних SQL Server

Відновлення починається з визначення типу збою і доступних резервних копій. У найпростішому випадку використовується повна резервна копія бази.

```
RESTORE DATABASE TrainingDB
FROM DISK = 'D:\Backup\TrainingDB_full.bak'
WITH RECOVERY;
```

Параметр NORECOVERY залишає базу в стані готовності до застосування наступних резервних копій.
Параметр RECOVERY завершує процес і робить базу доступною для роботи.
Параметр STANDBY використовується для доступу до бази в режимі читання між етапами відновлення.

Просунуті сценарії відновлення.

У моделях відновлення Full і Bulk-Logged SQL Server дозволяє застосувати журнал транзакцій до визначеного часу.

```
RESTORE DATABASE TrainingDB
FROM DISK = 'D:\Backup\TrainingDB_full.bak'
WITH NORECOVERY;

RESTORE LOG TrainingDB
FROM DISK = 'D:\Backup\TrainingDB_log.trn'
WITH STOPAT = '2026-03-12 11:45:00',
RECOVERY;
```

Іншим важливим сценарієм є відновлення окремих файлових груп.

```
RESTORE DATABASE TrainingDB
FILEGROUP = 'PRIMARY'
FROM DISK = 'D:\Backup\TrainingDB_full.bak'
WITH RECOVERY;
```

Просунуті сценарії відновлення.

Часткове відновлення бази. Воно використовується, коли необхідно відновити лише частину даних, наприклад архівні або робочі сегменти.

```
RESTORE DATABASE TrainingDB
PARTIAL
FROM DISK = 'D:\Backup\TrainingDB_partial.bak'
WITH RECOVERY;
```

Окремим сценарієм є відновлення бази з перенесенням файлів. Він використовується під час міграції або відновлення на інший сервер.

```
RESTORE DATABASE TrainingDB_New
FROM DISK = 'D:\Backup\TrainingDB_full.bak'
WITH MOVE 'TrainingDB' TO 'E:\Data\TrainingDB_New.mdf',
 MOVE 'TrainingDB_log' TO 'E:\Data\TrainingDB_New.ldf',
 RECOVERY;
```