МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ ДЕРЖАВНИЙ ПОЛІТЕХНІЧНИЙ КОЛЕДЖ

Для спеціальності:
151 Автоматизація та комп'ютерно-інтегровані технології

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни " Організація баз даних "

Харків 2020

Конспект лекцій з дисципліни " Організація баз даних " для студентів освітньо-професійної програми «Обслуговування інтелектуальних інтегрованих систем» спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»

Укладач: В.О.Величко - Харків: ХДПК, 2020, 116 с.

Затверджено на засіданні циклової комісії інформаційних технологій
Протокол від _____2020 р. №____

Голова циклової комісії _____М.М. Бочарніков
"____" _____ 2020 року

Схвалено методичною радою коледжу
Протокол від _____2020р. №____

Голова методичної ради _____
"_____" _____ 2020 р.

ЗМІСТ

ЛЕКЦІЯ 1. ВВЕДЕННЯ У ТЕОРІЮ БАЗ ДАНИХ

Лекція присвячена розгляду основних понять теорії баз даних і основних моделей даних, на яких побудовані сучасні БД.

Ціль: виявити основні структурні елементи баз даних і основні принципи, які використовуються при їх розробці.

Основні поняття

Існують різні визначення поняття *база даних* (БД). Найчастіше під БД розуміється пойменована сукупність структурованих даних, що відносяться до деякої предметної області. Однак в цьому випадку БД вельми важко відрізнити від звичайної картотеки або архіву документів.

Можна виділити три властивості, які відрізняють БД від простої сукупності даних:

1. БД зберігається і обробляється в обчислювальній системі.
2. Дані в БД добре структуровані, тобто виділені основні елементи, їх типи і зв'язки між елементами, а також обмеження на допустимі операції.
3. Забезпечується пошук і обробка даних.

Найбільш поширеним типом БД є реляційні бази даних.

Розгляньмо основні структурні елементи реляційної БД:

1. *Поле* – елементарна одиниця організації даних. Для опису поля використовують характеристики: ім'я, тип, довжина, точність тощо. Відповідає колонці в таблиці.
2. *Запис* – сукупність логічно пов'язаних полів. Відповідає рядку в таблиці.
3. Власне, *таблиця* (*відношення*).

Система баз даних

Система баз даних (СБД) – це комп'ютеризована система структурованих даних, основна ціль якої зберігання інформації та надання її на вимогу.

Розрізняють однокористувацькі і багатокористувацькі системи.

Однокористувацька система (Single-user system) – це система, в якій в один і той самий час до БД може отримати доступ тільки один користувач.

Багатокористувацька система (Multi-user system) – це система, в якій в кожен момент часу до БД можуть отримати доступ кілька користувачів. Основне завдання такої системи - дозволити користувачеві працювати з БД як з одного користувача.

Зазвичай в СБД виділяють чотири основні елементи:

1. Дані.
2. Апаратне забезпечення.
3. Програмне забезпечення (ПЗ).
4. Користувачі.

Спрощена схема СБД представлена на рис. 1.1.

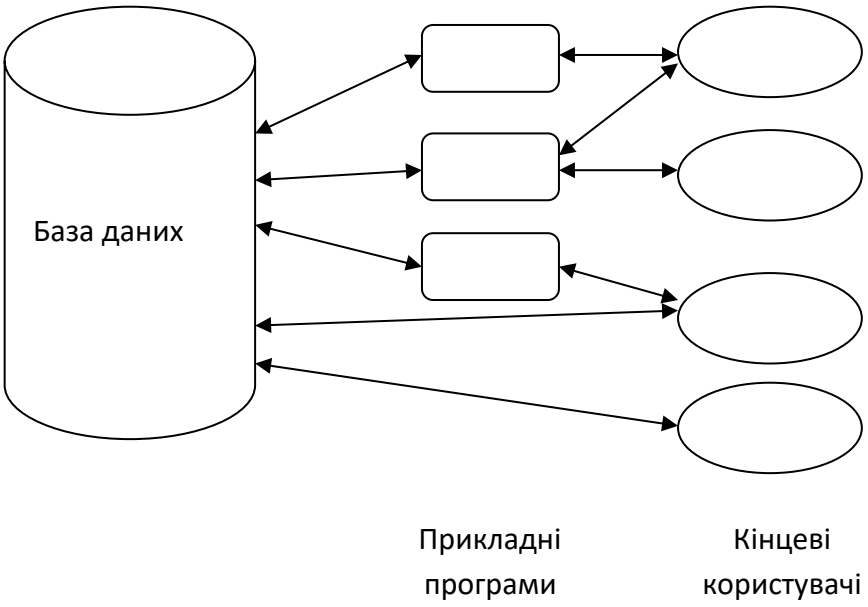


Рис. 1.1. Елементи системи баз даних

Дані

Дані в БД можна охарактеризувати як інтегровані і загальні.

Інтегровані дані можна представити як об'єднання декількох окремих файлів, що повністю або частково перекриваються. В разі *загальних* даних окремі області даних можна використовувати кількома різними користувачам.

Апаратне забезпечення

До нього відносяться:

- накопичувачі для зберігання інформації разом з пристроями вводу/виводу;
- процесор разом з основною пам'яттю, яка використовується для підтримки роботи ПЗ системи.

Програмне забезпечення

Основна частина ПЗ – це система керування базами даних, СКБД (DBMS – DataBase Management System – диспетчер БД).

Основна функція СКБД – надання користувачу можливості працювати з БД, не вникаючи в деталі на рівні апаратури.

СКБД підтримує користувацькі операції високого рівня. До таких операцій відносяться і операції, що виконуються за допомогою мови SQL (Structured Query Language, структурована мова запитів) - спеціальної мови БД. СКБД хоча й основний, але не єдиний програмний компонент системи, серед інших можна назвати утиліти, засоби розробки додатків, генератори звітів і інші.

Користувачі

Розрізняють три групи користувачів СБД:

1. *Прикладні програмісти*. Для цілей розробки прикладних програм, які використовують бази даних, можуть бути застосовані різні мови і середовища програмування: Visual Basic, C++, Java, C# та інші. Прикладні програми отримують доступ до бази даних за допомогою видачі відповідного запиту до СКБД (зазвичай це оператори SQL).
2. *Кінцеві (рядові) користувачі*. Кінцевий користувач може отримувати доступ до бази даних, застосовуючи одне з інтерактивних додатків. Багато СКБД надають не тільки кошти для виконання запитів SQL, але і графічні утиліти, що дозволяють створювати запити без знання SQL.
3. *Адміністратори БД*. Займаються керуванням роботи серверу БД.

Організація даних у БД

У базі даних виділяють наступні елементи:

- дані;
- об'єкти;
- зв'язки;
- властивості.

Дані

У БД дані зазвичай називають *постійними*, хоча вони звичайно не є такими в загальноприйнятому розумінні. Так їх назвали в порівнянні з мінливими даними – *транзитивними* (проміжні результати, вхідні, вихідні дані).

Вхідні дані – це інформація, що передається системі з терміналу або робочої станції. Коли ця інформація збережена в таблицях, вона стає частиною постійних даних або тягне за собою зміни постійних даних.

Вихідні дані – це повідомлення і результати, що видаються системою на екран, друк та інший пристрій виводу.

Об'єкти

У реляційних БД це *таблиці* (інша назва – *відношення),* що описують деякі об'єкти реального світу. Реляційні бази даних зберігають всі дані тільки в таблицях.

Зв'язки

Зв'язки відображають залежності між об'єктами. Як правило, вони бувають двосторонніми. Припустимо, є два об'єкти *Students* і *Groups*, по зв'язку між ними можна відповісти на два питання:

1) до якої групі належить даний студент;
2) які студенти входять у дану групу.

Схема, на якій представлені об'єкти та їх зв'язки, називається *Схема об'єкт-відношення* або *Діаграма об'єкт -відношення* (рис. 1.2.).

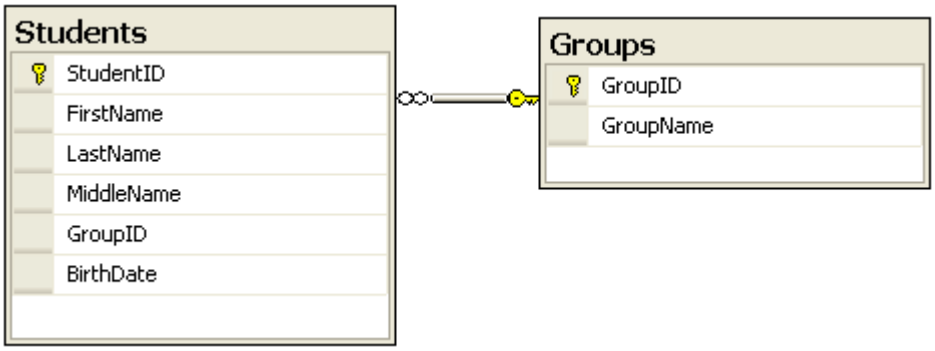


Рис. 1.2. Зв'язок між таблицями *Students* і *Groups*

У схему можуть бути зв'язки, що вказують на один і той самий тип об'єкту. Наприклад, викладач є наставником молодого викладача. У конкретному наборі об'єктів може бути скільки завгодно зв'язків. Між двома таблицями може бути більше однієї зв'язку.

Властивості

Всі об'єкти і зв'язки мають певні властивості. Властивості об'єктів виражаються полями таблиці. Властивості зв'язків виражаються в їх характеристиках при формуванні.

Види моделей даних

Ядром будь-якої БД є модель даних. *Модель даних* - це сукупність структури даних і операцій їх обробки.

Коротко розглянемо основні види моделей даних і виявимо їх основні переваги та недоліки, при цьому будемо враховувати фактори, що характеризують принципові особливості моделей, а також фактори, пов'язані з реалізацією цих моделей на ЕОМ.

1. **Ієрархічна модель даних.** Являє собою сукупність елементів, пов'язаних за суворо визначеними правилами. Об'єкти, пов'язані ієрархічними зв'язками утворюють орієнтований граф. Основними поняттями ієрархічної моделі даних є: *рівень, вузол* (або *елемент*) і *зв'язок*. Така модель даних має такі властивості:

- кожен вузол пов'язаний тільки з одним вищим вузлом, крім вершини;
- ієрархічна модель даних має тільки одну вершину, вузол не підпорядкований більш ніяким вузлам;
- від кожного вузла існує єдиний шлях до вершини;
- зв'язок не може бути встановлена між об'єктами, що знаходяться через рівень;
- зв'язок між вузлами першого рівня не визначається.

Приклади.

1) Файлова структура організації інформації.

2) Структура організації (директор, заступник, керівники відділів, співробітники) (рис.1.3).

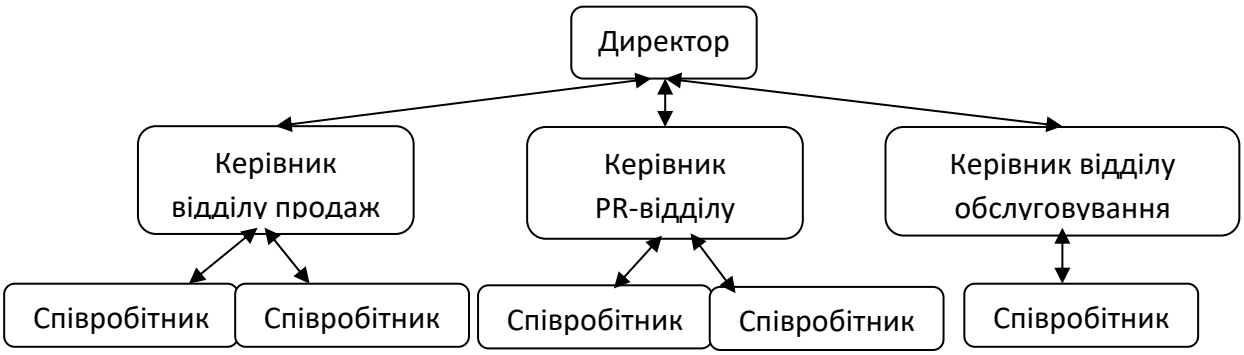


Рис. 1.3. Ієрархічна структура даних

Переваги:

1. Простота.
2. Мінімальна витрата пам'яті.

Недоліки:

1. Відсутність універсальності – не всяку інформацію можна виразити в ієрархічній моделі даних.
2. Виключно навігаційний принцип доступу до даних.
3. Доступ до даних тільки через кореневий елемент.

2. **Мережева модель даних.** Елементами цієї моделі є: рівень, вузол, зв'язок. Відмінності в тому, що елемент одного рівня може бути пов'язаний з будь-якою кількістю елементів сусіднього рівня, і не існує підпорядкованості рівнів один одному.

Властивості мережевої моделі:

- зв'язок не може бути встановлена між об'єктами, що знаходяться через рівень;
- зв'язок між вузлами першого рівня не визначається.

Приклад. Розглянемо роботу над проєктами: можна виділити три види об'єктів – співробітники, проєкти, замовники (рис.1.4).

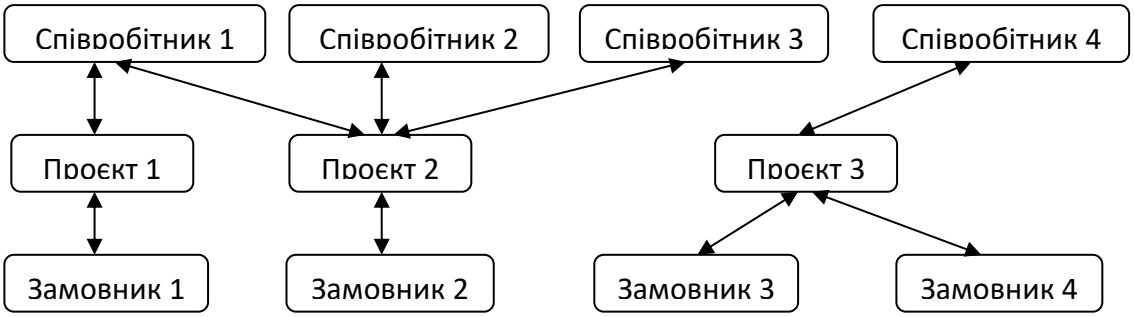


Рис. 1.4. Мережева структура даних

Переваги:

1. Універсальність.
2. Можливість доступу до даних через значення декількох відношень.

Недоліки:

1. Складність – велика кількість понять, варіантів їх взаємозв'язків і способів реалізації.
2. Допустимість тільки навігаційного принципу доступу до даних.

3. **Реляційна модель даних (таблична).** Це спосіб представлення даних у вигляді таблиць. Елементи: *поле* (стовпець), *запис* (рядок) і *таблиця* (відношення). Надалі ми будемо розглядати саме реляційну модель даних, яка використовується в реляційних системах. Під *реляційної системою* розуміється система, заснована на наступних принципах:

- дані користувача представлені тільки у вигляді таблиць;
- користувачу надаються оператори, які генерують нові таблиці зі старих (для вибірки даних).

Приклад. Розглянемо відношення *Студенти* і *Групи*:

Students:

StudentID	LastName	FirstName	MiddleName	GroupID
1	Козаков	Петро	Володимирович	1
2	Васильєв	Іван	Аркадійович	2
4	Шишкіна	Дар'я	Сергіївна	1

Groups:

GroupID	Supervisor
1	Царьов С.М.
2	Пестов Д.Н.

Переваги:

1. Простота. У такій моделі всього одна інформаційна конструкція, формалізує табличне представлення. Вона найбільш звична для користувача.
2. Теоретичне обґрунтування. Існують суворі методи нормалізації даних в таблицях (буде детально розглянуто у лекціях 10-11).
3. Незалежність даних. При зміні БД її структурі необхідні бувають лише мінімальні зміни прикладних програм.

Недоліки:

1. Низька швидкість, тому що потрібні операції з'єднання.
2. Велика витрата пам'яті в силу організації *всіх* даних у вигляді таблиць.

4. **Система інвертованих списків** – система індексів. Систему інвертованих списків можна розглядати як окремий випадок мережевої моделі даних, яка має два рівні. Основні елементи: *основний файл, інвертований список* (файл), *список зв'язків*. У такій системі є кілька основних файлів, що мають єдину наскрізну нумерацію.

Приклад. Розгляньмо об'єкти *Співробітники* і *Зарплата*.

Співробітники:

Співробітник	Посада
01 Іванов	програміст
02 Сидоров	лаборант
03 Шишкін	викладач
04 Васильєв	викладач

Зарплата:

Співробітник	Дата	Сума
05 Іванов	1.10.2008	5000
06 Сидоров	5.10.2008	7500
07 Іванов	3.12.2008	10000
08 Шишкін	3.12.2008	8000
09 Васильєв	25.01.2009	5000
10 Васильєв	27.01.2009	8750

Інвертований список може бути сформований по будь-якому полю основних списків, в ньому кожному значенню зіставляється список номерів (індексів).

Приклад: інвертований список *Посада*:

- програміст – 01;
- лаборант – 02;
- викладач – 03, 04

Список зв'язків складається тільки за основними стовпцями.

Приклад: розгляньмо два списки зв'язків:

- *Співробітники – Зарплата*:
  - 01 – 05, 07
  - 02 – 06
  - 03 – 08
  - 04 – 09, 10
- *Зарплата – Співробітники*:
  - 05 – 01
  - 06 – 02
  - 07 – 01
  - 08 – 03
  - 09 – 04
  - 10 – 04

Інвертовані списки є основою для створення інформаційно-пошукових систем (ІПС). В ІПС ключові атрибути відповідають ключовим словами, що визначають тематику пошуку.

Так як недоліки реляційної моделі даних компенсуються зростанням швидкодії і ресурсів сучасних комп'ютерів, то нині саме такі моделі набули найбільшого поширення.

Архітектура БД

Існує архітектура БД, запропонована дослідницькою групою ANSI/SPARC[1], і називається *архітектурою ANSI/SPARC*.

Кожна система баз даних не зобов'язана відповідати цьому визначенню, наприклад, «малі» системи, досить імовірно, не підтримуватимуть всі функції запропонованої архітектури. Проте розглянута архітектура з достатньою точністю описує більшість систем (і не тільки реляційних).

Прийнято виділяти три рівня в архітектурі СКБД.

1. *Внутрішній рівень* (що також називають *фізичним*) найбільш близький до фізичного сховища інформації, тобто пов'язаний зі способами зберігання інформації на фізичних пристроях. Внутрішній рівень відображає фізичні елементи для зберігання

[1] ANSI/SPARC (American National Standards Institute / Standards Planning and Requirements Committee – Американський національний інститут стандартів / Комітет планування стандартів) - дослідницька група в рамках Американського національного інституту стандартів. Архітектура ANSI/SPARC була запропонована у 1975 році.

інформації. Він являє собою нескінченне адресний простір, з якого інформація транслюється на зовнішні носії.

2. *Зовнішній рівень* (що називають також *користувацьким*) найбільш близький до користувачів, тобто пов'язаний зі способами представлення даних для окремих користувачів (прикладний програміст або кінцевий користувач). Для кожного користувача може існувати своя мова СКБД. Для прикладного програміста – це мова програмування, а для кінцевого користувача – це мова, заснований на меню, формах тощо. Але всі ці мови включають мову даних, вбудовану в СКБД. Для кожного окремого користувача може бути цікава деяка окрема частина БД. Такі частини, з якими працює користувач, називаються *зовнішнім представленням*.

3. *Концептуальний рівень* (що називають також *логічним*) є «проміжним» рівнем між двома першими. Це уявлення всієї інформації БД в більш абстрактній формі. На цьому рівні зберігаються власне дані, незалежні від форми їх подання. Концептуальне представлення складається з множини екземплярів даних. Дані тут представлені у вигляді концептуальної схеми. Крім самих даних в цю схеми можуть бути включені визначення додаткових засобів, наприклад, правила забезпечення цілісності.

Детальна схема архітектури системи баз даних представлена на рис. 1.5.

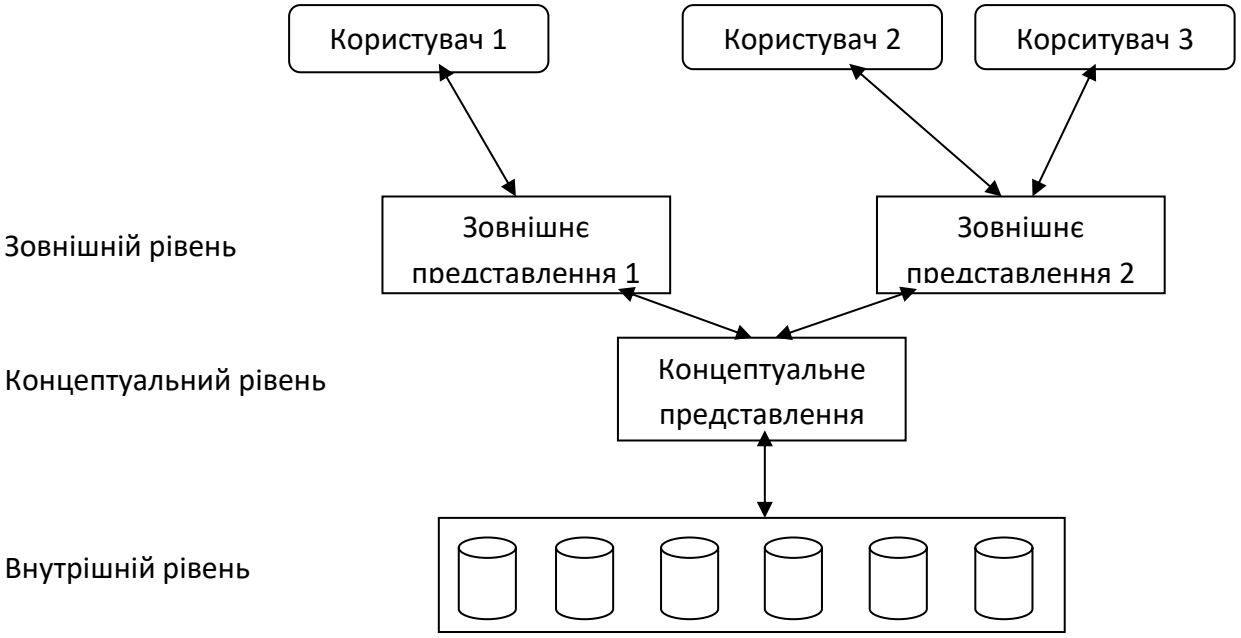


Рис. 1.5. Рівні архітектури систем баз даних

Класифікація БД

Бази даних класифікують за різними ознаками, розгляньмо основні з них.

1. За технологією обробки.
   - Централізовані. Зберігаються в пам'яті однієї обчислювальної системи.
   - Розподілені. Складаються з декількох, можливо пересічних частин, що зберігаються в різних вузлах обчислювальної мережі.

2. За способом доступу до даних.
   - З локальним доступом. Характеризується тим, що до такої БД є доступ користувача однієї ЕОМ.
   - З віддаленим (мережевим) доступом. Доступно для всіх користувачів мережі.
3. За архітектурою.
   - Файл-сервер. Передбачає виділення однієї машини в мережі як центральної (сервер файлів), на ній зберігається централізована БД, яка використовується спільно.
   - Клієнт-сервер. Передбачається виділення сервера БД, який крім зберігання здійснює обробку даних. Систему БД можна розглядати як систему, що складається з двох частин: сервер і набір клієнтів. *Сервером БД* називається власне СКБД, а *клієнтами* – різні додатки, які виконуються над СКБД.
4. За вмістом.
   - Географічні.
   - Історичні.
   - Наукові.
   - Мультимедійні.
   - тощо.

Короткі підсумки. Розглянуто основні поняття баз даних, представлені різні класифікації систем управління базами даних. Визначено основні моделі даних, які використовуються в конкретних реалізаціях СКБД: ієрархічна, мережева, реляційна і система індексів.

ЛЕКЦІЯ 2. КОМПОНЕНТИ MICROSOFT SQL SERVER 2008

У лекції розглядаються Microsoft SQL Server 2008, що функціонують на сервері баз даних, а також додатки, що постачаються разом з MS SQL Server 2008, які можуть використовуватися як на сервері, так і на клієнтських системах. Висвітлюються питання конфігурації сервера і призначення системи баз даних.

Ціль: визначити основні компоненти Microsoft SQL Server 2008 і продемонструвати їх основні можливості.

Microsoft SQL Server 2008

Microsoft SQL Server – це система управління клієнт-серверними реляційними базами даних, орієнтована на роботу під керуванням операційних систем Microsoft Windows. Microsoft SQL Server 2008 (MS SQL Server) підтримує операційні системи Windows Server 2003, Windows Server 2008, Windows XP, Windows Vista.

MS SQL Server включає в себе як серверну, так і клієнтську частину. Однак склад служб, включених в поставку сервера, залежить від версії. MS SQL Server 2008 доступний в шести версіях (редакціях):

- *Enterprise Edition*. Версія з максимальними можливостями для застосування в великих системах. Сюди включені більше 60-ти функцій, недоступних в інших версіях, наприклад: стиснення даних і резервних копій, аудит з використанням розширеного набору подій, утиліта для управління ресурсами *Resources Governor*, можливість гарячої заміни процесора.
- *Standard Edition*. Призначена для використання в системах середнього рівня, де не потрібні можливості *Enterprise* версії. Надає базові можливості з аналітики та створення звітів.
- *Workgroup Edition*. Підходить для установки в філіях компанії і надає засоби управління даними, створення звітності, віддаленої синхронізації і управління.
- *Web Edition*. Орієнтована на роботу в Інтернеті, дозволяє надавати клієнтам доступ до великомасштабних веб-додатків.
- *Express Edition*. Безкоштовна версія. Підходить для навчання, для створення настільних і невеликих серверних додатків, а також для поширення незалежними виробниками ПЗ.
- *Compact Edition*. Безкоштовна версія. Дозволяє створювати автономні або мало пов'язані додатки для мобільних пристроїв, настільних ПК і веб-клієнтів, що працюють під управлінням будь-яких версій Microsoft Windows.

Серверна частина системи

MS SQL Server реалізується у вигляді кількох самостійних служб, кожна з яких відповідає за виконання певних завдань.

- Служба *SQL Server (MSSQLServer)* є ядром цієї СКБД, від її функціонування залежать всі інші служби. Виконує наступні основні функції:
  - розподіляє ресурси комп'ютера між користувачами, які одночасно працюють з системою;
  - управляє файлами баз даних і журналами транзакцій;
  - виконує команди мови Transact-SQL (розглядається в лекції 3), запити і процедури (розглядаються в лекціях 4 і 5 відповідно), які вказуються користувачами;
  - забезпечує безпеку системи (наприклад, здійснює перевірку облікових записів користувачів; система безпеки розглядається в лекції 6);
  - відповідає за узгодженість і цілісність даних, запобігаючи виникненню логічних проблем.

Зауваження. Якщо дана служба не запущена, то ніякі користувачі не можуть підключитися до сервера і ніякі адміністративні завдання не можуть бути виконані!

- Служба *SQL Server Agent* відповідає за автоматичне виконання призначених адміністратором завдань, виконує відстеження певних подій і зіставлених їм завдань (наприклад, створення резервних копій, відправка повідомлення адміністратору про проблему тощо).
- Служба *Full-Text Filter Daemon* дозволяє реалізувати пошук символьної інформації в полях таблиць баз даних. За допомогою цієї служби здійснюється пошук слів і фраз, причому в результаті можуть бути відображені відмінювані форми дієслів та іменників.
- Служба *Integration Services* заміняє *DTS SQL Server 2000* і дозволяє здійснювати наступне:
  - відслідковувати виконання всіх пакетів служб *Integration Services*, що виконуються на комп'ютері;
  - відображати в ієрархічному вигляді пакети і папки служб *Integration Services*, які фізично зберігаються в різних місцях.
- Служба *Analysis Services* – ядро серверу OLAP[2], дозволяє створювати аналітичні додатки з мільйонами рядків даних і тисячами користувачів.
- Служба *Reporting Services* – ця служба представляє серверний компонент, який відповідає за генерацію звітів, надання їх користувачам, виконання різних службових операцій зі звітами.
- Служба *SQL Server Browser* призначена для формування списку доступних в мережі SQL- серверів.

[2] OLAP – On-Line Analytical Processing – оперативна аналітична обробка.

Клієнтська частина системи

MS SQL Server підтримує безліч різних типів клієнтів, кожен з яких може працювати на своїй апаратній і програмній платформі.

У комплект поставки MS SQL Server входять стандартні утиліти, які можуть використовуватися для управління роботою сервера і створення логічної структури баз даних, підтримуваних ним. Для розробки клієнтської програми можуть бути використані і різні засоби розробки додатків, наприклад, середовища візуального програмування Visual Studio .Net 2003-2008, Visual Basic, Delphi тощо.

До стандартних утиліт адміністрування відносяться такі додатки.

SQL Server Configuration Manager

Надає наступні можливості:

- З управління роботою всіх служб MS SQL Server, розглянутих вище. Можна запустити, призупинити або повністю зупинити будь-яку з описаних вище служб, а також вказати, від імені якого користувача її слід запускати.
- З визначення параметрів мережевих бібліотек, які забезпечують взаємодію з MS SQL Server. Можна обрати один або відразу декілька методів доступу до серверу:
  - *іменовані канали* (*Named Pipes*) – технологія схожа на використання сокетів, застосовується в разі недоступності протоколів TCP/IP;
  - *стек протоколів TCP/IP* (використовується за замовчуванням) – підходить для використання через мережу Інтернет;
  - *розподілена пам'ять* (*Shared Memory*) – підходить для локального використання, наприклад, веб-додаток і MS SQL Server знаходяться на одному комп'ютері. Забезпечує максимальну швидкість роботи;
  - *віртуальний інтерфейсний адаптер (Virtual Interface Adapter, VIA*) – використовується для підключень типу сервер-сервер із застосуванням спеціалізованого обладнання.
- З конфігурації мережевих бібліотек клієнта, використовуваних для доступу до MS SQL Server. Після настройки методів доступу до сервера, можна зробити конфігурацію клієнтських протоколів. Вузол *SQL Native Client 10.0 Configuration* містить два розділи: *Client protocols* і *Aliases* (рис. 2.1).

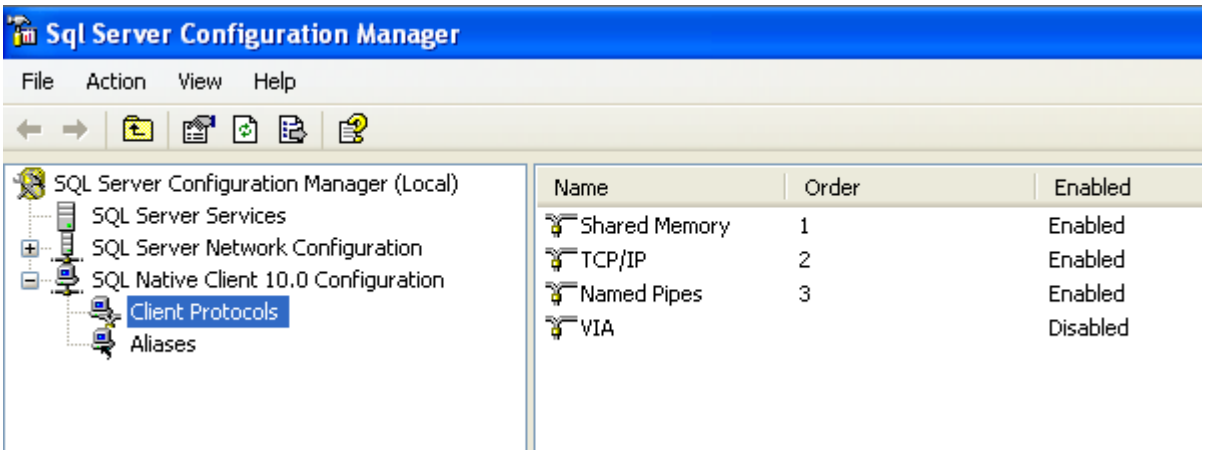


Рис. 2.1. Вікно утиліти *SQL Server Configuration Manager*

Починаючи з MS SQL Server 2000, з'явилася можливість підключення до сервера за допомогою декількох протоколів, наприклад, спочатку намагаємося підключитися через *Shared Memory*, якщо не вийшло, то через TCP/IP, і в останню чергу через *Named Pipes*. Для визначення порядку використання протоколів використовується властивість *Order*.

Вузол *Aliases* дозволяє створювати псевдоніми для підключення до сервера. *Псевдонім* (*Alias*) – це альтернативне ім'я з'єднання, яке може відрізнятися від імені сервера. При створенні псевдоніму можна вибрати протокол і порт, через які слід підключатися до серверу.

SQL Server Management Studio

Утиліта *Management Studio* дозволяє виконувати наступне:

- керувати налаштуваннями MS SQL Server;
- конфігурувати систему безпеки: управління ролями, обліковими записами, віддаленими серверами;
- працювати зі структурою баз даних: створювати, редагувати і видаляти БД і елементи БД;
- керувати виконанням завдань за розкладом;
- показувати поточну активність: поточні користувачі, які об'єкти заблоковані, інформацію про продуктивність.

Перед початком роботи з сервером необхідно підключитися до нього, вказавши наступну інформацію:

- *Server Type*. Тут слід вибрати, до якої саме служби необхідно підключитися: *Database Engine, Analysis Services, Report Server* або *Integration Services*.
- *SQL Server*. Дозволяє вказати, до якого сервера буде здійснюватися підключення. За замовчуванням ім'я *SQL Server* збігається з ім'ям комп'ютера.
- *Authentication Type* – спосіб автентифікації, можна вибрати *Windows Authentication* або *SQL Server Authentication*. Спосіб *Windows Authentication* використовує обліковий запис, під якою поточний користувач здійснив вхід в Windows (рис. 2.2). *SQL Server Authentication* використовує свою власну систему безпеки.

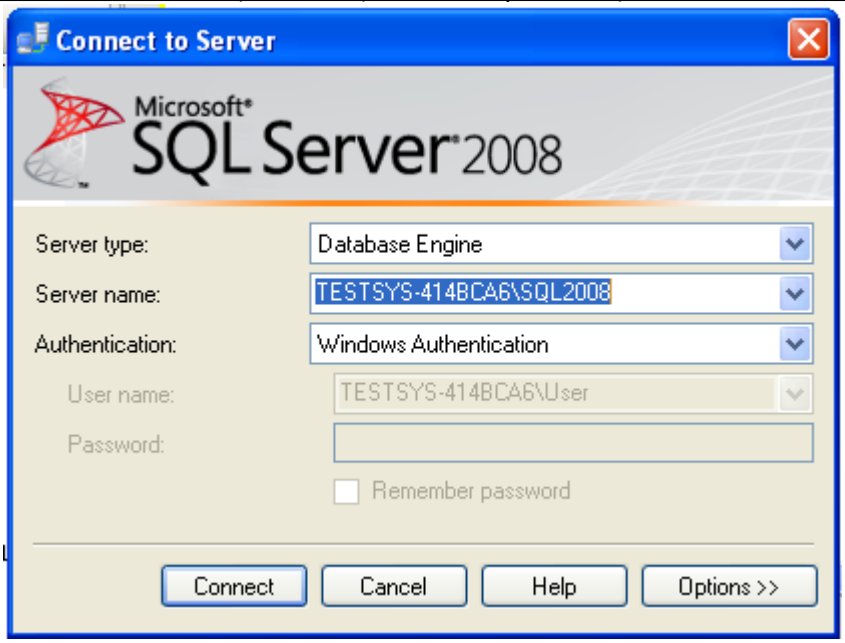


Рис. 2.2. Вікно з'єднання з SQL-сервером

Редактор запитів (Query Editor)

Для того щоб написати новий запит до бази даних, необхідно виконати команду **New Query**, розташовану на панелі інструментів *Management Studio*. В результаті відкриється нова вкладка, в якій можна писати SQL-код (див. рис. 2.3).

Виконаймо наступний запит (мова написання запитів буде висвітлена в лекціях 3 і 4):

```
SELECT * FROM INFORMATION_SCHEMA.TABLES;
```

Зауваження. Для виконання запиту необхідно виконати команду **Query – Execute (F5)**. Щоб просто перевірити правильність синтаксичного запису можна скористатися командою **Query – Parse (Ctrl+F5)**, при цьому сам запиту не буде виконаний**.**

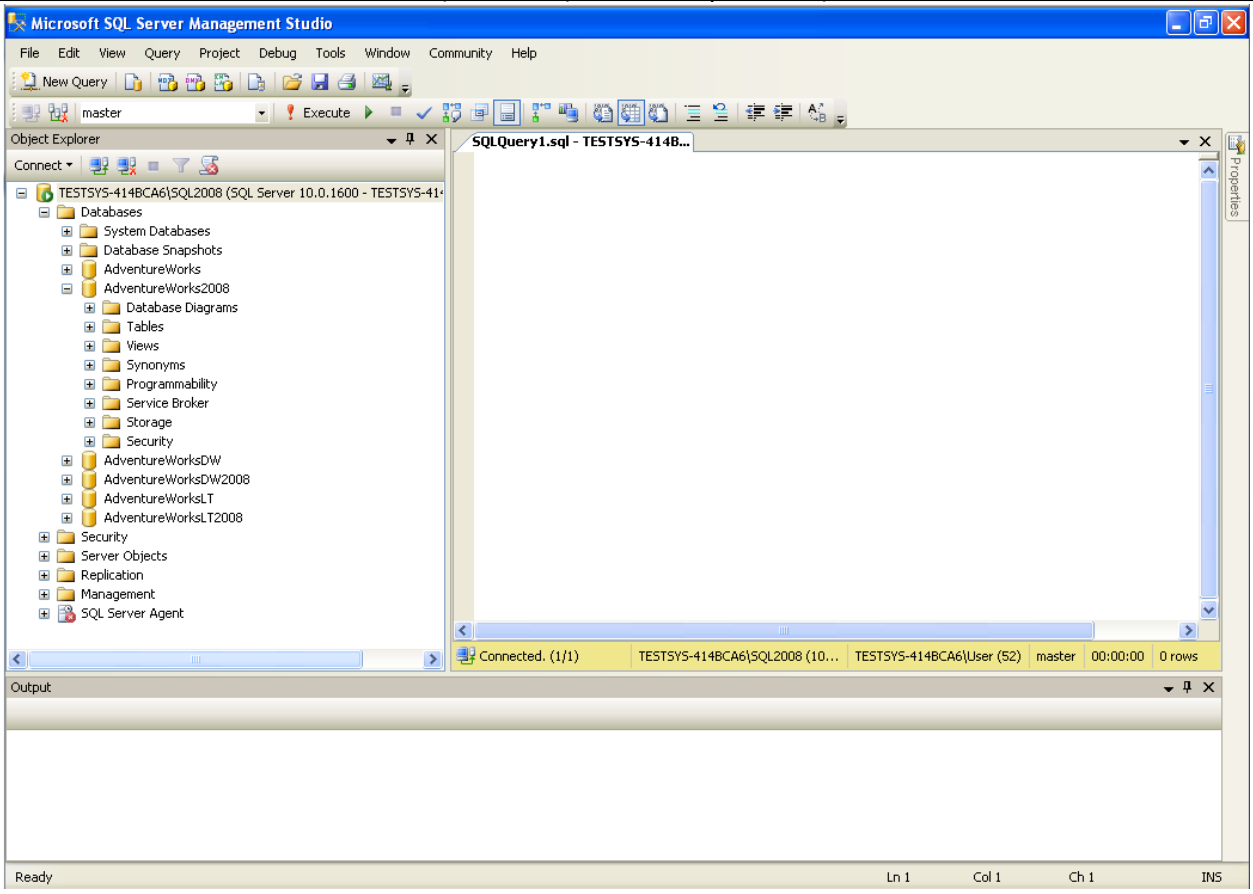


Рис. 2.3. Головний віконний додаток *Management Studio* з вікном *Редактора запитів*

Тепер буде доступно наступне:

- заголовок, в якому вказується логічне ім'я сервера, поточна база даних і ім'я користувача, який встановив з'єднання;
- область запиту, яка використовується для введення запитів, що передаються MS SQL Server;
- область результатів, в якій відображаються результати виконання запиту. Способи відображення результатів можуть бути наступними:
  - *Results in Text* – результати виводяться у вигляді звичайного тексту.
  - *Results in Grid* – результат виводиться у вигляді таблиці, в якій можна змінювати ширину стовпців, виділяти потрібні комірки/рядки/стовпці.
  - *Results to File* – аналогічно *Results in Text*, тільки вивід здійснюється не на екран, а в файл.

Management Studio дозволяє відкривати кілька вікон запитів і працювати з декількома базами даних одночасно. В кожному вікні встановлюється власне з'єднання з MS SQL Server, яке описано у *SQL Server Configuration Manager*, на основі різних облікових записів користувачів і їх паролів. Для створення нового підключення використовується команда **File – New – Database Engine Query.**

Вміст області запиту поточного підключення може бути збережено в файлі на зовнішньому носії командою **File – Save**.

Object Explorer

Дозволяє здійснювати навігацію по базі даних: переглядати доступні об'єкти, виконувати запити на перегляд вмісту таблиць, створювати скрипти для об'єктів тощо. (рис. 2.4).

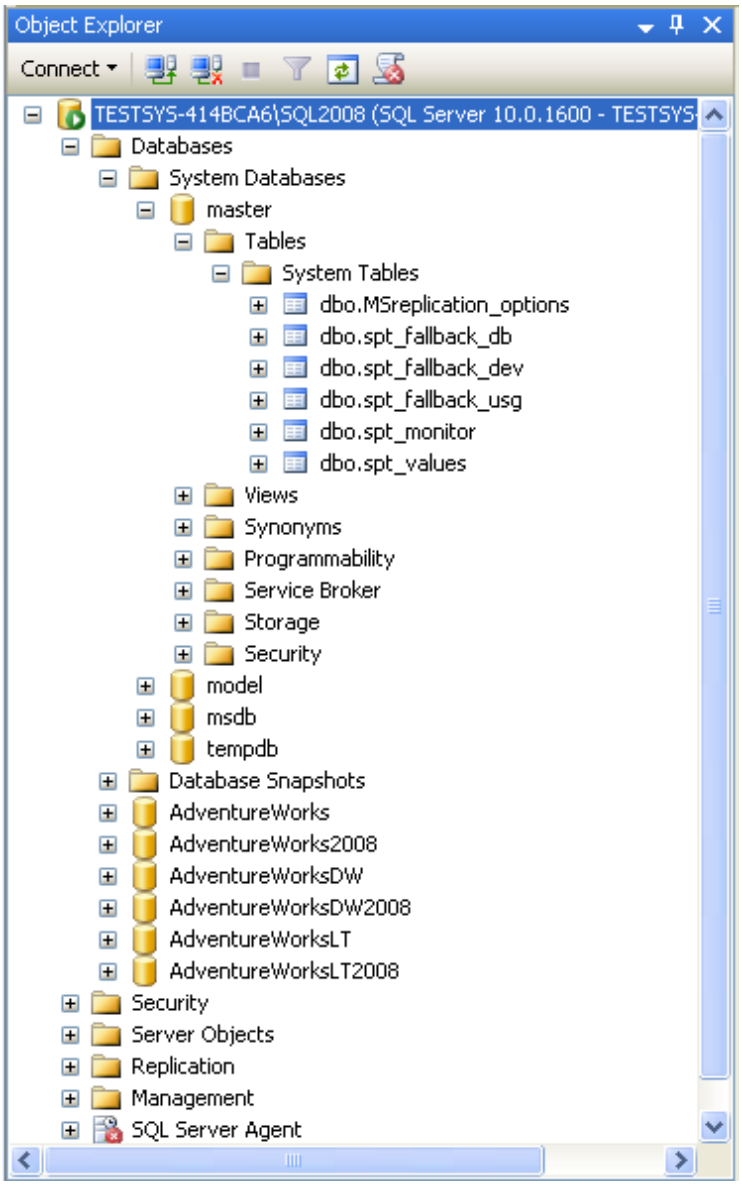


Рис. 2.4. Панель *Object Explorer*

Випадний список баз даних

База даних, обрана в цьому списку, використовується в редакторі запитів як база даних за замовчуванням (див. рис. 2.5). Тому важливо перед виконанням запитів, переконатися, що обрана потрібна БД. Це можна зробити або через випадний список, або за допомогою команди SQL:

```
USE AdventureWorks2008
```

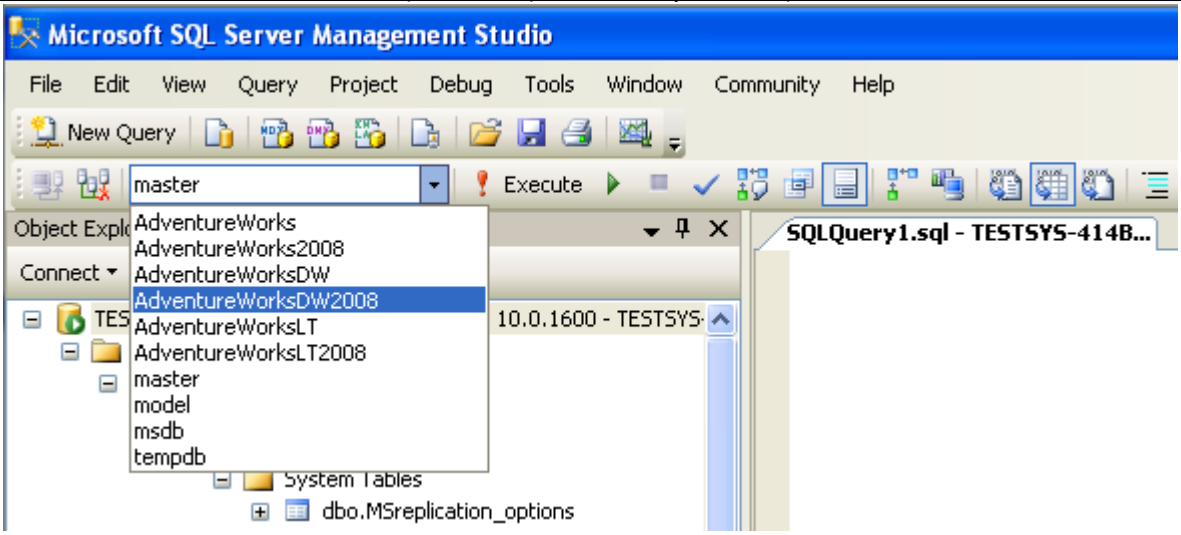


Рис. 2.5. Вікно вибору поточної бази даних

Reporting Services Configuration

Використовується для конфігурації служб звітів. MS SQL Server 2008 включає в себе вбудований web-сервер, тому немає необхідності в установці та налаштуванні служб інтернет-серверів IIS (*Internet Information Services*). Для створення звітів використовується Report Definition Language (RDL) – мова, заснована на XML[3].

Bulk Copy Program

Утиліта командного рядка, призначена для перенесення форматованих даних великого обсягу в MS SQL Server чи з нього. Наприклад, відформатовані дані можуть бути автоматично перенесені зі звичайного текстового файлу в таблицю MS SQL Server.

SQL Server Profiler

Дозволяє в реальному часі відстежувати виконання всіх команд. Профайлер може знаходити «вузькі» місця в базі даних, визначати запити, які довго виконуються, і найбільш часто виконувані запити.

Утиліта sqlcmd

Утиліта командного рядка, яка дозволяє виконувати SQL-скрипти. Дана утиліта може виявитися набагато ефективнішою, ніж *Management Studio*, коли не потрібен графічний користувальницький інтерфейс.

[3] XML (eXtensible Markup Language – розширювана мова розмітки) – текстовий формат, призначений для зберігання структурованих даних.

SQL Server Integration Services (SSIS)

Дозволяє легко отримувати дані з будь-яких джерел через механізм OLE DB[4] або провайдерів даних.NET і поміщати їх в таблиці MS SQL Server. Під час перенесення даних до них може бути застосована трансформація.

SQL Server Business Intelligence Development Studio

Представляє собою версію Visual Studio і дозволяє створювати пакети для *Integration Services*, звіти для *Reporting Services* і працювати з проєктами *Analysis Services*.

Конфігурування MS SQL Server

Конфігурація роботи служби *MSSQLServer* може бути виконано або спеціальною збереженою процедурою, що виконується в утиліті *Management Studio,* або графічним шляхом засобами цієї ж утиліти. Вибір способу не має значення, тому що графічний спосіб здійснює доступ до системних даними за допомогою цієї ж збереженої процедури, тільки в більш наочній формі.

Для зміни параметрів служби за допомогою *Management Studio* необхідно вибрати потрібний сервер і в контекстному меню вибрати команду **Properties**, і в діалоговому вікні, що з'явилося, виконати конфігурування сервера.

На вкладці *General* відображаються основні відомості про систему: версія операційної системи, обсяг пам'яті, кількість процесорів та ін., а також параметри запуску служб серверу.

Вкладка *Memory* дозволяє управляти виділенням пам'яті для виконання дій MS SQL Server: або динамічне управління пам'яттю, або встановлення фіксованого розміру.

Вкладка *Processors* дозволяє управляти тим, на яких процесорах можна виконувати запити SQL.

За допомогою вкладки *Security* визначається тип автентифікації користувачів, також визначаються параметри аудиту доступу до сервера. Можна налаштувати сервер на використання певного облікового запису, під яким буде запускатися служба *MSSQLServer***.**

Вкладка *Connections* дозволяє конфігурувати клієнтські підключення до сервера. Якщо параметр дорівнює 0, то дозволяється підключення максимальної кількості користувачів - 32767 підключень.

За допомогою вкладки *Database Settings* вказуються налаштування новостворюваних баз даних: параметри індексів і роботи з пристроями резервного копіювання, час відновлення бази даних.

Вкладка *Advanced* містить деякі загальні установки сервера. Наприклад, визначається мова за замовчуванням для повідомлень сервера або регулюється підтримка 2000 року, яка визначає, як будуть інтерпретуватися дві останні цифри року.

Вкладка *Permissions* дозволяє управляти іменами входу і ролями, а також керувати правами на виконання дій в MS SQL Server.

[4] OLE DB (Object Linking and Embedding, Database – впровадження і зв'язування об'єктів в базах даних) – набір інтерфейсів, які дозволяють додаткам звертатися до даних з різних джерел.

Системні бази даних

База даних зазвичай являє собою сукупність таблиць і інших об'єктів, таких як представлення, збережені процедури і інші.

MS SQL Server 2008 містить чотири системні бази даних:

- ***master***;
- ***model***,
- ***msdb***.
- ***tempdb***.

Всі ці бази даних необхідні для коректної роботи сервера. Без деяких з них робота MS SQL Server і зовсім буде неможлива. Розглянемо призначення кожної системної бази даних.

БД *master*

Будь-який SQL сервер незалежно від версії містить БД ***master***, в якій фіксується все, що відбувається в системі. Наприклад, при створенні нової БД додається запис в таблицю *sys.databases* БД ***master***. Також всі системні збережені процедури знаходяться в цій БД. Оскільки практично всі, що описує MS SQL Server, зберігається тут, то ця БД критично важлива і не може бути видалена.

БД *model*

Дана БД використовується як шаблон для створення будь-якої нової бази даних. Таким чином, можна внести зміни в цю БД, щоб створювані знову бази містили потрібні зміни. Наприклад, можна додати групу користувачів, яка повинна бути за замовчуванням у всіх нових БД. Оскільки ***model*** використовується як шаблон, то вона обов'язково має бути присутня для нормального функціонування сервера. Крім того, слід мати на увазі, що нові БД повинні мати розмір не менше, ніж БД ***model***.

БД *msdb*

У цій БД служба *SQL Server Agent* зберігає всі системні завдання. Наприклад, якщо задано резервування за розкладом, то в ***msdb*** з'явиться спеціальний запис. Аналогічним чином дану БД використовують і інші підсистеми MS SQL Server, наприклад, *SQL Server Integration Services*.

БД *tempdb*

Фактично ***tempdb*** є робочою областю для MS SQL Server. Наприклад, якщо виповнюється великий складний запит, для виконання якого MS SQL Server потрібно створити тимчасову таблицю, то така таблиця буде створена в БД ***tempdb***. Те ж саме відбудеться і в разі, якщо користувач сам створює тимчасову таблицю, хоча користувачеві може здаватися, що він створює її в поточній базі. На відміну від інших БД, ***msdb*** і сама є тимчасовою: вона відтворюється кожного разу заново при запуску MS SQL Server.

Короткі підсумки. Розглянуто служби MS SQL Server 2008, що забезпечують як базову функціональність, так і додаткову: кошти звітів, інтеграції тощо. Дана характеристика системних баз даних і основне їхнє призначення.

ЛЕКЦІЯ 3. ЗАГАЛЬНІ ВІДОМОСТІ ПРО TRANSACT-SQL

У лекції розглядаються основні типи даних, що використовуються в MS SQL Server 2008, правила оголошення змінних, а також алгоритмічні конструкції, властиві всім мовам програмування, і стандартні функції для обробки даних.

Ціль: дати уявлення про основні можливості мови Transact-SQL.

Загальні відомості про Transact-SQL

У 1970-х роках була розроблена спеціальна мова SEQUEL (Structured English Query Language), який пізніше був перейменований в SQL. У 1986 році комітетом ANSI (American National Standards Institute) був прийнятий перший стандарт мови. Останнім прийнятим стандартом є ISO SQL:2008.
Мова SQL розроблялась як непроцедурна мова, яка буде доступна звичайним користувачам, а не тільки програмістам. Однак з часом мова ставала все складніше і зараз призначена в більшій мірі для програмістів. Незважаючи на існування стандартів мови SQL, в кожній СКБД використовується своє розширення цієї мови. Основною причиною використання розширень SQL є потреба застосовувати SQL не тільки як мову запитів, але і як мову програмування, для цього, як правило, вводяться так звані *збережені процедури*. СКБД Microsoft SQL Server використовує своє розширення мови, яке називається Transact-SQL і включає в себе наступне:

- керівні конструкції;
- локальні змінні;
- додаткові функції (математичні, рядкові та інші);
- підтримка автентифікації Windows.

Типи даних

MS SQL Server підтримує всі основні прості типи даних, що використовуються в сучасних мовах програмування. У версії MS SQL Server 2008 були додані декілька нових типів, а деякі перестали рекомендуватися до використання.
Типи даних в MS SQL Server можна розділити на сім категорій.

1. Цілі числа:
   - **Bit** (1 байт). Насправді перший біт в таблиці буде займати один байт, проте наступні сім біт в цій таблиці будуть зберігатися в тому ж байті.
   - **Bigint** (8 байтів). 64-розрядне ціле число, дозволяє зберігати числа від -2^{63} до $+2^{63}-1$.

   1. **Int** (4 байти). Діапазон значень від –2 147 483 648 до +2 147 483 647.
   2. **SmallInt** (2 байти). Діапазон значень від –32768 до +32767.
   3. **TinyInt** (1 байтів). Діапазон значень від 0 до 255.

2. Числа з фіксованою комою:

- **Decimal** або **Numeric**. Діапазон значень від $-10^{38}-1$ до $+10^{38}-1$.
- **Money** (8 байтів). Грошовий формат, діапазон значень від -2^{63} до $+2^{63}$ з чотирма знаками після коми.
- **SmallMoney** (4 байта). Грошовий формат, діапазон значень від –214748,3648 до +214748,3647.

3. Числа з плаваючою комою:
   4. **Float**. Діапазон значень від –1,79E +308 до +1,79E +308.

4. Дата й час:
   5. **DateTime** (8 байтів). Діапазон значень від 1січня 1753 року до 31 грудня 9999 року з точністю до трьох сотих секунди.
   6. **DateTime2** (6-8 байтів). Новий тип даних, підтримує точність до 0,1 мс.
   7. **SmallDateTime** (4 байти). Діапазон значень від 1 січня 1900 року до 6 червня 2079 року з точністю одна хвилина.
   8. **DateTimeOffset** (8-10 байт). Аналогічний типу **DateTime**, але зберігає ще здвиг відносно часу UTC[5].
   9. **Date** (3 байти). Зберігає тільки дату. Діапазон значень від 1 січня 0001 року по 31 грудня 9999 року.
   10. **Time** (3-5 байтів). Зберігає тільки час з точністю до 0,1 мс.

5. Символьні рядки:
   11. **Char**. Рядок фіксованої довжини. Максимальна довжина рядка 8000 символів.
   12. **VarChar**. Рядок змінної довжини. Максимальна довжина рядка 8000, але при використанні ключового слова «max» може зберігати до 2^{31} байтів.
   13. **Text**. Застосовувався для зберігання великих рядків, зараз рекомендується використання **varchar(max)**, замість **text**. Залишений для зворотної сумісності.
   14. **Nchar**. Рядок фіксованої довжини в Юнікоді. Максимальна довжина рядка 4000 символів.
   15. **NvarChar**. Рядок змінної довжини в Юнікоді. Максимальна довжина рядка 4000 символів, але при використанні ключового слова «max» може зберігати до 2^{31} байтів.
   16. **Ntext**. Аналогічний типу **text**, але призначений для роботи з Юнікодом. Зараз рекомендується використовувати **nvarchar(max)**. Залишений для зворотної сумісності.

6. Двійкові дані:
   17. **Binary**. Дозволяє зберігати двійкові дані розміром до 8000 байтів.
   18. **VarBinary**. Тип даних змінної довжини, дозволяє зберігати до 8000 байт, але при використанні ключового слова «max» до 2^{31} байтів.

[5] UTC (Universal Time Coordinated) – універсальний синхронізований час.

19. **Image**. Використовувався для зберігання великих обсягів даних, зараз рекомендується використовувати **varbinary(max)**. Залишений для забезпечення сумісності.

7. Інші типи даних:
20. **Table**. Особливий тип даних, використовуваний в основному для тимчасового зберігання таблиць і для передачі в якості параметра у функції.
21. **HierarchyID**. Використовується для подання положення в ієрархічній структурі.
22. **Sql_variant**. Тип даних, який зберігає значення різних типів даних, підтримуваних MS SQL Server.
23. **XML**. Дозволяє зберігати XML-дані.
24. **Cursor** (1 байт). Тип даних для змінних або вихідних параметрів збережених процедур, які містять посилання на курсор.
25. **Timestamp / rowversion** (8 байтів). Це тип даних, який представляє собою автоматично сформовані унікальні двійкові числа в базі даних. Значення даного типу генерується БД автоматично при вставці або зміни запису.
26. **UniqueIdentifier** (16 байтів). Представляє собою GUID (Special Globally Unique Identifier). Гарантується унікальність даного значення.

Змінні у Transact-SQL

Будь-який об'єкт бази даних повинен володіти унікальним ім'ям всередині цієї бази. На основі імені відбувається звернення до цього об'єкта. Імена об'єктів називаються *ідентифікаторами*. Transact-SQL накладає ряд обмежень на порядок іменування об'єктів:

- перший символ імені повинен бути одним із символів латинського алфавіту (A-Z, a-z), або символом @, # чи _, тобто не допускається використання цифр та інших спеціальних символів. Інша частина імені може включати будь-які символи алфавіту, цифри і деякі спеціальні символи;
- загальна довжина імені звичайного об'єкта не повинна перевищувати 128 символів, для тимчасових об'єктів – 116;
- всередині імені забороняється використання пробілів, дужок і таких символів, як ~,!,%, ^, & й ін.
- ім'я об'єкта не повинна збігатися із зарезервованим словом та з ім'ям вже наявного об'єкта;
- якщо ім'я об'єкта містить прогалини або збігається з зарезервованим словом, то його необхідно помістити всередину квадратних дужок [].

Зауваження. Transact-SQL є CASE-нечутливою мовою, тобто не розрізняє регістр символів.

Імена локальних змінних повинні задовольняти перерахованим правилам іменування об'єктів і завжди повинні починатися з символу @. Область дії змінної обмежена пакетом операторів або процедурою, в якій вона була оголошена.

Оголошення змінної

Синтаксис команди (всередині квадратних дужок в описі синтаксису розташовуються необов'язкові елементи):

DECLARE @Ім'яЗмінної ТипДаних [,…n]

Приклад оголошення однієї змінної:
DECLARE @sum int

Приклад оголошення кількох змінних:
DECLARE @temp int, @count float

Присвоєння значення

Синтаксис команди:
SET @Ім'яЗмінної = Вираз

Приклад присвоєння значення:
SET @sum = 0

У версії MS SQL Server 2008 з'явилися такі нововведення.

- Ініціалізувати змінну стало можна відразу при оголошенні, наприклад:

DECLARE @iCounter int = 0

- З'явилися оператори + =, - =, * =, / = для короткої форми запису арифметичних конструкцій.

Керівні конструкції Transact-SQL

Групування команд

Групування двох і більше команд в єдиний блок здійснюється за допомогою конструкції:

BEGIN
...
END

Таке групування використовується в умовних і циклічних конструкціях.

Конструкція розгалуження

Виконання тієї чи іншої групи команд в залежності від виконання або невиконання деякої умови реалізується за допомогою конструкції:

```
IF <умова>
        Оператор
[ELSE
        Оператор]
```

При відсутності команд, які виконуються при недотриманні умови, ключове слово ELSE можна не вказувати.

Слід зазначити особливість перевірки значень на NULL (спеціальний маркер, що позначає відсутність інформації). Замість звичайного порівняння: IF @myvar = NULL, слід використовувати оператор IS: IF @myvar IS NULL.

Конструкція CASE

Аналогічна оператору CASE у мовах програмування. В MS SQL Server 2008 оператор CASE має два можливих варіанти використання.

1. З вхідним виразом:

```
CASE <вхідний вираз>
        WHEN <вираз when > THEN <результат>
        [...]
        [ELSE <результат>]
END
```

Приклад:

```
SELECT TOP 10 SalesOrderID, SalesOrderID % 10 AS 'Last Digit', Position = CASE
SalesOrderID % 10
        WHEN 1 THEN 'Один'
        WHEN 2 THEN 'Два'
        WHEN 3 THEN 'Три'
        WHEN 4 THEN 'Чотири'
        ELSE 'Інше'
END
FROM Sales.SalesOrderHeader;
```

Результат запиту представлений на рис. 3.1.

	SalesOrderID	Last Digit	Position
1	43793	3	Third
2	51522	2	Second
3	57418	8	Something Else
4	43767	7	Something Else
5	51493	3	Third
6	72773	3	Third
7	43736	6	Something Else
8	51238	8	Something Else
9	53237	7	Something Else
10	43701	1	First

Рис. 3.1. Результат використання оператора CASE

2. Без вхідного виразу:

```
CASE
        WHEN <логічний вираз> THEN <результат>
        [...]
        [ELSE <результат>]
END
```

Використовується, як правило, для пошуку.

Приклад:

```
SELECT TOP 10 SalesOrderID % 10 AS 'OrderLastDigit',
ProductID % 10 AS 'ProductLastDigit',
"How Close?" = CASE
        WHEN (SalesOrderID % 10) < 3 THEN 'Менше трьох'
        WHEN ProductID = 6 THEN 'ProductID дорівнює 6'
        WHEN ABS(SalesOrderID % 10 - ProductID) <= 1 THEN 'У межах одного'
        ELSE 'Більше одного'
END
FROM Sales.SalesOrderDetail
ORDER BY SalesOrderID DESC;
```

Результат виконання запиту представлений на рис. 3.2.

	OrderLastDigit	ProductLastDigit	How Close?
1	3	2	More Than One Apart
2	3	9	More Than One Apart
3	3	8	More Than One Apart
4	2	2	Ends With Less Than Three
5	2	8	Ends With Less Than Three
6	1	7	Ends With Less Than Three
7	1	0	Ends With Less Than Three
8	1	1	Ends With Less Than Three
9	0	2	Ends With Less Than Three
10	0	4	Ends With Less Than Three

Рис. 3.2. Результат використання оператора CASE

Циклічна конструкція

Transact-SQL підтримує єдиний тип циклу – цикл WHILE, синтаксис якого наступний:

WHILE умова
    Оператор
    [BREAK | CONTINUE]

Зауваження. Вертикальна риса в описі синтаксису означає «або», тобто в даному прикладі може бути вказаний або BREAK, або CONTINUE.

Тіло циклу виконується до тих пір, поки умова істинна. Цикл можна примусово зупинити, якщо виконати в тілі циклу команду BREAK. Якщо ж потрібно почати цикл заново, не чекаючи виконання команд тіла циклу, необхідно виконати команду CONTINUE.

Конструкція WAITFOR

У деяких випадках потрібно відкласти виконання тієї чи іншої команди. Для цих цілей можна скористатися оператором WAITFOR. Ця команда має наступний синтаксис:

WAITFOR DELAY <’time’> | TIME <’time’>

Якщо використовується параметр *DELAY*, то вказується, скільки часу необхідно чекати MS SQL Server. Максимально можлива затримка – 24 години. Приклад використання: WAITFOR DELAY ‘01:00’, який призупинить виконання на одну годину.

Якщо використовується параметр *TIME*, то виконання буде припинено до настання заданого часу. Приклад використання: WAITFOR TIME ‘01:00’ – призупинення виконання коду до настання першої години ночі.

Блок TRY/CATCH

Дану конструкцію можна використовувати для обробки виняткових ситуацій. Вперше це конструкція з'явилася в MS SQL Server 2005.

Блок TRY/CATCH у MS SQL Server працює так само, як і в інших мовах програмування. Використовується наступний синтаксис:

```
BEGIN TRY
{ <вираз SQL> }
END TRY
BEGIN CATCH
{ <вираз SQL> }
END CATCH [ ; ]
```

В блоке BEGIN TRY... END TRY виконуються потенційно небезпечні команди, якщо при цьому станеться помилка рівня 11-19, то виконання буде передано в блок CATCH.

Рівні помилок:

- 1–10 Інформаційні повідомлення. Наприклад, виявлення NULL значень при виконанні агрегатних функцій. Так як керування в блок CATCH ці помилки не передають, то для отримання інформації про помилки можна використовувати функцію @@ERROR.
- 11-19 Відносно серйозні помилки. Наприклад, порушення обмежень зовнішнього ключа.
- 20-25 Дуже серйозні помилки. Це помилки на рівні системи, тому в коді не можна дізнатися про її виникненні. Такі помилки розривають поточне з'єднання і переривають виконання команд.

Коментарі

Існує два види коментарів:

- *однорядкові* – в цьому випадку ігнорується текст праворуч від символів коментаря: -- (подвійний дефіс);
- *багаторядкові* – ігнорується текст, записаний між двома парами символів: /* … */.

Функції Transact-SQL

MS SQL Server має ряд вбудованих функцій для полегшення і прискорення обробки даних. Розрізняють три типи функцій:

- *функції наборів записів* – результатом виконання є об'єкт, який може бути використаний як таблиця даних;
- *агрегатні функції* – результатом є єдине значення заданого атрибуту з деякої множини записів;

- *скалярні функції* – результатом також є одне значення з чітко визначеного набору аргументів.

Скалярні функції

Виділяють наступні категорії скалярних функцій – математичні, рядкові, для роботи з датами, конфігураційні і системні. Розглянемо кожну категорію.

Математичні функції

Більшість математичних функцій повертають результат того ж типу, що і вихідні значення. Наприклад, при перекладі величини кута з градусів в радіани у випадку *Radians(90)* результат дорівнює 1, що невірно, так як в якості аргументу функції використано ціле число. правильний запис: *Radians (90.0).*

Abs (вираз) – обчислення абсолютного значення виразу. Можна використовувати як цілочислові, так і нецілочислові величини.

IsNumeric (вираз) – перевірка, чи має вказане вираз числовий тип даних. Результат дорівнює 1, якщо вираз має числовий тип, інакше - 0.

Rand () – повертає випадкове значення на основі системного часу в діапазоні від 0 до 1.

Floor (вираз) – округлення вказаного значення до найближчого мінімального цілого числа.

Ceiling (вираз) – повертає найближче ціле число, більше або рівне даному.

Power (вираз, степінь) – піднесення до степеня вираження. Тип значення, що повертається збігається з вихідним типом.

Sqrt (вираз) – обчислює квадратний корінь.

Рядкові функції

Ascii (рядок) – повертає ASCII-код самого лівого символу рядка.

Char (вираз) – повертає символ, ASCII-код якого відповідає зазначеному числовому виразу.

Len (рядок) – обчислює довжину рядка в символах.

LTrim (рядок), RTrim (рядок) – видаляють початкові і кінцеві пробіли відповідно.

Left (рядок, число), Right (рядок, число) – повертають вказану кількість символів рядка, починаючи з лівого і правого краю рядка відповідно.

SubString (рядок, початок, довжина) – повертає для рядка підрядок зазначеної довжини, починаючи із зазначеного символу.

CharIndex (рядок1, рядок2 [, старт]) – пошук підрядка *рядок1* в рядку *рядок2*. Повертає порядковий номер першого символу, з якого починається перше входження підрядка в рядок. Додатково можна вказати стартову позицію, з якої буде розпочато пошук.

Stuff (рядок1, початок, довжина, рядок2) – видаляє певну кількість символів *рядку1*, починаючи з зазначеного, і замінює їх новим *рядком2*.

Replace (рядок1, рядок2, рядок3) – замінює всі входження рядка *рядок2* в заданому рядку *рядок1* на *рядок3*.

Reverse (рядок) – повертає рядок, символи якої записані в зворотному порядку по відношенню до початкового рядка.

Функції для роботи з датами

GetDate () – повертає поточний системний час.

IsDate (вираз) – перевіряє правильність виразу на відповідність одному з можливих форматів введення дати.

Day (дата), Month(дата), Year(дата) – повертають день, місяць і рік із вказаної дати.

DateName (тип, дата) – виділяє з дати зазначену в типі частина і повертає її в символьному форматі. Формат частин: yy або yyyy - рік, qq або q - квартал, mm або m - місяць, dd або d - день, wk або ww - тиждень, hh - година, mi або m - хвилина, ss або s - секунда, ms - мілісекунда.

DatePart (тип, дата) – виділяє з дати вказану частину і повертає її в числовому форматі.

DateAdd (тип, число, дата) – додає до зазначеної дати число, тип якого вказаний в першому параметрі.

DateDiff (тип, початок, кінець) – повертає різницю між зазначеними частинами дат в зазначеному типі.

Конфігураційні функції

Повертають інформацію про поточну конфігурацію MS SQL Server. Наприклад:

@@Version – повертає інформацію про дату, версію і тип процесора сервера.

@@ServerName – символьне ім'я локального MS SQL Server.

@@Max_Connections – максимально дозволена кількість одночасних підключень до сервера.

Системні функції

Повертають інформацію про значеннях, об'єкти і поточні параметри MS SQL Server.

DataLength(вираз) – повертає число, що відповідає кількості байт, необхідних для зберігання результату виразу.

@@Error – код останньої помилки, що сталася в поточному з'єднанні. Якщо помилок немає, результат дорівнює 0.

Host_Name() – символьне ім'я комп'ютера в мережі, на якому виконується команда.

System_User і *Session_User* – повертають відповідно ім'я облікового запису користувача для входу і ім'я користувача поточної бази даних.

@@IDLE – визначає кількість мілісекунд, що минула з часу останнього запуску MS SQL Server.

NewID() – генерує нове значення типу **UniqueIdentifier**.

Permission ([ObjectID[, 'column']]) – повертає інформацію про права доступу для поточного користувача. Аргумент *ObjectID* вказує ідентифікаційний номер об'єкта бази даних. Для отримання ідентифікаційного номера об'єкту за його ім'ям використовується функція *Object_ID('ім'я')*.

Результатом даної функції є 32-бітове значення, кожен біт якого відповідає тому чи іншому праву доступу. Якщо значення цього біта дорівнює 1, то доступ до об'єкта дозволений.

Визначимо, наприклад, чи має користувач право вибірки даних з таблиці *Product* БД ***AdventureWorks2008***:

```
SELECT Permissions (object_id('production.product'))
```

Якщо перший молодший біт результату дорівнює 1, то вибірку даних з таблиці *Product* дозволено.

Щоб перевірити право доступу до поля деякої таблиці, необхідно вказати ідентифікаційний номер цієї таблиці даних і в аргументі *'column'* вказати ім'я цього поля. Наприклад, для перевірки можливості вибірки даних з поля *Stor_id* таблиці *Sales* необхідно:

```
SELECT Permissions (object_id('production.product'),'ProductID')
```

Якщо перший молодший біт результату дорівнює 1, то вибірка даних з поля *ProductID* таблиці *Product* також дозволена.

Налагодження коду в Management Studio

У MS SQL Server 2008 з'явилася можливість налагоджувати SQL-код за допомогою покрокового виконання. Для цього існують команди **Debug – Step Into** (F11) і **Debug – Step Over** (F10). Команда **Debug – Toggle Breakpoint** (F9) встановлює точку зупину, до якої буде відбуватися виконання коду.

Під час налагодження доступні такі вікна:

- *Locals* – автоматично містить список всіх локальних змінних і показує їх поточні значення;
- *Watch* – дозволяє вручну додавати змінні для відстеження їх значень;
- *Callstack* – показує стек викликів функцій.

Знімок екрану під час процесу налагодження показаний на рис. 3.3.

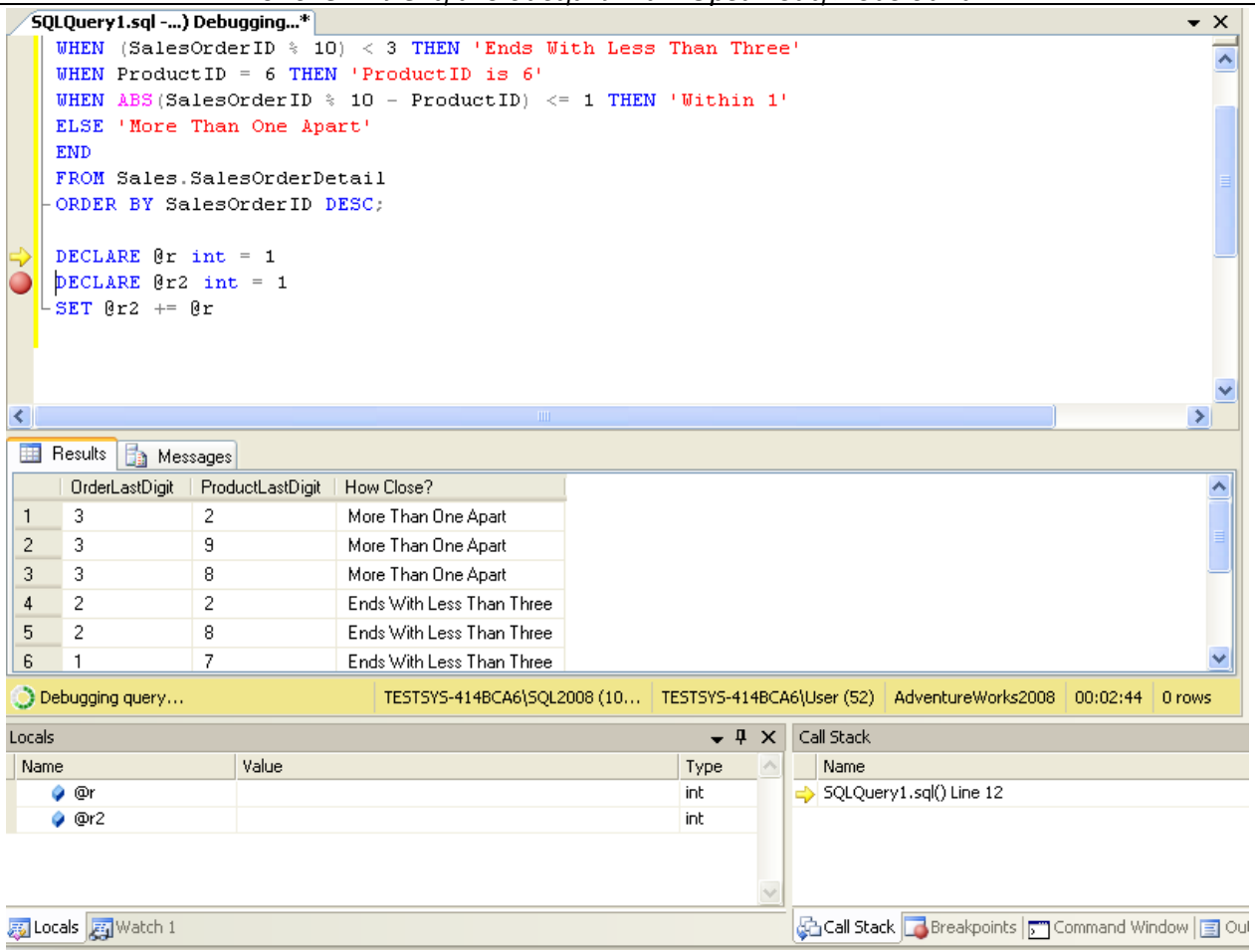


Рис. 3.3. Вікно налагодження у *Management Studio*.
Жовта стрілка вказує на поточну виконувану рядок;
червоний гурток – точка зупинки.

Короткі висновки. Розглянуто типи даних, доступні в MS SQL Server 2008. Продемонстровано використання базових алгоритмічних конструкцій і локальних змінних, а також показана можливість налагодження SQL-коду.

ЛЕКЦІЯ 4. ВИБІРКА ДАНИХ

У лекції детально розглядається оператор вибірки SELECT. Наводяться приклади різних видів з'єднання таблиць при вибірці даних. Також порушуються питання аналітичної вибірки з використанням групування даних і агрегатних функцій.

Ціль: сформувати уявлення про можливості та способи використання оператора SELECT.

Метою будь-якої системи управління базами даних є здійснення операцій над даними: введення, зміна, видалення і вибірка. При цьому вибірка даних є найбільш часто використовуваним аспектом управління даними. Вибірка здійснюється за допомогою однієї єдиної команди SELECT, що є частиною мови DML[6] (Data Manipulation Language – мова керування даними).

Результатом виконання даної команди є *підсумковий набір даних*, що складається з таких частин, як заголовки стовпців і запису даних. Цей набір повинен містити не менше одного стовпчика, записи можуть бути, а можуть і не бути.

Команда SELECT складається з семи основних частин: переліку вибірки і розділів FROM, WHERE, GROUP BY, HAVING, ORDER BY і COMPUTE – обов'язковим є лише список вибірки, при використанні ж інших частин необхідно їх використовувати в тому порядку, в якому вони наведені вище.

Проста вибірка даних

Для простої вибірки даних використовується скорочений синтаксис оператора SELECT:

SELECT [ALL | DISTINCT] [TOP n [PERCENT] *СписокВибірки*
FROM *Ім'яТаблиці*
WHERE *УмоваВідбору*

СписокВибірки визначає поля, що включаються в підсумковий набір даних, *Ім'яТаблиці* вказує таблицю БД, з якої повертаються записи, а *УмоваВідбору* дозволяється обмежити число записів, що повертаються, за допомогою логічних операторів.

За замовчуванням команда SELECT повертає всі записи, включаючи дублікати, що визначається ключовим словом ALL, яке може бути опущено. Для отримання набору унікальних неповторюваних записів необхідно вказувати ключове слово DISTINCT.

Використання ключового слова TOP наказує виводити не всі записи підсумкового набору, а тільки n перших. Можна вибирати не фіксовану кількість

[6] DML – це сімейство комп'ютерних мов, які використовуються в комп'ютерних програмах або користувачами баз даних для отримання, вставки, видалення або зміни даних в базах даних. SQL є найвідомішим і найбільш використовуваним представником мов даного сімейства.

записів, а певний відсоток від усіх рядків – для цього вказується ключове слово PERCENT.

Список вибірки

Список вибірки може містити-включати наступні один або кілька елементів:

* | Ім'яПоля | Вираз [AS Псевдонім], [...n].

Для вибірки всіх полів з таблиці в списку вибірки необхідно вказати зірочку (*).

Ключове слово AS дозволяє замінити в підсумковому наборі даних звичайні імена полів псевдонімами. Ім'я псевдоніма має задовольняти стандартним правилам іменування об'єктів. При необхідності включити неприпустимі символи, пробіли або національні алфавіти, ім'я псевдоніма записують в квадратні дужки.

Наприклад, для отримання списку країн із зазначенням їх коду та останньої дати зміни запису з таблиці CountryRegion бази даних AdventureWorks2008 необхідно вибрати поля CountryRegionCode, Name, ModifiedDate:

```
SELECT     CountryRegionCode AS 'Код',
           [Name] AS 'Країна',
           ModifiedDate AS 'Дата зміни'
FROM Person.CountryRegion
```

При цьому БД ***AdventureWorks*** повинна бути поточною. Зверніть увагу, що перед назвою таблиці використовується ще назва схеми *Person*, призначена для управління об'єктами, пов'язаними з працівниками та департаментами. Поля даних будуть представлені користувачеві в порядку, визначеному в списку вибірки.

Елемент *Вираз* задає вираз, який включається в підсумковий набір даних. Вираз може містити константи, імена полів, функції та їх комбінації. За замовчуванням ім'я колонки з виразом не визначене, тому можна вказати псевдонім.

Наприклад, список співробітників із зазначенням прізвища і першого символу імені та ідентифікаційного номера може бути отриманий в результаті запиту:

```
SELECT LastName+' '+Substring(FirstName,1,1)+'.' AS [Співробітник],
       ContactID
FROM Person.Contact
```

Зручність одержуваного набору даних може бути підвищена шляхом його сортування в збільшувальному або спадному порядку. Сортування можливе за ім'ям поля (навіть якщо воно і не вказано в списку вибірки), за псевдонімом або за позицією в списку вибірки, які вказуються в розділі ORDER BY Ім'яПоля [,…n] [ASC | DESC].

За замовчуванням сортування здійснюється за зростанням, що відповідає зарезервованому слову ASC, яке може опускатися, для сортування в порядку спадання вказується – *DESC*.

Для відображення, розглянутого раніше, списку співробітників упорядкованого в алфавітному порядку необхідно доповнити запит:

```
SELECT LastName+' '+Substring(FirstName,1,1)+'.' AS [Співробітник],
        ContactID
FROM Person.Contact
ORDER BY [Співробітник]
```

Умова відбору

Умову відбору визначає критерій відбору записів, що включаються в підсумковий набір. В результат будуть включені тільки ті рядки, які відповідають накладеним умовам.

Умова може включати вирази, утворені за допомогою операторів порівняння або логічних операторів. Умови можуть також об'єднуватися і за допомогою логічних операндів AND, OR і NOT.

Наприклад, щоб отримати список товарів, ціна яких знаходиться в діапазоні від $ 12 до $ 20, необхідно виконати наступний запит:

```
SELECT [Name], ListPrice
FROM Production.Product
WHERE (ListPrice>=12) and (ListPrice<=20)
ORDER by 2
```

Ці два оператора порівняння можуть бути замінені одним логічним оператором BETWEEN, за допомогою якого можна отримати відповідь на питання, чи лежить величина в зазначеному діапазоні.

```
SELECT [Name], ListPrice
FROM Production.Product
WHERE ListPrice BETWEEN 12 and 20
ORDER BY ListPrice
```

Даний оператор призначений лише для того, щоб полегшити логіку сприйняття алгоритму.

Для пошуку за шаблоном символьних рядків використовується логічний оператор LIKE, який найчастіше використовується в ситуаціях, коли невідомо точний збіг.

У шаблоні можна використовувати такі універсальні символи:

- % – означає будь-який рядок, що складається з 0 і більше символів;
- _ – рівно один символ;
- [] – будь-який символ із заданої множини (Наприклад, [adfh]) або діапазону (Наприклад, [0-9]),
- [^] – будь-який символ, який не потрапляє в заданий діапазон або множину.

Наприклад, щоб вивести імена співробітників і їх посад, які проживають за межами США, необхідно виконати запит:

```
SELECT LastName, JobTitle
FROM HumanResources.vEmployee
```

```
WHERE CountryRegionName Not LIKE '%United States%'
```

Для визначення відповідності виразу одному з перерахованих в заданому списку значень застосовується логічний оператор IN. Даний оператор завжди може бути записаний і у вигляді групи умов, об'єднаних операндом OR.

Наприклад, за допомогою наступного запиту може бути отриманий список співробітників з США і Канади:

```
SELECT LastName, JobTitle
FROM HumanResources.vEmployee
WHERE CountryRegionName IN ('United States', 'Canada')
```

Однак в список значень не можна включати невизначене значення NULL, для роботи з такими значеннями використовується функція вибірки IS NULL.

Наприклад, наступний запит повертає список товарів, у яких не вказано колір:

```
SELECT [Name]
FROM Production.Product
WHERE Color IS NULL
```

Вибірка даних з кількох таблиць

Така вибірка даних передбачає з'єднання декількох таблиць для отримання єдиного набору результатів, що включають записи і поля кожної таблиці. З'єднання дозволяє зібрати дані, розділені в процесі нормалізації.

Існує три види з'єднань: внутрішнє, зовнішнє і перехресне. Для об'єднання трьох і більше таблиць можна застосовувати послідовність з'єднань.

Для з'єднання таблиць необхідно розділ FROM доповнити ключовими словами JOIN, яке визначає з'єднувані таблиці і метод з'єднання, і ON, яке вказує загальні для таблиць поля.

Внутрішнє з'єднання

При такому вигляді з'єднання порівнюються значення загальних полів двох таблиць, повертаються тільки записи, що задовольняють критерію зв'язування в обох таблицях. Записи, для яких немає пари в пов'язаній таблиці, в результат не включаються.

Наприклад, в таблиці Product нормалізованої бази даних AdventureWorks зберігатися тільки ідентифікатор категорії товару. Щоб отримати список товарів із зазначенням їх категорій, необхідно з'єднати таблиці Product і ProductSubcategory:

```
SELECT Product.Name, ProductSubcategory.Name, Product.ListPrice
FROM Production.Product INNER JOIN Production.ProductSubcategory
ON Product.ProductSubcategoryID = ProductSubcategory.ProductSubcategoryID
```

При такому з'єднанні товари, для яких не зазначено їх категорія, не включаються до набору результатів.

Зверніть увагу, що в розділі FROM необхідно вказати ім'я схеми Production, а в інших випадках перед зазначенням поля використовується ім'я таблиці Product і ProductSubcategory для вирішення конфліктів, тому що в обох таблицях присутні поля Name.

Внутрішнє з'єднання

Таке з'єднання також повертає об'єднані записи, що відповідають умові об'єднання. Однак ліві та праві з'єднання повертають і записи, що не відповідають вказаній умові об'єднання. Такі сполуки застосовуються для отримання повного набору записів однієї з таблиць.

За *лівого з'єднання* в результат будуть включені всі записи лівої таблиці (ім'я якої розташовано ліворуч від JOIN), незалежно від того, є для них відповідний запис в правій таблиці (ім'я таблиці розташоване праворуч від JOIN) чи ні.

Наприклад, наступний запит повертає ім'я контактної особи і дату розміщення замовлення:

```
SELECT FirstName, LastName, OrderDate
FROM Person.Contact LEFT JOIN Sales.SalesOrderHeader
ON Contact.ContactID = SalesOrderHeader.ContactID
```

Для осіб, які не розміщували замовлення, в поле OrderDate міститься значення NULL.

За *правого з'єднання* (ключове слово RIGHT JOIN) в результат включаються всі записи правої таблиці, незалежно від того, є для них відповідний рядок в лівій таблиці.

Змініть розглянутий запит так, щоб він видавав такі ж результати при використанні правого з'єднання.

Перехресні з'єднання

При такому з'єднанні виводяться всі комбінації записів таблиць, при цьому не потрібно вказувати значень полів, що співпадають, тому умова ON опускається.

У нормалізованих базах даних перехресні з'єднання найчастіше використовуються для отримання списку всіх можливих комбінацій записів двох таблиць, тобто число записів підсумкового набору дорівнює добутку числа записів першої таблиці на число записів другої.

Наприклад, за допомогою перехресного з'єднання можна перерахувати всі можливі способи поставки товарів в базі даних ***Northwind*** (поставлялася з MS SQL Server до появи ***AdventureWorks***):

```
SELECT DISTINCT Suppliers.Country, Orders.ShipCountry
FROM Suppliers CROSS JOIN Orders
```

Об'єднання декількох наборів результатів

Оператор UNION об'єднує результати двох і більше операторів SELECT і застосовується в разі, коли дані не можна отримати за допомогою одного запиту.

Для отримання єдиного підсумкового набору даних необхідно написати окремі оператори SELECT і об'єднати їх за допомогою оператора UNION, при цьому, на відміну від з'єднання, записи в підсумковий набір додаються один за одним:

```
SELECT ...
UNION [ALL]
SELECT ...
[,...]
```

За замовчуванням повторювані записи видаляються, для отримання всіх записів необхідно вказати ключове слово ALL. Необхідно також враховувати, що список полів, порядок і всі їхні властивості повинні бути однакові у всіх використовуваних запитах.

Імена полів підсумкового набору беруться з першого запиту, тому створення псевдонімів полів виконується в ньому. Для отримання відсортованого набору даних розділ ORDER BY вказується після останнього оператора SELECT.

Аналітична вибірка даних

Аналітична вибірка даних з бази даних спирається на можливості Transact-SQL створювати запити, нерозривно пов'язані з агрегатними функціями:

- *Avg ([all | distinct] вираз)* – середнє арифметичне всіх значень.
- *Count ([all | distinct] вираз | *)* – кількість значень у списку, відмінних від NULL. При використанні символу * підраховується кількість значень, включаючи значення NULL або повторювані значення.
- *Sum ([all | distinct] вираз)* – сума всіх значень списку.
- *Max ([all | distinct] вираз)* – максимальне значення.
- *Min ([all | distinct] вираз)* – мінімальне значення.

Ключове слово **all** наказує виконувати агрегування всіх записів в результуючому наборі даних, **distinct** - агрегування тільки унікальних записів. За замовчуванням використовується **all**.

Наприклад, обчислення середньої ціни товарів здійснюється за допомогою наступного запиту:

```
SELECT AVG(ListPrice) FROM Production.Product
```

При виконанні агрегатної функції здійснюється об'єднання значень окремого поля таблиці або частини записів, після чого виконується вказане агрегування.

Агрегатна функція повертає одне єдине значення, тому використання інших імен полів в списку вибірки заборонено.

Підзапити

Підзапит – це оператор SELECT, включений в інші запити. Підзапити застосовуються для розбивки складного запиту на серію логічних етапів. Їх застосування ефективно, якщо запит використовує записи, повернуті іншим запитом.

Існує два види підзапитів.

1) Вкладені підзапити – повертають єдине значення або список значень. Вкладений запит виконується один раз, а потім результуюче значення використовується в зовнішньому запиті.

Наприклад, щоб визначити імена замовників, які розмістили замовлення в останній обліковий день, можна скористатися наступним запитом:

```
SELECT CustomerID
FROM Sales.SalesOrderHeader
WHERE OrderDate =
      (SELECT Max(OrderDate) FROM Sales. SalesOrderHeader)
```

Виконання цього запиту здійснюється в два етапи: на першому – здійснюється виконання підзапиту, як самостійного запиту, що повертає значення, яке використовується на другому етапі при виконанні зовнішнього запиту.

В результаті отримуємо імена і прізвища, які можуть бути використані в підзапиті для отримання повного імені замовника:

```
SELECT FirstName + ' ' + LastName AS 'CustomerName'
FROM Sales.vIndividualCustomer
WHERE CustomerID in
      (SELECT CustomerID
      FROM Sales.SalesOrderHeader
      WHERE OrderDate=
            (SELECT Max(OrderDate) FROM Sales. SalesOrderHeader))
```

Такий запит може бути оформлений і в вигляді з'єднання таблиць. Такі запити обробляються значно швидше, тому якщо виконання запиту не потрібно виконувати в кілька етапів, використання підзапитів не обов'язково.

2) Пов'язані підзапити – використовуються дані зовнішнього запиту, причому пов'язаний запит виконується один раз для кожного запису зовнішнього запиту.

Наприклад, для визначення переліку замовників, які замовили за один раз більше 300 одиниць товару, необхідно виконати наступний запит:

```
SELECT SalesOrderID, CustomerID
FROM Sales.SalesOrderHeader oh
WHERE 300 < (SELECT sum(OrderQty)
            FROM Sales.SalesOrderDetail od
            WHERE od.SalesOrderID = oh.SalesOrderID)
```

В пов'язаних підзапитах, щоб розрізняти імена таблиць необхідно застосовувати їх псевдоніми. У прикладі для таблиці *SalesOrderHeader* визначено псевдонім *oh,* а для *SalesOrderDetail – od.*

Однак використання пов'язаних підзапитів неефективно, краще перетворювати їх в з'єднання таблиць, що дозволяє оптимізатору запитів вибирати найкращий спосіб обробки даних.

Наприклад, наступний запит для кожного товару відображає відомості про найбільше його замовлення: кількість замовленого товару і номер замовлення.

```
SELECT Name, OrderQty, SalesOrderID
FROM Production.Product p
INNER JOIN Sales.SalesOrderDetail od1 ON od1.ProductID=p.ProductID
WHERE OrderQty = (SELECT Max(OrderQty)
                  FROM Sales.SalesOrderDetail od2
                  WHERE od1.ProductID = od2.ProductID)
```

Групування записів

Для групування записів за полями чи виразами застосовується розділ GROUP BY оператора SELECT, що дозволяє застосовувати для кожної групи функції агрегування.

Синтаксис цієї частини наступний:

```
[GROUP BY ВиразГрупування, [...n]]
```

Наприклад, щоб визначити кількість товарів в кожному замовленні, необхідно згрупувати записи з однаковим ідентифікаційним номером замовлення SalesOrderID і підрахувати кількість записів в кожній групі:

```
SELECT SalesOrderID, count(ProductID) AS [Количество]
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
```

При використанні GROUP BY для кожної визначеної групи значень виводиться тільки один запис в підсумковому наборі даних.

Якщо для кожного замовлення ще необхідно підрахувати і загальну кількість товарів у замовленні, то запит повинен бути доповнений:

```
SELECT SalesOrderID,
        count(ProductID) AS [Кількість],
        sum(OrderQty) AS [Сума товарів]
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
```

За групування записів допускається також використання розділу WHERE, в цьому випадку групуються записи, що задовольняють цій умові.

Розділ WHERE дозволяє визначити, які записи повинні піддатися угрупованню, а розділ HAVING - які групи повинні бути виведені в підсумковий набір даних. Ключове слово HAVING можна використовувати тільки в розділі GROUP BY.

Так за допомогою цього розділу можна в підсумковий набір даних помістити тільки ті замовлення, сума товарів в яких перевищує 150 одиниць:

```
SELECT SalesOrderID,
        count(ProductID) AS [Кількість],
        sum(OrderQty) AS [Сума Товарів]
FROM Sales.SalesOrderDetail
GROUP BY SalesOrderID
HAVING sum(OrderQty) >= 150
```

Короткі висновки. Були розглянуті основні варіанти використання оператора SELECT, можливість і необхідність з'єднання таблиць (внутрішнє, зовнішнє, перехресне). Крім того були вивчені агрегатні функції MS SQL Server і способи їх застосування в операторі SELECT.

ЛЕКЦІЯ 5. ДОПОМІЖНІ ОБ'ЄКТИ БАЗДАНИХ

У лекції розглядаються часто використовувані об'єкти в базах даних: збережені процедури і подання. Демонструється створення і керування цими об'єктами за допомогою команд SQL.

Ціль: сформувати поняття про призначення збережених процедур і подань.

Поняття збереженої процедури

Збережена процедура (Stored procedure) – це іменований набір операторів Transact-SQL, що зберігається на сервері.

Організація взаємодії між клієнтом і сервером за допомогою збережених процедур передбачає наступне: клієнт за відомим іменем викликає блок команд, що зберігається на сервері бази даних, сервер виконує цей блок команд і повертає клієнту результат. Таким чином, використання збережених процедур знижує мережевий трафік і скорочує число запитів клієнтів, тому що, замість пересилання мережею декількох операторів, передається лише ім'я процедури, що виконується.

Збережені процедури є самостійними об'єктами бази даних, до яких можна дозволяти або забороняти доступ командами GRANT і DENY. Наприклад, виконання наступної команди заборонить виконання команд збереженої процедури **hello** для користувача *mng*:

```
DENY exec ON hello TO mng.
```

Збережені процедури схожі з процедурами інших мов програмування і дозволяють:

- вмикати різні оператори і викликати інші процедури, що зберігаються;
- приймати вхідні параметри і повертати значення у вигляді вихідних параметрів.

MS SQL Server підтримує наступні види збережених процедур:

- *системні процедури* – зберігаються в системній базі даних **master**, їх імена починаються з символів sp_. Використовуються для вирішення спеціалізованих системних задач: адміністрування, безпеки та ін.;
- *користувацькі процедури* – створюються, зберігаються і виконуються користувачами в контексті тільки тієї бази даних, для якої були створені;
- *тимчасові процедури* – доступні тільки в активному з'єднанні, після закриття з'єднання видаляються автоматично. Імена таких процедур повинні починатися з символу #.

Для роботи зі збереженими процедурами призначені системні збережені процедури:

- **sp_helptext** *Ім'яПроцедури* – виводить код зазначеної збереженої процедури;

- **sp_help** *Ім'яПроцедури* – виводить список параметрів і їх типів даних для зазначеної процедури;
- **sp_stored_procedures** – повертає список збережених процедур поточної бази даних.

Як і більшість об'єктів MS SQL Server збережена процедура може бути створена за допомогою засобів Transact-SQL або із застосуванням графічного інтерфейсу Management Studio.

При створенні збереженої процедури необхідно враховувати такі особливості:

- процедура може містити необмежено кількість операторів, крім операторів створення наступних об'єктів: процедури, подання, правила, замовчування;
- створення процедури може виконати користувач ролі sysadmin, db_owner або db_ddladmin, а також той, хто має право на виконання команди CREATE PROC;
- кількість параметрів не повинно перевищувати 2100.

Створення процедури засобами Transact-SQL

Створення збереженої процедури полягає у виконанні наступної команди:

```
CREATE PROCEDURE|PROC <sproc name>
[<parameter name> [<schema>.]<data type> [VARYING]
[= <default value>] [OUT[PUT]] [READONLY]
[, n...]
[WITH
RECOMPILE| ENCRYPTION | [EXECUTE AS { CALLER|SELF|OWNER|'<user name>'}]
[FOR REPLICATION]
AS
<code> | EXTERNAL NAME <assembly name>.<assembly class>.<method>
```

- Ім'я процедури повинно задовольняти правилам іменування об'єктів MS SQL Server;
- parameter name визначає ім'я параметра (має починатися з символу @), який буде використовуватися для передачі вхідних або вихідних даних (при вказівці ключового слова OUTPUT);
- data type вказує, до якого типу даних повинні відноситись значення параметра;
- default value дозволяє визначити значення за замовчуванням, якщо при виклику процедури параметр був не вказаний;
- опція READONLY створює параметр доступний тільки для читання, якщо параметр має тип table, то зазначення READONLY обов'язково;
- режим WITH ENCRYPTION забороняє подальший перегляд коду створюваної збереженої процедури, шифруючи його;

- режим RECOMPILE вказує, що сервер не кешує план виконання процедури, і процедура компілюється тільки під час виконання.

Після ключового слова AS йдуть або команди Transact-SQL, які і складають тіло процедури, або прописується метод із вказаної збірки .Net Framework.

Приклад створення збереженої процедури з шифруванням:

```
CREATE PROCEDURE HumanResources.uspEncryptThis
WITH ENCRYPTION
AS
 SELECT BusinessEntityID, JobTitle, NationalIDNumber,
                VacationHours, SickLeaveHours
 FROM HumanResources.Employee;
```

Щоб переконатися, що вихідний текст процедури недоступний, можна виконати наступний код:

```
EXEC sp_helptext 'HumanResources.uspEncryptThis';
```

Результатом выполнения будет сообщение:

```
The text for object 'HumanResources.uspEncryptThis' is encrypted
(Текст об'єкта 'HumanResources.uspEncryptThis' зашифрований.)
```

Виконання процедури

Процедура може бути виконана за допомогою оператора EXECUTE:

```
EXEC[UTE]
      [@СтатусПовернення =] Ім'яПроцедури
      [ [@параметр=] {Значення | Вираз} [OUTPUT] ] [,...]
```

При використанні вихідного параметра, його необхідно описати з використанням ключового слова OUTPUT при створенні процедури, а також використати це слово і при вказівці відповідного параметра при виклику процедури. В іншому випадку процедура не передає вихідне значення.

Параметр @ СтатусПовернення використовується для отримання значення коду повернення зі збереженої процедури, виконаного за допомогою оператора RETURN Статус. Користувачеві рекомендовано використовувати позитивні значення статусу.

Керування збереженими процедурами

Зміна

Для зміни наявної процедури використовується оператор ALTER PROC, параметри цієї команди аналогічні параметрам команди створення процедури.

Перейменування

Для цього необхідно використовувати спеціальну системну збережену процедуру:

sp_rename 'ІмяОб'єкту' 'НовеІмяОб'єкту'.

Видалення

Для видалення збереженої процедури використовується команда Transact-SQL:
DROP PROC Ім'яПроцедури.

Подання

Подання (View) є ще одним об'єктом, що становить логічну структуру будь-якої бази даних. Подання для кінцевих користувачів виглядає як таблиця, проте при цьому не містить даних, а лише подає їх. Фізично ж подані дані розташовані в різних таблицях бази даних.

Подання реалізується у вигляді збереженого запиту, на основі якого і проводиться вибірка з різних таблиць бази даних.

Подання мають наступні переваги:

забезпечують конфіденційність інформації, тому що дозволяють відобразити тільки необхідну інформацію, приховуючи певні поля;

спрощують подання даних, тому що користувач працює з поданням як з єдиною таблицею, яка створена на основі вибірки даних з декількох таблиць;

керують правами доступу до даних, наприклад, замість того, щоб надавати права на виконання запитів до певних полів таблиць, простіше дозволити виконання запитів через подання.

MS SQL Server надає різні способи створення подань: за допомогою засобів Transact-SQL і в утиліті адміністрування *Management Studio*.

Створення подань за допомогою Transact-SQL

Для створення подання використовується команда CREATE VIEW, право її виконання мають члени ролей *sysadmin, db_owner, db_dlladmin*:

```
CREATE VIEW Ім'яПодання [(поле [,...n])]
[WITH ENCRYPTION]
AS
ЗапитВибірки
```

При вказанні *Ім'яПодання* необхідно дотримуватися раніше визначених правил іменування об'єктів, також це ім'я не повинно збігатися з ім'ям вже наявної таблиці в базі даних. Параметр WITH ENCRYPTION визначає шифрування коду запиту і гарантує, що користувачі не зможуть переглянути і використовувати його.

ЗапитВибірки є оператор SELECT, параметри якого і визначають вміст подання. Імена полів подання задаються або з допомогою псевдонімів в операторі вибірки, або вказуються в параметрі поля.

Наприклад, створимо подання, що містить лише таку інформацію про співробітників компанії **AdventureWorks**, як: посаду і логін співробітника, дата народження.

```
CREATE VIEW InfoEmployees ([Номер], [Фамилия], [Дата рождения]) AS
SELECT BusinessEntityID, JobTitle + '(' + LoginID + ')',
        CONVERT (char(10), BirthDate, 104)
FROM HumanResources.Employee
```

Для перегляду вмісту подання скористайтеся наступним запитом:

```
SELECT * FROM InfoEmployees
```

За допомогою даного подання обмежений доступ до деяких полях вихідної таблиці *Employee*, в цьому випадку говорять, що на таблицю накладений *вертикальний фільтр*, тобто обмежений доступ до частини полів таблиці без захисту на рівні стовпців.

Якщо в коді запиту вибірки визначено умову відбору записів, то кажуть, що на таблицю накладений *горизонтальний фільтр*. Наприклад, наступне подання забезпечує доступ до інформації про виробників, що мають онлайн-служби для замовлення товару:

```
CREATE VIEW OnlineVendors
AS
SELECT [Name]
FROM Purchasing.Vendor
WHERE PurchasingWebServiceURL IS NOT NULL
```

У запиті вибірки може бути вказана команда SELECT будь-якої складності, однак при цьому забороняється використовувати розділу ORDER BY, який в подальшому можна застосувати при вибірці даних зі створеного подання. Також рекомендується створювати подання тільки на основі таблиць, для яких виконано внутрішнє з'єднання.

Наприклад, створимо подання, що відображає сумарну вартість кожного замовлення із зазначенням замовника і його номера:

```
CREATE VIEW InfoOrders
AS
SELECT FirstName + ' ' + LastName as [Назва компанії],
        SalesOrderHeader.SalesOrderID as [Номер замовлення],
        Convert (money, sum(UnitPrice*OrderQty*(1-UnitPriceDiscount)),0) as [Результат]
```

```
FROM (Person.Contact INNER JOIN Sales.SalesOrderHeader
ON Contact.ContactID=SalesOrderHeader.ContactID)
INNER JOIN Sales.SalesOrderDetail
ON SalesOrderHeader.SalesOrderID=SalesOrderDetail.SalesOrderID
GROUP BY SalesOrderHeader.SalesOrderID, FirstName + ' ' + LastName
```

Слід пам'ятати, що використання уявлень не сприяє продуктивності. Звернення до подання викликає виконання його внутрішнього коду, таким чином, в кращому разі подання НЕ знизять продуктивність БД.

Управління поданнями

Створене подання може бути змінено виконанням команди ALTER VIEW, що має аналогічний синтаксис команді CREATE VIEW. Для видалення подання необхідно виконати команду:

```
DROP VIEW Ім'яПодання
```

Для отримання інформації про подання використовується збережена процедура: **sp_help** *Ім'яПодання*, яка відображає список полів подання з описом їх властивостей. Для відображення коду, за допомогою якого створено подання, можна скористатися збереженою процедурою: **sp_helptext** *Ім'яПодання.*

Список об'єктів, від яких залежить подання, може бути отриманий виконанням збереженої процедури: **sp_depends** *Ім'яПодання.*

Короткі висновки. Розглянуто збережені процедури і подання, а також основні команди SQL для створення, використання і керування цими об'єктами.

ЛЕКЦІЯ 6. СИСТЕМА БЕЗПЕКИ У БАЗАХ ДАНИХ

У лекції розглядаються основні механізми безпеки в MS SQL Server 2008: облікові записи для входу і користувачі бази даних, серверні ролі і ролі бази даних. Демонструється призначення прав користувачу як за допомогою графічного інтерфейсу, так і за допомогою SQL команд.

Ціль: показати способи захисту інформації в MS SQL Server 2008.

Будь-яка система зберігання інформації повинна бути максимально захищена як від випадкового, так і від навмисного пошкодження або спотворення інформації, тобто при плануванні бази даних необхідно чітко визначати повноваження кожного користувача системи.

У MS SQL Server передбачено три рівні безпеки:

1. автентифікація при реєстрації;
2. дозвіл на доступ до бази даних, підтримуваний сервером;
3. повноваження (дозволи) користувача.

Автентифікація користувача

Безпека в усіх додатках насамперед заснована на використанні *імені користувача* (login) і *паролю*. Під час встановлення MS SQL Server створюється обліковий запис **sa** (system administrator – системний адміністратор), до MS SQL Server 2000 вона створювалася з порожнім паролем, у MS SQL Server 2008 використовувати порожній пароль для **sa** за замовчуванням заборонено і при встановленні сервера потрібно задати складний пароль відповідно до поточних політик безпеки Windows.

Розгляньмо елементарні правила безпеки, які на практиці часто ігноруються.

- Одне ім'я (login) на одного користувача. Незважаючи на всю простоту вимоги, нерідко кілька користувачів використовують одне і те ж ім'я для входу. Особливо це погано в ситуації, коли цьому обліковому запису надані привілеї адміністратора. Рекомендується кожному користувачеві створити окремий обліковий запис з мінімально необхідним набором прав. Такий підхід не тільки сприяє підвищенню безпеки, але і дозволяє здійснювати аудит дій користувачів.
- Установка терміну дії паролю (Password expiration). Обмеження за часом дії пароля може убезпечити в разі, якщо ви комусь дали свій пароль для виконання якоїсь тимчасової роботи. Після закінчення терміну дії пароля, доступ іншого користувача буде закритий автоматично, якщо ви не повідомите йому новий пароль. Однак дана політика може мати більше негативних наслідків, ніж позитивних. Уявіть ситуацію, коли користувачі змушені міняти паролі кожні 30 днів: дуже швидко користувачам набридне придумувати нові паролі і запам'ятовувати їх, внаслідок цього паролі будуть складатися максимально простими.
- Обмеження на мінімальну довжину пароля і його складність (Password Length). Крім довжини паролів, рекомендується ставити вимогу на використання в паролі і літер, і цифр, що значно ускладнює можливість підбору паролів. Також

рекомендується не використовувати прості слова (імена, назви, торговельні марки) та набори чисел такі, як телефонні номери, номер паспорта, день народження тощо.

- Обмеження на кількість спроб входу (Number of Tries to Log In). Головне завдання цього правила – запобігти підбору паролів хакерами. Тому будь-яке невелике значення, наприклад, 3-5 спроб входу, цілком допустиме.

Для реалізації цих правил на практиці необхідно розглянути, що являє собою процес автентифікації у MS SQL Server 2008 і які типи автентифікації він підтримує.

Автентифікація користувача – це процес, при якому користувач у залежності від зазначеного імені користувача та пароля допускається чи ні до встановлення з'єднання з MS SQL Server.

MS SQL Server може працювати в двох режимах автентифікації користувачів, використовуючи або режим *автентифікації Windows* (Windows Authentication), або *режим автентифікації засобами MS SQL Server* (SQL Server Authentication).

Ці режими автентифікації використовують різні облікові записи користувачів. При автентифікації засобами MS SQL Server обліковий запис і пароль створює адміністратор баз даних MS SQL Server, при автентифікації засобами Windows – системний адміністратор мережі (в цьому випадку для підключення до MS SQL Server користувачеві не потрібний обліковий запис MS SQL Server).

Автентифікація MS SQL Server дозволяє визначити ім'я користувача для входу (login) і пароль для встановлення з'єднання з сервером. При встановленні MS SQL Server створюється лише один обліковий запис користувача для входу – **sa**.

Зауваження. Обліковий запис **sa** призначений для виконання всіх функцій адміністрування сервера і має всі права на доступ до всіх об'єктів всіх баз даних серверу.

За допомогою збережених процедур існує можливість додавання інших облікових записів для організації роботи з MS SQL Server, а також можливість керувати списком прав користувача, тобто набором дозволених йому дій.

Ролі серверу

Іншим фундаментом системи безпеки сервера є ролі. Під *роллю* розуміються певні права на виконання операторів і роботу з об'єктами бази даних. Ролі об'єднують декількох користувачів в групу, наділену певними правами, причому одному користувачеві може бути призначено кілька ролей.

Існує 8 *постійних ролей серверу*, які надають адміністративні привілеї на рівні сервера, незалежно від бази даних:

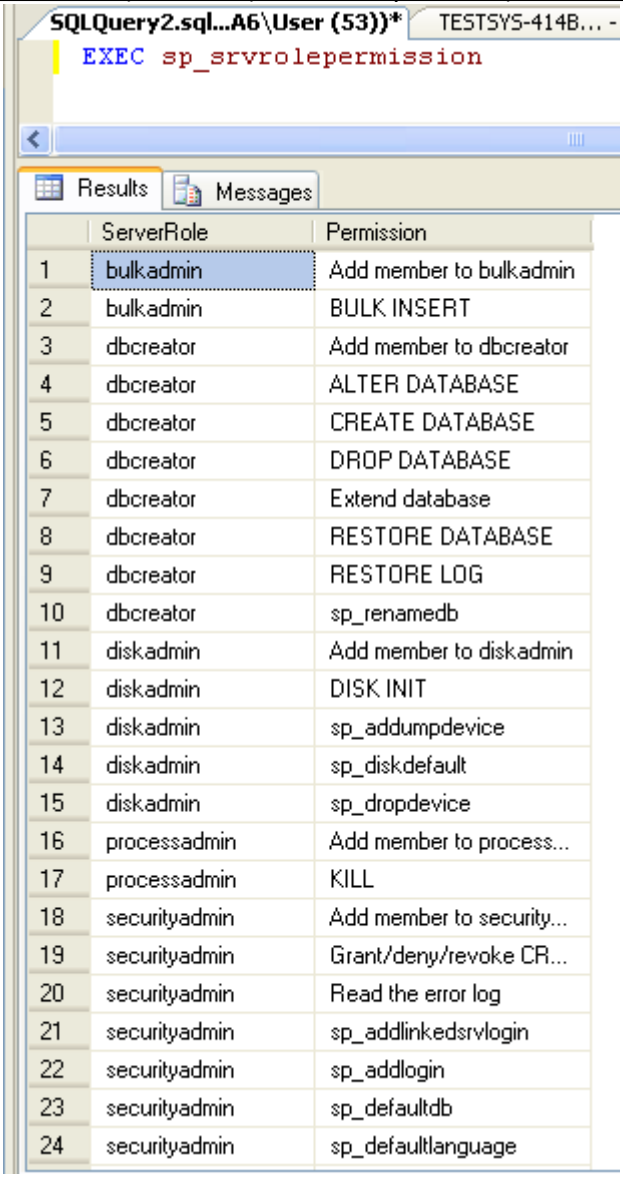
- *sysadmin* – може виконувати будь-які дії на MS SQL Server. За замовчуванням сюди входить обліковий запис **sa** і всі члени групи адміністраторів Windows;
- *setupadmin* – управляє пов'язаними серверами (linked servers) і процедурами, які виконуються разом із запуском серверу;
- *securityadmin* – може створювати і управляти логінами, читати журнал помилок і створювати БД;
- *processadmin* – володіє правами управління процесами всередині MS SQL Server, Наприклад, член цієї ролі може завершувати завдання, які виконуються дуже довго;
- *dbcreator* – дозволено створення і зміна баз даних;
- *diskadmin* – керує файлами баз даних: призначає файли в групи, приєднує/від'єднує бази даних тощо;
- *bulkadmin* – дозволяє виконувати команду BULK INSERT для вставки відразу великої кількості записів в таблицю;

Для перегляду інформації про вбудовані ролі використовуються збережені процедури:

- **sp_helpsrvrole** – повертає список ролей серверу і опис кожної ролі;
- **sp_helpsrvrolemember** *['ім'я ролі']* – повертає список ролей і облікових записів, яким надано ці ролі;
- **sp_srvrolepermission** *['ім'я ролі']* – повертає список дозволів, наданим цим ролям.

Зауваження. Якщо зазначений необов'язковий параметр *['ім'я ролі']*, то виводиться інформація, яка стосується тільки зазначеної ролі.

На рис. 6.1. показаний приклад виклику збереженої процедури.



	ServerRole	Permission
1	bulkadmin	Add member to bulkadmin
2	bulkadmin	BULK INSERT
3	dbcreator	Add member to dbcreator
4	dbcreator	ALTER DATABASE
5	dbcreator	CREATE DATABASE
6	dbcreator	DROP DATABASE
7	dbcreator	Extend database
8	dbcreator	RESTORE DATABASE
9	dbcreator	RESTORE LOG
10	dbcreator	sp_renamedb
11	diskadmin	Add member to diskadmin
12	diskadmin	DISK INIT
13	diskadmin	sp_addumpdevice
14	diskadmin	sp_diskdefault
15	diskadmin	sp_dropdevice
16	processadmin	Add member to process...
17	processadmin	KILL
18	securityadmin	Add member to security...
19	securityadmin	Grant/deny/revoke CR...
20	securityadmin	Read the error log
21	securityadmin	sp_addlinkedsrvlogin
22	securityadmin	sp_addlogin
23	securityadmin	sp_defaultdb
24	securityadmin	sp_defaultlanguage

Рис. 6.1. Приклад виклику збереженої процедури

Керування обліковими записами для входу

1. Для додавання облікових записів користувачів для входу в MS SQL Server використовується збережена процедура **sp_addlogin:**

sp_addlogin [@loginame=] ‘обліковий запис користувача для входу’
[, [@passwd=] ‘пароль користувача’]
[, [@defdb=] ‘база даних’]
sp_addlogin [@loginame =] 'login'
[, [@passwd =] 'password']
[, [@defdb =] 'database']
[, [@deflanguage =] 'language']
[, [@sid =] sid]
[, [@encryptopt=] 'encryption_option']

@defdb – база даних, до якої буде підключати MS SQL Server цього користувача за замовчуванням. Якщо цей параметр не визначений, то буде використовуватися системна база даних ***master***.

@deflanguage – мова за замовчуванням; якщо параметр не вказано, то буде використовуватися мова за замовчуванням, задана для серверу.

@sid – ідентифікаційний номер (security identification number); якщо параметр не заданий, то sid буде згенеровано.

@encryptopt – вказує, чи потрібно проводити хешування пароля, або замість пароля вже використовується хеш.

Ця збережена процедура широко використовується на сьогоднішній день. Однак в наступних версіях MS SQL Server вона буде видалена. Замість неї рекомендується використовувати наступну конструкцію:

```
CREATE LOGIN loginName { WITH <option_list1> | FROM <sources> }

<option_list1> ::=
PASSWORD = { 'password' | hashed_password HASHED } [ MUST_CHANGE ]
[ , <option_list2> [ ,... ] ]

<option_list2> ::=
SID = sid
| DEFAULT_DATABASE = database
| DEFAULT_LANGUAGE = language
| CHECK_EXPIRATION = { ON | OFF}
| CHECK_POLICY = { ON | OFF}
| CREDENTIAL = credential_name

<sources> ::=
WINDOWS [ WITH <windows_options> [ ,... ] ]
| CERTIFICATE certname
| ASYMMETRIC KEY asym_key_name

<windows_options> ::=
DEFAULT_DATABASE = database
| DEFAULT_LANGUAGE = language
```

Параметри WITH дозволяє задати пароль і налаштування безпеки, а розділ FROM вказує, що логін не просто належить MS SQL Server, а асоціюється або з обліковим записом Windows, або з сертифікатом, або з ключем.

MS SQL Server накладає ряд обмежень на імена користувачів, ролі і паролі:

- імена користувачів, ролі і паролі повинні мати розмір від 1 до 128 символів і не містити зворотних слешів (\);
- імена користувачів не повинні збігатися з ключовими словами і вбудованими іменами користувачів;

- пароль може не містити ніяких символів, але в разі якщо активовані політики паролів, то він повинен задовольняти їх вимогам.

2. Для перегляду інформації за іменами користувачів, що використовуються для входу, використовується збережена процедура sp_helplogins. Приклад виконання цієї процедури показаний на рис. 6.2.

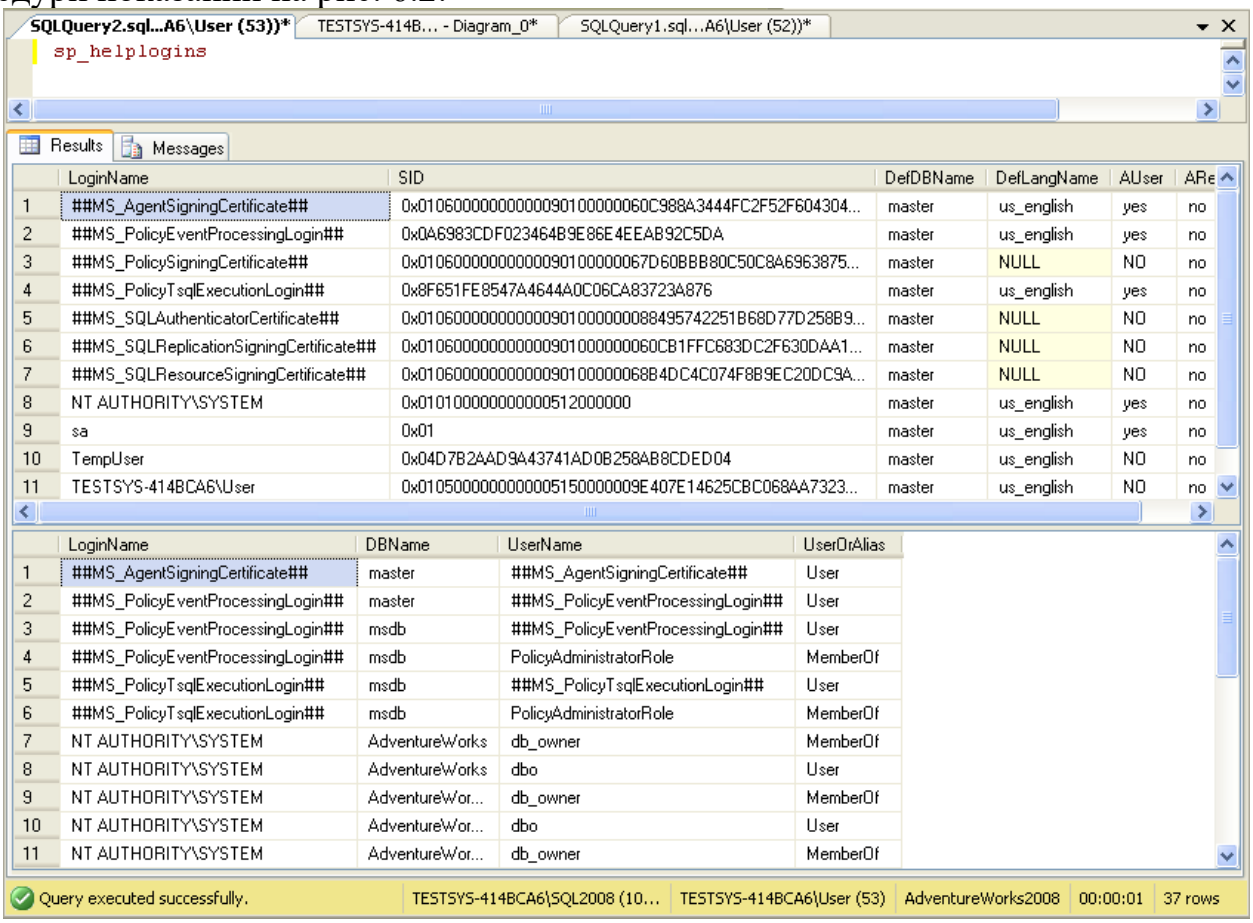


Рис. 6.2. Отримання списку користувачів MS SQL Server

3. Зміна пароля облікового запису користувача для входу виконується за допомогою процедури **sp_password**.

Зауваження. Для отримання довідки за командами Transact-SQL і збереженим процедурам можна скористатися утилітою *Management Studio*. Для цього необхідно виділити ім'я оператора і натиснути клавішу **F1**.

Ця збережена процедура широко використовується на сьогоднішній день. Однак в наступних версіях MS SQL Server вона буде видалена. Замість неї, рекомендується використовувати конструкцію ALTER LOGIN.

4. Видалення облікового запису здійснює збережена процедура:

sp_droplogin 'обліковий запис користувача для входу'.

Ця збережена процедура широко використовується на сьогоднішній день. Однак в наступних версіях MS SQL Server вона буде видалена. Замість неї, рекомендується використовувати настпуну конструкцію:

DROP LOGIN 'обліковий запис користувача для входу'

5. Для присвоєння облікового запису для входу вбудованій серверній ролі використовується процедура:

sp_addsrvrolemember [@loginame=] 'обліковий запис користувача для входу' , [@rolename=] 'роль'

Наприклад:

EXEC sp_addsrvrolemember 'Corporate\HelenS', 'sysadmin'

6. Скасування присвоєння облікового запису певної ролі виконується за допомогою збереженої процедури:

sp_dropsrvrolemember [@loginame=] 'обліковий запис користувача', [@rolename=] 'роль'

Наприклад:

EXEC sp_dropsrvrolemember 'JackO', 'sysadmin'

Доступ до бази даних

База даних – це найважливіша одиниця в системі безпеки MS SQL Server. Доступ до бази даних надається *користувачеві бази даних* (не плутати з ім'ям входу в MS SQL Server*),* наділеному певними правами, які визначаються приналежністю користувача до ролі бази даних.

Ролі баз даних

Ролі баз даних надають набори адміністративних привілеїв на рівні бази даних. При використанні ролей бази даних кожен обліковий запис серверу матиме різні повноваження залежно від того, з якою базою даних здійснюється робота. Існує 10 вбудованих ролей бази даних:

- *db_owner* – включає в себе права всіх інших ролей бази даних. Користувач отримує права власника бази.
- *db_accessadmin* – схожа на серверну роль securityadmin, за винятком того, що обмежена однією базою даних. Вона не дозволяє створювати нові логіни MS SQL Server, але дозволяє додавати нових користувачів в базу даних.
- *db_datareader* – дозволяє виконання оператора SELECT для всіх таблиць бази даних.

- *db_datawriter* – дозволяє виконувати INSERT, UPDATE і DELETE для всіх таблиць бази даних.
- *db_ddladmin* – дозволяє додавати, видаляти і змінювати об'єкти в базі даних.
- *db_securityadmin* – ще одна роль схожа на серверну роль securityadmin. На відміну від db_accessadmin, вона не дозволяє створювати нових користувачів в базі, але дозволяє керувати ролями і членством в ролях, а також правами на доступ до об'єктів бази даних.
- *db_backupoperator* – дозволяє створювати резервні копії бази даних.
- *db_denydatareader* – забороняє виконання SELECT для всіх таблиць бази даних.
- *db_denydatawriter* – забороняє виконання INSERT, UPDATE і DELETE для всіх таблиць бази даних.

Перегляд інформації про ролі баз даних (як вбудованих, так і певних користувачів) здійснюється за допомогою процедури **sp_helprole,** перегляд членів ролей баз даних – **sp_helprolemember.**

Зауваження. У кожній базі даних є спеціальна роль *Public*. Вона не може бути видалена. Кожен користувач бази даних обов'язково член цієї ролі. Зазвичай цю роль використовують для надання деяких дозволів всім користувачам даного серверу.

Керування користувачами баз даних

За замовчуванням MS SQL Server має спеціальну обліковий запис користувача в базі даних: власник БД (*dbo* – data base owner). Власник бази даних має повну владу над БД, ним автоматично стає той, хто створив цю базу даних.

1. Для створення користувача БД використовують процедури:

```
sp_adduser [@loginame=] 'обліковий запис для входу'
                [, [@name_in_db=] 'ім'я користувача']
                [, [@grpname=] 'роль']
sp_grantdbaccess [@loginame=] 'обліковий запис для входу'
                    [, [@name_in_db=] 'ім'я користувача']
```

Їх відмінність полягає в тому, що sp_adduser дозволяє відразу присвоїти користувачу певну роль бази даних. Ця процедура залишена для забезпечення зворотної сумісності і при роботі викликає процедуру sp_grantdbaccess.
Обидві ці збережені процедури будуть видалені в наступних версіях MS SQL Server. Замість них, рекомендується використовувати наступну конструкцію:

```
CREATE USER user_name
 [ { { FOR | FROM }
 {
 LOGIN login_name
```

```
| CERTIFICATE cert_name
| ASYMMETRIC KEY asym_key_name
}
| WITHOUT LOGIN
]
[ WITH DEFAULT_SCHEMA = schema_name ]
```

Опис параметрів:

- *user_name* – ім'я користувача в БД, має бути не довше 128 символів.
- *LOGIN login_name* – вказує з яким логіном необхідно асоціювати даного користувача.
- *CERTIFICATE cert_name* – вказує сертифікат, з яким необхідно асоціювати даного користувача.
- *ASYMMETRIC KEY asym_key_name* – вказує ключ, з яким необхідно асоціювати даного користувача.
- *WITH DEFAULT_SCHEMA = schema_name* – задає схему за замовчуванням для користувача. При пошуку об'єктів вони спочатку будуть шукатися в цій схемі.
- *WITHOUT LOGIN* – вказує, що користувач не повинен бути асоційований з існуючим логіном.

2. Відображення списку користувачів здійснюється в результаті виконання процедури **sp_helpuser**.

3. Для видалення користувача БД використовуються процедури:

```
sp_dropuser [@name_in_db=] 'ім'я користувача'
```

Ця збережена процедура буде видалена в наступних версіях MS SQL Server. Замість неї, рекомендується використовувати наступну конструкцію:

```
DROP USER 'ім'я користувача'
```

Користувач не може бути видалений, якщо він є власником об'єктів у базі даних.

4. Надання користувачеві певної ролі здійснюється процедурою:

```
sp_addrolemember [@rolename=] 'роль',
                 [@membername=] 'користувач'
```

5. Скасування присвоєної користувачеві ролі може бути виконана за допомогою процедури:

```
sp_droprolemember [@rolename=] 'роль',
                  [@membername=] 'користувач'
```

6. З метою спрощення адміністрування системи безпеки серверу адміністратор має можливість створювати призначені для користувача ролі баз даних і наділяти їх певним набором повноважень.

Для цього використовується збережена процедура:

sp_addrole [@rolename=] 'роль'

Для видалення створеної раніше ролі:

sp_droprole [@rolename=] 'роль'

Таким чином доступ до MS SQL Server здійснюється за допомогою *системи імен користувачів для входу*, в системі же безпеки MS SQL Server *користувачу* надаються певні права доступу до об'єктів даних. Не слід плутати **ім'я користувача для входу** і **ім'я користувача бази даних** для доступу до об'єктів.

Все описанные выше действия по организации системы безопасности можно выполнить также и с помощью графического интерфейса утилиты администрирования *Management Studio***.**

Керування обліковими запитами для входу

Перегляд списку наявних облікових записів і їх параметрів здійснюється вибором групи *Logins* у папці **Security** серверу. Облікові записи користувачів відображаються в полі *Name*, у полі *Default Database* – ім'я бази даних, до якої користувач підключається за замовчуванням.

Для створення нового облікового запису для входу необхідно виконати команду **New Login…** контекстного меню вузла **Logins,** в діалоговому вікні вказати (див. рис. 6.3):

вкладка *General*: ім'я користувача, тип автентифікації (при автентифікації засобами MS SQL Server – задати пароль), базу даних, до якої підключається автоматично, і мову за замовчуванням;

вкладка *Server Roles*: ролі сервера, в які буде входити створюваний обліковий запис;

вкладка *User Mapping*: доступ до однієї із створених на сервері бази даних, в поле *User* ввести ім'я користувача бази даних і включити створюваного користувача в одну існуючих ролей.

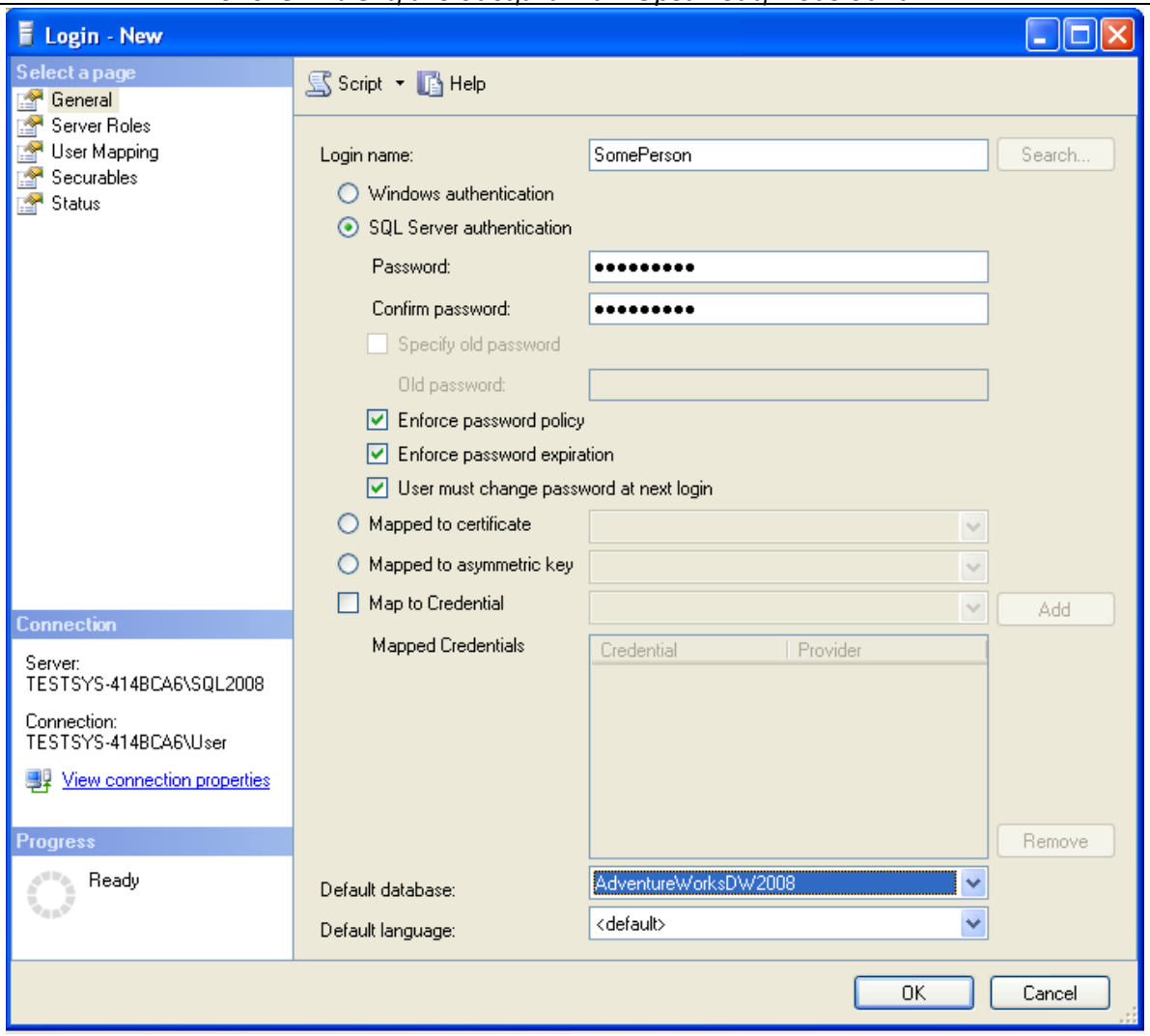


Рис. 6.3. Вікно створення нового облікового запису

Для зміни параметрів наявного облікового запису користувача для входу необхідно вибрати її зі списку і виконати команду контекстного меню **Properties**, для видалення – **Delete.**

Керування ролями сервера

Для відображення списку ролей серверу необхідно вибрати групу *Server Roles* у папці **Security** серверу. Перегляд користувачів, які входять в цю роль і дозволів, наданих їй, здійснюється виконанням команди **Дія – Властивості**.

Вбудовані ролі серверу не можуть бути вилучені із системи, і не можна змінити визначені для них дозволи. Також заборонено створювати і власні серверні ролі.

Керування користувачами баз даних

Для перегляду і керування параметрами користувачів певної бази даних призначена група *Security/Users* цієї бази даних. Облікові записи відображаються в полі *User Name*, а у полі *Login Name* – відповідні їм облікові записи для входу.

Для створення нового користувача бази даних необхідно виконати команду **New User…,** потім у поле *User name* ввести ім'я користувача, а в списку *Login Name* вибрати відповідний обліковий запис для входу (див. рис. 6.4). Можна також включити користувача в ролі бази даних.

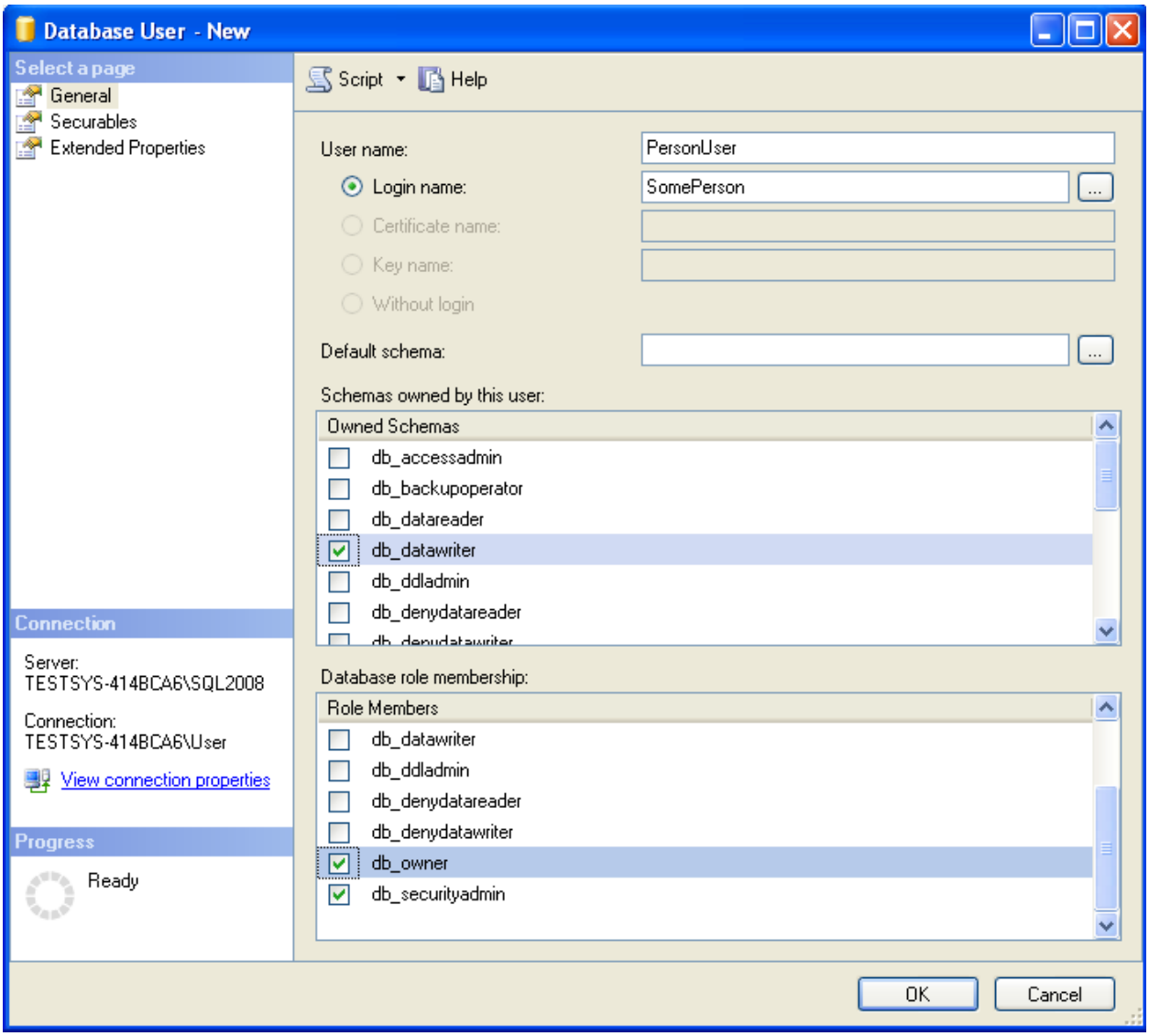


Рис. 6.4. Вікно створення нового користувача БД

Для зміни параметрів облікового запису служить команда **Properties,** а для видалення – **Delete.**

Керування ролями баз даних

Для відображення списку ролей бази даних використовується група *Roles*. Для перегляду користувачів, що входять в цю групу необхідно виконати команду **Properties.**

Дозволи користувача

Виконання операторів мови Transact-SQL і дій в системі може бути виконано тільки після успішної автентифікації користувача. При отриманні оператора Transact-SQL від користувача сервер перевіряє, чи має право користувач на виконання відповідних дій. Якщо права є, то дія виконується, в іншому випадку повертається повідомлення про помилку.

Дії, які можуть бути виконані користувачем, визначаються правами, виданими йому безпосередньо, або роллю, в якій він перебуває. Таким чином, користувач повинен мати відповідні дозволи на виконання тих чи інших дій.

Права у MS SQL Server можно разделить на три категории: разрешения для объектов, разрешения для команд Transact-SQL і неявні дозволи.

Для різних об'єктів застосовуються такі набори дозволів як:

- SELECT. Користувач з цим привілеєм може виконувати запити в таблиці.
- INSERT. Користувач з цим привілеєм може додавати дані в таблицю.
- UPDATE. Користувач з цим привілеєм може змінювати дані в таблиці. Не можна призначити цей привілей для певних полів таблиці.
- DELETE. Користувачеві з цієї дозволено видаляти дані з таблиці.
- ALL. Користувачеві надані будь-які дозволи.

Дозволи для команд Transact-SQL контролюють можливість створення об'єктів в базі даних, створення самої бази даних і виконання процедури резервного копіювання.

Надання доступу

Управління дозволами користувача на доступ до об'єктів бази даних здійснюється за допомогою команди GRANT:

```
GRANT
{дозвіл [,...n] }{ ON таблиця | подання [(поле[,...n])]
| ON [збережена процедура[,...n] }
ТО обліковий запис [,...n][WITH GRANT OPTION]
```

Призначення аргументів цієї команди:

- *дозвіл* – список дозволів, що надаються користувачеві. Якщо надається декілька дозволів одночасно, то вони розділяються комами;
- *таблиця, подання, збережена процедура* – вказуються імена конкретних об'єктів поточної бази даних, для яких необхідно надати доступ. MS SQL Server має можливість визначати права доступу не тільки на рівні таблиці, але і на рівні поля;
- *обліковий запис* – вказується ім'я об'єкта системи безпеки, якому надаються права. В якості об'єкта може виступати як облікові записи користувачів, так і їх групи;
- *WITH GRANT OPTION* – дозволяє користувачеві, якому надаються права, призначати їх і для інших користувачів.

Наприклад, за допомогою такої команди користувачеві *TestUser* бази даних ***AdventureWorks*** надаються права вибірки і зміни даних таблиці *Orders* цієї бази даних:

GRANT SELECT, UPDATE ON AdventureWorks2008.Production.WorkOrder TO TestUser

Наступна команда надає користувачеві *Andy* права тільки вибірки даних полів *Name* і *ListPrice* таблиці *Product* бази даних ***AdventureWorks****:*

GRANT SELECT ON AdventureWorks2008.Production.Product (Name, ListPrice) TO Andy

Заборона доступу

Для заборони користувачеві доступу до об'єктів бази даних використовується команда DENY:

DENY
{дозвіл [,...n] }{ ON таблиця | подання [(поле[,...n])]
| ON [збережена процедура[,...n] }
TO обліковий запис [,...n][CASCADE]

Параметри цієї команди аналогічні параметрам командам GRANT. Параметр *CASCADE* дозволяє відкликати права не тільки у конкретного користувача, але також і у всіх користувачів, кому він надав дані права.

Неявне відхилення доступу

Неявне відхилення подібно забороні доступу з тією відмінністю, що воно діє тільки на тому рівні, на якому визначено. Тобто якщо користувачеві на певному рівні доступ відхилений, то він може його отримати на іншому рівні, Наприклад, через членство в ролі, що має на це право.

Зауваження. **За замовчуванням доступ користувача до даних неявно відхилений.**

Для неявного відхилення доступу до даних використовується наступна команда:

REVOKE [GRANT OPTION FOR]
{дозвіл [,...n] }{ ON таблиця | подання [(поле[,...n])]
| ON [збережена процедура[,...n] }
TO обліковий запис [,...n][CASCADE]

Сенс параметрів аналогічний параметрам команд GRANT і DENY. Параметр *GRANT OPTION FOR* використовується, коли необхідно відкликати право, надане параметром *WITH GRANT OPTION* команди GRANT. При цьому користувач зберігає дозвіл на доступ до об'єкта, але втрачає можливість надавати дозвіл іншим користувачам.

Зауваження. Необхідно пам'ятати наступний принцип: дозвіл має **найнижчий** пріоритет, а заборона - **найвищий**. Тобто доступ до даних може бути отриманий тільки явним його наданням при відсутності заборони доступу на будь-якому іншому рівні. Якщо доступ явно не надано, користувач не може працювати з даними.

Робота з дозволами користувача може виконуватися і за допомогою графічного інтерфейсу утиліти адміністрування *Management Studio*. Щоб призначити повноваження об'єкту безпеки необхідно вибрати його в групі *Users* (для зміни дозволу конкретного користувача бази даних) або в групі *Roles* (для дозволів певної ролі). Для цих цілей використовується вкладка *Securables* (див. рис. 6.5).

У вкладці, що з'явилася, перераховані всі об'єкти бази даних, з можливими правами доступу. Можна встановити одне з трьох станів доступу: надання (галочка), заборона (хрестик) і неявне відхилення (пусте поле) – у відповідному полі.

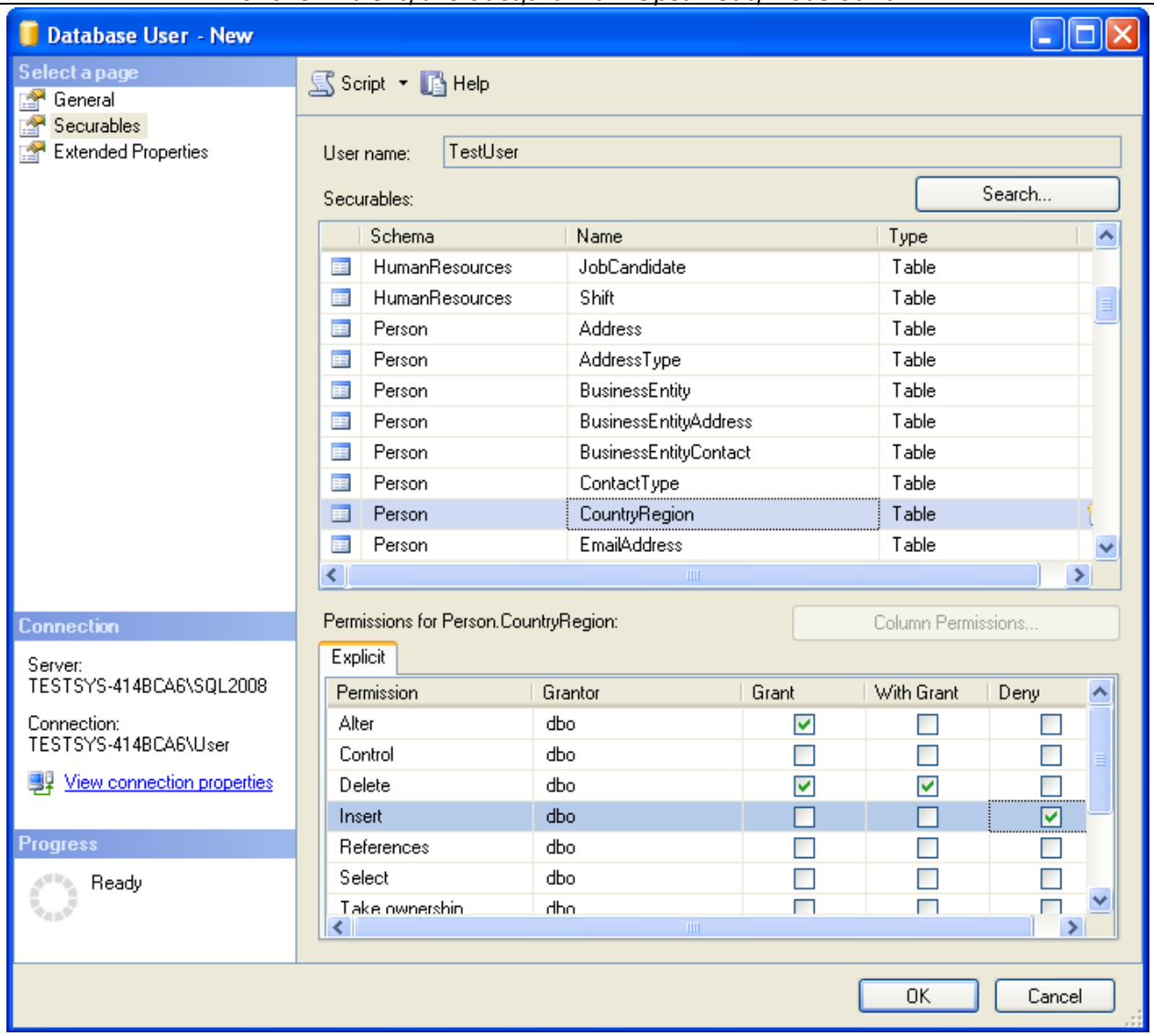


Рис. 6.5. Призначення прав користувачу БД

Короткі висновки. Були розглянуті ключові компоненти системи безпеки MS SQL Server 2008, визначено призначення серверних ролей і ролей БД. На прикладах показані способи вирішення або заборони певних дій користувача на сервері.

ЛЕКЦІЯ 7. СТРУКТУРА БАЗ ДАНИХ У MS SQL SERVER

У лекції розглядаються способи створення і налаштування БД, а також її структура: первинні файли, файлові групи, журнал. Порушуються питання стиснення БД і резервного копіювання та відновлення.

Ціль: познайомити зі структурою бази даних у MS SQL Server 2008 і навчити створювати бази даних і керувати ними.

Планування фізичної організації бази даних – найважливіша частина адміністративної роботи з базами даних, файлами і файловими групами. Погано фізично організована працюватиме з меншою продуктивністю.

Створення і налаштування баз даних

СКБД MS SQL Server пропонує два можливі варіанти створення бази даних:
1. Використання графічного інтерфейсу *Management Studio.*
2. Використання команд SQL.

Створення бази даних – це процес зазначення імені файлу, визначення розмірів і розміщення файлів бази даних, а також визначення параметрів файлу журналу транзакцій.

Можна виділити три типи файлів в базах даних MS SQL Server:

1. *Первинні файли даних*. Як правило, використовується розширення MDF. У будь-якій базі даних є один первинний файл, який містить дані і розташування всіх інших файлів БД.

2. *Вторинні файли даних*. Як правило, використовується розширення NDF. Вторинним є будь-який файл, крім первинного та файлів журналів. БД можуть не містити жодного втраченого файлу.

3. *Файл журналів*. Як правило, використовується розширення LDF. У кожній БД існує щонайменше один файл журналу. Журнал транзакцій містить відомості про зміни, що відбуваються в БД, тобто при здійсненні деякої транзакції (операції) в цей журнал заносяться відомості. Згодом цей журнал стає все більше, тому потрібно стежити за його розміром. Основне призначення журналу транзакцій - це забезпечення цілісності даних. Він дозволяє скасовувати зроблені зміни в БД.

Для зручності адміністрування і розподілу навантаження файли можна об'єднувати в файлові групи, які діляться на два види.

1. Первинні файлові групи. Сюди входять первинний файл і всі файли, які явно не були поміщені в іншу групу.

2. *Користувацькі файлові групи* – це будь-яка група створювана користувачем в БД.

Файли журналів не входять в одну файлову групу, вони обробляються окремо від звичайних файлів.

Нова база даних являє собою копію бази даних ***model***, всі параметри якої копіюються в нову базу даних. За замовчуванням бази даних мають створювати тільки ті користувачі, яким призначено ролі *sysadmin* і *dbcreator*.

Створення бази даних здійснюється за допомогою команди:

```
CREATE DATABASE ім'я_бази_даних
[ON [PRIMARY] (NAME = 'логічне_ ім'я _файлу',
FILENAME = 'фізичне_ ім'я_файлу'
[, SIZE = размер]
[, MAXSIXE = {максимальний_розмір | UNLIMITED} ]
[, FILEGROWTH = крок_збільшення_розміру [Mb | Kb | %] )
[ {FILEGROUP ім'я _файлової_групи} ]
[, ...n ]
[LOG ON (NAME = 'логічне_ ім'я _файлу',
FILENAME = 'фізичне_ ім'я_файлу'
[, SIZE = размер]
[, MAXSIXE = { максимальний_розмір | UNLIMITED} ]
[, FILEGROWTH = крок_збільшення_розміру [Mb | Kb | %] )
[, ...n ]
```

Опис параметрів:

- *PRIMARY* – визначає файл як первинний або як член первинної файлової групи, якщо опущено, то основним файлом стає перший файл в операторі і для зберігання використовується первинна файлова група;
- *NAME* – визначає логічне ім'я файлу. За замовчуванням збігається з фізичним ім'ям файлу, визначеному в параметрі *FILENAME;*
- *FILENAME* - вказує повний шлях і ім'я фізичної файлу*;*
- *SIZE* – вказує розмір файлу: в мегабайтах, кілобайтах. Мінімально можливе значення 512 Кб. Розмір основного файлу за замовчуванням дорівнює розміру БД ***model***. За замовчуванням розмір додаткових файлів даних і журналу дорівнює 1 Мб;
- *MAXSIZE* – вказує максимальний розмір, до якого може збільшуватися файл. Якщо цей параметр не вказано, то встановлюється значення *UNLIMITED*, що дозволяє збільшувати файлів розмір без обмежень;
- *FILEGROWTH* – задає крок збільшення файлу, причому нуль означає заборону збільшення розміру. Значення вказується в мегабайтах, кілобайтах або відсотках. За замовчуванням приріст – 10%, якщо не вказані одиниці, то цифра сприймається в мегабайтах;
- *FILEGROUP* – визначає ім'я групи файлів, в яку поміщається файл.

Для перегляду інформації про базу даних, файли і групи файлів використовуються наступні збережені процедури:

- **sp_helpdb** *[база_даних]* – інформація про базу даних та її налаштування. Якщо база даних не вказана, то відображається звіт по всіх базах даних, які підтримуються в MS SQL Server.

- **sp_helpfile** *['ім'я']* – інформація про файли, що відносяться до поточної бази даних. Якщо ім'я файлу не вказано, то відображається інформація про всі файли цієї бази даних.
- **sp_helpfilegroup** *['ім'я']* – інформація про всі файлові групи в поточній базі даних. Якщо вказано ім'я файлової групи, то виводиться інформація по кожному файлу зазначеної групи.
- **sp_spaceused** *['об'єкт']* – відомості про дисковий простір, що використовується зазначеним об'єктом.

Крім перерахованих вище фізичних параметрів база даних має ще й логічні параметри. Тільки власник і системний адміністратор може змінити ці параметри. Управління параметрами здійснюється за допомогою системної збереженої процедури **sp_dboption**:

sp_dboption [[@dbname=] 'ім'я_бази'] [, [@option=] ''] [, [@value=] ON | OFF]

Зміна бази даних

Видалення бази даних здійснюється за допомогою оператора:

DROP DATABASE ім'я_бази_даних [, …n]

В результаті видаляються всі файли, які використовуються базою даних. Правом на видалення володіє власник бази і користувачі ролі *sysadmin*, це право не може бути передано іншим обліковим записам.

Зміна власника бази даних проводиться за допомогою спеціальної збереженої процедури. Власником можна зробити будь-який обліковий запис, який в даний момент не є користувачем бази, у такий спосіб:

sp_changedbowner [[@loginname=] 'ім'я_кристувача'

Перейменування бази даних:

sp_renamedb [@old_name=] 'старе_ ім'я', [@new_name=] 'нове_ ім'я'

Для перейменування бази даних її необхідно перевести в однокористувацький режим роботи.

Для управління вже наявними файлами журналу і файлами даних, додавання додаткових файлів даних або журналу, видалення файлів, а також для роботи з файловими групами використовується команда:

ALTER DATABASE база_даних

{ ADD FILE <посилання_на_файл> [TO FILEGROUP наименование]
| ADD LOG FILE <посилання_на_файл>
| REMOVE FILE логічне_ ім'я _файлу
| ADD FILEGROUP ім'я _групи
| REMOVE FILEGROUP ім'я _групи
| MODIFY FILE <посилання_на_файл>
| MODIFY FILEGROUP ім'я _групи свойство_группы }
где <посилання_на_файл> =
(NAME = 'логічне_ ім'я _файлу',
FILENAME = 'фізичне_ім'я _файлу'
[, SIZE = размер]
[, MAXSIXE = {максимальний_розмір | UNLIMITED}]
[, FILEGROWTH = крок_збільшення_розміру [Mb | Kb | %])

Дана команда дозволяє додавати файл в наявну файлову групу, видаляти файли (при цьому видаляється і фізичний файл), додавати і видаляти файлові групи, змінювати фізичні параметри вже наявних файлів, а також змінювати властивості файлових груп: READONLY, READWRITE, DEFAULT (при визначенні цієї властивості, в цю групу буде заноситься файли, у яких в параметрах не визначена належність до групи; встановленої за замовчуванням спочатку вважається первинна файлова група).

Стиснення бази даних

Стиснення бази даних – це процес зменшення розмірів файлів бази даних за рахунок видалення невживаних частин файлу. Існує три способи стиснення бази даних:

- автоматичне стиснення при встановленні відповідного параметра в налаштуваннях бази даних;
- видалення вільного простору з файлів бази даних за допомогою утиліт адміністрування MS SQL Server;
- зменшення розміру зазначених файлів (або файлових груп), а також очищення вмісту файлів для їх подальшого видалення.

Автоматичне стиснення даних виконується постійно з певними інтервалами, якщо встановлено параметр бази даних autoshrink. При операціях автоматичного стиснення можна визначити, яку частину бази даних необхідно стиснути. MS SQL Server намагається звільнити значну частину бази даних самостійно. Ці операції виконуються в період найменшої активності користувачів.

Стиснення всієї бази даних вручну здійснюється з використанням наступної команди:

DBCC SHRINKDATABASE ('ім'я_БД', ['процент'] [, NOTRUNCATE | TRUNCATEONLY])

Опис параметрів:

- *ім'я _БД* – ім'я бази даних, яку необхідно стиснути;
- *процент* – кількість відсотків вільного простору, яке бажано залишити після стиснення;
- *NOTRUNCATE* – зведений простір не повертається операційній системі, а резервується в файлах, тобто фізично зменшення розміру бази даних не відбувається;
- *TRUNCATEONLY* – вільний простір видаляється за останнім використовуваним в файлі екстентом[7], при цьому дані не переміщаються, а параметр відсоток ігнорується.

Права на стиск бази даних видані тільки членам ролі sysadmin і власникам бази даних. Після стиснення бази даних виводиться звіт, в якому вказується:

- кількість сторінок, до яких стискається файл;
- розрахункове число сторінок, в які можуть бути поміщені всі дані файлу;
- кількість сторінок, що містять дані;
- кількість сторінок, на які файл може бути ще стиснутий.

Не можна стиснути базу даних до розміру менше початкового.

Стиснення бази даних можна здійснити також і шляхом **стиснення кожного її файлу** за допомогою наступної команди:

DBCC SHRINKFILE ('ім'я _файлу', ['кінцевий_розмір']
[, EMPTYFILE | NOTRUNCATE | TRUNCATEONLY])

Опис параметрів:

- *ім'я _файлу* – логічне ім'я файлу, який необхідно стиснути;
- *кінцевий_розмір* – бажаний розмір (ціле число в мегабайтах), який повинен мати файл після виконання стиснення. Якщо цей параметр не вказаний або менше мінімально допустимого розміру, то файл стискається до мінімально можливого розміру;
- *EMPTYFILE* – виконується перенесення даних з файлу в інші файли файлової групи;
- *NOTRUNCATE* – звільнене місце не повертається операційній системі, тобто розмір файлу не зменшується насправді. При цьому дані розташовуються більш компактно і зміщуються до початку файлу;
- *TRUNCATEONLY* – відбувається обрізання файлу, починаючи з останньої використовуваної сторінки. Ніякого переміщення даних не відбувається.

Резервне копіювання даних

[7] Екстент – безперервна область пам'яті на накопичувачі.

Необхідно приділяти особливу увагу цілісності інформації, з якою працює користувач. MS SQL Server пропонує наступні типи резервного копіювання інформації:

- *повна копія* бази даних, яка є відправною точкою при відновленні бази даних після збою, проте в залежності від обсягу даних цей процес може займати багато часу, тому не рекомендується виконувати його занадто часто. Повна копія містить всі дані, що містяться в базі даних на момент закінчення резервування;
- *копія журналу транзакцій*, необхідна для фіксування всіх змін даних, що відбулися в системі з моменту останнього резервного копіювання. Сама копія журналу містить відомості про транзакції і лише тільки разом з копією бази даних дозволяє повернутися до стану, що передує збою;
- *диференціальна копія* даних містить зміни даних, що відбулися з моменту останнього створення повної копії бази даних. При цьому зберігаються тільки сторінки які зазнали змін. Таким чином, для відновлення бази даних досить самої останньої диференціальної копії.

Щоб створити резервну копію необхідно вибрати носій, тобто визначити пристрій, який буде використовуватися для створення копій. **Щоб додати пристрій** використовується збережена процедура:

sp_addumpdevice 'тип_пристрою', 'логічне_ім'я', 'фізичне_ім'я'

Опис параметрів:

- *тип_пристрою* – тип пристрою резервного копіювання. Можна вибрати зі значень TAPE (магнітна стрічка), DISK (магнітний диск);
- *логічне_ім'я, фізичне_ім'я* – логічне і фізичне ім'я пристрою резервного копіювання відповідно.

Для створення резервної копії бази даних, журналу транзакцій, файлів і файлових груп необхідно скористатися командою:

BACKUP {LOG | DATABASE } ім'я _БД
[FILE = 'логічне_ ім'я _файлу', ...]
[FILEGROUP = 'ім'я _групи']
TO логічне_ім'я_устройства
[WITH
[DESCRIPTION = 'коментар']
[DIFFERENTIAL]
[EXPIREDATE = 'дата']
[INIT | NOINIT] ...]

Опис параметрів:

- *DIFFERENTIAL* – створюється диференціальна копія бази даних;
- *EXPIREDATE* – визначається дата, після якої резервна копія вважається застарілою і може бути переписана*;*
- *INIT | NOINIT* – система здійснює чи ні ініціалізацію пристрою.

Відновлення бази даних

При відновленні бази даних з резервної копії наявна база даних буде переписана. Для відновлення бази даних використовується команда:

RESTORE {LOG | DATABASE } имя_БД
'файл_или_файловая_ группа'
[FROM логічне_ім'я_устройства]
[WITH
[DBO_ONLY]
[MOVE 'логічне_ ім'я _файлу' TO 'фізичне_ім'я'] ...]

Опис параметрів:

- *DBO_ONLY* – дозволяється доступ до відновленої бази тільки власникам;
- *MOVE* – вказує, яке фізичне ім'я буде відповідати відновлюваному файлу. За замовчуванням файл відновлюється з тим же фізичним ім'ям, яке було визначено при резервному копіюванні.

Короткі висновки. Вивчено основні фізичні елементи бази даних: первинні файли, файлові групи, журнал. Продемонстровано створення нової БД і керування нею. Показано можливості резервного копіювання та стиснення БД.

ЛЕКЦІЯ 8. РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ

У лекції розглядаються основні елементи реляційних баз даних, а також питання цілісності даних. Даються визначення первинних і зовнішніх ключів.

Ціль: розглянути основні елементи реляційних баз даних.

Реляційні об'єкти даних

Реляційна модель включає три складові частини:
1. Об'єкти.
2. Цілісність.
3. Оператори.

Розгляньмо об'єкт на рис. 8.1.

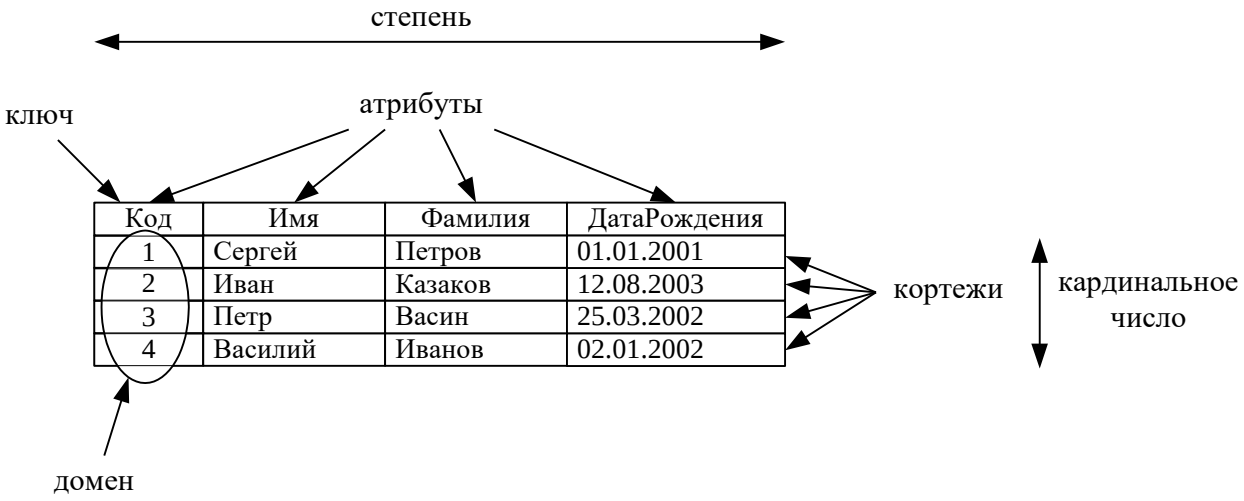


Рис. 8.1. Основні елементи реляційної БД

Відношення відповідає тому, що ми зазвичай називаємо таблицею. *Кортеж* відповідає примірнику запису, *атрибут* – полю таблиці, *степінь (степень)* – кількості атрибутів, *кардинальне число* – кількості кортежів.

Домени

Домен – це загальна множина значень, з яких беруться конкретні значення певного атрибута деякого відношення.

Домени також можна визначити як іменовану множину скалярних значень одного типу.

Скалярне значення (скаляр) – це найменша семантична одиниця даних, яка є окремим значенням даних. У скалярів немає внутрішньої структури, тобто вони не розкладені в даній реляційної моделі. Насправді, в інших контекстах скаляри можуть мати внутрішню структуру (наприклад, прізвище складається з літер), але для

конкретної таблиці це розкладання не має сенсу, тому що втрачається його значення. Кожен атрибут повинен бути визначений на єдиному домені. Наприклад, атрибут **StudentID** визначений на домені {1, 2, 3, ..., 10}, атрибут **GroupID** – на домені {1, 2, 3, ..., 10}, але це будуть різні домени, хоча і містять однакові елементи.

Не обов'язково всі елементи домену повинні використовуватися в конкретному відношенні.

Основне значення доменів полягає в тому, що вони обмежують операції порівняння.

Приклад: розгляньмо запит

```
SELECT * FROM Stduents, Groups
WHERE GroupsID.GroupID = Students.StudentID
```

Такий запит не має сенсу, оскільки ми порівнюємо числові значення з *різних* доменів. Правильно буде так:

```
WHERE GroupsID.GroupID = Students.GroupID
```

Відношення

Відношення можна розглядати з двох сторін:

1) *змінна відношення* – це звичайна змінна (як в будь-якій мові програмування), тобто іменований об'єкт, значення якого може змінюватися;

2) *значення відношення* – це значення цієї змінної в конкретний момент часу.

Уточнімо визначення відношення:

Відношення R, задане на множині доменів $D_1, D_2, \ldots, D_n$, які не обов'язково різні, містить дві частини: заголовок і тіло.

Заголовок містить фіксовану множину пар $A_i : D_i$, де A_i – ім'я атрибута.

Тіло містить множину кортежів, кожен з яких в свою чергу містить множину значень Z_{ji}, де i – номер атрибуту, j – номер кортежу.

Властивості відношень:

1) немає однакових кортежів, оскільки тіло відношення являє собою множину;

2) кортежі невпорядковані, тобто немає таких понять, як «перший» або «десятий» кортеж, немає понять «попередній» і «наступний»;

3) атрибути невпорядковані, тому що заголовок відношень теж визначений як множина;

4) всі значення атрибутів неподільні, тому що домен містить неподільні елементи.

Такі відношення, які не містять подільних атрибутів, називаються *нормалізованими*, або *представленими у першій нормальній формі*.

Цілісність реляційних даних

У кожен момент часу будь-яка БД містить конкретну конфігурацію значень, яка представляє певний стан об'єкта реального світу. Отже, БД потребує визначення правил цілісності, щоб інформувати СКБД про обмеження реального світу. Наприклад, для атрибутів «зростання», «вага» необхідно обмеження невід'ємності. Такого роду правила, характерні для конкретної БД, називаються *специфічними*. Крім специфічних правил існують загальні правила цілісності для всіх БД. Такі правила пов'язані з потенційними, первинними і зовнішніми ключами і будуть розглянуті далі в цій лекції.

Потенційні ключі

Нехай *R* – певне відношення. Тоді *потенційний ключ K* для *R* - це підмножина атрибутів R, що володіють такими властивостями:

1) унікальність, тобто немає двох різних кортежів в поточному значенні змінної відношення R з однаковим значенням *K*;

2) ненадмірність, тобто жодне з підмножин *K* не володіє властивістю унікальності.

Це властивість можна застосувати тільки для випадку, якщо потенційний ключ складається більш ніж з одного атрибуту. Наприклад, не можна призначити потенційним ключем сукупність полів **StudentID** і **Name**, інакше цей ключ буде надлишковим.

Можуть існувати відношення, в яких єдиним природним потенційним ключем буде комбінація всіх атрибутів, але це може бути незручно. Тоді вводять штучний потенційний ключ. Наприклад, в таблиці *Students* сукупність атрибутів **{Name, GroupID, Birthdate}** є ключ, але зручніше ввести штучний ключ - **StudentID**.

StudentID	Name	GroupID	BirthDate
1	Козаков Петро	2	23.04.1990
2	Васильєв Іван	1	11.05.1991
4	Шишкіна Дар'я	2	23.09.1991

Потенційні ключі призначені для забезпечення основного механізму адресації на рівні кортежів, тобто досить вказати значення потенційного ключа, за яким можна знайти будь-який кортеж.

Первинні й альтернативні ключі

Відношення може мати більше одного потенційного ключа, але один з них завжди вибирається в якості *первинного*, інші потенційні ключі будуть в цьому випадку називатися *альтернативними*.

Наприклад, у таблиці *Groups*:

GroupID	Name
1	ПМ-41

2	ПМ-51

GroupID – первинний ключ, а **Name** – альтернативний. В кожному відношенні завжди повинен бути один і тільки один первинний ключ.

Зовнішні ключі

Нехай *R2* – відношення. Тоді *зовнішній ключ FK* щодо *R2* - це підмножина множини атрибутів *R2* таке, що:

1) існує базове відношення *R1* з потенційним ключем *CK;*

2) кожне значення *FK* в поточному значенні *R2* завжди збігається зі значенням *CK* деякого кортежу в поточному значенні *R1*.

З даного визначення можна вивести такі наслідки:

1) за визначенням кожне значення зовнішнього ключа повинно бути значенням відповідного потенційного ключа, але зворотне не потрібно, тобто потенційний ключ може містити значення, які в даний момент не є значенням зовнішнього ключа;

2) даний зовнішній ключ буде складеним тоді і тільки тоді, коли відповідний потенційний ключ теж складений;

3) кожен атрибут, що входить в даний зовнішній ключ, повинен бути визначений на тому ж домені, що і відповідний атрибут потенційного ключа;

4) R1 і R2 не обов'язково різні.

Розгляньмо відношення *Students*:

StudentID	Name	GroupID	BirthDate
1	Козаков Петро	2	23.04.1990
2	Васильєв Іван	1	11.05.1991
4	Шишкіна Дар'я	2	23.09.1991

Атрибут **GroupID** буде зовнішнім ключем, тому що до нього проведена зв'язок від таблиці *Groups.*

У зв'язку з зовнішніми ключами вводиться ще ряд термінів. Кажуть, що значення зовнішнього ключа представлено посиланням до кортежу, який містить відповідне значення потенційного ключа. Цей кортеж називається *посилальним, або цільовим.*

Відношення, яке містить контрольний ключ, називається *відношенням, що посилається,* а відношення, яке містить відповідний ключ, називається *посилальним,* або *цільовим* (target relation).

Існує *правило посилальної цілісності*: БД не повинна містити неузгоджених значень зовнішніх ключів. *Неузгоджене значення* - це таке значення, для якого немає потенційного ключа в посилальному відношенні. Це правило еквівалентно визначенню зовнішнього ключа.

Правила зовнішніх ключів

Правило цілісності пов'язано зі станом БД в конкретний момент часу. Отже, необхідний механізм, що дозволяє підтримувати БД цілісною. Для цього потрібно визначити операції, які можуть порушити цілісність, і або заборонити їх, або ввести додаткові *компенсаційні операції*, які будуть виправляти тимчасове порушення цілісності БД.

Наприклад, нам необхідно видалити групу ПМ-51 з відношення *Groups*. Якщо ми просто видалимо відповідний кортеж з таблиці *Groups*, ми порушимо цілісність, тому що в таблиці *Students* залишаться студенти, що належать групі, що вже не існує. Саме для таких випадків розробляються компенсаційні операції.

Таким чином, для БД необхідно передбачити компенсаційні операції для двох моментів:

1) видалення об'єкта посилання зовнішнього ключа, тобто посилального кортежу;

2) зміна (оновлення) потенційного ключа, на який є посилання.

Для компенсації цих операцій існують як мінімум дві можливості:

1. Обмежити виконання операції. Для операції видалення – не видаляти кортеж, поки не видалять всі кортежі, що посилаються, тобто відкласти видалення;

2. Каскадувати. Тут можливо кілька варіантів, наприклад при видаленні:

- видалити сам кортеж і всі відповідні кортежі, що посилаються;
- видалити сам кортеж, а для всіх кортежів, що посилаються, виправити значення на правильне, наприклад, встановити NULL-значення для даного атрибуту.

NULL-значення

Іноді потрібно можливість позначити відсутність інформації, яка є необов'язковою в даному відношенні. Наприклад, для відношення *Students* таким необов'язковим атрибутом може бути **Height** (зріст студента). Проблему можна вирішити двома способами.

1. Використовувати спеціальні значення того ж типу даних, що і сам атрибут. Наприклад, атрибут **Height** має тип **int**, тоді можна використовувати -1 для позначення відсутності інформації про зріст студента, а будь-яке інше число буде вказувати сам зріст.

2. Використовувати спеціальні універсальні маркери – NULL-значення. Едгар Кодд[8] запропонував другий варіант, головною перевагою якого є те, що NULL-значення деякого атрибуту свідчить саме про його відсутність, тобто це не те ж саме, що і число нуль або порожній рядок. Однак такі невизначені значення можуть викликати ускладнення при забезпеченні цілісності БД.

Серед фахівців розділилися думки щодо необхідності цих міток. Е. Кодд вважає, що вони повинні бути невід'ємною частиною БД, а К. Дейт[9], навпаки, вважає, що вони навіть шкідливі.

[8] Едгард Ф. Кодд (Edgar F. Codd) (1923-2003) – американський вчений англійського походження. Довгий час працював в корпорації IBM. Створив основи теорії реляційних баз даних. Сформулював 12 законів аналітичної обробки даних і ввів термін OLAP (On-Line Analytical Processing – оперативна аналітична обробка).

[9] Кристофер Дж. Дейт (Christopher J. Date) (н.1941) – англійський учений, який працював над теорією реляційних баз даних спільно з Е. Коддом.

У загальному випадку в БД для кожного атрибуту можна задати, дозволено або заборонено містити NULL-значення.

Розглянемо вплив NULL-значення на різні ключі. З використанням цього значення вводиться правило цілісності об'єктів: жоден елемент первинного ключа базового відношення не може бути NULL значенням. Це правило пояснюється наступним: кортежі відношень відповідають об'єктам реального світу, отже, ці об'єкти різні, тобто деяким чином розпізнані, а первинні ключі виконують функцію унікальної ідентифікації.

Правило забезпечення цілісності може бути застосовано тільки:

1) до базових відношень, а не обчислюваним (похідних);

2) до первинних ключів, а для альтернативних може бути дозволено або заборонено використання NULL-значення.

NULL-значення можуть використовуватися або при вставці і зміні запису (для позначення відсутності інформації) або при каскадному видаленні. Наприклад, ми хочемо видалити групу, але при цьому не видаляти студентів цієї групи. В цьому випадку в поле **GroupID** відношення *Students* ми можемо вказати NULL-значення для всіх студентів, які належать групі, що видаляється.

StudentID	Name	GroupID	BirthDate
1	Козаков Петро	2	23.04.1990
2	Васильєв Іван	1	11.05.1991
4	Шишкіна Дар'я	2	23.09.1991

GroupID	Name
1	ПМ-41
2	ПМ-51

Після видалення групи ПМ-51, отримаємо наступний стан відношення *Students*:

StudentID	Name	GroupID	BirthDate
1	Козаков Петро	NULL	23.04.1990
2	Васильєв Іван	1	11.05.1991
4	Шишкіна Дар'я	NULL	23.09.1991

Короткі висновки. Розглянуто основні елементи реляційних баз даних, а також питання цілісності даних. Дано визначення первинних і зовнішніх ключів.

ЛЕКЦІЯ 9. ОПЕРАТОРИ РЕЛЯЦІЙНОЇ АЛГЕБРИ

У лекції розглядаються основні оператори реляційної алгебри і наводяться приклади їх реалізації на мові SQL.

Ціль: познайомити з основними реляційними операторами.

Поняття реляційної алгебри

Реляційна алгебра – це набір операцій, які приймають відношення в якості операндів і повертають відношення в якості результату. Перша версія цієї алгебри була визначена Е. Коддом.

В основі всіх реляційних БД лежить використання реляційної алгебри, яка забезпечує запис виразів для реалізації на деякій мові, наприклад, SQL. Якщо можливості мови як мінімум відповідають можливостям, забезпеченим операціями алгебри, то її називають *реляційно повною*.

Зазвичай виділяють 8 основних операторів реляційної алгебри і кілька додаткових (їх кількість змінюється з часом). Ми розглянемо два додаткових оператора: розширення і підбиття підсумків.

Основні оператори реляційної алгебри

Реляційна алгебра включає вісім основних операцій, які поділяються на дві групи, в кожну з яких входить чотири операції.

1. Традиційні операції з множинами, модифіковані для таблиць (відношень).

1) Об'єднання: **A UNION B**

Результатом операції об'єднання є відношення, що містить всі кортежі, що належать одному з двох або обом відношенням.

На відміну від об'єднання множин, результатом об'єднання відношень має стати відношення, а не набір різнорідних кортежів.

При об'єднанні повинні дотримуватися дві умови:

- відношення повинні бути сумісні за типом, тобто мати одну і ту саму множину атрибутів, визначених на одних і тих же доменах;
- результатом кожної операції має бути також відношення (властивість замкнутості).

Нехай є таблиця *Students*:

StudentID	Name	GroupID	BirthDate
1	Козаков Петро	2	23.04.1990
2	Васильєв Іван	1	11.05.1991
4	Шишкіна Дар'я	2	23.09.1991

І відношення *Teachers*:

TeacherID	Name	BirthDate
1	Кісліцин О.П.	1.2.1970
2	Царьов С.М.	10.03.1964
4	Пестов Д.Н.	2.05.1980

Нам потрібно вивести список всіх викладачів і студентів із зазначенням їх дня народження. Для цього можна використовувати такі оператори SQL:

```
SELECT Name, BirthDate FROM Students
UNION
SELECT Name, BirthDate FROM Teachers
```

Результатом виконання буде наступне відношення:

Name	BirthDate
Козаков Петро	23.04.1990
Васильєв Іван	11.05.1991
Шишкіна Дар'я	23.09.1991
Кісліцин О.П.	1.2.1970
Царьов С.М.	10.03.1964
Пестов Д.Н.	2.05.1980

2) Перетин: **A INTERSECT B**

Результатом операції перетину є відношення, що містить кортежі, які належать обом відношенням.

Для операції перетину необхідні ті ж дві умови, що і для об'єднання: сумісність за типом і замкнутість.

Приклад: розгляньмо відношення *Teachers* (відношення) і *Supervisors* (куратори, можуть керувати викладачами):

Teachers:

TeacherID	Name	BirthDate
1	Кісліцин О.П.	1.2.1970
2	Царьов С.М.	10.03.1964
4	Пестов Д.Н.	2.05.1980

Supervisors:

SupervisorID	Name	BirthDate
1	Кісліцин О.П.	1.2.1970
2	Царьов С.М.	10.03.1964
4	Нечаєв Н.В.	12.08.1970

Потрібно вивести всіх викладачів (з таблиці Teachers), які одночасно є кураторами.

```
SELECT Name FROM Teachers
INTERSECT
SELECT Name FROM Supervisors
```

Результатом:

Name
Кісліцин О.П.
Царьов С.М.

3) Віднімання: **A MINUS B**

Результатом операції віднімання є відношення, яке містить всі кортежі, що належать першому відношенню і не належать другому відношенню.

Умови необхідні ті ж: сумісність за типом і замкнутість.

Приклад: розгляньмо відношення *Teachers* (викладачі) і *Supervisors* (куратори, можуть керувати викладачами):

Teachers:

TeacherID	Name	BirthDate
1	Кісліцин О.П.	1.2.1970
2	Царьов С.М.	10.03.1964
4	Пестов Д.Н.	2.05.1980

Supervisors:

SupervisorID	Name	BirthDate
1	Кісліцин О.П.	1.2.1970
2	Царьов С.М.	10.03.1964
4	Нечаев Н.В.	12.08.1970

Потрібно вивести всіх викладачів (з таблиці Teachers), які одночасно є кураторами.

```
SELECT Name FROM Teachers
EXCEPT
SELECT Name FROM Supervisors
```

Результатом выполнения запроса будет:

Name
Пестов Д.Н.

4) Множення: **A TIMES B**

Результатом є відношення, що містить всі можливі кортежі, які є поєднанням двох кортежів, що належать двом відношенням.

Для множення необхідно властивість замкнутості, тобто результатом є не просто множина пар кортежів, а множина цілих кортежів. В реляційній алгебрі кожна пара кортежів замінюється одним завдяки зчепленню (*зчеплення* – це об'єднання в сенсі теорії множин, а не реляційної алгебри).

При множенні може виникнути проблема однакових імен атрибутів. У цьому випадку одне з конфліктних полів необхідно перейменувати.

Приклад: розгляньмо відношення *Students* і *Courses* (навчальні курси):

Students:

StudentID	Name	GroupID	BirthDate
1	Козаков Петро	2	23.04.1990
2	Васильєв Іван	1	11.05.1991
4	Шишкіна Дар'я	2	23.09.1991

Courses:

CourseID	Name
1	Бази даних
2	Технології програмування
3	Програмування у середовищі C++

Потрібно для кожного студента вивести список всіх доступних навчальних курсів. Складемо наступний код:

```
SELECT Students.Name AS 'Student', Courses.Name AS 'CourseName'
FROM Students
CROSS JOIN Courses
```

Результат виконання запиту:

Student	CourseName
Козаков Петро	Бази даних
Васильєв Іван	Бази даних
Шишкіна Дар'я	Бази даних
Козаков Петро	Технології програмування
Васильєв Іван	Технології програмування
Шишкіна Дар'я	Технології програмування
Козаков Петро	Програмування у середовищі C++
Васильєв Іван	Програмування у середовищі C++
Шишкіна Дар'я	Програмування у середовищі C++

2. Спеціальні реляційні операції.

1) ***Вибірка (або обмеження)***

Результатом вибірки є відношення, що містить всі кортежі з вихідного відношення, які задовольняють визначеній умові.

Нехай дано таке відношення Students:

StudentID	Name	GroupID
1	Козаков Петро	2
2	Васильєв Іван	1
4	Шишкіна Дар'я	2
5	Драгомирів Євгеній	1
6	Васнєцова Євгенія	2

Оберімо лише тих студентів, які належать групі 2:

SELECT * FROM Students
WHERE GroupID = 2

Результатом выполнения будет отношение:

StudentID	Name	GroupID
1	Козаков Петро	2
4	Шишкіна Дар'я	2
6	Васнєцова Євгенія	2

2) Проекція

Результатом проекції є відношення, що містить всі кортежі після видалення з них деяких атрибутів. В цьому випадку кортежі-результати називаються *подкортежами*.

Звернімо увагу на два моменти:

1) можливе зазначення всіх атрибутів вихідного відношення для проекції – вийде *тотожна проекція*;

2) можливо вказати порожній список атрибутів – вийде *нульова проекція*.

Здійснити проекцію можна зазначенням після SELECT списку необхідних атрибутів.

Нехай дано наступне відношення *Students*:

StudentID	Name	GroupID	BirthDate
1	Козаков Петро	2	23.04.1990
2	Васильєв Іван	1	11.05.1991
4	Шишкіна Дар'я	2	23.09.1991

Виберімо з таблиці лише імена та дати народження:

SELECT Name, BirthDate FROM Students

Результатом виконання стане відношення:

Name	BirthDate
Козаков Петро	23.04.1990
Васильєв Іван	11.05.1991
Шишкіна Дар'я	23.09.1991

3) З'єднання

Результат з'єднання – це відношення, кортежі якого є поєднанням двох кортежів, що належать двом початковим відношенням і мають спільні значення для одного або декількох атрибутів (спільне значення у відношенні-результаті з'являється тільки один раз).

Нехай дано наступне відношення *Students*:

StudentID	Name	GroupID	BirthDate
1	Козаков Петро	2	23.04.1990
2	Васильєв Іван	1	11.05.1991
4	Шишкіна Дар'я	2	23.09.1991

і відношення *Groups*:

GroupID	GroupName
1	ПМ-11
2	ПМ-12
3	ПМ-21

З'єднаємо ці таблиці по полю **GroupID** і виведемо ім'я студента і назву навчальної групи:

```
SELECT Name, GroupName FROM Students
INNER JOIN Groups ON Students.GroupID = Groups.GroupID
```

Результатом виконання буде наступне відношення:

Name	GroupName
Козаков Петро	ПМ-12
Васильєв Іван	ПМ-11
Шишкіна Дар'я	ПМ-12

Це з'єднання має властивості асоціативності і комутативності.

4) Ділення: **A DIVIDEBY B**

Результатом ділення двох відношень (бінарного і унарного) є відношення, що містить всі значення атрибута першого бінарного відношення, які відповідають всім значенням унарного відношення.

Ця операція не має аналога в MS SQL Server 2008, тому розглянемо на прикладі поділу відношення *R1* на *R2*:

R1:

A	X
A	Y
B	Z
B	X
C	Y

R2:

X
Y

У результаті отримаємо відношення:

R:

A

Операції розширення та підбиття підсумків

Після того, як Е. Кодд запропонував вісім основних операцій, численні автори запропонували нові алгебраїчні операції. Ми розглянемо лише дві з них – розширення і підбиття підсумків. Ці операції вдало доповнюють основний набір і є найбільш затребуваними.

1. Розширення

За допомогою операції розширення з відношення створюється нове відношення, що містить новий атрибут, значення якого отримані за допомогою скалярних обчислень.

Нехай дано наступне відношення *Teachers*:

TeacherID	Name	BirthDate
1	Кісліцин О.П.	1.2.1970
2	Царьов С.М.	10.03.1964
4	Пестов Д.Н.	2.05.1980

Добавим новый атрибут **Age**, который будет показывать, сколько полных лет исполнилось преподавателю:

SELECT TeacherID, Name, BirthDate, DATEDIFF(YEAR, BirthDate, GetDate()) AS 'Age' FROM Teachers

Результатом виконання буде відношення:

TeacherID	Name	BirthDate	Age
1	Кісліцин О.П.	1.2.1970	39
2	Царьов С.М.	10.03.1964	45
4	Пестов Д.Н.	2.05.1980	29

2. Підбиття підсумків

Операція підбиття підсумків дає можливість "вертикальних" обчислень. Для цього використовуються агрегатні функції, які для набору значень повертають одне єдине. Найбільш поширені функції: *Sum, Count, Avg, Min, Max.*

Нехай дано відношення *Students*:

StudentID	Name	GroupID
1	Козаков Петро	2
2	Васильєв Іван	1
4	Шишкіна Дар'я	2
5	Драгомирів Євгеній	1
6	Васнєцова Євгенія	2

Потрібно підрахувати, скільки студентів у кожній групі:

SELECT GroupID, **Count**(StudentID) as 'StudentCount' FROM Students
GROUP BY GroupID

Результатом виконання стане відношення:

GroupID	StudentCount
1	2
2	3

Оператори оновлення

Призначені для керування даними в таблицях. Існує три операції оновлення.

1) Вставка запису

Дозволяє додавати одну або кілька нових записів у відношення.

Нехай є відношення *Students*:

StudentID	Name	GroupID
1	Козаков Петро	2
2	Васильєв Іван	1
4	Шишкіна Дар'я	2

Щоб додати в нього два нових записи, можна використати наступні оператори SQL:

INSERT INTO Students (Name, GroupID) VALUES ('Драгомирів Євгеній', 1)
INSERT INTO Students (Name, GroupID) VALUES ('Васнєцова Євгенія', 2)

После выполнения, отношение *Students* будет выглядеть следующим образом:

StudentID	Name	GroupID
1	Козаков Петро	2
2	Васильєв Іван	1
4	Шишкіна Дар'я	2
5	Драгомирів Євгеній	1
6	Васнєцова Євгенія	2

2) Видалення запису

Видаляє всі записи з відносини або тільки записи, що задовольняють заданому критерію.

Нехай є відношення *Students*:

StudentID	Name	GroupID
1	Козаков Петро	2
2	Васильєв Іван	1
4	Шишкіна Дар'я	2
5	Драгомирів Євгеній	1
6	Васнєцова Євгенія	2

Щоб видалити з нього студентів, що входять в групу з номером 1, можна використати наступні оператори SQL:

```
DELETE Students
WHERE GroupID = 1
```

Після їх виконання відношення *Students* набуде такого вигляду:

StudentID	Name	GroupID
1	Козаков Петро	2
4	Шишкіна Дар'я	2
6	Васнєцова Євгенія	2

Якщо не вказати розділ WHERE в операторі DELETE, то будуть видалені всі записи з таблиці.

3) Оновлення запису

Дозволяє оновлювати значення зазначених полів: усіх або тільки певних записів.

Нехай є відношення *Students*:

StudentID	Name	GroupID
1	Козаков Петро	2
2	Васильєв Іван	1
4	Шишкіна Дар'я	2
5	Драгомирів Євгеній	1
6	Васнєцова Євгенія	2

Щоб перевести всіх студентів, що входять в групу з номером 1, в групу з номером 2, можна використати наступні оператори SQL:

```
UPDATE Students SET GroupID = 2
WHERE GroupID = 1
```

Після виконання, відношення *Students* буде виглядати наступним чином:

StudentID	Name	GroupID
1	Козаков Петро	2
2	Васильєв Іван	2
4	Шишкіна Дар'я	2
5	Драгомирів Євгеній	2
6	Васнєцова Євгенія	2

Короткі висновки. Розглянуто основні операції над таблицями, а також були наведені приклади використання цих операцій на мові SQL.

ЛЕКЦІЯ 10. ПЕРШІ НОРМАЛЬНІ ФОРМИ

У лекції розглядаються перші три нормальні форми і дається уточнення третьої нормальної форми. Наводяться приклади відношень, які не відповідають нормальним формам, і демонструються способи їх нормалізації.

Ціль: ввести визначення перших трьох нормальних форм і обґрунтувати необхідність їх застосування.

Процес нормалізації

Процес нормалізації ґрунтується на концепції *нормальних форм*. Кажуть, що відношення знаходиться в певній нормальній формі, якщо воно задовольняє заданому набору умов.

На рис. 10.1 показано кілька нормальних форм, які визначені до теперішнього часу. Перші три були описані Е. Коддом, наступну формулу вивели Р. Бойс і Е. Кодд (НФБК – нормальна форма Бойса-Кодда), а четверту і п'яту нормальні форми визначив Р. Фейгін.

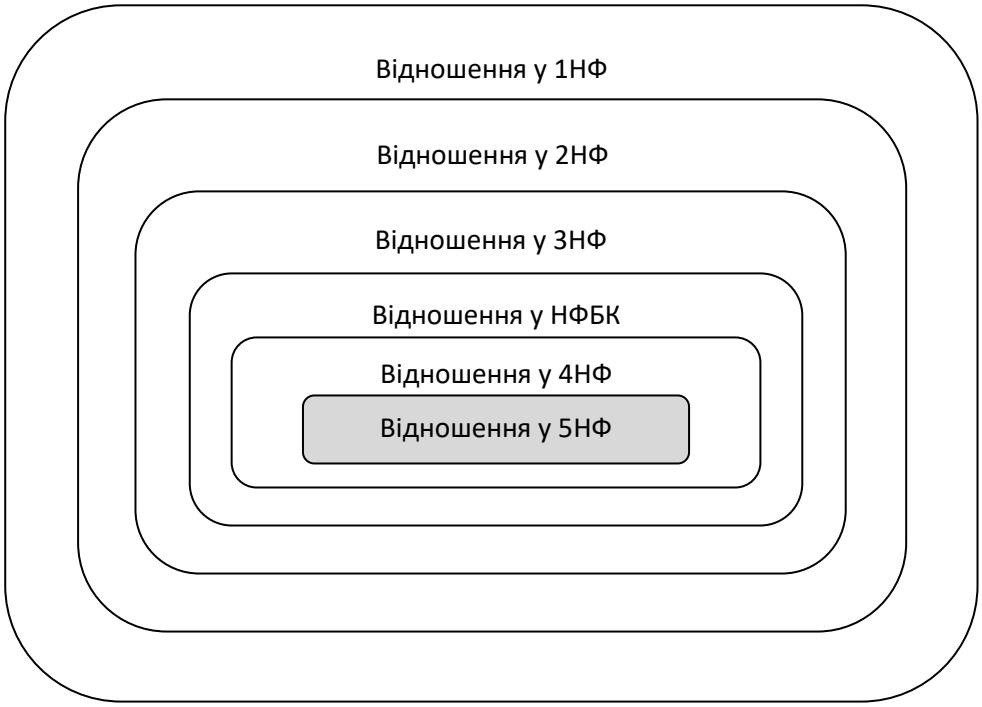


Рис. 10.1. Рівні нормалізації

З рис. 10.1 видно, що відношення в деякій нормальній формі передбачає приведення його до попередньої нормальної форми.

Поняття перших нормальних форм засноване на *функціональних залежностях* (ФЗ) і *процесі декомпозиції*. Тому перш за все потрібно дати визначення функціональних залежностей, розглянути правила їх виведення, а також можливість і необхідність декомпозиції деяких відношень.

Поняття функціональної залежності

Нехай *R* – це відношення. З одного боку, воно має конкретне (постійне) значення в даний момент часу. З іншого боку, це змінна, яка в кожен момент часу може прийняти деяке нове значення.

Поняття ФЗ можна застосувати і до першого, і до другого випадку. Однак ми будемо розглядати тільки другий випадок, тому що він більше відповідає реальності.

Визначення функціональної залежності. Нехай *R* – змінна відношення. *X* та *Y* – випадкові підмножини множини атрибутів *R*. Тоді *Y функціонально залежить* від *X*, що у символьному вигляді записується так $X \rightarrow Y$ (читається як «*X* функціонально визначає *Y*») тоді і тільки тоді, коли для будь-якого допустимого значення *R* кожне значення *X* пов'язано точно з одним значенням *Y*.

Тут *X* називають *детермінантом* ФЗ, а *Y – залежною частиною* ФЗ.

Приклад: Нехай *R* – відношення *Students*. *X* – код студента, а *Y* – множина всіх атрибутів студента. Тоді $X \rightarrow Y$, оскільки *X* є первинним ключем, який унікально ідентифікує запис у таблиці *Students*.

Таке твердження буде правильне і для більш загального випадка: якщо *X* – це потенційний ключ, то множина всіх атрибутів *R* завжди функціонально залежить від *X*.

Однак слід мати на увазі, що якщо в *R* є ФЗ, ліва частина якої не включає потенційний ключ, то *R* має *надмірність*, що ускладнює забезпечення цілісності даних і займає зайві ресурси системи.

Якщо жоден атрибут не може бути опущений з лівої частини, то таку функціональну залежність називають *непривідною* (точніше, *непривідною ліворуч*).

Приклад:

{**StudentID**, **FirstName**, **LastName**, **MiddleName**} → {**BirthDate**} – привідна ФЗ.
{**StudentID**} → {**BirthDate**} – непривідна ФЗ.

Множина функціональних залежностей називається непривідною тоді і тільки тоді, коли вона має всі три перелічені нижче властивості:

1) Залежна частина кожної функціональної залежності містить тільки один атрибут.
2) Детермінант кожної функціональної залежності є непривідною.
3) Жодна функціональна залежність з множини не може бути видалена без втрати інформації про зв'язки.

Розгляд множини непривідних ФЗ важливо для нормалізації відносин.

Виділяють два види ФЗ:

1. *Тривіальні ФЗ* – це ФЗ, в яких права частина (*Y*) є підмножиною лівої частини (*X*). З практичної точки зору вони не уявляють значного інтересу, проте з точки зору формальної теорії залежностей необхідно враховувати їх наявність.

2. *Нетривіальні ФЗ*. Вони дійсно є обмеженнями цілісності даних, тому в подальшому ми будемо розглядати саме нетривіальні ФЗ.

Для визначення того в якій нормальній формі знаходиться відношення, потрібно знайти всі ФЗ. Існують три *правила Армстронга* (шведський математик), що дозволяють з початкової множини ФЗ вивести можливі ФЗ.

Нехай *A, B, C* – це підмножини множини атрибутів відношення *R, AB* – об'єднання цих підмножин.

1. *Правило рефлексивності***.** Якщо множина *B* є підмножиною множини *A*, то $A \rightarrow B$. (По суті, це визначення тривіальної залежності.)

2. *Правило доповнення***.** Якщо $A \rightarrow B$, то $AC \rightarrow BC$.

3. *Правило транзитивності***.** Якщо $A \rightarrow B$ і $B \rightarrow C$, то $A \rightarrow C$.

Кожне з цих правил може бути доведено на основі визначення ФЗ.

Однак з метою спрощення отримання всіх ФЗ можна вивести ще кілька додаткових правил (нехай *D* – це ще одна довільна підмножина множини атрибутів *R*):

4. *правило самовизначення***.** $A \rightarrow A$.

5. *Правило декомпозиції***.** Якщо $A \rightarrow BC$, то $A \rightarrow B$ і $A \rightarrow C$.

6. *Правило об'єднання***.** Якщо $A \rightarrow B$ і $A \rightarrow C$, то $A \rightarrow BC$.

7. *Правило композиції*. Якщо $A \rightarrow B$ і $C \rightarrow D$, то $AC \rightarrow BD$.

8. *Теорема загального об'єднання*. Якщо $A \rightarrow B$ і $C \rightarrow D$, то $A(C – B) \rightarrow BD$.

Назва теореми вказує на те, що деякі з перерахованих вище правил можуть бути виведені як окремі випадки цієї теореми.

Однак слід мати на увазі, що ці правила не забезпечують чіткого алгоритму отримання всіх ФЗ. Більш того, такого алгоритму не існує. Єдиний шлях – це перебір всіх варіантів.

Декомпозиція без втрат

Суттєвим аспектом процедури нормалізації є декомпозиція – розбиття відношення на похідні. При цьому таке розбиття не повинно спричинити за собою втрату інформації. Якщо відношення розбите на два без втрат, то це розбиття можна назвати зворотним, тобто можна зробити зворотне з'єднання.

Приклад: розгляньмо відношення *Students*:

StudentID	LastName	FirstName	MiddleName	GroupID	BirthDate
1	Козаков	Петро	Володимирович	1	1.1.1992
2	Васильєв	Іван	Аркадійович	2	23.09.1990
4	Шишкіна	Дар'я	Сергіївна	1	12.5.1991

1-й варіант декомпозиції. Розіб'ємо *Students* на два наступних відношення:

StudentID	LastName	FirstName	MiddleName	GroupID
1	Козаков	Петро	Володимирович	1
2	Васильєв	Іван	Аркадійович	2
4	Шишкіна	Дар'я	Сергіївна	1

GroupID	BirthDate
1	1.1.1992
2	23.09.1990

При такому варіанті розбиття деяка інформація втрачається: неможливо визначити вірну дату народження (*BirthDate*) кожного студента.

2-й варіант декомпозиції. Розіб'ємо *Students* на два наступних відношення:

StudentID	LastName	FirstName	MiddleName	GroupID
1	Козаков	Петро	Володимирович	1
2	Васильєв	Іван	Аркадійович	2
4	Шишкіна	Дар'я	Сергіївна	1

StudentID	BirthDate
1	1.1.1992
2	23.09.1990
4	12.5.1991

Це декомпозиція без втрат, оскільки ці два відношення можна об'єднати і вийде початкове відношення *Students*.

Можна помітити, що процес декомпозиції фактично являє собою операції *проекції,* тому кожне отримане при декомпозиції відношення складається з підмножини полів початкового відношення.

Щоб гарантувати, що після декомпозиції зворотне з'єднання відношень дасть початкове, скористаємося *теоремою Хіта*.

Теорема Хіта

Нехай $R\{A, B, C\}$ – це відношення, де A, B і C – множини атрибутів цього відношення. Якщо R задовольняє функціональній залежності $A \rightarrow B$, то R дорівнює з'єднанню її проекцій по атрибутам $\{A, B\}$ і $\{A, C\}$.

Перша, друга та третя нормальні форми

Перша нормальна форма (1НФ). Відношення знаходиться в 1НФ тоді і тільки тоді, коли всі використовувані домени містять тільки скалярні значення, тобто значення всіх полів відношення повинні бути неподільні.

Нехай дано таке відношення *Students*:

StudentID	Name
1	Козаков Петро Володимирович
2	Васильєв Іван Аркадійович
4	Шишкіна Дар'я Сергіївна

Поле **Name** містить одночасно прізвище, ім'я та по батькові. Це поле можна розділити на три, тоді отримаємо відношення в 1НФ:

StudentID	LastName	FirstName	MiddleName
1	Козаков	Петро	Володимирович
2	Васильєв	Іван	Аркадійович
4	Шишкіна	Дар'я	Сергіївна

Слід мати на увазі, що значення полів повинні бути **логічно** неподільними, тому що прізвище складається з окремих букв, але такий поділ не матиме сенсу для відношення *Students*.

Друга нормальна форма (2НФ). Відношення знаходиться в 2НФ тоді і тільки тоді, коли воно знаходиться в 1НФ і кожен неключових атрибут непривідно залежить від первинного ключа.

Розгляньмо наступне відношення:

StudentID	LastName	FirstName	MiddleName	Course	Score
1	Козаков	Петро	Володимирович	інформатика	5
2	Васильєв	Іван	Аркадійович	математика	4
4	Шишкіна	Дар'я	Сергіївна	математика	5
2	Васильєв	Іван	Аркадійович	інформатика	3

Первинним ключем тут буде {**StudentID**, **Course**}, тоді проаналізуємо ФЗ цього відношення, де детермінантою є первинний ключ, а в правій частині неключовий атрибут. Розгляньмо наступну ФЗ:

{**StudentID**, **Course**} → {**LastName**}

Очевидно, що вона є привідною, тому що прізвище залежить тільки від **StudentID**, а значить існує ФЗ: {**StudentID** } → {**LastName**}.

Отже дане відношення не знаходиться у 2НФ. Зробімо декомпозицію вихідного відношення на наступні два.

Students:

StudentID	LastName	FirstName	MiddleName
1	Козаков	Петро	Володимирович
2	Васильєв	Іван	Аркадійович
4	Шишкіна	Дар'я	Сергіївна

Scores:

StudentID	Course	Score
1	інформатика	5
2	математика	4
4	математика	5
2	інформатика	3

Третя нормальна форма (3НФ). Відношення знаходиться в 3НФ тоді і тільки тоді, коли воно знаходиться у 2НФ і кожен неключовий атрибут не є транзитивно залежним від первинного ключа (це означає, що у відношенні відсутні будь-які *взаємні* залежності).

Розгляньмо наступне відношення *Students*:

StudentID	LastName	FirstName	MiddleName	GroupID	Supervisor
1	Козаков	Петро	Володимирович	1	Царьов С.М.
2	Васильєв	Іван	Аркадійович	2	Пестов Д.Н.
4	Шишкіна	Дар'я	Сергіївна	1	Царьов С.М.

Так як існують ФЗ: **StudentID → GroupID** і **GroupID → Supervisor**, то атрибут **Supervisor** транзитивно залежить від первинного ключа **StudentID**. Відтак відношення не перебуває у 3НФ. Наведемо ставлення до 3НФ.

Students:

StudentID	LastName	FirstName	MiddleName	GroupID
1	Козаков	Петро	Володимирович	1
2	Васильєв	Іван	Аркадійович	2
4	Шишкіна	Дар'я	Сергіївна	1

Groups:

GroupID	Supervisor
1	Царьов С.М.
2	Пестов Д.Н.

Уточнення третьої нормальної форми – нормальна форма Бойса-Кодда

При визначенні 3НФ було зроблено припущення про те, що відношення має тільки один потенційний ключ, який і є первинним. Якщо розглянути більш загальний випадок, то первинне визначення, дане Е. Коддом для ЗНФ, виявляється не у всіх випадках задовільним. Зокрема, воно неадекватно при виконанні наступних умов:

1) відношення має два (або більше) потенційних ключа;
2) ці потенційні ключі є складеними;
3) вони перекриваються (тобто мають принаймні один спільний атрибут).

Тому згодом вихідне визначення ЗНФ було замінено більш суворим визначенням Бойса-Кодда.

На практиці комбінація всіх трьох умов зустрічається рідко, і для відносин, в яких не виконуються всі ці три умови, 3НФ і нормальна форма Бойса-Кодда повністю еквівалентні.

Нормальна форма Бойса-Кодда (НФБК). Відношення знаходиться в нормальній формі Бойса-Кодда тоді і тільки тоді, коли кожна її нетривіальна і непривідна ліворуч функціональна залежність має в якості свого детермінанта деякий потенційний ключ.

Розгляньмо наступне відношення:

StudentID	CourseID	TeacherID
1	20	23
2	20	12
4	15	9

Кожен кортеж означає, що деякий студент вивчає певну дисципліну у зазначеного викладача. При цьому існують наступні обмеження:

1. Кожен викладач веде тільки одну дисципліну.

2. Одну дисципліну може вести кілька викладачів.
3. Для деякого студента одну дисципліну веде тільки один викладач.

В даному відношенні є два потенційних ключа {**StudentID**, **CourseID**} і {**StudentID**, **TeacherID**}. Обидва ключі є складовими і вони мають спільний атрибут **StudentID**, тобто перекриваються. Таким чином виконані всі три умови, які можуть привести до ситуації, коли ставлення може перебувати в 3НФ, але не знаходиться в НФБК.

Очевидно, що відношення знаходиться в 3НФ:

- значення всіх атрибутів неподільні (1НФ);
- кожен неключових атрибут непривідно залежить від первинного ключа (2НФ);
- всі неключові атрибути нетранзитивно залежать від потенційного ключа (3НФ).

Однак, в даному відношенні існує ФЗ **TeacherID → CourseID** і **TeacherID** при цьому не є потенційним ключем, отже відношення не перебуває у НФБК.

Якщо зробити декомпозицію цього відношення на два: {**StudentID**, **CourseID**} і {**CourseID**, **TeacherID**}, то втратимо інформацію про те, який саме викладач веде дисципліну для конкретного студента. Це пояснюється тим, що початкове відношення є *атомарним*, тобто його не можна розбити на дві незалежні проекції.

Таким чином, в процесі нормалізації не завжди є сенс прагнути до атомарних відношень, але, тим не менше, для основних об'єктів бази даних рекомендується домагатися атомарності.

Якщо відношення знаходиться тільки в 1НФ, але не знаходиться у 2НФ і 3НФ, то можна говорити про *надмірності інформації*. Надмірність не тільки збільшує обсяг і трудомісткість заповнення бази даних, але і може привести до аномалій оновлення. Аномалії можуть призводити до труднощів при вставці, оновленні та видаленні, а головне до спотворення або втрати інформації.

Короткі висновки. Розглянуто перша, друга, третю нормальні форми і нормальна форма Бойса-Кодда. На прикладах розібраний процес приведення відносини в задану нормальну форму.

ЛЕКЦІЯ 11. ЧЕТВЕРТА І П'ЯТА НОРМАЛЬНІ ФОРМИ

У лекції розглядаються четверта і п'ята нормальні форми. Наводиться остаточна схема нормалізації БД. Даються визначення альтернативних нормальних форм.

Ціль: ввести поняття четвертої і п'ятої нормальних форм і обґрунтувати необхідність їх застосування.

Для визначення 4НФ необхідно ввести поняття *багатозначної залежності (БЗ)*, яке є узагальненням поняття функціональної залежності.

Багатозначна залежність

Нехай R – відношення, а A, B і C є довільними підмножинами множини атрибутів відносини R.

Тоді підмножина B *багатозначно залежить* від підмножини A, що символічно виражається наступним записом $A \rightarrow\rightarrow B$ (читається як «A багатозначно визначає B»), тоді і тільки тоді, коли в кожному допустимому значенні R множина значень B, відповідне заданій парі значень A, C, залежить від значення A і не залежить від значення C.

Розгляньмо відношення {**CourseID**, **TeacherID**, **RoomID**} з наступними обмеженнями:

1. Будь-яка дисципліна може вестися будь-якою кількістю викладачів і в будь-якій кількості кабінетів.
2. Викладачі та кабінети не залежать одне від одного.
3. Викладач може вести кілька різних дисциплін у різних кабінетах.

З цих обмежень видно, що відношення виходить надлишковим, тому що якщо існують такі два кортежі:

CourseID	TeacherID	RoomID
2	10	400
2	9	401

то в даному відношенні має існувати і такі два записи:

CourseID	TeacherID	RoomID
2	9	400
2	10	401

Таким чином, якщо ми хочемо додати інформацію про те, що якусь дисципліну може вести певний викладач, ми повинні вставити стільки записів, скільки кабінетів підходить для даної дисципліни.

Крім того, можна легко перевірити, що дане відношення знаходиться в НФБК: якщо відношення знаходиться в 1НФ і воно повністю ключове (єдиний потенційний ключ складається з всієї множини атрибутів відношення), то можна стверджувати, що воно знаходиться в НФБК. Це пояснюється тим, що неключових атрибутів немає, а, отже, всі вимоги 2НФ, 3НФ і НФБК виконуються автоматично.

Для нормалізації відношення, нам потрібно розбити його на два, але раніше ми виробляли декомпозицію на основі транзитивних ФЗ, тут же всі ФЗ тривіальні, тобто всі атрибути безпосередньо залежать від первинного ключа. Необхідно визначити спосіб декомпозиції даного відношення і довести, що декомпозиція буде проведена без втрат.

Правило багатозначної залежності: для відношення *R*, в якому існують підмножини безлічі атрибутів *A*, *B*, *C*, $A \rightarrow\rightarrow B$ тоді і тільки тоді, коли $A \rightarrow\rightarrow C$. Зазвичай це записують так: $A \rightarrow\rightarrow B \mid C$.

Теорема Фейгіна. Нехай *A*, *B* і *C* є множинами атрибутів змінної відносини *R*{*A*, *B*, *C*}. У такому випадку відношення *R* дорівнюватиме з'єднанню його проекцій по атрибутам {*A*, *B*} і {*A*, *C*} тоді і тільки тоді, коли для відносини *R* виконується багатозначна залежність $A \rightarrow\rightarrow B \mid C$.

Знайдімо всі БЗ у нашому відношенні:

- {**CourseID**} → {**TeacherID**}
- {**CourseID**} → {**RoomID**}

По теоремі Фейгіна ми можемо зробити декомпозицію за цими двома БЗ і при цьому ніяка інформація не буде втрачена. Отримаємо два відношення: {**CourseID**, **TeacherID**} і {**CourseID**, **RoomID**}. Так як ці відносини повністю ключові, то вони знаходяться в НФБК.

Тепер, дотримуючись теореми Фейгіна, можна дати визначення *четвертої нормальної форми.*

Четверта нормальна форма

Відношення *R* знаходиться у четвертій нормальній формі (4НФ) тоді і тільки тоді, коли в разі існування таких підмножин *A* і *B* атрибутів цього відношення *R*, для яких виконується нетривіальна багатозначна залежність $A \rightarrow\rightarrow B$, всі атрибути відношення *R* також *функціонально* залежать від атрибуту *A*.

Можна дати альтернативне формулювання: відношення R знаходиться в 4НФ тоді і тільки тоді, коли воно знаходиться в НФБК і всі багатозначні залежності щодо R фактично представляють собою функціональні залежності від його потенційних

ключів. Зверніть увагу на те, що, виходячи з цього визначення, знаходження в 4НФ передбачає обов'язкове знаходження в НФБК.

Відповідно, початкове відношення {**CourseID**, **TeacherID**, **RoomID**} не знаходиться в 4НФ, оскільки існує БЗ **CourseID** →→ **TeacherID**, яка не є ФЗ, і детермінант не є ключем.

Отримані дві проекції {**CourseID**, **TeacherID**} і {**CourseID**, **RoomID**} будуть у 4НФ, оскільки існують тільки ФЗ від єдиного потенційного ключа.

Залежність з'єднання

До сих пір за замовчуванням передбачалося, що єдиною необхідною або допустимої операцією в процесі нормалізації є заміна відношення за правилами декомпозиції без втрат *точно двома* її проекціями. Однак існують відносини, для яких не можна виконати декомпозицію без втрат на дві проекції, але які *можна* піддати декомпозиції без втрат на три або більшу кількість проекцій. Подібні відносини позначимо терміном *«n-декомпоноване відношення»,* де *n* – кількість проекцій, на які можна розбити без втрат.

Приклад. Розгляньмо приклад відношення, яке важко розбити без втрат на два відношення:

Teacher	Course	Group
Іванов С.Ю.	Технології програмування	ПМ-41
Іванов С.Ю.	Технології програмування	ПМ-51
Іванов С.Ю.	Бази даних	ПМ-41
Пестов О.А.	Бази даних	ПМ-51
Веснін Р.А.	Бази даних	ПМ-51

Разобьем на три отношения:

Teacher	Course
Іванов С.Ю.	Технології програмування
Іванов С.Ю.	Бази даних
Пестов О.В.	Бази даних
Веснін Р.А.	Бази даних

Course	Group
Технології програмування	ПМ-41
Технології програмування	ПМ-51
Бази даних	ПМ-41
Бази даних	ПМ-51

Teacher	Group
Іванов С.Ю.	ПМ-41
Іванов С.Ю.	ПМ-51

Пестов О.В.	ПМ-51
Веснін Р.А.	ПМ-51

Якщо провести з'єднання тільки двох відношень, то ми отримаємо зайві записи, яких не було в початковому відношенні. Але якщо ми з'єднаємо всі три проекції, то отримаємо в точності початкове відношення. Тому можна сказати, що вихідне відношення 3-декомпоноване.

Дамо визначення залежності з'єднання, необхідного для 5НФ.

Залежність з'єднання. Нехай *R* – відношення, а *A, B, ..., Z* – довільні підмножини множини його атрибутів. Відношення *R* задовольняє *залежності з'єднання* *{*A, B, . . . , Z*} (читається «зірочка *A, B, ..., Z*») тоді і тільки тоді, коли будь-яке допустиме значення відношення *R* еквівалентно з'єднанню його проекцій по підмножині *A, B, ..., Z* множини атрибутів.

Для нашого початкового відношення існує залежність з'єднання:

*{ {**Teacher**, **Course**}, {**Course**, **Group**}, {**Teacher**, **Group**} }.

Залежності з'єднань включають в себе багатозначні залежності, а багатозначні залежності включають в себе функціональні.

П'ята нормальна форма

Відношення *R* знаходиться в п'ятій нормальній формі (5НФ), яку іноді інакше називають *проекційно-сполучної нормальною формою*, тоді і тільки тоді, коли кожна нетривіальна залежність з'єднання у відношенні R визначається потенційним ключем (або ключами) *R*.

Відношення {**Teacher**, **Course**, **Group**} не перебуває у 5НФ, оскільки у відношенні існує тільки один потенційний ключ (первинний), то можна скласти тільки одну залежність з'єднання:

*{ {**Teacher**, **Course**}, {**Course**, **Group**}, {**Teacher**, **Group**} }

Ця залежність з'єднання не передбачається первинним ключем, тобто потенційний ключ не зустрічається в підмножинах цієї залежності

Розгляньмо відношення *Groups*: {**GroupID**, **GroupName**, **Faculty**, **Supervisor**}. Тут два потенційних ключа: **GroupID** і **GroupName**. Можна скласти наступні залежності з'єднання:

- *{ {**GroupID**, **Supervisor**}, {**GroupID**, **GroupName**, **Faculty**} }

- *{ {**GroupID**, **GroupName**}, {**GroupID**, **Faculty**}, {**GroupName**, **Supervisor**} }
- тощо

Слід зазначити, що в кожній підмножині є хоча б один потенційний ключ, і кожен потенційний ключ зустрічається хоча б один раз в цих підмножинах.

Втім, знайти в деякому відношенні всі залежності з'єднання набагато складніше, ніж багатозначні і функціональні залежності, тому процес перевірки на 5НФ недостатньо точний. Однак відносини, які доведені до 4НФ, але не перебувають у 5НФ, на практиці зустрічаються вкрай рідко.

Фактично 5НФ є *остаточною нормальною формою* у відношенні до операцій проекції і з'єднання (що відображено в її альтернативній назві – *проекційно-сполучна нормальна* форма).

Таким чином, якщо змінна відношення знаходиться в 5НФ, то *гарантується, що вона не містить аномалій*, які можуть бути виключені за допомогою її розбиття на проекції.

Підсумкова схема нормалізації

До сих пір весь процес нормалізації ґрунтувався на декомпозиції без втрат. Основна ідея полягала в наступному: нехай дано деяке відношення *R* у 1НФ і для нього визначені функціональні залежності, багатозначні залежності і залежності з'єднання. Завдання полягає в систематичному розбитті вихідного відношення *R* на такий набір менших (тобто таких, що мають меншу ступінь) відношень, який в деякому сенсі буде еквівалентний відношенню *R*, але з іншого боку буде кращим. Кожен етап процесу такого перетворення полягає в розбитті на проекції відносин, отриманих на попередньому етапі. При цьому на кожному етапі перетворення існуючі обмеження використовуються для вибору тих проекцій, які будуть отримані в цей раз.

Весь процес нормалізації можна неформально визначити за допомогою наступних **правил**:

1. Відношення в 1НФ слід розбити на такі проекції, які дозволять виключити всі функціональні залежності, які не є привідними. В результаті буде отриманий набір змінних відношення у 2НФ.

2. Отримані відношення у 2НФ слід розбити на такі проекції, які дозволять виключити всі існуючі транзитивні функціональні залежності. В результаті буде отриманий набір відношень у 3НФ.

3. Відношення в 3НФ слід розбити на проекції, що дозволяють виключити всі інші функціональні залежності, в яких детермінанти не є потенційними ключами. В результаті буде отриманий набір відношень у НФБК.

Зауважимо, що ці три правила можна об'єднати в одне: «Початкове відношення слід розбити на проекції, що дозволяють виключити функціональні залежності, в яких детермінанти не є потенційними ключами».

4. Відносини в НФБК слід розбити на проекції, що дозволяють виключити всі багатозначні залежності, які не є також функціональними. В результаті буде отриманий набір відношень в 4НФ. На практиці такі багатозначні залежності зазвичай виключаються перед виконанням етапів 1-3.

5. Відношення в 4НФ слід розбити на проекції, що дозволяють виключити всі залежності з'єднання, які не визначаються потенційними ключами. В результаті буде отриманий набір відношень в 5НФ.

З приводу наведених вище правил можна зробити кілька додаткових зауважень:

1) Процес розбиття на проекції на кожному етапі повинен бути виконаний без втрат і зі збереженням залежностей (там, де це можливо).

2) Іноді зручно користуватися альтернативними визначеннями нормальних форм:

- відношення R знаходиться в НФБК тоді і тільки тоді, коли кожна функціональна залежність визначається його потенційними ключами;
- відношення R знаходиться в 4НФ тоді і тільки тоді, коли кожна багатозначна залежність визначається його потенційними ключами;
- відношення R знаходиться в 5НФ тоді і тільки тоді, коли кожна залежність з'єднання визначається його потенційними ключами.

3) Загальне призначення процесу нормалізації полягає в наступному:

- виключення надмірності;
- усунення аномалій оновлення;
- розробка проєкта бази даних, який є досить «якісним» поданням реального світу, інтуїтивно зрозумілим і може служити гарною основою для подальшого розширення;
- спрощення процедури накладення обмежень цілісності (ця ціль забезпечується створенням зв'язку «один до багатьох», від ключового до неключових).

4) Необхідно пам'ятати, що дані рекомендації з приводу нормалізації є всього лише рекомендаціями і, ймовірно, можуть існувати міркування, за якими нормалізацію не слід виконувати до кінця або можна змінювати послідовність дій.

5) Ідеї нормалізації надзвичайно корисні для проєктування баз даних, але вони аж ніяк не є універсальним засобом з наступних причин:

- нормалізація дозволяє реалізувати певні обмеження цілісності, але на практиці, крім залежностей з'єднання, функціональних і багатозначних залежностей, існують й інші типи обмежень;

- декомпозиція може бути не унікальною (як правило, є кілька способів приведення заданого відношення до 5НФ), але існує дуже мало об'єктивних критеріїв, що дозволяють вибрати один з альтернативних варіантів декомпозиції;
- переслідування одночасно двох цілей (тобто приведення до НФБК і збереження залежностей) в деяких випадках призводить до конфліктної ситуації;
- не всяку надмірність даних можна усунути розбивкою на проекції.

Слід також зазначити, що існують хороші методики низхідного проєктування, які дозволяють тим або іншим конкретним способом створювати повністю нормалізований проєкт бази даних.

Інші нормальні форми

Крім вже описаних, існують і інші нормальні форми. Справа в тому, що *теорія нормалізації (теорія залежностей)* розвинулася в широку самостійну галузь знань. Дослідження в цій галузі тривають і понині, причому досить успішно, і після 5НФ були виведені деякі інші:

1. **Доменно-ключова нормальна форма (ДКНФ)**. Ця форма була запропонована Р. Фейгін, але на відміну від інших нормальних форм, вона не визначається в термінах функціональних залежностей, багатозначних залежностей або залежностей з'єднання. Замість цього стверджується, що відношення *R* знаходиться у ДКНФ тоді і тільки тоді, коли кожне накладене на нього обмеження є логічним наслідком *обмежень доменів* і *обмежень ключів* відносини *R*.

Тут обмеження домену – це обмеження, що пропонує використання для певного атрибута значень тільки з деякого заданого домену. А обмеження ключа – це обмеження, яке стверджує, що деякий атрибут або комбінація атрибутів являє собою потенційний ключ.

Незважаючи на те, що Р. Фейгін довів, що відношення в ДКНФ знаходиться і в 5НФ, не всі відносини можна привести до ДКНФ.

2. **Нормальна форма типу «вибірка - об'єднання».** Такі нормальні форми пов'язані з іншим напрямком в дослідженні нормалізації: проведення декомпозиції не на основі проекції, а на основі інших операцій. В даному випадку взяті операції вибірки і об'єднання.

Так, наприклад, таблицю *Students* на основі вибірки можна розбити на таблиці по кожній групі або по курсу.

Таким чином, завжди можна створити нову теорію нормалізації на основі вибірки, і нормальні форми цієї теорії будуть типу «вибірка-об'єднання». Тоді нормальні форми, розглянуті нами, можна назвати нормальними формами типу «проекція-з'єднання».

Короткі висновки. Розглянуті четверта і п'ята нормальні форми, які вважаються остаточними. Сформульовано основні правила нормалізації баз даних. Наведено альтернативні нормальні форми.

ЛЕКЦІЯ 12. ВИКОРИСТАННЯ MS SQL SERVER 2008 СУМІСНО З MS VISUAL STUDIO 2008

У лекції розглядаються нові технології роботи з даними: LINQ і ADO.NET. На прикладі демонструється можливість MS SQL Server виконувати функції і використовувати нові типи даних, створені в MS Visual Studio для платформи .Net Framework.

Ціль: познайомити з новими можливостями в області роботи з даними.

Створення функцій для MS SQL Server з використанням платформи .Net Framework

Дана можливість з'явилася ще в MS SQL Server 2005, коли стало можна підключати до MS SQL Server свої власні збірки, створені для платформи .Net Framework.

Завдяки цьому користувачі отримали можливість створювати в MS Visual Studio за допомогою однієї з мов програмування (C #, Visual Basic.Net та ін.) об`єкти, які потім можуть використовуватися всередині MS SQL Server. Дозволяється створення наступних об'єктів:

- нові типи даних;
- призначені для користувача функції;
- збережені процедури;
- агрегатні функції;
- тригери.

За замовчуванням MS SQL Server заборонено використання CLR (Common Language Runtime), тому потрібно явно включити цю опцію на сервері.

Зауваження. CLR – це загальномовна виконуване середовище, яке входить до складу Microsoft .Net Framework і відповідає за виконання будь-якого коду, створеного для платформи .Net Framework.

C# – об'єктно-орієнтована мова, розроблена спеціально для платформи .Net Framework. Однак можна використовувати і інші мови програмування для даної платформи.

Розглянемо процес створення звичайної функції для MS SQL Server на мові C # в MS Visual Studio 2008.

1. У MS Visual Studio 2008 створили новий проєкт (меню **File – New – Project…**)
2. У діалозі створення проєкта виберемо потрібний тип проєкта і вкажемо ім'я (рис. 12.1).

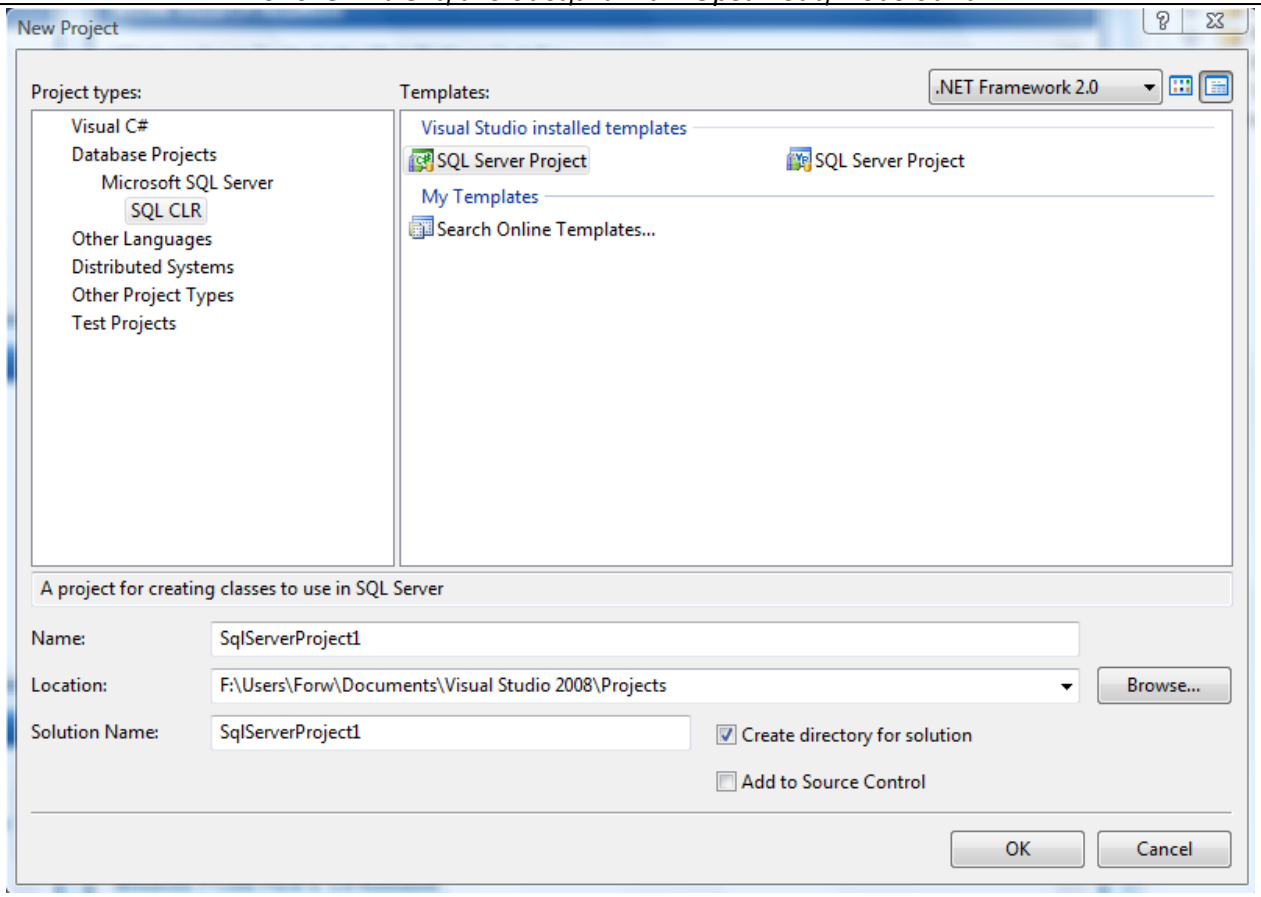


Рис. 12.1. Створення нового проєкту в MS Visual Studio 2008

3. Далі можна вказати сервер і базу даних, які згодом можна буде використовувати для налагодження коду (див. рис. 12.2).

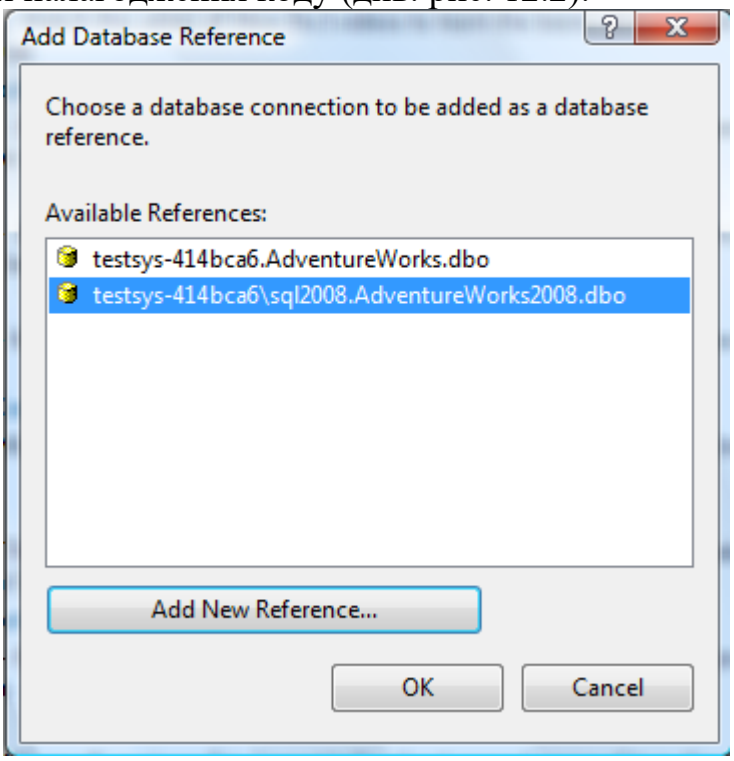


Рис. 12.2. Вибір серверу і бази даних

4. У створений проєкт додамо новий об'єкт – користувацьку функцію (див. рис. 12.3).

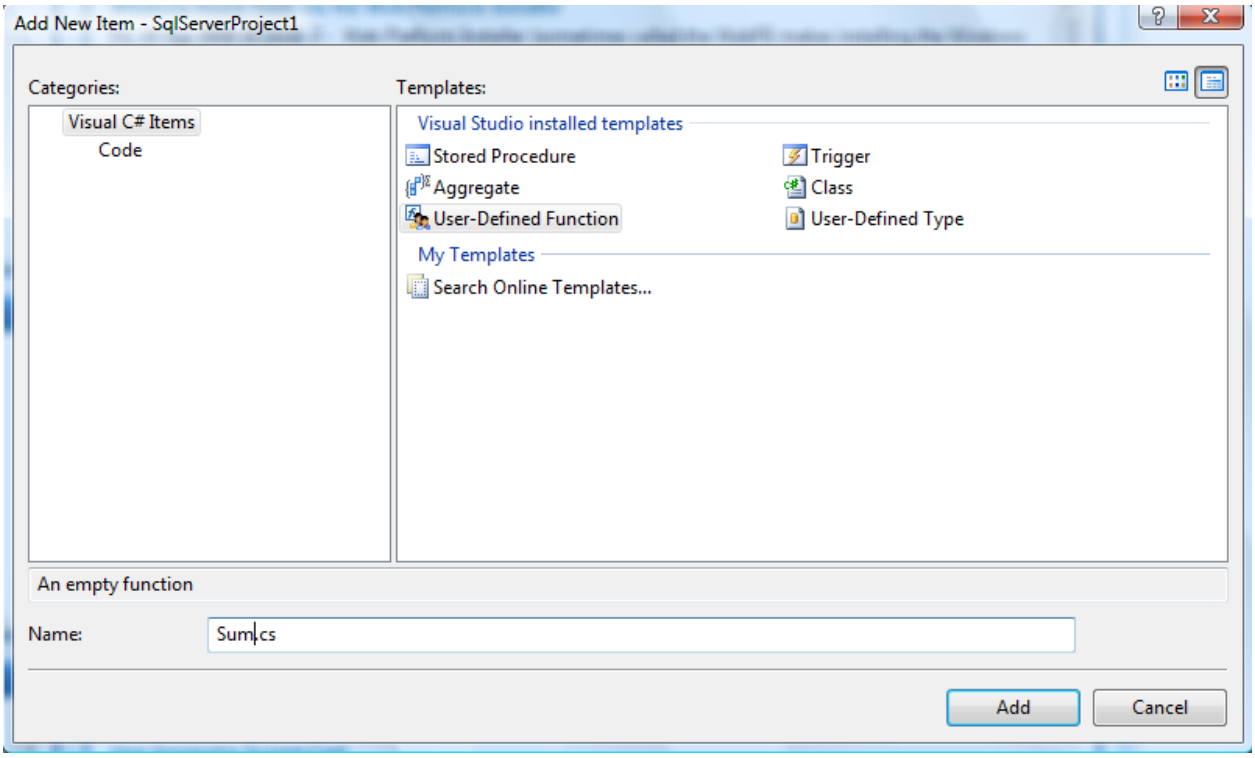


Рис. 12.3. Додавання функції у проєкт

5. Напишімо наступний код у цій функції:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;

public partial class UserDefinedFunctions
{
 [Microsoft.SqlServer.Server.SqlFunction]
 public static SqlInt32 Sum(SqlInt32 a, SqlInt32 b)
 {
        return a + b;
 }
};
```

Зверніть увагу, що використовуються не звичайні типи даних, які є в C#, а специфічні для MS SQL Server, такі типи мають префікс «**Sql**»

6. Зареєструємо збірку на сервері MS SQL Server. Для цього виконаємо команду меню **Build** – **Deploy Solution**. При цьому буде проведена компіляція збірки, а сама збірка буде поміщена на сервер в зазначену БД (див. п.3.).

7. Дозволимо виконання CLR коду на сервері:

```
sp_configure 'clr enabled', 1
RECONFIGURE;
```

8. Після цього можна викликати створену функцію:

```
SELECT dbo.Sum(10, 20)
```

Аналогічним чином можна створити і агрегатную функцію. Наприклад, агрегатна функція SUM в MS SQL Server працює тільки з числами, тому якщо виникає необхідність використовувати таку функцію для рядків (щоб набір рядків об'єднався в єдиний рядок, а значення були б розділені комами), то це можна реалізувати засобами C#.

Технології доступу до даних: LINQ і ADO.NET

Технологія ADO.NET

ADO.NET являє собою основну модель доступу до даних в платформі .Net Framework. Вона не є розширенням існуючої раніше технології ADO, а реалізує нову модель роботи з даними. Наприклад, вона передбачає від'єднану модель роботи, тобто підключення з сервером баз даних встановлюється тільки на момент виконання запиту, після чого воно розривається. Раніше підключення з БД підтримувалося протягом усього сеансу роботи.

ADO.NET включає наступні ключові компоненти:

- *наборы данных* (DataSet). Представляют собой некоторую часть реальной БД, включая в себя не только таблицы, но и связи между таблицами и ограничения;
- *провайдеры данных* (DataProvider). Благодаря наличию различных провайдеров, технология ADO.NET может работать с различными типами СКБД: MS SQL Server, MS Access, Oracle, а также с любой БД, используя технологию ODBC (Open DataBase Connectivity).

Загальна схема роботи з використанням ADO.NET:

1. Створити і встановити підключення до сервера. Наприклад, підключімося до локального сервера до БД ***AdventureWorks***:

```
string connectionString = "Data Source=(local);Initial
                Catalog=AdventureWorks; Integrated Security=SSPI;";
SqlConnection connection = new SqlConnection(connectionString);
connection.Open();
```

2. Створити команду і виконати на сервері. Наприклад, отримаймо всі записи з таблиці *Orders*:

```
SqlCommand cmd = new SqlCommand("SELECT * FROM Orders", connection);
SqlDataReader reader = cmd.ExecuteReader();
```

3. Обробити результати виконання команди. Наприклад, виведімо вміст таблиці на консоль:

```
while (reader.Read())
{
 Console.WriteLine(String.Format("{0}, {1}", reader[0], reader[1]));
}
reader.Close();
```

4. Закрити з'єднання:

```
conn.Close();
```

Технологія LINQ

LINQ (Language Integrated Query) – проєкт Microsoft з додавання синтаксису мови запитів, що нагадує SQL, в мови програмування платформи .NET Framework.

LINQ дозволяє виконувати запити до об'єктів, що знаходяться в пам'яті, в типізованої бази даних і в XML документі. Для цього використовуються відповідні технології: LINQ, DLINQ XLINQ.

Наприклад, створімо масив arr і заповнімо його числами від 0 до 10, а потім за допомогою LINQ виберімо елементи більші 4 і менші 8:

```
int[] arr = { 7, 9, 3, 4, 5, 6, 0, 8, 9, 10 };
var newArray = from i in arr
                    where i > 4 && i < 8
                    select i;
```

У результаті змінна **newArray** буде містити колекцію цілих чисел: 7, 5, 6.

Перед початком роботи LINQ з базами даних нам потрібно отримати опис об'єктів бази даних в MS Visual Studio. Існують різні інструменти для генерації сутностей для MS Visual Studio з баз даних. Наприклад, утиліта командного рядка SQLMetal, яку можна знайти в папці: C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin.

Приклад використання:

```
SQLMetal /server:.\SQL2008 /database:AdventureWorks /pluralize
 /code:AdventureWorks.cs
```

В результаті будуть згенеровані об'єкти на C#, що описують всі об'єкти БД ***AdventureWorks2008***.

Розгляньмо застосування LINQ на практиці. Перед виконанням запитів LINQ до баз даних необхідно створити контекст даних:

```
var db = new MyDataContext(@"server=.\SQLEXPRESS; database=my;
                                    integrated security=SSPI");
if (!db.DatabaseExists())
 db.CreateDatabase();
```

Розгляньмо приклади виконання стандартних операцій над даними за допомогою LINQ.

1. Вибірка одного єдиного значення:

```
var first = db.Customers.FirstOrDefault(c => c.CustID=="CHIPS");
```

2. Вибірка за умовою:

```
where, null, contains & type
var r = new string[] { "WA", "OR" };
var customers = from c in db.Customers
        where c is Customer &&
                (c.Region==null || r.Contains(c.Region))
        select c;
```

3. Операція з'єднання таблиць:

```
var labels = (from c in db.Customers join o in db.Orders
        on c.CustID equals o.CustID
        select new { name = c.ContactName,
        address = o.ShipAddress
        }).Distinct();
```

4. Виконання агрегатних функцій:

```
var totals = from c in db.Customers
        group c by c.Country into g
        select new Summary { Country = g.Key,
        CustomerCount = g.Count(),
        OrdCount = g.Sum(a=> a.Orders.Count)};
```

5. Операція вставки нового запису:

```
var customer = new Customer() {
```

```
    CustID = "CHIPS",
    CompanyName = "Mr. Chips" };
db.Customers.InsertOnSubmit(customer);
db.SubmitChanges();
```

Короткі висновки. Розглянуто нові технології роботи з даними: LINQ і ADO.NET. Продемонстровані нові можливості MS SQL Server – виконувати функції і використовувати нові типи даних, створені в MS Visual Studio для платформи .Net Framework.

Рекомендуемая литература

1. Дейт К. Дж. Введение в системы баз данных. – М.: Вильямс, 2008.
2. Дибетта П. Знакомство с Microsoft SQL Server 2005. – М.: Русская редакция, 2005.
3. Lobel L., Brust A. J., Forte S. Programming Microsoft SQL Server 2008. – Microsoft Press, 2008.
4. Уолтерс Р. Э., Коулс М., Рей Р., Феррачати Ф., Фармер Д. SQL Server 2008: ускоренный курс для профессионалов. – М.: Издательский дом «Вильямс», 2008.
5. Волоха А. В. Microsoft SQL Server 2005. Новые возможности. – СПб.: Питер, 2006.
6. Каленик А. И. Использование новых возможностей Microsoft SQL Server 2005. – М.: Русская редакция; СПб.: Питер, 2006.
7. Вьейра Р. SQL Server 2000 Программирование. Ч. 1. – М.: Изд-во «БИНОМ. Лаборатория знаний», 2004.
8. Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. – М.: Издательский дом «Вильямс», 2003.
9. Дэвидсон Л. Проектирование баз данных на SQL Server 2000. – М.: Изд-во «БИНОМ. Лаборатория знаний», 2003.
10. Жилинский А. Самоучитель Microsoft SQL Server 2008. – СПб.: БХВ-Петербург, 2009.