Міністерство освіти і науки України
Миколаївський національний університет
імені В.О.Сухомлинського

Г. С. Погромська, Н. А. Махровська

БАЗИ ДАНИХ: ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ

Навчально-методичний посібник

для студентів спеціальності
122 Комп'ютерні науки

Місто
Видавництво
2019

УДК 004.655(076.5)
ББК 32.81
П 43

РЕЦЕНЗЕНТИ:
Євдокимов В.Д. – доктор технічних наук, професор Одеського національного морського університету (м. Одеса)
Фісун М.Т. – доктор технічних наук, професор, завідувач кафедри інженерії програмного забезпечення Чорноморського національного університету імені Петра Могили (м. Миколаїв)
Борисенко В. Д. – доктор технічних наук, професор, завідувач кафедри комп'ютерної інженерії Миколаївського національного університету імені В. О. Сухомлинського (м. Миколаїв)

Рекомендовано до друку рішенням
Вченої ради Миколаївського національного університету
імені В.О.Сухомлинського (протокол № від р.)

Погромська Г.С.
П 43 Бази даних: проектування та реалізація/ Г. С. Погромська, Н.А. Махровська. – Місто: Видавництво, 2019. – 183 с.

ISBN

Розглядаються основні поняття теорії баз даних, моделі даних, архітектура і класифікація БД. Особлива увага приділяється реляційній моделі, а також процесам нормалізації. Розкриваються питання проектування та безпеки БД.

Призначений для вивчення теоретичних основ і набуття практичних навичок проектування, реалізації та експлуатації баз даних в системі управління базами даних Microsoft SQL Server версії 2008 та вище.

Розрахований на студентів та викладачів ЗВО, які спеціалізуються в галузі комп'ютерних наук, зокрема вивчають основи баз даних.

УДК 004.655(076.5)
ББК 32.81

ISBN

ЗМІСТ

СПИСОК УМОВНИХ СКРОЧЕНЬ

БД – База даних
БЗ – Багатозначна залежність
ДКНФ – Доменно-ключова нормальна форма
ЕОМ – Електронно-обчислювальна машина
ІПС – Інформаційно-пошукова система
НФ – Нормальна форма
НФБК – Нормальна форма Бойса-Кодда
ПЗ – Програмне забезпечення
СБД – Система баз даних
СУБД – Система управління базами даних
ФЗ – Функціональна залежність
ANSI/ SPARC – American National Standards Institute / Standards Planning and Requirements Committee
DML – Data Manipulation Language
IIS – Internet Information Services
OLAP – On-Line Analytical Processing
RDL –Report Definition Language
SQL – Structured Query Language
UTC – Universal Time Coordinated
XML – eXtensible Markup Language

ПЕРЕДМОВА

Пропонований навчальний посібник дозволяє вивчити теоретичні основи баз даних (БД), сформувати вміння щодо ефективного вирішення завдань адміністрування та управління даними в широко поширеній системі управління базами даних (СУБД) Microsoft SQL Server 2008, яка орієнтована на побудову корпоративної системи управління базами даних.

Компетенції, які формуються під час вивченні курсу «Організація баз даних», сприяють набуття у студентів практичних навиків щодо:

- вивчення порядку, особливості встановлення та налаштування програмного забезпечення MS SQL Server;

- створення екземпляра, користувачів та таблиць бази даних, виконання змін структури таблиць;

- виконання операцій зі створення структурних компонентів СУБД за допомогою програми командного рядку;

- застосування операцій з маніпулювання табличними даними за допомогою середи MS SQL Server;

- здійснення простих запитів за масивами таблиць бази даних MS SQL Server, виконання богатотабличних запитів щодо відбору даних.

- виконання запитів щодо групування та сортування даних, обчислення даних за допомогою вбудованих функцій;

- порядку та особливості встановлення клієнтського програмного забезпечення MS SQL Server.

У змісті навчального посібника викладені основні поняття теорії баз даних, моделі даних, архітектура і класифікація БД. Особлива увага приділяється реляційної моделі, а також процесам нормалізації. Розглядаються питання проектування та безпеки БД. Матеріал посібника включає 12 тем.

Перша тема присвячена розгляду основних понять теорії баз даних, таких як база даних, її властивості і елементи, система баз даних і її елементи, моделі даних - ієрархічною, мережевий, реляційної і на основі інвертованих списків. Також в лекції розглядаються архітектура та класифікація БД.

У другій темі наводиться огляд компонентів СУБД Microsoft SQL Server 2008 – служб, що функціонують на сервері баз даних, і додатків, що поставляються разом з даної СУБД, Висвітлюються питання конфігурації сервера і призначення системних баз даних.

Третя тема призначена для знайомства з мовою програмування Transact-SQL – процедурним розширенням мови структурованих запитів SQL. У лекції розглядаються основні типи даних, що використовуються в MS SQL Server 2008, правила оголошення змінних, а також алгоритмічні конструкції Transact-SQL і стандартні функції для обробки даних.

У четвертій темі розглядається ключовий оператор мови SQL - оператор вибірки SELECT. Наводяться приклади різних видів з'єднання таблиць при вибірці даних. Також порушуються питання аналітичної вибірки з використанням угруповання даних і агрегатних функцій.

П'ята тема присвячена допоміжним, але часто використовуваних об'єктів в базах даних: збереженим процедурам і уявленням. У лекції показані способи створення і управління цими об'єктами за допомогою команд SQL.

Шоста тема знайомить з основними механізмами безпеки в MS SQL Server 2008: обліковими записами для входу і користувачами бази даних, серверними ролями і ролями бази даних. На прикладах продемонстровано управління даними механізмами як за допомогою графічного інтерфейсу, так і за допомогою SQL команд.

У сьомій темі розглядаються структура БД в MS SQL Server 2008 (первинні файли, файлові групи, журнал), способи створення і

налаштування БД. Зачіпаються питання стиснення БД, резервного копіювання і відновлення.

Предметом восьмий теми є реляційна модель даних: розглядаються основні елементи реляційних баз даних; питання цілісності даних. Також даються визначення первинних і зовнішніх ключів.

У дев'ятій темі дається огляд основних операторів реляційної алгебри - об'єднання, перетину, віднімання, твори, вибірки, проекції, з'єднання і ділення, а також двох допоміжних - розширення і підбиття підсумків. Наводяться приклади їх реалізації на мові SQL.

У десятій темі описуються перші три нормальні форми і дається уточнення третьої нормальної форми. Наводяться приклади відносин, невідповідних нормальним формам, і демонструються способи їх нормалізації.

Одинадцята тема присвячена четвертій і п'ятій нормальним формам. У ній наводиться остаточна схема нормалізації БД та даються визначення альтернативних нормальних форм.

У дванадцятій темі, заключної, 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.

Таким чином, перша тема є вступної. У темах з другої по сьому розглядаються основні компоненти, механізми та можливості СУБД Microsoft SQL Server 2008. Теми з восьмої по одинадцяту присвячені реляційної моделі даних, реляційної алгебри і процесу нормалізації. В 12-тій темі висвітлюються питання взаємодії Microsoft SQL Server 2008 і середовища програмування Microsoft Visual Studio 2008.

Представлена послідовність тем обумовлена наступними міркуваннями. Як правило, в ході вивчення курсу лабораторні роботи

проводяться паралельно з лекціями; а для виконання робіт необхідні знання конкретної СУБД – Microsoft SQL Server 2008. Тому теми, пов'язані з Microsoft SQL Server 2008, винесені в першу половину змісту навчального посібника, а питання реляційних баз даних залишилися у другій половині. Така послідовність, на наш погляд, не заважає цілісному сприйняттю матеріалу, підвищуючи при цьому ефективність спільного опрацювання теоретичного матеріалу і лабораторних робіт.

Залежно від наявного обладнання і вимог мережного адміністрування установка MS SQL Server 2008 може бути проведена як на фізичний комп'ютер, так і на віртуальну машину. Однак при установці на віртуальну машину вимоги до обсягу оперативної пам'яті підвищуються.

Для поглибленого розуміння структури MS SQL Server і поставляються з ним додатків рекомендується встановити серверну і клієнтські частини на різні комп'ютери (фізичні або віртуальні). Такий підхід сприяє більш чіткому усвідомленню самої серверної служби, яка виконує всі SQL-запити, і клієнтських додатків, що здійснюють підключення до сервера, його налаштування і передачу йому запитів.

Для успішного опанування матеріалу посібника бажані (але не необов'язкові) базові знання однієї з мов програмування (Паскаль, C++, Java).

ТЕМА 1. ВВЕДЕННЯ В ТЕОРІЮ БАЗ ДАНИХ

Тема присвячена розгляду основних понять теорії баз даних і основних моделей даних, на яких будуються сучасні БД.

Мета: виявити основні структурні елементи баз даних і основні принципи, використовувані при їх розробці.

Зміст теми 1:

1.1. Основні поняття

1.2. Система БД

1.3. Організація даних в БД

1.4. Види моделей даних

1.5. Архітектура БД

1.6. Класифікація БД

1.1. Основні поняття

Існують різні визначення поняття база даних (БД). Найчастіше під БД розуміється пойменована сукупність структурованих даних, які стосуються певної предметної області. Однак у 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 та обробка даних.

Найбільш поширеним типом БД є реляційні бази даних. Розглянемо основні структурні елементи реляційної БД:

1. *Поле* – елементарна одиниця організації даних. Для опису поля використовують характеристики: ім'я, тип, довжина, точність і т.д. Відповідає стовпцю в таблиці.

2. *Запис* – сукупність логічно пов'язаних полів. Відповідає рядку в таблиці.

3. Власне *таблиця (відношення*).

1.2. Система БД

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

Розрізняють користувацькі і багатокористувацькі системи.

Однокористувацька система (Single-user system) – це система, в якій в один і той же 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 декілька користувачів. Основне завдання такої системи – дозволити користувачеві працювати з БД як з однокористувацькою.

Зазвичай в СБД виділяють чотири основні елементи: дані, апаратне забезпечення, програмне забезпечення (ПЗ), користувачі.

Спрощена схема СБД представлена на рис. 1.1.

Дані

Дані в БД можна охарактеризувати як інтегровані та загальні. *Інтегровані* дані можна представити як об'єднання декількох окремих файлів, які повністю або частково не перекриваються. У разі загальних даних окремі області даних можна використовувати декільком різним користувачам.

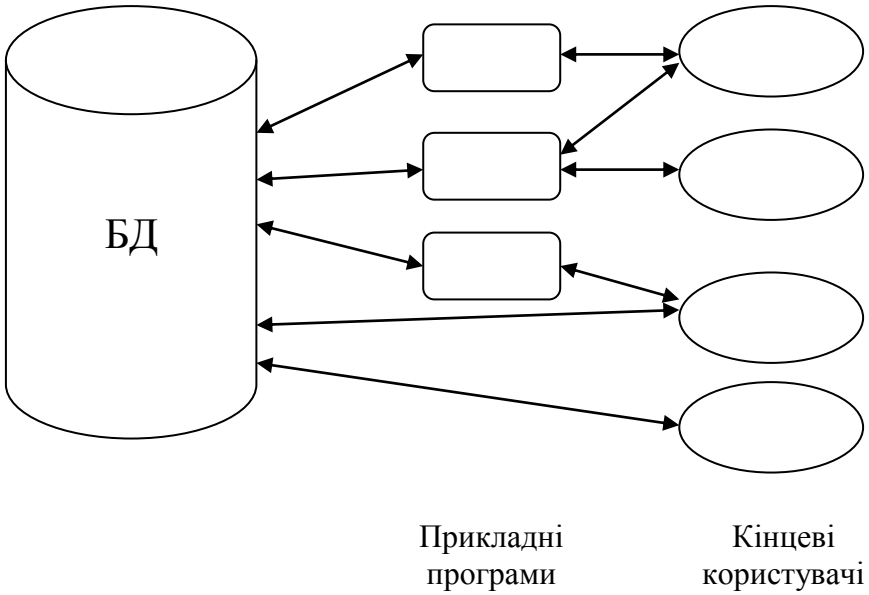


Рис. 1.1. Елементи системи баз даних

Апаратне забезпечення

До нього належать:

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

Програмне забезпечення

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

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

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

Користувачі

Розрізняють три групи користувачів СБД:

1. *Прикладні програмісти*. Для цілей розробки прикладних програм, які використовують бази даних, можна застосовувати 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 СУБД (зазвичай це оператори SQL).

2. *Кінцеві (рядові) користувачі*. Кінцевий користувач може отримувати доступ до бази даних, застосовуючи одне з інтерактивних додатків. Багато СУБД надають не тільки засоби для виконання запитів SQL, але й графічні утиліти, що дозволяють створювати запити без знання SQL.

3. *Адміністратори БД*. Займаються управлінням роботи сервера БД.

1.3. Організація даних в БД

У базі даних виділяють такі елементи: дані, об’єкти, зв’язки, властивості.

Дані

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

Вхідні дані – це інформація, яка 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 і 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) якої групи належить даний студент;

2) які студенти входять в дану групу.

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

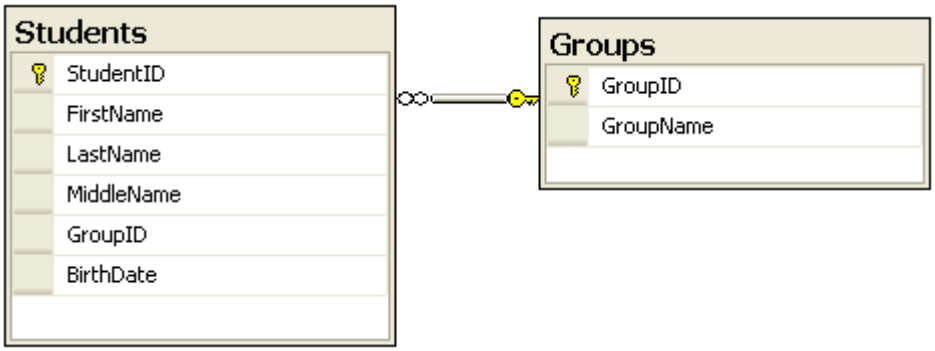


Рис. 1.2. Зв’язок між таблицями Students та Groups

У схемі можуть бути зв’язку, що вказують на один і той же тип об’єкта. Наприклад, викладач є наставником молодого викладача. У конкретному наборі об’єктів може бути будь-яка кількість зв’язків. Між двома таблицями може бути більше одного 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.

1.4. Види моделей даних

Ядром будь БД є модель даних. *Модель даних* – це сукупність структури даних та 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 відношеннями утворюють орієнтований граф. Основними поняттями ієрархічній моделі даних є: рівень, вузол (або елемент) і зв'язок. Така модель даних має такі властивості:

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

П р и к л а д и .

1) Файлова структура організації інформації.

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

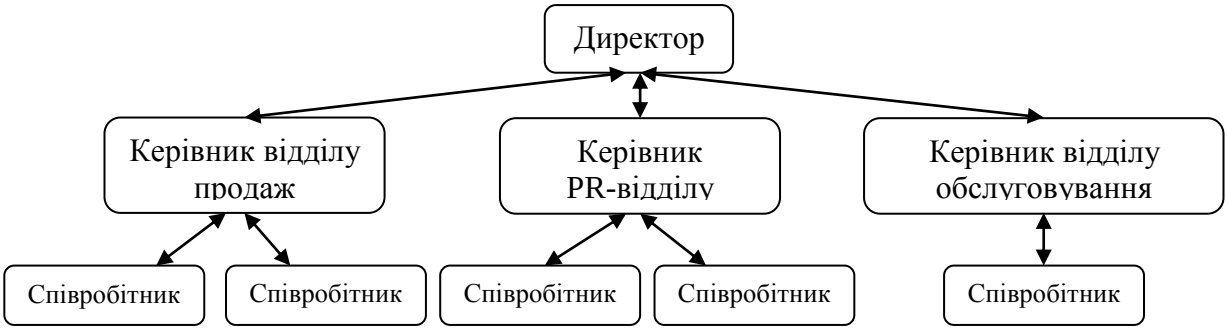


Рис. 1.3. Ієрархічна структура даних

Переваги:

1. Простота.
2. Мінімальні витрати 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 встановлений між об’єктами, що знаходяться через рівень;
- зв’язок між вузлами першого рівня не визначається.

П р и к л а д . Розглянемо роботу над проектами: можна виділити три види об’єктів – співробітники, проекти, замовники (рис.1.4).

Переваги:

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

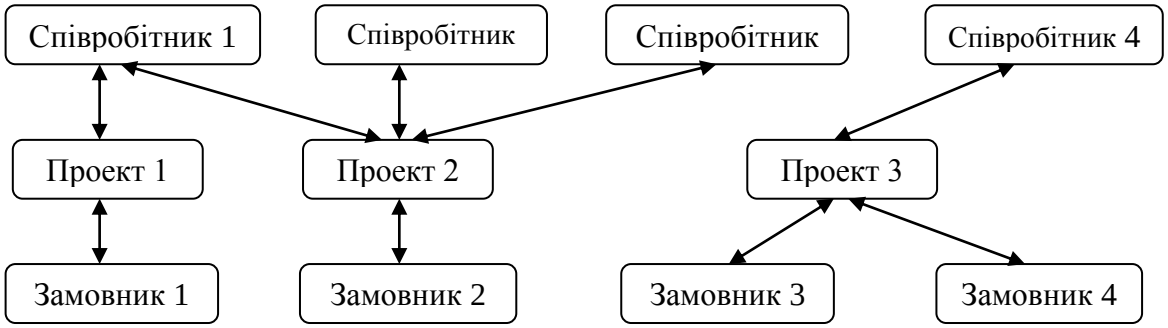


Рис. 1.4. Мережева структура даних

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;
- користувачеві надаються оператори, які генерують нові таблиці зі старих (для вибірки даних).

П р и к л а д . Розглянемо відносини *Студенти* та *Групи*:

Students:

StudentID	LastName	FirstName	MiddleName	GroupID
1	Давидов	Петро	Сергійович	1
2	Павлов	Богдан	Іванович	2
4	Кузьменко	Дарина	Петрівна	1

Groups:

GroupID	Supervisor
1	Сидоров С.М.
2	Курилов 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 в темах 10-11).

3. Незалежність даних. При зміні БД, її структури необхідні бувають лише мінімальні зміни прикладних програм.

Недоліки:

1. Низька швидкість, тому потрібні операції з'єднання.

2. Велика витрата пам'яті завдяки 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 декілька основних файлів, що мають єдину наскрізну нумерацію.

П р и к л а д . Розглянемо об'єкти Співробітники та 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, у ньому кожному значенню зіставляється список номерів (індексів).

П р и к л а д : інвертований список *Посада*:

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

Список зв’язків складається лише за основними стовпцями.

`Приклад`: розглянемо 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 ключовими словами, які визначають тематику пошуку.

Так як недоліки реляційної моделі даних компенсуються зростанням швидкодії і ресурсів сучасних комп’ютерів, то в даний час саме такі моделі набули найбільшого поширення.

1.5. Архітектура БД

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

Кожна система баз даних не зобов'язана відповідати цьому визначенню, наприклад, «малі» системи, вельми ймовірно, не будуть підтримувати всі функції запропонованої архітектури. Проте, розглянута архітектура з достатньою точністю описує більшість систем (і не тільки реляційні).

Прийнято виділяти три рівня в архітектурі СУБД.

1. *Внутрішній рівень* (*фізичний*) найбільш близький до фізичного сховища інформації, тобто пов'язаний зі способами зберігання інформації на фізичних пристроях. Внутрішній рівень відображає фізичні елементи для зберігання інформації. Він являє собою нескінчений адресний простір, з якого інформація проектується на зовнішні носії.

2. *Зовнішній рівень* (*користувацький*) 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 СУБД. Для прикладного програміста – це мова програмування, а для кінцевого користувача – це мова, заснований на меню, формах і т.д. Але всі ці мови включають мову даних, вбудовану в СУБД. Для кожного окремого користувача може бути цікава деяка окрема частина БД. Такі частини, з якими працює користувач, називаються *зовнішнім поданням.*

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

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

Детальна схема архітектури системи баз даних представлена на рисунку 1.5.

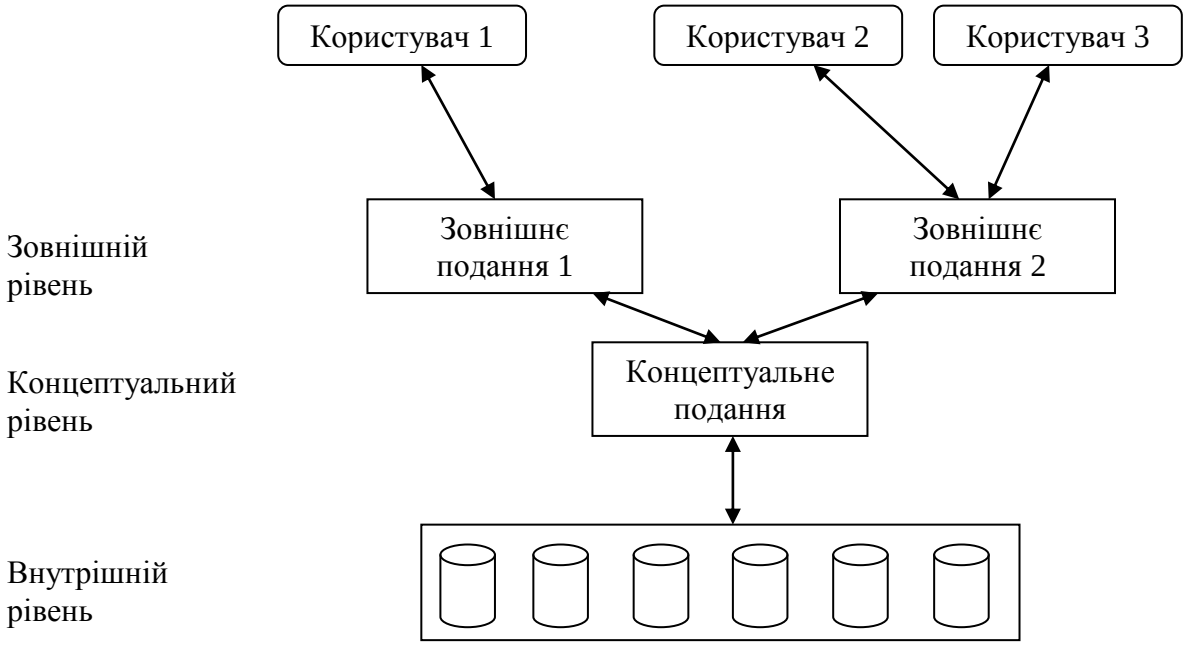


Рис. 1.5. Рівні архітектури систем баз даних

1.6. Класифікація БД

Бази даних класифікують за різними ознаками, розглянемо 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 серверу БД, який окрім зберігання здійснює обробку даних. Систему БД можна розглядати як систему, що складається з двох частин: сервер і набір клієнтів. Сервером БД називається власне СУБД, а клієнтами – різні додатки, які виконуються над СУБД.

4. *За вмістом*.

- Географічні.
- Історичні.
- Наукові.
- Мультимедійні і т.ін.

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

ТЕМА 2. КОМПОНЕНТИ MICROSOFT SQL SERVER 2008

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

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

Зміст теми 2:

2.1. Microsoft SQL Server 2008

2.2. Серверна частина системи

2.3. Клієнтська частина системи

2.4. Конфігурація MS SQL Server

2.5. Системні бази даних

2.1. Microsoft SQL Server 2008

Microsoft SQL Server – це система управління клієнт-серверними базами даних, орієнтована на роботу під управлінням 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 у 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 по аналітиці та створенню звітів.
- *Workgroup Edition*. Підходить для установки в філіях компанії і надає засоби керування 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.

2.2. Серверна частина системи

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

- Служба *SQL Server (MS SQL Server*) є ядром цієї СУБД, 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 темі 3), запити і збережені процедури (розглядаються в темі 4 і 5 відповідно), які вказуються користувачами;
- забезпечує безпеку системи (наприклад, здійснює перевірку облікових записів користувачів; система безпеки розглядається в темі 6);
- відповідає за узгодженість і цілісність даних, запобігаючи логічним проблемам.

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 виниклу проблему і т.п.).
- 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* замінює службу *DTS SQL Server 2000* і дозволяє виконувати наступне:
  - відстежувати 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 сервера OLAP2, дозволяє створювати аналітичні додатки з мільйонами рядків даних і тисячами користувачів.

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

- Служба Reporting Services – ця служба являє 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.

2.3. 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 керування роботою сервера і створення логічної структури баз даних, підтримуваних ним. Для розробки клієнтського застосування можуть бути використані і різні засоби розробки додатків, наприклад, середовища візуального програмування Visual Studio .Net 2003-2008, 2010, 2012, Visual Basic, Delphi та ін.

До стандартних утиліт адміністрування відносяться наступні додатки.

SQL Server Configuration Manager

Надає наступні можливості:

- 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.
- Визначення параметрів мережевих бібліотек, які забезпечують взаємодію з MS SQL Server. Можна вибрати один або відразу декілька методів доступу до сервера:

- *іменовані канали (Named Pipes)* – технологія схожа на використання сокетів, застосовується в разі недоступності протоколів TCP / IP;
- *стек протоколів TCP/IP* (використовується за умовчанням) – підходить для використання через мережу Інтернет;
- *колективна 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.

- Конфігурування мережевих бібліотек клієнта, які використовуються для доступу до MS SQL Server. Після налаштування 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).

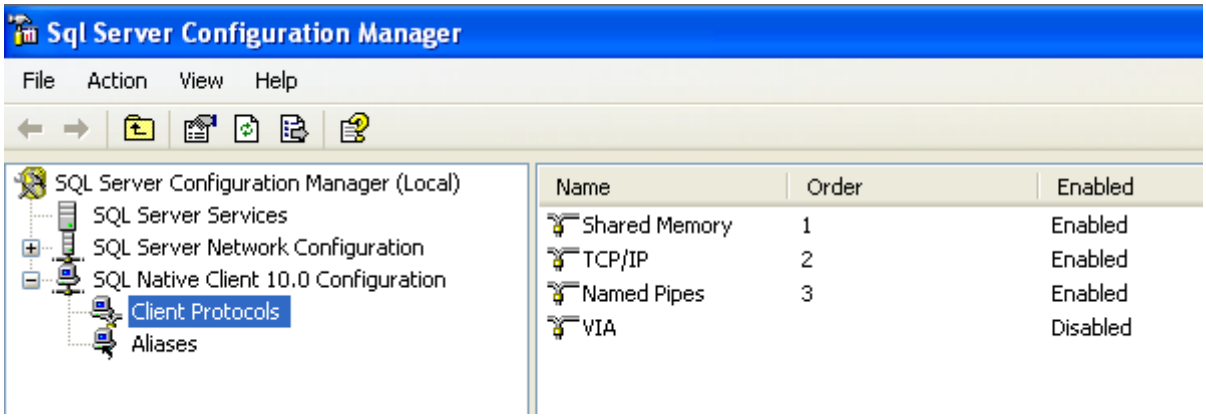


Рис. 2.1. Вікно утиліти SQL Server Configuration Manager

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*.

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

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;
- управляти 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 підключиться: *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 аутентифікації, можна вибрати *Windows Authentication* або *SQL Server Authentication*. Спосіб *Windows Authentication* використовує обліковий запис, під яким поточний

користувач здійснив вхід в ОС 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).

Виконаємо наступний запит (мова складання 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 синтаксичної записи можна скористатися командою **Query – Parse (Ctrl+F5)**, при цьому сам клопотання не буде виконаний.

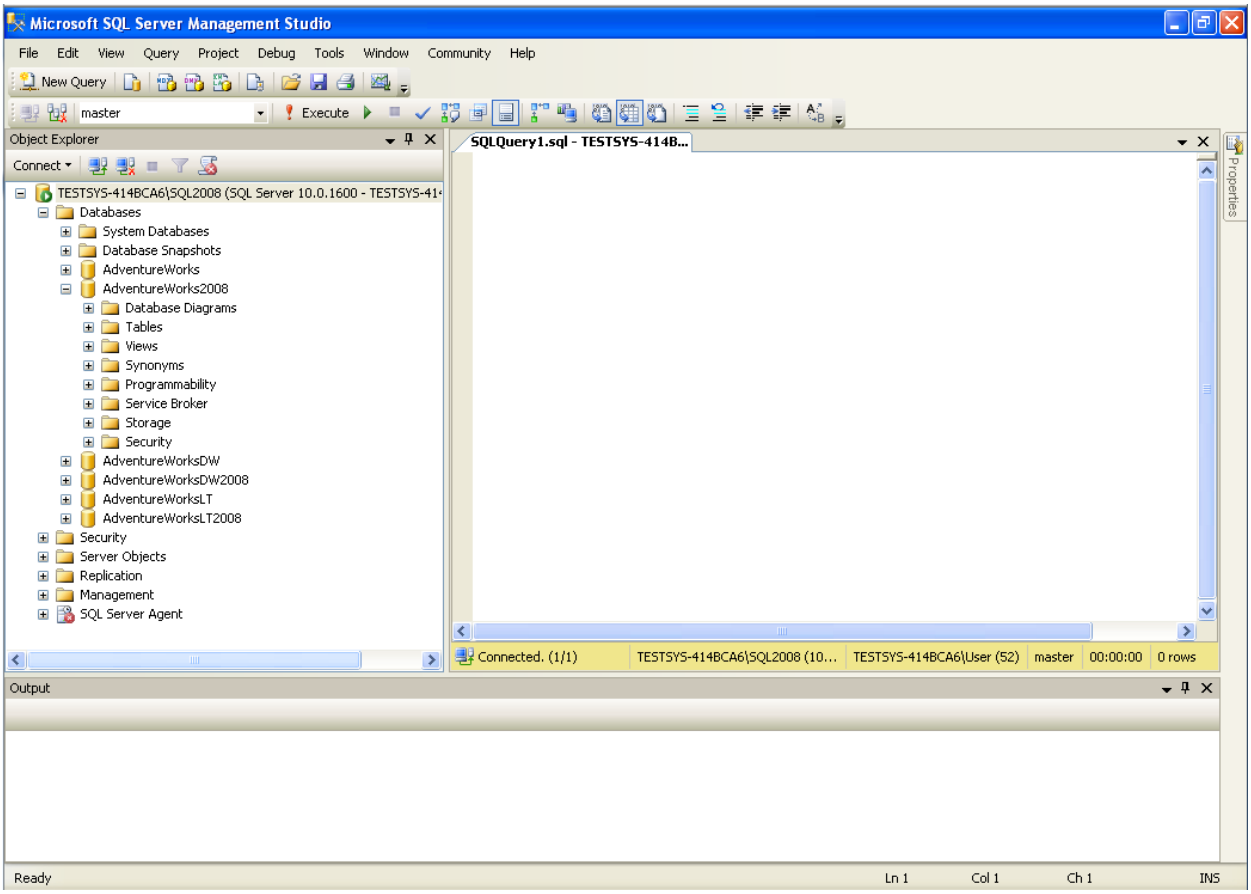


Рис. 2.3. Головне вікно додаткка *Management Studio* з вікном *Редактора запитиів*

Після цього буде доступне наступне:

- заголовок, в якому вказується логічне ім’я сервера, поточна база даних та ім’я користувача, який встановив з’єднання;
- область запиту, використовувана для введення запитів, переданих 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 / рядка / стовпці.
- *Results to File* – аналогічно *Results in Text*, тільки висновок здійснюється не на екран, а у 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 в *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 у файлі на зовнішньому носії командою **File – Save**.

Об'єкт Explorer

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

Випадає 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 випадаючий список, або за допомогою команди SQL:

```
USE AdventureWorks2008
```

Reporting Services Configuration

Використовується для конфігурації служб звітів. MS SQL Server 2008 включає в себе вбудований веб-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 форматі XML[3].

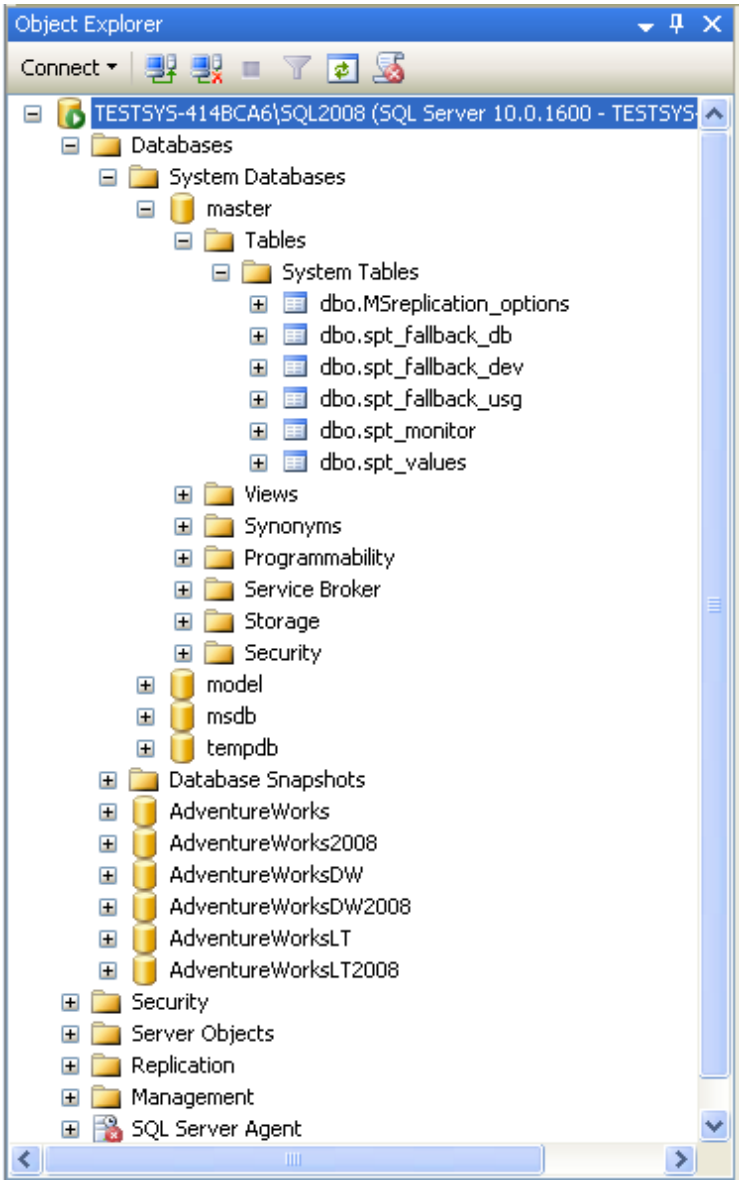


Рис. 2.4. Панель *Object Explorer*

Bulk Copy Program

Утиліта командного рядка, призначена для перенесення форматованих даних великого обсягу в MS SQL Server або з нього.

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

Наприклад, відформатовані дані можуть бути автоматично перенесені зі звичайного текстового файлу в таблицю MS SQL Server.

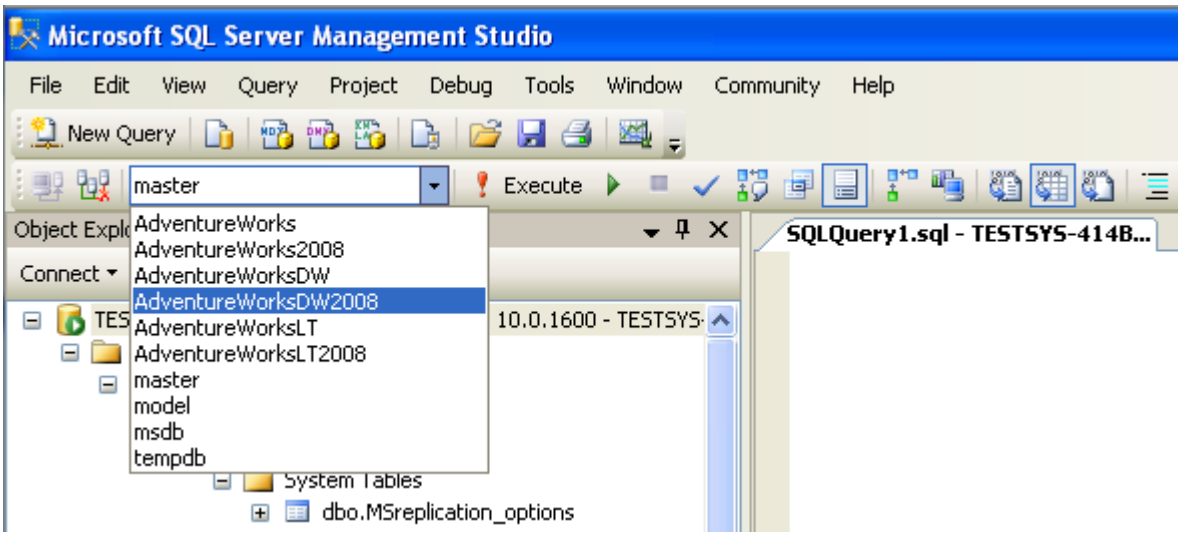


Рис. 2.5. Вікно вибору поточної бази даних

SQL Server Profiler

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

Утиліта SQLCMD

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

SQL Server Integration Services (SSIS)

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

SQL Server Business Intelligence Development Studio

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

Представляє особливу версію Visual Studio і дозволяє створювати пакети для *Integration Services*, звіти для *Reporting Services* і працювати з проектами служби *Analysis Services*.

2.4. Конфігурація MS SQL Server

Конфігурування роботи служби *MS SQL Server* може бути виконано або спеціальною збереженою процедурою, що виконується в утиліті *Management Studio*, або графічним способом засобами цієї ж утиліти. Вибір способу не має значення, тому графічний спосіб здійснює доступ до системних даними за допомогою цієї ж процедури, що зберігається, 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 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 сервера .

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

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

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

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 управляти правами на виконання дій в MS SQL Server.

2.5. Системні бази даних

База даних зазвичай являє собою сукупність таблиць та інших об’єктів, таких як представлення, збережені процедури та 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 БД *sys.databases* БД ***master***. Також всі системні збережені процедури знаходяться в цій БД. Оскільки практично все, 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 нові бази, які створюються, 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 у випадку, 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 щоразу заново при запуску MS SQL Server.

Стислі підсумки. Розглянуто служби MS SQL Server 2008, що забезпечують як базову функціональність, так і додаткову: засоби звітів, інтеграції і т.д. Дана характеристика системних баз даних і основне їх призначення.

ТЕМА 3. ЗАГАЛЬНІ ВІДМОСТІ ПРО TRANSACT-SQL

У темі 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.

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

Зміст теми 3:

3.1. Загальні відомості про Transact-SQL

3.2. Типи даних

3.3. Змінні в Transact-SQL

3.4. Управляючі конструкції Transact-SQL

3.5. Коментарі

3.6. Функції Transact-SQL

3.7. Налагодження коду в Management Studio

3.1. Загальні відомості про Transact-SQL

У 1970-х роках була розроблена спеціальна мова SEQUEL (Structured English Query Language), яка пізніше була перейменована 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 розроблялася як непроцедурна мова, яка буде доступною 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 СУБД використовується своє розширення цієї мови. Основною причиною використання розширень SQL є потреба застосовувати SQL не тільки як мови запитів, але і як складової мови програмування, для цього, як правило, вводяться так звані *збережені*

процедури. СУБД Microsoft SQL Server використовує своє розширення мови, яке називається Transact-SQL і включає в себе наступне:

- керуючі конструкції;
- локальні змінні;
- додаткові функції (математичні, строкові та інші);
- підтримка аутентифікації Windows.

3.2. Типи даних

MS SQL Server підтримує всі основні прості типи даних, що використовуються в сучасних мовах програмування. У версії MS SQL Server 2008 було додано кілька 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 байт). 64-разрядне ціле число, що дозволяє зберігати числа від –263 до +263–1.
- **Int** (4 байта). Діапазон значень від –2 147 483 648 до +2 147 483 647.
- **SmallInt** (2 байта). Діапазон значень від –32768 до +32767.
- **TinyInt** (1 байт). Діапазон значень від 0 до 255.

2. Числа з фіксованою комою:

- **Decimal** або **Numeric**. Діапазон значень від –1038–1 до +1038–1.

- **Money** (8 байт). Грошовий формат, діапазон значень від –263 до +263 з чотирма знаками після коми.
- **SmallMoney** (4 байта). Грошовий формат, діапазон значень від – 214748,3648 до +214748,3647.

3. Числа с плаваючою комою:

- **Float**. Діапазон значень від –1,79E +308 до +1,79E +308.

4. Дата та час:

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

5. Символьні рядки:

- **Char**. Рядок фіксованої довжини. Максимальна довжина рядка 8000 символів.
- **VarChar**. Рядок змінної довжини. Максимальна довжина рядка 8000, але при застосуванні ключового слова «max» може зберігати до 231 байт.
- **Text**. Застосовувався для зберігання великих рядків, зараз рекомендується використання *varchar(max)* замість *text*. Залишений для забезпечення сумісності.
- **Nchar**. Рядок фіксованої довжини в Юнікоді. Максимальна довжина рядка 4000 символів.

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

- **NvarChar**. Рядок змінної довжини в Юнікоді. Максимальна довжина рядка 4000 символів, але при використанні ключового слова «max» може зберігати до 231 байт.
- **Ntext**. Аналогічний типу *text*, але призначений для роботи з Юнікодом. Зараз рекомендується використовувати *nvarchar(max)*. Залишений для забезпечення сумісності.

6. Двійкові дані:

- **Binary**. Дозволяє зберігати двійкові дані розміром до 8000 байт.
- **VarBinary**. Тип даних змінної довжини, дозволяє зберігати до 8000 байт, але при використанні ключового слова «max» до 231 байт.
- **Image**. Використовувався для зберігання великих обсягів даних, зараз рекомендується використовувати *varbinary(max)*. Залишений для забезпечення сумісності.

7. Інші типи даних:

- **Table**. Особливий тип даних, який використовується в основному для тимчасового зберігання таблиць і для передачі в якості параметра у функції.
- **HierarchyID**. Використовується для подання положення в ієрархічній структурі.
- **Sql_variant**. Тип даних, який зберігає значення різних типів даних, які підтримуються MS SQL Server.
- **XML**. Дозволяє зберегти XML-дані.
- **Cursor** (1 байт). Тип даних для змінних і вихідних параметрів збережених процедур, які містять посилання на курсор.
- **Timestamp / rowversion** (8 байт). Тип даних, який представляє собою автоматично сформовані унікальні двійкові числа

в базі даних. Значення даного типу генерується БД автоматично при вставці або зміни запису.

- **UniqueIdentifier** (16 байт). Представляє собою GUID (Special Globally Unique Identifier). Гарантується унікальність даного значення.

3.3. Змінні в 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 вводить ряд обмежень на порядок іменування об’єктів:

- перший символ імені повинен бути одним із символів латинського алфавіту (A-Z, a-z), або символом @, # або _, тобто не допускається використання цифр і 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 прогалин, дужок і таких символів, як ~, !, %, ^, & та ін.
- ім’я об’єкта не повинно збігатися з зарезервованим словом і з іменем вже існуючого 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 регістра 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 декількох змінних:

```
DECLARE @temp int, @count float
```

Присвоєння значення

Синтаксис команди:

```
SET @Ім'яЗмінної = Вираз
```

Приклад присвоєння значення:

```
SET @sum = 0
```

З версії MS SQL Server 2008 з'явилися наступні нововведення.

- Ініціалізувати змінну стало можна відразу при оголошенні, наприклад:

  ```
  DECLARE @iCounter int = 0
  ```

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

3.4. Управляючі 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 в мовах програмування. В MS SQL Server 2008 оператор CASE має два можливих варіанти застосування.

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 'Четыре'
ELSE 'Другое'
END
FROM Sales.SalesOrderHeader;
```

Результат запиту поданий на рисунку 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 пошука.

Приклад:

```
SELECT TOP 10 SalesOrderID % 10 AS
'OrderLastDigit',
ProductID % 10 AS 'ProductLastDigit',
"How Close?" = CASE
WHEN (SalesOrderID % 10) < 3 THEN 'Меньше трёх'
WHEN ProductID = 6 THEN 'ProductID равен 6'
```

```
WHEN ABS(SalesOrderID % 10 - ProductID) <= 1
THEN ' В пределах одного'
ELSE 'Больше одного'
END
FROM Sales.SalesOrderDetail
ORDER BY SalesOrderID DESC;
```

Результат виконання запиту поданий на рисунку 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 застосування оператора CASE

Циклічна конструкція

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

```
WHILE умова
Оператор
[BREAK | CONTINUE]
```

Зауваження. Вертикальна риска в описі синтаксису означає «або», тобто в даному прикладі може бути вказано 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 однієї 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* в MS SQL Server працює також як і в інших мовах програмування. Використовується наступний синтаксис:

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

В блоці 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

управління в блок CATCH дані 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.

3.5. 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: /* ... */.

3.6. 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, строкові, для роботи з датами, конфігураційні і системні. Розглянемо кожну категорію.

Математичні функції

Більшість математичних функцій повертають результат того ж типу, що й вихідні значення. Наприклад, при перекладі величини кута з градусів в радіани в разі *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 цілочисельні, так і нецілочисельне величини.

IsNumeric (вираз) – перевірка, чи має вказаний 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 зазначеного значення до найближчого мінімального цілого числа.

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

Power (вираз, ступінь) – підведення в ступінь виразу. 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 прогалини 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 з 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 рядка1, починаючи з зазначеного, і замінює їх новим рядком2.

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

Reverse (рядок) – повертає рядок, символи якого записані в зворотному порядку по відношенню до вихідного 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 частину 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 вказаного в першому параметрі.

DateDiff (тип, початок, закінчення) – 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, версії 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 значення, 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 минув 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, кожний 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 вибірка даних з таблиці *Product* дозволена.

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.

3.7. Налагодження коду в Management Studio

В 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 останову, 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 рисунку 3.3.

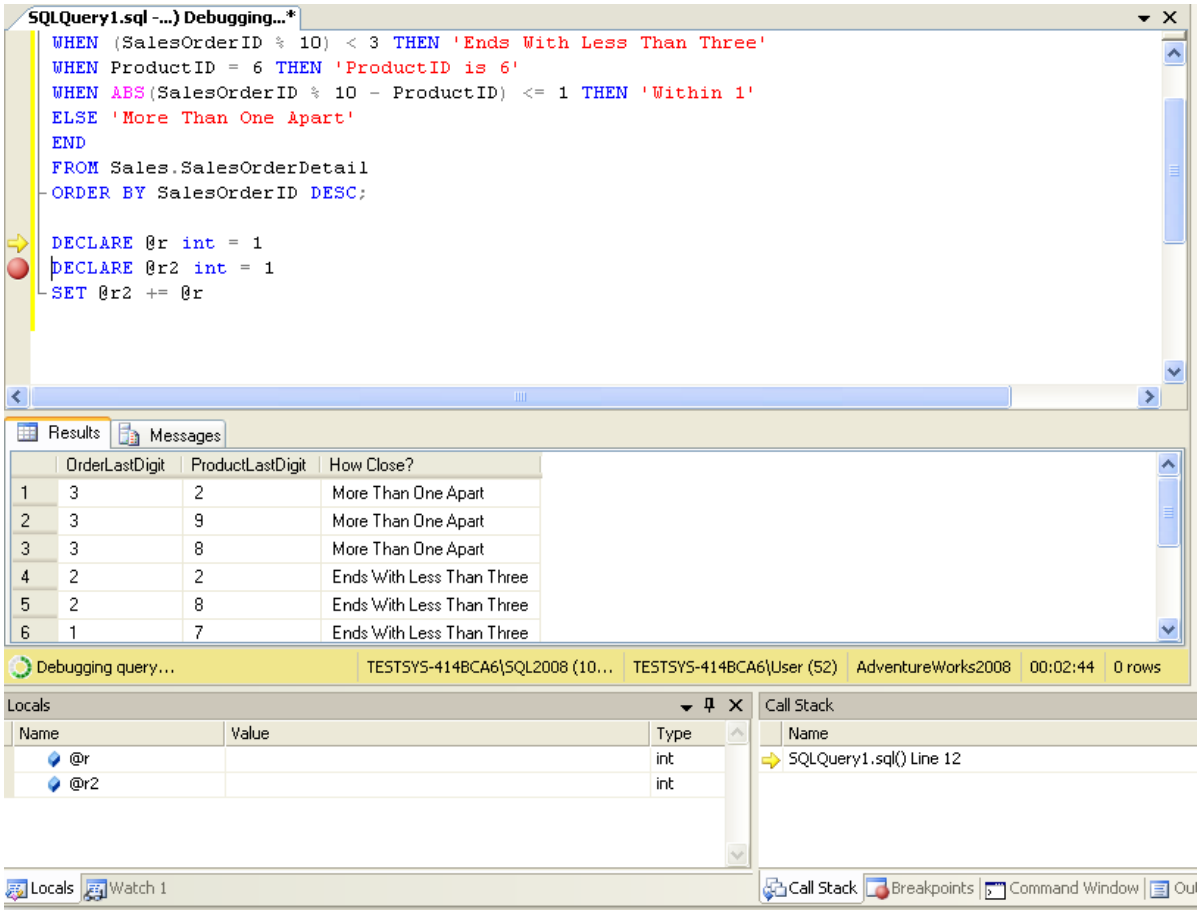


Рис. 3.3. Вікно налагодження* в Management Studio

*Жовта стрілка вказує на поточний виконуваний рядок; червоне коло – точка зупинки

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

ТЕМА 4. ВИБІРКА ДАНИХ

У темі розглядається оператор вибірки SELECT. Наводяться приклади різних видів з'єднань таблиць під час вибірки даних. Окреслюються питання аналітичної вибірки із застосуванням групування даних та агрегатних функцій.

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

Зміст теми 4:

4.1. Проста вибірка даних

4.2. Вибірка даних з декількох таблиць

4.3. Аналітична вибірка даних

4.4. Підзапити

4.5. Групування записів

Метою будь-якої СУБД є здійснення операцій над даними: введення, зміна, видалення і вибірка. При цьому вибірка даних є тим аспектом управління даними, що найчастише використовується.

Вибірка здійснюється за допомогою однієї єдиної команди SELECT, що є частиною мови DML[6] (Data Manipulation Language – мова управління даними).

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

Команда SELECT складається з семи основних частин: списку вибірки і розділів FROM, WHERE, GROUP BY, HAVING, ORDER BY і COMPUTE – обов'язковим є лише список вибірки, при використанні

[6] DML – это семейство компьютерных языков, используемых в компьютерных программах или пользователями баз данных для получения, вставки, удаления или изменения данных в базах данных. SQL является самым известным и используемым представителем языков данного семейства.

інших частин необхідно їх застосовувати в тому порядку, в якому вони наведені вище.

4.1. Проста вибірка даних

Для простої вібірки даних застосовується скорочений с и н т а к с и с 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* дозволяє обмежити число записів, які повертаються за допомогою логічних операторів.

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

Використання ключового слова *TOP n* задає виведення не всіх записів підсумкового набору, а тільки *n* перших. Можна вибирати не фіксовану кількість записів, а певний відсоток від всіх рядків – для цього вказується ключове слово *PERCENT*.

Список вибірки

Список вибірки може містити наступні один або кілька елементів:

```
* | Им'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 у квадратних дужках.

Наприклад, для отримання списку країн із зазначенням їх коду та останньої дати зміни записи з таблиці *CountryRegion* бази данних ***AdventureWorks2008*** слід обрати поля *CountryRegionCode, Name, ModifiedDate*:

```
SELECT CountryRegionCode AS 'Код',
[Name] AS 'Страна',
ModifiedDate AS 'Дата изменения'
FROM Person.CountryRegion;
```

При цьому БД ***AdventureWorks*** повинна бути поточної. 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 [Сотрудник],
 ContactID
FROM Person.Contact
```

Зручність зчитування одержуваного набору даних може бути підвищена шляхом його сортування в зростаючому або спадному порядку. Сортування можливе по імені поля (навіть якщо воно і не вказано в списку вибірки), за псевдонімом або за позицією в списку вибірки, які вказуються в розділі `ORDER BY ИмʼяПоля [,…n] [ASC | DESC]`.

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

Наприклад. 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

Умова 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 невідомий точний збіг.

В шаблоні можна використовувати такі універсальні символи:

- **%** – мається на увазі будь-який рядок, що складається з 0 і більше символів;
- **_** – рівно один символ;
- **[ ]** – будь-який символ із заданої множини (наприклад, [adfh]) або диапазону (наприклад, [0-9]),
- **[^]** – будь-який символ, що не потрапляє в заданий діапазон або безліч.

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 із США та 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;
```

4.2. Вибірка даних з декількох 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 таблиці, що з'єднуються і метод з'єднання, і *ON*, яке вказує загальні для таблиць поля.

Внутрішнє з'єднання

При такому виді 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 зв'язувальній таблиці, в результат не включаються.

Наприклад, в таблиці *Product* нормалізованої бази даних ***AdventureWorks*** зберігається 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 решті випадків перед зазначенням поля використовується ім'я таблиці *Product* і *ProductSubcategory* для вирішення конфліктів, тому що в обох таблицях присутні поля *Name*.

Зовнішнє з'єднання

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

При *лівому з'єднанні* в результат будуть включені всі записи лівої таблиці (ім'я якої розташовано зліва від *JOIN*), незалежно від того, є

для них відповідний запис в правій таблиці (ім'я таблиці розташоване праворуч від *JOIN*) чи ні.

Н а п р и к л а д , 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.

При *правому з'єднанні* (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 співпадаючих значень полів, 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.

Н а п р и к л а д , 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*.

4.3. Аналітична вибірка даних

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

- *Avg ([all | distinct] выраз)* – середнє арифметичне всіх значень.

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

Ключове слово **all** передбачає 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.

4.4. 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 вили 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, який 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 у 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 товара 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);
```

4.5. Групування записів

Для групування записів за полями або 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
[Кількість]
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;
```

При групуванні записів допускається також використання розділу *WHERE*, в цьому випадку групуються записи, що задовольняють цій умові.

Розділ *WHERE* дозволяє визначити, які записи повинні піддатися групуванню, 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;
```

Стислі підсумки. 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*.

ТЕМА 5. СТРУКТУРА БАЗ ДАНИХ В MS SQL SERVER

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

Мета: познайомити зі структурою бази даних в MS SQL Server 2008 і навчити створювати бази даних і управляти ними.

Зміст теми 5:

5.1. Створення та налагодження БД

5.2. Зміна бази даних

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

5.1. Створення та налагодження БД

СУБД MS SQL Server пропонує два можливих варіантів створення бази даних:

1. Використання графічного інтерфейсу *Management Studio.*
2. Використання команд SQL.

Створення бази даних – це процес вказування 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 вторинного файлу.

3. *Файли журналів*. Як правило, використовується розширення LDF. У кожної 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, що створена користувачем в БД.

Файли журналів не входять в жодну 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 призначені ролі *sysadmin* і *dbcreator*.

Створення бази даних здійснюється за допомогою команди:

```
CREATE DATABASE ім'я_бази_даних
[ON [PRIMARY] (NAME = 'логічне_ім'я_файла',
FILENAME = 'фізичне_ім'я_файла'
[, SIZE = розмір]
[, MAXSIXE = {максимальний_розмір | UNLIMITED}
]
```

```
    [, FILEGROWTH = крок_приросту_розміра [Mb | Kb
| %] )
    [ {FILEGROUP ім'я_файлової_групи} ]
    [, ...n ]
    [LOG ON (NAME = 'логічне_ім'я_файла',
    FILENAME = 'фізичне_ім'я_файла'
    [, SIZE = розмір]
    [, MAXSIXE = {максимальний_розмір | UNLIMITED}
]
    [, FILEGROWTH = крок_приросту_розміра [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, визначеним в параметрі *FILENAME*;
- *FILENAME* – вказує повний шлях та 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 вказаний, то встановлюється значення *UNLIMITED*, що дозволяє збільшувати файлам 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, файлах і групах файлів використовуються наступні збережені процедури:

- **sp_helpdb** *[база_данних]* – інформація про базу даних та її налагодженнях. 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 є 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]
```

5.2. Зміна бази даних

Видалення бази даних відбувається 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, в такий спосіб:

```
sp_changedbowner [ [@loginname=]
'ім'я_користувача
```

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 існуючими 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 <вказівка_на_файл> [TO FILEGROUP
найменування]
| ADD LOG FILE < вказівка_на_файл>
| REMOVE FILE логічне_ім'я_файла
| ADD FILEGROUP ім'я_групи
| REMOVE FILEGROUP ім'я_групи
| MODIFY FILE < вказівка_на_файл>
| MODIFY FILEGROUP ім'я_групи властивість_групи
}
где <вказівка_на_файл> =
(NAME = логічне_ім'я_файла',
FILENAME = 'фізичне_ім'я_файла'
```

```
    [, SIZE = розмір]
    [, MAXSIXE = {максимальний_розмір | UNLIMITED}
]
    [, FILEGROWTH = крок_приросту_розміра [Mb | Kb
| %] )
```

Дана команда дозволяє додавати файл в існуючу файлову групу, видаляти файли (при цьому видаляється і фізичний файл), додавати і видаляти файлові групи, змінювати фізичні параметри вже існуючих файлів, а також змінювати властивості файлових груп: *READONLY, READWRITE, DEFAULT* (при визначенні цієї властивості, в цю групу будуть заноситися файли, у яких в параметрах не визначена приналежність до групи; встановленою 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. Існують три способи стиснення бази даних:

- автоматичне стиснення при встановленні відповідного параметра в налагодженнях 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', ['відсоток'] [,
NOTRUNCATE | TRUNCATEONLY])
```

Опис параметрів:

- *ім'я_БД* – ім'я бази даних, яку необхідно стиснути;
- *відсоток* – кількість відсотків вільного простору, яке бажано залишити після стиснення;
- *NOTRUNCATE* – вільний 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 стиснення 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 меншого за початковий.

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

```
DBCC SHRINKFILE ('ім'я_файла',
['кінцевий_розмір]
[, EMPTYFILE | NOTRUNCATE | TRUNCATEONLY ])
```

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

Опис параметрів:

- *ім'я_файла* – 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* – місце, що звільнилося, 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

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, який був перед збоєм;

- *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_файла', ...]
[ FILEGROUP = 'ім'я_групи' ]
TO логічне_ім'я_пристрою
[ WITH
[ DESCRIPTION = 'коментарій' ]
[ 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 перезаписана;
- *INIT* | *NOINIT* – система здійснює або не здійснює ініціалізацію пристрою.

Відновлення бази даних

При відновленні бази даних з резервної копії існуюча база даних буде перезаписана. Для відновлення бази даних використовується команда:

```
RESTORE {LOG | DATABASE } ім'я_БД
'файл_або_файлова_ група'
[ FROM логічне_ім'я_пристрою ]
[ WITH
[ DBO_ONLY ]
[ MOVE 'логічне_ім'я_файла' 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 під час резервного копіювання.

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

ТЕМА 6. ДОПОМІЖНІ ОБ'ЄКТИ БАЗИ ДАНИХ

У темі розглядаються часто використовувані об'єкти в базах даних: збережені процедури і уявлення. Демонструється створення і управління цими об'єктами за допомогою команд SQL.

Мета: сформувати поняття про призначення збережених процедур і уявлень.

Зміст теми 6:

6.1. Поняття збереженої процедури

6.2. Створення процедури засобами Transact-SQL

6.3. Виконання процедури

6.4. Управління збереженими процедурами

6.5. Уявлення

6.6. Створення уявлень за допомогою Transact-SQL

6.7. Управління уявленнями

6.1. 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 імені 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 по мережі декількох операторів передається лише ім'я процедури, що викликається.

Збережені процедури є самостійними об'єктами бази даних, до яких можна дозволити або заборонити 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:

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

MS SQL Server підтримує наступні види 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 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 з застосуванням графічного інтерфейсу *Management Studio*.

При створенні збереженої процедури необхідно враховувати такі особливості:

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

6.2. 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, якому типу даних відповідає значення параметра;
- *default value* – дозволяє визначити значення за замовчуванням, якщо при виклику процедури параметр був невідомий;
- опція *READONLY* створює параметр доступний тільки для читання, якщо параметр має тип ***table***, то вказівка *READONLY* обов'язкова;
- режим *WITH ENCRYPTION* забороняє подальший перегляд коду створюваної процедури, шифруючи його;
- режим *RECOMPILE* вказує, що сервер не кеширує план виконання процедури, і процедура компілюється тільки під час виконання.

Після ключового слова *AS* слідують або команди Transact-SQL, які і складають тіло процедури, або прописується метод із зазначеної 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 виконання буде повідомлення:

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

6.3. Виконання процедури

Збережена процедура може бути виконана за допомогою оператора EXECUTE:

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

При використанні вихідного параметра, його слід описати з використання ключового слова *OUTPUT* при створенні процедури, а також використовувати це слово і при вказівці відповідного параметра при виклику процедури. В протилежному 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 додатні значення статусу.

6.4. Управління збереженими процедурами

Зміна. Для зміни існуючої 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'єкта' 'НовеІмяОб'єкта'.
```

Видалення. Для видалення збереженої процедури використовується команда Transact SQL:

```
DROP PROC ІмяПроцедури.
```

6.5. Уявлення

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

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

Уявлення мають наступні переваги:

- забезпечують конфіденційність інформації, тому що дозволяють відобразити тільки необхідну інформацію, приховуючи певні поля;
- спрощують уявлення даних, тому що користувач працює з уявленням як з єдиної 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 уявлення.

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

6.6. Створення уявлень за допомогою Transact-SQL

Для створення уявлення використовується команда CREATE VIEW, правом її виконання володіють члени ролей *sysadmin, db_owner, db_dlladmin*:

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

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

ЗапитВибірки є оператор SELECT, параметри якого і визначають вміст уявлення. Імена полів уявлення задаються або за допомогою псевдонімів в операторі вибірки, або вказуються в параметрі поле.

Наприклад, створимо подання, що містить лише таку інформацію про співробітників компанії ***AdventureWorks***, як: посада 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 змісту проекції виконується наступний запит:

```
SELECT * FROM InfoEmployees
```

За допомогою даного подання обмежений доступ до деяких полів 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, таке уявлення 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 розділ *ORDER BY*, який в подальшому можна застосувати при вибірці даних зі створеного уявлення. Також рекомендується створювати уявлення тільки на основі таблиць, для яких виконано внутрішнє з'єднання.

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

```
CREATE VIEW InfoOrders
AS
SELECT FirstName + ' ' + LastName as [Название
компании],
```

```
    SalesOrderHeader.SalesOrderID as [Номер
заказа],
        Convert (money, sum(UnitPrice*OrderQty*(1-
    UnitPriceDiscount)),0) as [Итог]
   FROM (Person.Contact INNER JOIN
Sales.SalesOrderHeader
   ON
Contact.ContactID=SalesOrderHeader.ContactID)
   INNER JOIN Sales.SalesOrderDetail
   ON
SalesOrderHeader.SalesOrderID=SalesOrderDetail.Sal
esOrderID
   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 уявлення *НЕ* знизять продуктивність БД.

6.7. Управління уявленнями

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

```
DROP VIEW Ім’яУявлення
```

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

Список об’єктів, від яких залежить уявлення, може бути отриманий виконанням збереженої процедури: **sp_depends** Ім’яУявлення.

Стислі підсумки. Розглянуто збережені процедури і уявлення, а також основні команди SQL для створення, використання і управління цими об’єктами.

ТЕМА 7. СИСТЕМА БЕЗПЕКИ В БАЗАХ ДАНИХ

У темі 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.

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

Зміст теми 7:

7.1. Аутентифікація користувача

7.2. Ролі сервера

7.3. Управління обліковими записами для входу

7.4. Доступ до бази даних

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.

В MS SQL Server передбачено три рівня безпеки:

1. Аутентифікація при реєстрації.

2.Дозвіл на доступ до бази даних, яка підтримується сервером.

3. Повноваження (дозволи) користувача.

7.1. Аутентифікація користувача

Безпека в усіх додатках насамперед заснована на використанні *імені користувача* (login) та *пароля*. При установці MS SQL Server створюється обліковий запис **sa** (system administrator – системний

адміністратор), до MS SQL Server 2000 він створювався з порожнім паролем, в MS SQL Server 2008 використовувати порожній пароль для **sa** за замовчуванням заборонено і при установці сервера потрібно задати складний пароль у відповідності до поточних політик безпеки Windows.

Розглянемо 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 саме ім'я для входу. Особливо це погано в ситуації, коли для цього облікового 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 пароля (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 другого 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 пароля 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 і т.ін.

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

Для реалізації зазначених вище правил на практиці необхідно розглянути, що являє собою процес аутентифікації в MS SQL Server 2008 і які типи аутентифікації він підтримує.

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

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

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

Аутентифікація MS SQL Server дозволяє визначити ім'я користувача для входу (login) і пароль для встановлення з'єднання з сервером. При установці MS SQL Server створюється лише однин 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 сервера.

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.

7.2. Ролі сервера

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 сервера*, 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 сервера;
- *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 занадто довго;
- *dbcreator* – дозволено створення та зміна баз даних;
- *diskadmin* – управляє файлами баз даних: призначає файли в групи, приєднує / від'єднує бази даних і т. ін;
- *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 сервера та опис кожної ролі;
- **sp_helpsrvrolemember** *[‘ім’я ролі’]* – повертає список ролей та облікових записів, яким присвоєні ці ролі;
- **sp_srvrolepermission** *[‘ім’я ролі’]* – повертає список дозволів, наданих цим ролям.

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

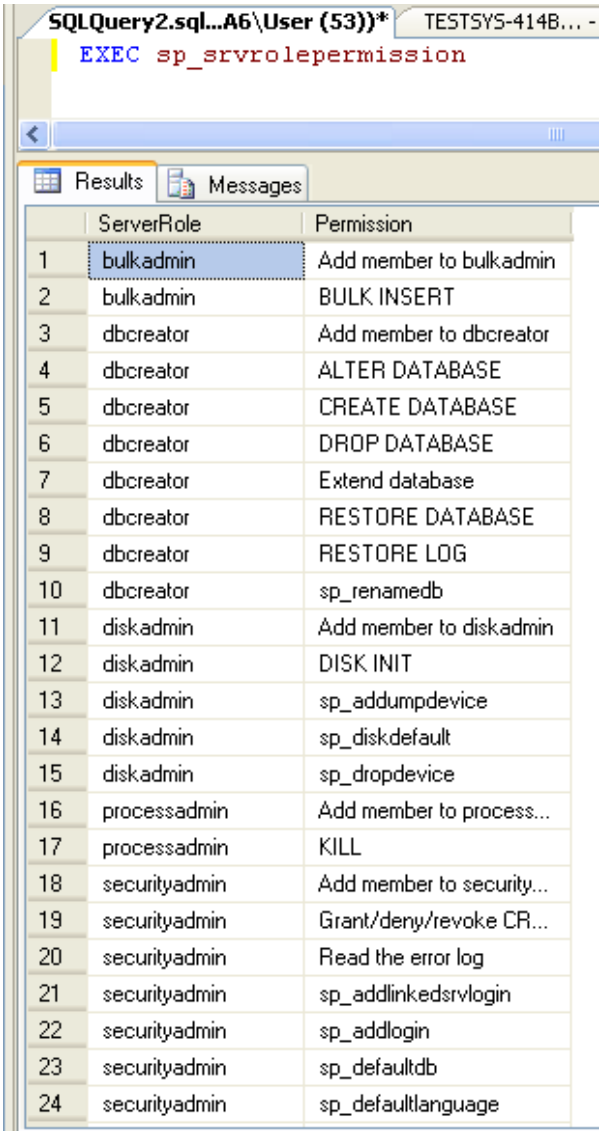
На рис. 7.1. показаний приклад виклику збереженої процедури.

7.3. Управління обліковими записами для входу

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



	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. 7.1. Приклад виклику збереженої процедури

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

@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 використовується в MS SQL Server 2005 та 2008. Однак в наступних версіях MS SQL Server вона є видаленою. 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
```

Параметр *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 зворотний слеш (\);
- 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 по іменах користувачів, що використовуються для входу, застосовується збережена процедура **sp_helplogins**. Приклад виконання цієї процедури показаний на рис. 7.2.

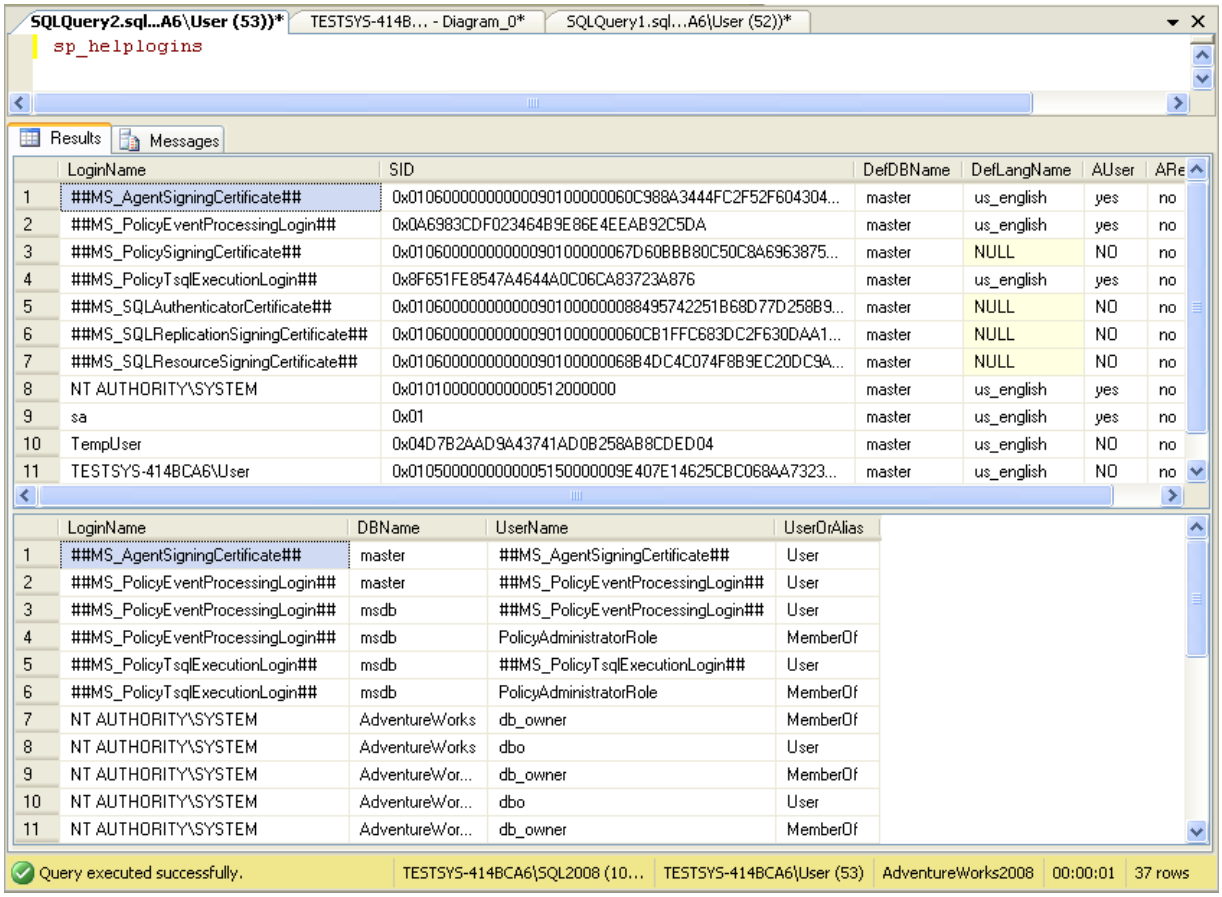


Рис. 7.2. Отримання списку користувачів MS SQL Server

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

Зауваження. Для отримання довідки по командам 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 версіях MS SQL Server 2010 та вище вона видалена. Замість неї рекомендується використовувати конструкцію ALTER LOGIN.

4. Видалення облікового запису здійснюється збереженою процедурою:

```
sp_droplogin 'обліковий запис користувача для входа'
```

Ця збережена процедура широко використовується на сьогоднішній день. Однак в версіях MS SQL Server 2010 та вище вона видалена. Замість неї рекомендується використовувати наступну конструкцію:

```
DROP LOGIN 'обліковий запис користувача для входа'
```

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

```
sp_addsrvrolemember [@loginame=] 'обліковий запись користувача для входа' , [@rolename=] 'роль'
```

Наприклад:

```
EXEC sp_addsrvrolemember 'Corporate\HelenS', 'sysadmin'
```

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

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

```
Наприклад:
EXEC sp_dropsrvrolemember 'JackO', 'sysadmin'
```

7.4. 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 бази даних

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

- *db_owner* – включає у 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 управляти 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 визначених користувачем) 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 сервера.

Управління користувачами баз даних

За замовчуванням MS SQL Server має спеціальний обліковий запис користувача в базі даних: власник БД (*dbo* – data base owner). Власник бази даних має повну владу над БД, власником 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 процедури, що були видалені в версіях MS SQL Server 2010 та вище. 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 була видалена в версіях MS SQL Server 2010 та вище. Замість неї рекомендується використовувати наступну конструкцію:

```
DROP USER 'ім'я користувача'
```

Користувач не може бути видалений, якщо він є власником об’єктів в базі даних.

4. Присвоєння 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 сервера адміністратор має можливість створювати користувацькі 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 безпеки MS SQL Server *користувачеві* надаються певні права доступу до об'єктів даних. Не слід плутати ***ім'я користувача для входу*** та ***ім'я користувача бази даних для доступу до об'єктів***.

Всі описані вище дії з організації системи безпеки можна виконати також і за допомогою графічного інтерфейсу утиліти адміністрування *Management Studio*.

Управління обліковими записами для входу

Перегляд списку наявних облікових записів і їх параметрів здійснюється вибором групи *Logins* в папці **Security** сервера. Облікові записи користувачів відображаються в полі *Name*, в 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, що з'явиться, вказати (див. рис. 7.3):

- вкладка *General*: ім'я користувача, тип аутентифікації (при аутентифікації засобами MS SQL Server – задати пароль), базу даних, до якої користувач підключається автоматично, і мову за замовчуванням;
- вкладка *Server Roles*: ролі сервера, в які буде входити створюваний обліковий запис;
- вкладка *User Mapping*: доступ до однієї зі створених на сервері бази даних, в поле User ввести ім'я користувача бази даних і включити створюваного користувача в одну з існуючих ролей.

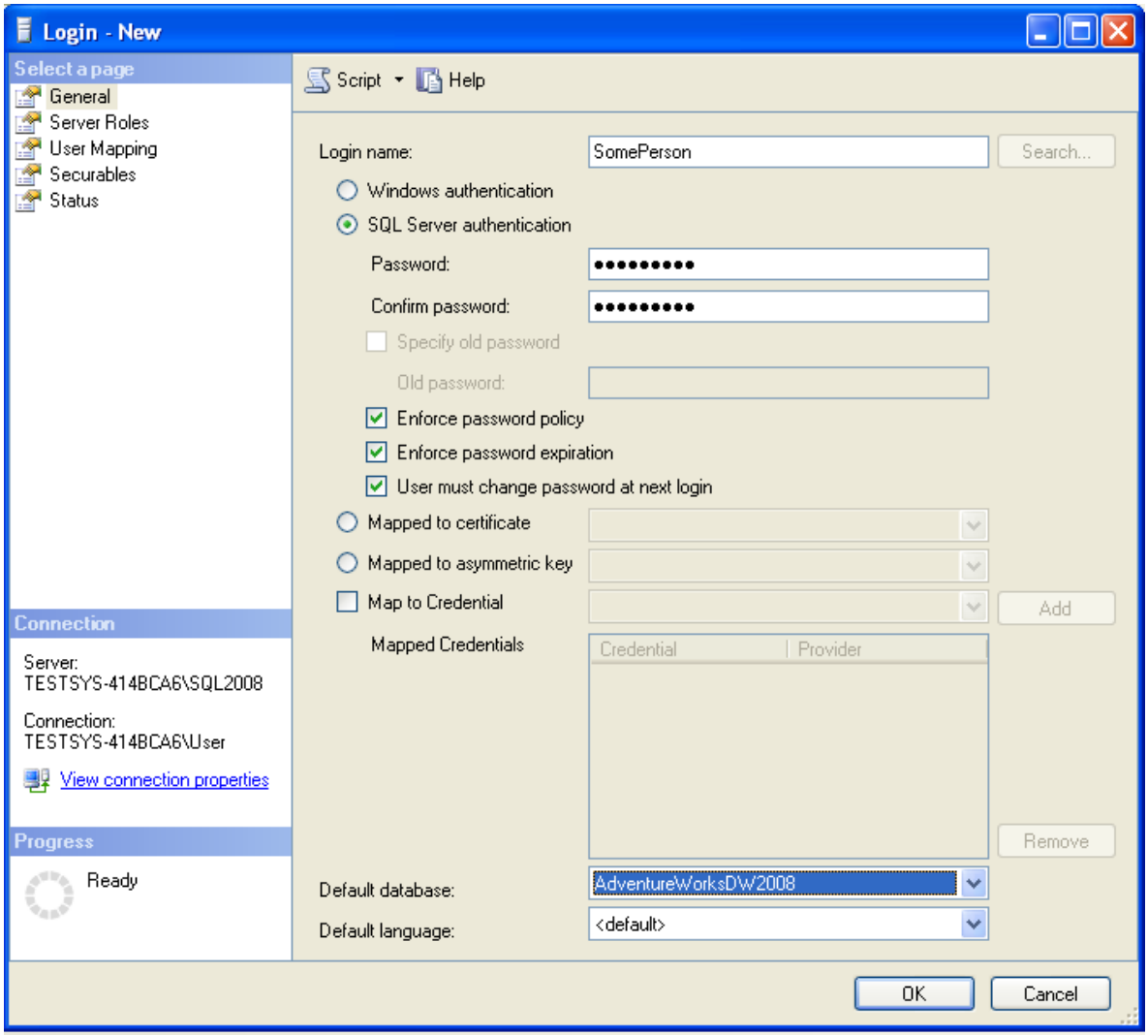


Рис. 7.3. Вікно створення нового облікового запису

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

Управління ролями сервера

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

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

Управління користувачами баз даних

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

Для створення нового користувача бази даних необхідно виконати команду **New User ...**, потім в полі *User name* ввести ім'я користувача, а в списку *Login Name* вибрати відповідний обліковий запис для входу (див. рис. 7.4). Можна також включити користувача в ролі бази даних.

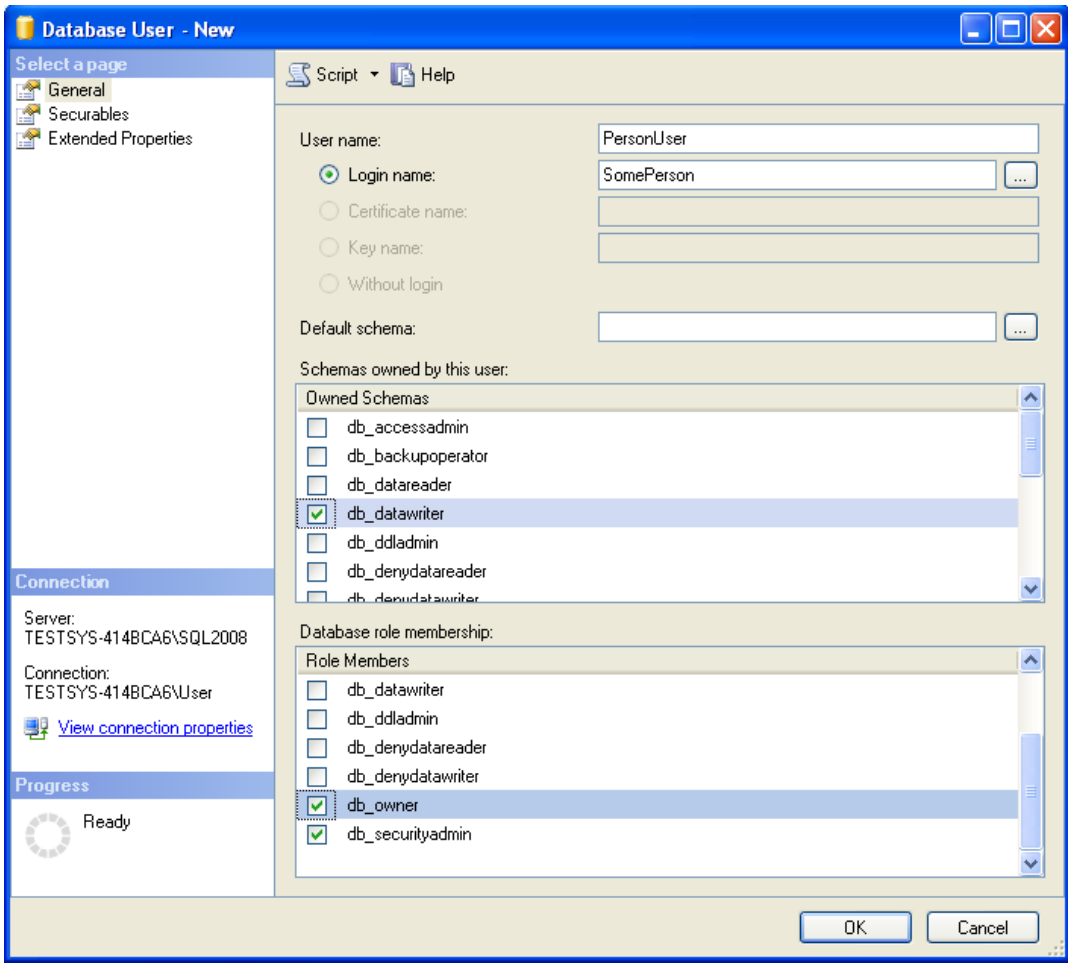


Рис. 7.4. Вікно створення нового користувача БД

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

Управління ролями бази 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 аутентифікації 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 в MS SQL Server можна розділити на три категорії: дозволи для об’єктів, дозволи 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 цим привілеєм дозволено видаляти дані з таблиці.

- ALL. Користувачеві надані будь-які дозволи.

Разрешения для команд Transact-SQL контролируют возможность создания объектов в базе данных, создание самой базы данных и выполнение процедуры резервного копирования.

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

Надання доступу

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

```
GRANT
{ дозвіл [,...n] }{ ON таблиця | уявлення
[(поле[,...n])]
| ON [збережена процедура[,...n] }
TO 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, уявлення, збережена процедура* – вказуються імена конкретних об'єктів поточної БД, 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 можуть 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 наступної команди користувачеві *TestUser* БД ***AdventureWorks*** надаються права вибірки і зміни даних таблиці *Orders* цієї БД:

```
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 | уявлення
[(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 | уявлення
[(поле[,...n])]
| ON [збережена процедура[,...n] }
TO обліковий запис [,...n][CASCADE]
```

Смисл 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 заборону – **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. 7.5).

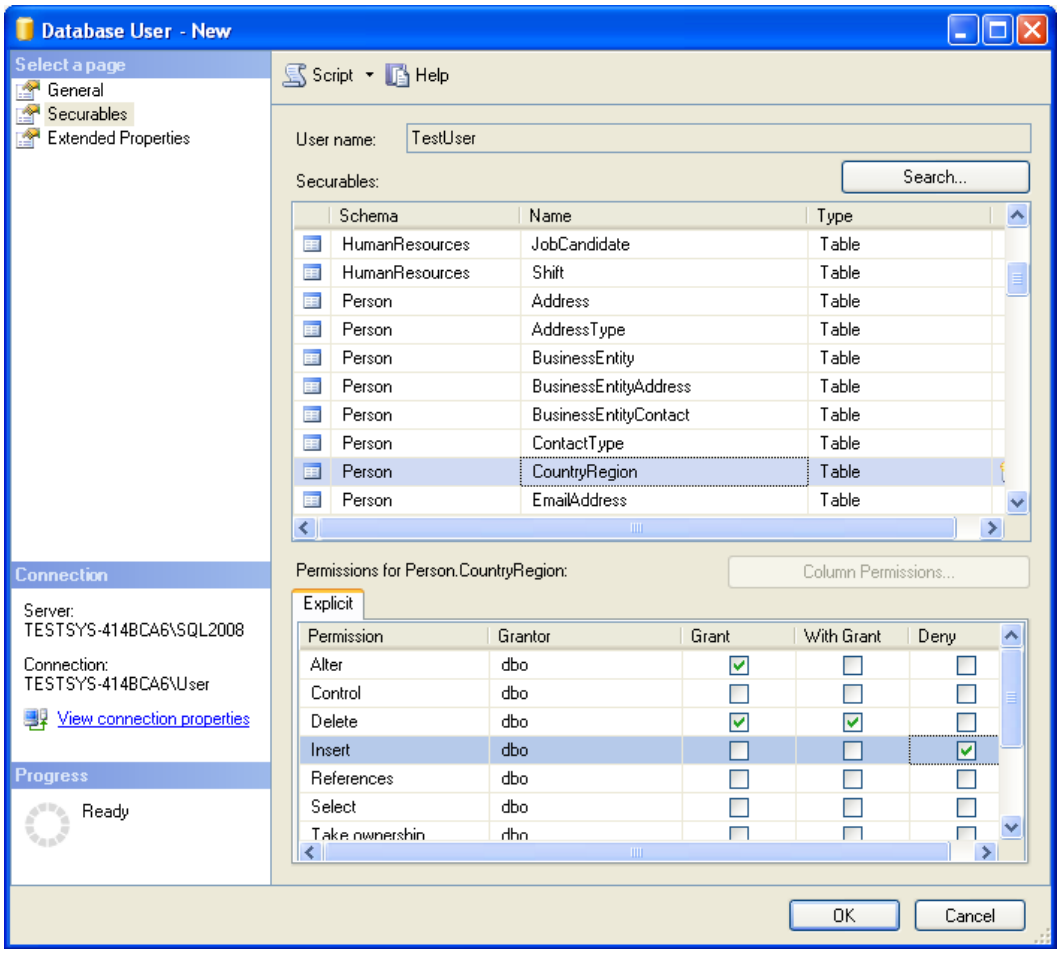


Рис. 7.5. Призначення прав користувачу БД

У вкладці, що з’явиться, перераховані всі об’єкти БД, з можливими правами доступу. Можна встановити один з трьох станів доступу: надання (галочка), заборона (хрестик) і неявне відхилення (порожнє поле) – у відповідному полі.

Стислі підсумки. 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.

ТЕМА 8. РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ

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

Мета: Розглянути основні елементи реляційних баз даних.

Зміст теми 8:

8.1. Реляційні об’єкти даних

8.2. Домени

8.3. Відношення

8.4. Цілісність реляційних даних

8.5. Потенційні ключі

8.6. Первинні та альтернативні ключі

8.7. Зовнішні ключі

8.8. NULL-значення

8.1. Реляційні об’єкти даних

Реляційна модель включає три складові частини: об’єкти, цілісність, оператори.

Розглянемо об’єкт на рис. 8.1.

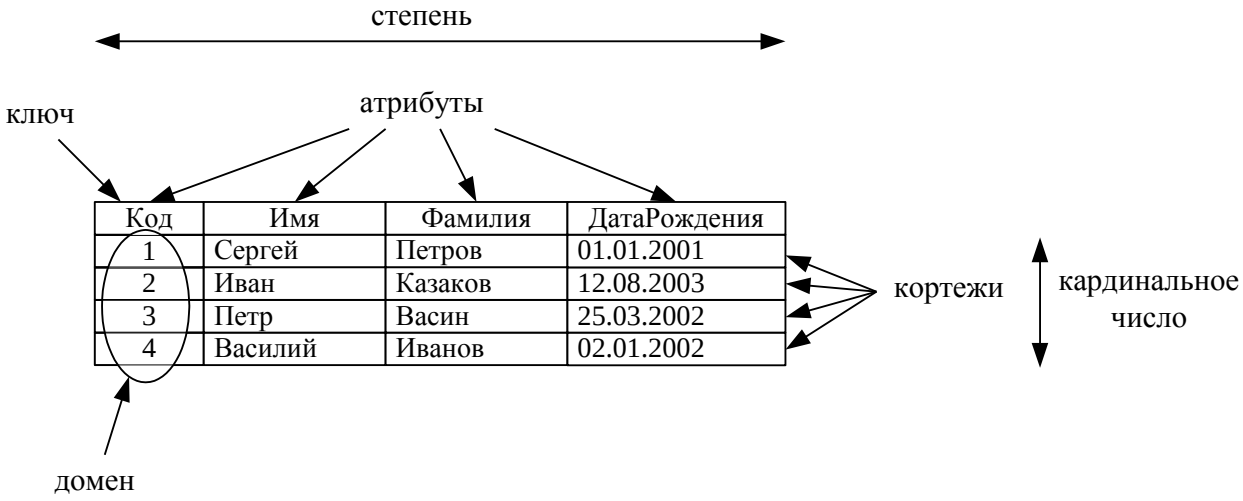


Рис. 8.1. Основні елементи реляційної БД

Відношення відповідає тому, що ми зазвичай називаємо таблицею. *Кортеж* відповідає екземпляру запису, *атрибут* – полю таблиці, *ступінь* – кількості атрибутів, *кардинальне число* – кількості кортежів.

8.2. 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 розкладються 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.

П р и к л а д : Розглянемо 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
```

8.3. Відношення

Відношення можна розглядати з двох боків:

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.

Уточнимо визначення відношення:

Відношення R, задане на множині доменів D_1, D_2, ..., 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 відношення 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 в першій нормальній формі*.

8.4. 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 СУБД про обмеження реального світу. Наприклад, для атрибутів «зріст», «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 темі.

8.5. Потенційні ключі

Нехай *R* – деяке 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 жодна з підмножин *K* не володіє властивістю унікальності.

Цю властивість можна застосувати тільки для випадку, якщо потенційний ключ складається більш ніж з одного атрибута. 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 ключем, але зручніше ввести штучний ключ – **StudentID**.

Таблиця 8.1

Відношення Students

StudentID	Name	GroupID	BirthDate
1	Курзаков Микола	2	13.10.1997
2	Вовк Семен	1	11.05.1998
4	Шишкун Дарина	2	23.09.1998

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

8.6. Первинні та 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, в таблиці *Groups*:

Таблиця 8.2

Зміст таблиці Groups

GroupID	Name
1	КН-41
2	КН-51

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

8.7. Зовнішні ключі

Нехай *R2* – відношення. Тоді *зовнішній ключ FK* у відношенні *R2* – це підмножина множини атрибутів *R2* така, 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 отримати 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 складеним;

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.

Розглянемо відношення *Students* (див. табл. 8.1)

Атрибут **GroupID** буде зовнішнім ключем, тому що до нього проведено 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 посилальний ключ, називається *посилальним відношенням*, 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 або заборонити їх, або ввести додаткові *компенсуючі 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 до вже неіснуючої групи. Саме для таких випадків розробляються компенсуючі операції.

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

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

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

Для компенсації цих операцій існують як мінімум дві можливості:

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

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

- видалити сам кортеж і всі відповідні посилальні кортежі;
- видалити сам кортеж, а для всіх посилальних кортежів виправити значення на правильне, наприклад, встановити NULL-значення для даного атрибута.

8.8. NULL-значення

Іноді потрібна 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 атрибута свідчить саме про його відсутності, 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.

У загальному випадку в БД для кожного атрибута 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 упізнавані, а первинні ключі виконують функцію унікальної ідентифікації.

Правило забезпечення цілісності може бути застосовано тільки:

[8]Едгар Ф. Кодд (Edgar F. Codd) (1923-2003 рр.) – 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] Крістофер Дж. Дейт (Christopher J. Date) (1941 р.) – англійський вчений, який працював над теорією реляційних баз даних спільно з Е.Коддом.

1) до базових відношень, а не обчислювальним (похідним);

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 полі **GroupID** відношення *Students* (див. табл. 8.1) ми можемо вказати NULL-значення для всіх студентів, які належать групі, що видаляється.

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

Таблиця 8.3

StudentID	**Name**	**GroupID**	**BirthDate**
1	Курзаков Микола	NULL	13.10.1997
2	Вовк Семен	1	11.05.1998
4	Шишкун Дарина	NULL	23.09.1998

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

ТЕМА 9. ОПЕРАТОРИ РЕЛЯЦІЙНОЇ АЛГЕБРИ

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

Мета: Знайомство з основними реляційними операторами.

Зміст теми 9:

9.1. Поняття реляційної алгебри

9.2. Основні оператори реляційної алгебри

9.3. Спеціальні реляційні операції.

9.3. Операції розширення і підведення підсумків

9.4. Оператори поновлення

9.1. 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 алгебраїчними операціями, то його називають *реляційно повним*.

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

9.2. 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 ж множину атрибутів, визначених на одних і тих же доменах;
- результатом кожної операції має бути також відношення (властивість замкнутості).

Нехай є відношення *Students* (див. табл. 8.1)

І відношення *Teachers*:

Таблиця 9.1

Відношення Teachers

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

Приклад 9.1. 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 відношення, представлено у таблиці 9.2.

Таблиця 9.2

Результат виконання запиту з прикладу 9.1

Name	BirthDate
Курзаков Микола	13.10.1997
Вовк Семен	11.05.1998
Шишкун Дарина	23.09.1998
Кислицин О.П.	1.2.1970
Царьов С.М.	10.03.1964
Пестов Д.Н.	2.05.1980

Приклад 9.2. Розглянемо відношення *Teachers* (викладачі) (див. табл. 9.1) і *Supervisors* (куратори, можуть керувати викладачами) (див. табл. 9.3).

Таблиця 9.3

Відношення Supervisors

SupervisorID	Name	BirthDate
1	Кислицин 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 виконання запиту буде відношення, представлено у таблиці 9.4

Таблиця 9.4

Результат виконання запиту з прикладу 9.2

Name
Кислицин 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.

П р и к л а д 9 . 3 . Розглянемо відношення *Teachers* (викладачі) – табл. 9.1 та *Supervisors* (куратори, можуть керувати викладачами) – табл. 9.3:

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

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

Результатом виконання запиту буде:

Таблиця 9.4

Результат виконання запиту з прикладу 9.3

Name
Пестов Д.Н.

4) Добуток*:* A TIMES B

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

Для добутку необхідна 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 добутку може виникнути проблема однакових імен атрибутів. У цьому випадку одне з конфліктуючих полів необхідно перейменувати.

П р и к л а д 9 . 4 . Розглянемо відношення *Students* (табл. 8.1) і *Courses* (навчальні курси) (табл. 9.5)

Таблиця 9.5

Відношення Courses

CourseID	Name
1	Програмування
2	Бази даних
3	Проектування програмних систем

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 з прикладу 9.4 представлено у таблиці 9.6.

Таблиця 9.6

Результат виконання запиту з прикладу 9.4

Name	Name
Курзаков Микола	Програмування
Вовк Семен	Програмування
Шишкун Дарина	Програмування
Курзаков Микола	Бази даних
Вовк Семен	Бази даних
Шишкун Дарина	Бази даних
Курзаков Микола	Проектування програмних систем
Вовк Семен	Проектування програмних систем
Шишкун Дарина	Проектування програмних систем

9.3. Спеціальні реляційні операції

1) В*ибірка (або обмеження)*

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

Приклад 9.5. Нехай дано таке відношення *StudentsNew* (див. табл. 9.7).

Таблиця 9.7

Відношення StudentsNew

StudentID	Name	GroupID
1	Курзаков Микола	2
2	Вовк Семен	1
4	Шишкун Дарина	2
5	Драгомиров Євген	1
6	Васнєцова Євгенія	2

Виберемо лише тих студентів, які належать групі 2:

```
SELECT * FROM Students
WHERE GroupID = 2
```

Результатом виконання прикладу 9.5 буде відношення, представлено у таблиці 9.8.

Таблиця 9.8

Результат виконання запиту з прикладу 9.5

StudentID	Name	GroupID
1	Курзаков Микола	2
4	Шишкун Дарина	2
6	Васнецова Євгенія	2

2) Проекція

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

Звернемо увагу на два моменти:

1) можливо зазначення всіх атрибутів вихідного відношення для проекції – отримається *тотожна проекція*;

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

Здійснити проекцію можна зазначенням після SELECT списку необхідних атрибутів.

Приклад 9.6. Нехай дано таке відношення *Students* (див. табл. 8.1). Виберемо з таблиці тільки імена і дати народжень:

```
SELECT Name, BirthDate FROM Students
```

Результатом виконання запиту з прикладу 9.6 подано у таблиці 9.9.

Таблиця 9.9

Результат виконання запиту з прикладу 9.6

Name	BirthDate
Курзаков Микола	13.10.1997
Вовк Семен	11.05.1998
Шишкун Дарина	23.09.1998

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 в результуючому відношенні з'являється тільки один раз).

Приклад 9.7. Нехай дано таке відношення *Students* (таб. 8.1) і відношення *GroupsNew* (див. табл. 9.10)

Таблиця 9.10

Відношення GroupsNew

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

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

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

Результатом виконання буде відношення, представлено у таблиці 9.11.

Таблиця 9.11

Результат виконання запиту з прикладу 9.7

Name	**GroupName**
Курзаков Микола	КН -12
Вовк Семен	КН -11
Шишкун Дарина	КН -12

Це з'єднання має властивості асоціативності і комутативності.

***4) Ділення:* A DIVIDEBY B**

Результатом поділу 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 ділення відношення *R1* на *R2*:

R1

A	X
A	Y
B	Z
B	X
C	Y

R2

X
Y

В результаті отримаємо відношення *R*:

R:

A

9.4. Операції розширення і підведення підсумків

Після того, як Е. Кодд запропонував вісім основних операцій, численні автори запропонували нові алгебраїчні операції. Ми розглянемо лише дві з них – розширення і підведення 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.

Приклад 9.8. Нехай дано відношення *Teacher* (табл. 9.1). Додамо новий атрибут **Age**, який буде показувати, скільки повних років виповнилося викладачеві:

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

Результатом виконання запиту з прикладу 9.8 буде відношення, представлено у таблиці 9.12.

Таблиця 9.12

Результат виконання запиту з прикладу 9.8

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

2. Підведення підсумків

Операція підведення 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*.

Приклад 9.9. Нехай дано відношення *StudentsNew* (табл. 9.7). Потрібно підрахувати, скільки студентів в кожній групі:

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

Результатом виконання запиту з прикладу 9.9 буде відношення, надане у таблиці 9.13.

Таблиця 9.13

Результат виконання запиту з прикладу 9.9

GroupID	StudentCount
1	2
2	3

9.5. Оператори поновлення

Призначені для управління даними в таблицях. Існує три операції поновлення.

1) Вставка запису. Дозволяє додавати одну або кілька нових записів у відношення.

Приклад 9.10. Нехай є відношення *Students* (див. табл. 8.1). Щоб додати в нього дві нові записи, можна використовувати наступні оператори SQL:

```
INSERT INTO Students (Name, GroupID) VALUES ('Драгомиров Євген', 1)
INSERT INTO Students (Name, GroupID) VALUES ('Васнецова Євгенія', 2)
```

Після виконання запиту з прикладу 9.10, відношення *Students* буде виглядати так, як представлено в таблиці 9.14.

Таблиця 9.14

Результат виконання запиту з прикладу 9.10

StudentID	Name	GroupID
1	Курзаков Микола	2
2	Вовк Семен	1
4	Шишкун Дарина	2
5	Драгомиров Євген	1
6	Васнецова Євгеніч	2

2) Видалення запису. Видаляє всі записи з відношення, або тільки записи, що задовольняють заданому критерію.

Приклад 9.11. Нехай є відношення *StudentsNew* (табл. 9.7). Щоб видалити з нього студентів, що входять в групу з номером 1, можна використовувати наступні оператори SQL:

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

Після виконання запиту з прикладу 9.11, відношення *StudentsNew* буде виглядати так, як представлено в таблиці 9.15.

Таблиця 9.15

Результат виконання запиту з прикладу 9.11

StudentID	Name	GroupID
1	Курзаков Микола	2
4	Шишкун Дарина	2
6	Васнецова 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.

Приклад 9.12. Нехай є відношення *StudentsNew* (табл. 9.7). Щоб перевести всіх студентів, що входять в групу з номером 1, в групу з номером 2, можна використовувати наступні оператори SQL:

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

Після виконання запиту з прикладу 9.12, відношення *StudentsNew* буде виглядати так, як представлено в таблиці 9.16.

Таблиця 9.16

Результат виконання запиту з прикладу 9.12

StudentID	Name	GroupID
1	Курзаков Микола	2
2	Вовк Семен	2
4	Шишкун Дарина	2
5	Драгомиров Євген	2
6	Васнецова Євгенія	2

Стислі підсумки. Розглянуто основні операції над таблицями, а також наведено приклади використання цих операцій на мові SQL.

ТЕМА 10. ПЕРШІ НОРМАЛЬНІ ФОРМИ

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

Мета: Ввести визначення перших трьох нормальних форм і обґрунтувати необхідність їх застосування.

Зміст теми 10:

10.1. Процес нормалізації

10.2. Поняття функціональної залежності

10.3. Декомпозиція без втрат

10.4. Перша, друга і третя нормальні форми

10.5. Уточнення третьої нормальної форми – нормальна форма Бойса-Кодда

10.1. 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. показані нормальні форми, 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.

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.

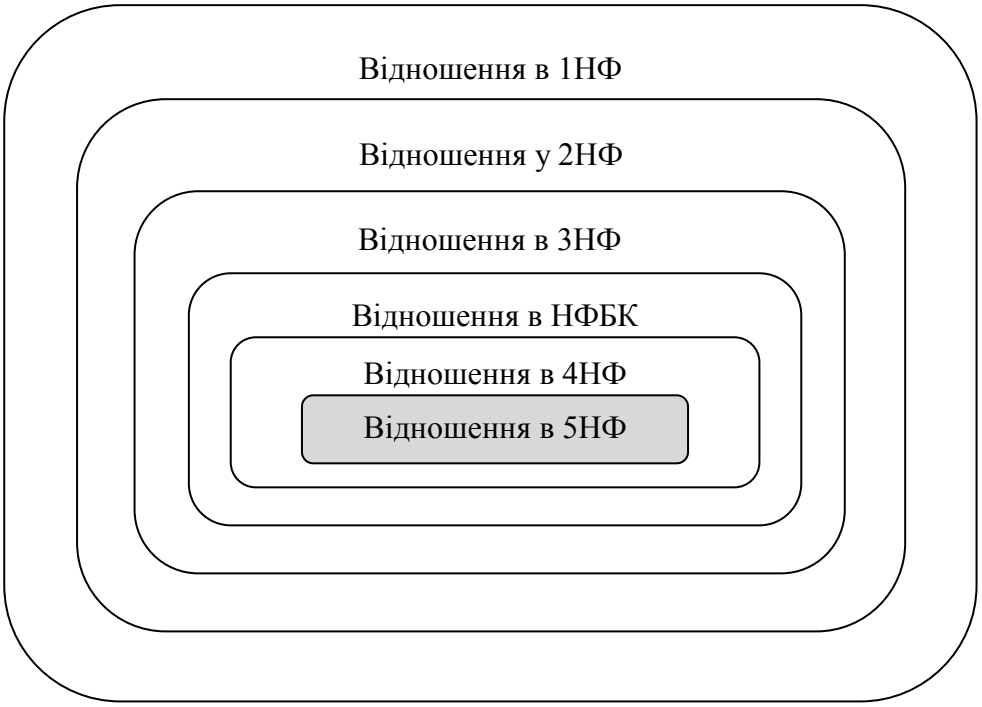


Рис. 10.1. Рівні нормалізації

10.2. 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 і Y – довільні підмножини множини атрибутів R. Тоді Y *функціонально залежить* від X, що в символьному вигляді записується як $X \rightarrow Y$ (читається як «X функціонально визначає Y») тоді і тільки тоді, коли для будь-якого допустимого значення R кожне значення X пов’язане точно з одним значенням Y.

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

П р и к л а д : Нехай *R* – це відношення *Students*. *X* – код студента, а *Y* – множина всіх атрибутів студента. Тоді $X \rightarrow Y$, тому що X являє собою первинний ключ, який унікально ідентифікує запис у таблиці *Students*.

Таке твердження буде вірно і для більш загального випадку: 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* володіє *надмірністю*, що ускладнює забезпечення цілісності даних і займає зайві ресурси системи.

Якщо жоден атрибут не може бути опущений з лівої частини, то така функціональна залежність називається *неприводимою* (точніше, *неприводимою зліва*).

П р и к л а д .

{StudentID, FirstName, LastName, MiddleName} → {BirthDate} – приводима ФЗ.

{StudentID} → {BirthDate} – неприводима ФЗ.

Множина функціональних залежностей називається *неприводимою* тоді і тільки тоді, коли вона володіє всіма трьома перерахованими нижче властивостями:

1. Залежна частина кожної функціональної залежності містить тільки один атрибут.

2. Детермінант кожної функціональної залежності є неприводимим.

3. Жодна функціональна залежність з множини не може бути видалена без втрати інформації про зв’язки.

Розгляд множини неприводимих 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 являють 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.

10.3. 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 оборотним, тобто можна зробити зворотне з'єднання.

П р и к л а д . Розглянемо відношення *Students*:

StudentID	LastName	FirstName	MiddleName	GroupID	BirthDate
1	Курзаков	Микола	Володимирович	1	13.10.1997
2	Вовк	Семен	Аркадійович	2	11.05.1998
4	Шишкун	Дарина	Сергіївна	1	23.09.1998

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

StudentID	LastName	FirstName	MiddleName	GroupID
1	Курзаков	Микола	Володимирович	1
2	Вовк	Семен	Аркадійович	2
4	Шишкун	Дарина	Сергіївна	1

GroupID	BirthDate
1	13.10.1997
2	11.05.1998

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	Курзаков	Микола	Володимирович	1
2	Вовк	Семен	Аркадійович	2
4	Шишкун	Дарина	Сергіївна	1

StudentID	BirthDate
1	13.10.1997
2	11.05.1998
4	23.09.1998

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

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

Щоб гарантувати, що після декомпозиції зворотне з'єднання відношень дасть вихідне, 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*}.

10.4. Перша, друга і 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	Курзаков Микола Володимирович
2	Вовк Семен Аркадійович

4	Шишкун Дарина 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	Курзаков	Микола	Володимирович
2	Вовк	Семен	Аркадійович
4	Шишкун	Дарина	Сергіївна

Слід мати на увазі, що значення полів повинні бути логічно неподільні, тому що прізвище складається з окремих букв, але такий розподіл 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 неключовий атрибут неприводимо залежить від первинного ключа.

Розглянемо наступне відношення:

StudentID	LastName	FirstName	MiddleName	Course	Score
1	Курзаков	Микола	Володимирович	Інформатика	5
2	Вовк	Семен	Аркадійович	Математика	4
4	Шишкун	Дарина	Сергіївна	Математика	5
2	Васильєв	Іван	Аркадійович	Інформатика	3

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

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

Вочевидь, що вона є приводимою, тому що прізвище залежить тільки від **StudentID**, тобто існує ФЗ: {**StudentID**} → {**LastName**}.

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

Students:

StudentID	LastName	FirstName	MiddleName
1	Курзаков	Микола	Володимирович
2	Вовк	Семен	Аркадійович
4	Шишкун	Дарина	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).

Розглянемо наступне відношення *Students*:

StudentID	LastName	FirstName	MiddleName	GroupID	Supervisor
1	Курзаков	Микола	Володимирович	1	Царьов С.М.
2	Вовк	Семен	Аркадійович	2	Пестов Д.Н.
4	Шишкун	Дарина	Сергіївна	1	Царьов С.М.

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

Students:

StudentID	LastName	FirstName	MiddleName	GroupID
1	Курзаков	Микола	Володимирович	1
2	Вовк	Семен	Аркадійович	2
4	Шишкун	Дарина	Сергіївна	1

Groups:

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

10.5. 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 3НФ, 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 3НФ було замінено більш суворим визначенням Бойса-Кодда.

На практиці комбінація всіх трьох умов зустрічається рідко, і для відношень, 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 неприводима зліва функціональна залежність має в якості свого детермінанта деякий потенційний ключ.

Розглянемо 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 складеними і вони мають загальний атрибут **StudentID**, тобто перекриваються. Таким чином виконані всі три умови, які можуть привести до ситуації, коли відношення може перебувати в 3НФ, але не знаходиться в НФБК.

Очевидно, що відношення знаходиться в 3НФ:

- значення всіх атрибутів неподільні (1НФ);
- кожен неключовий атрибут неприводимо 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 вихідне 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 *надмірність 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.

Стислі підсумки. Розглянуто перша, друга, третя нормальні форми і нормальна форма Бойса-Кодда. На прикладах розібраний процес приведення відношення в задану нормальну форму.

ТЕМА 11. ЧЕТВЕРТА І П'ЯТА НОРМАЛЬНІ ФОРМИ

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

Мета: Ввести поняття четвертої і п'ятої нормальних форм і обґрунтувати необхідність їх застосування.

Зміст теми 11:

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

11.1. Багатозначна залежність

Нехай R – відношення, а A, B і C є довільними підмножинами множини атрибутів відношення 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, відповідна заданій парі значень A, C, залежить тільки від значення A і не залежить від значення C.

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

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

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

CourseID	TeacherID	RoomID
2	10	400
2	9	401

то в даному відношенні мають 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 викладати 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 усієї 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 робили 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 множини 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 відношення *R {A, B, C}*. В такому випадку відношення *R* дорівнюватиме з'єднанню його проекцій по атрибутам {*A, B*} і {*A, C*} тоді і тільки тоді, коли для відношення *R* виконується багатозначна залежність $A \rightarrow\rightarrow B \mid C$.

Знайдемо всі БЗ в нашому відношенні:

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

За теоремою 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 відношення повністю ключові, то вони знаходяться в НФБК.

Тепер, слідуючи теоремі Фейгіна, можна дати визначення *четвертої нормальної форми.*

11.2. Четверта нормальна форма

Відношення *R* знаходиться в 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 атрибута A.

Можна дати альтернативне формулювання: відношення *R* знаходиться в 4НФ тоді і тільки тоді, коли воно знаходиться в НФБК і всі багатозначні залежності у відношенні *R* фактично представляють собою функціональні залежності від його потенційних ключів. Зверніть увагу на те, що, виходячи з цього визначення, находження в 4НФ передбачає обов'язкове находження в НФБК.

Отже, вихідне відношення {**CourseID, TeacherID, RoomID**} не перебуває у 4НФ, тому що існує БЗ **CourseID** →→ **TeacherID**, яка не є ФЗ, і детермінант не є ключем.

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

11.3. Залежність з'єднання

До сих пір за замовчуванням передбачалося, що єдиною необхідною або допустимою операцією в процесі нормалізації є заміна відношення за правилами декомпозиції без втрат *точно двома* її проекціями. Однак існують відношення, 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 відношення позначимо терміном *«n-декомпонуєме відношення»,* де *n* – кількість проекцій, на які можна розбити без втрат.

П р и к л а д . Розглянемо приклад відношення, яке важко розбити без втрат на два відношення:

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

Розіб'ємо на три відношення:

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	КН-41
Технології програмування	КН-51
Бази даних	КН-41
Бази даних	КН-51

Teacher	**Group**
Іванов С.Ю.	КН-41
Іванов С.Ю.	КН-51
Пестов О.В.	КН-51
Веснін Р.А.	КН-51

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

Дамо визначення залежності з'єднання, необхідне 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 припустиме значення відношення *R* еквівалентно з'єднанню його проекцій за підмножинами *A, B, ..., Z* множини атрибутів.

Для нашого вихідного 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.

11.4. П'ята нормальна форма

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

Відношення {**Teacher, Course, Group**} не знаходиться у 5НФ, тому що в 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.

Розглянемо 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**}} і т.под.

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 відношення, які доведені до 4НФ, але не перебувають у 5НФ, на практиці зустрічаються вкрай рідко.

Фактично 5НФ є *остаточною нормальною формою* по відношенню до операцій проекції і з'єднання (що відображено в її альтернативній назві – *проекційно-з'єднальна* 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.

11.5. Підсумкова схема нормалізації

До цього весь процес нормалізації базувався на декомпозиції без втрат. Основна ідея полягала в наступному: нехай дано деяке відношення *R* в 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 більш прийнятним. Кожен етап процесу такого перетворення полягає в розбитті на проекції відношень, 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 неприведеними. 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 в 3НФ.

3. Відношення в 3НФ слід розбити на проекції, що дозволяють виключити всі функціональні залежності, які залишилися, в яких детермінанти не є потенційними ключами. В результаті буде отриманий набір відношень в НФБК.

Зауважимо, що ці три правіла можна об'єднати в одне: «Початкове відношення слід розбити на проекції, що дозволяють виключити всі функціональні залежності, в яких детермінанти не є потенційними ключами».

4. Відношення 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 з 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 проекту бази даних, який є досить «якісним» поданням реального світу, інтуїтивно зрозумілий і може служити хорошою основою для подальшого розширення;
- спрощення процедури накладення обмежень цілісності (ця мета забезпечується створенням зв'язку «один до багатьох», від ключового до не ключового атрибутів).

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 проектування 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 і 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 зручні методики спадного проектування, які дозволяють тим або іншим конкретним способом створювати повністю нормалізований проект бази даних.

11.6. 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 в даний час, причому досить успішно, і після 5НФ були виведені деякі інші:

1. Доменно-ключова нормальна форма (ДКНФ). Ця форма була запропонована Р. Фейгіном, 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 в ДКНФ тоді і тільки тоді,

коли кожне накладене на нього обмеження є логічним наслідком *обмежень доменів* і *обмежень ключів* відношення ***R***.

Тут обмеження домену – це обмеження, що пропонує використання для певного атрибута значень тільки з деякого заданого домена. 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 задіяні 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».

Стислі підсумки. Розглянуто четверта і п'ята нормальні форми, які вважаються остаточними. Сформульовано основні правила нормалізації баз даних. Наведено альтернативні нормальні форми.

ТЕМА 12. ВИКОРИСТАННЯ MS SQL SERVER 2008 СПІЛЬНО З MS VISUAL STUDIO 2008

У темі розглядаються 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.

Мета: Познайомити з сучасними можливостями в області роботи з даними.

Зміст теми 12:

12.1. Створення функцій для MS SQL Server з використанням платформи .Net Framework

12.2. Технології доступу до даних: LINQ і ADO.NET

12.1. Створення функцій для MS SQL Server з використанням платформи .Net Framework

Дана можливість з'явилася ще в MS SQL Server 2005 року, 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 і ін.) об'єкти, які згодом можуть використовуватися всередині MS SQL Server. Дозволяється створення наступних об'єктів:

- нові типи даних;
- користувацькі функції;
- збережені процедури;
- агрегатні функції;
- тригери.

За замовчуванням на MS SQL Server заборонено використання CLR (Common Language Runtime), тому потрібно явно включити цю опцію на сервері.

Зауваження. CLR – це загальномовне виконуюче 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. В MS Visual Studio 2008 створимо новий проект (меню **File – New – Project ...)**

2. У діалозі створення проекту виберемо потрібний тип проекту і вкажемо ім'я (рис. 12.1).

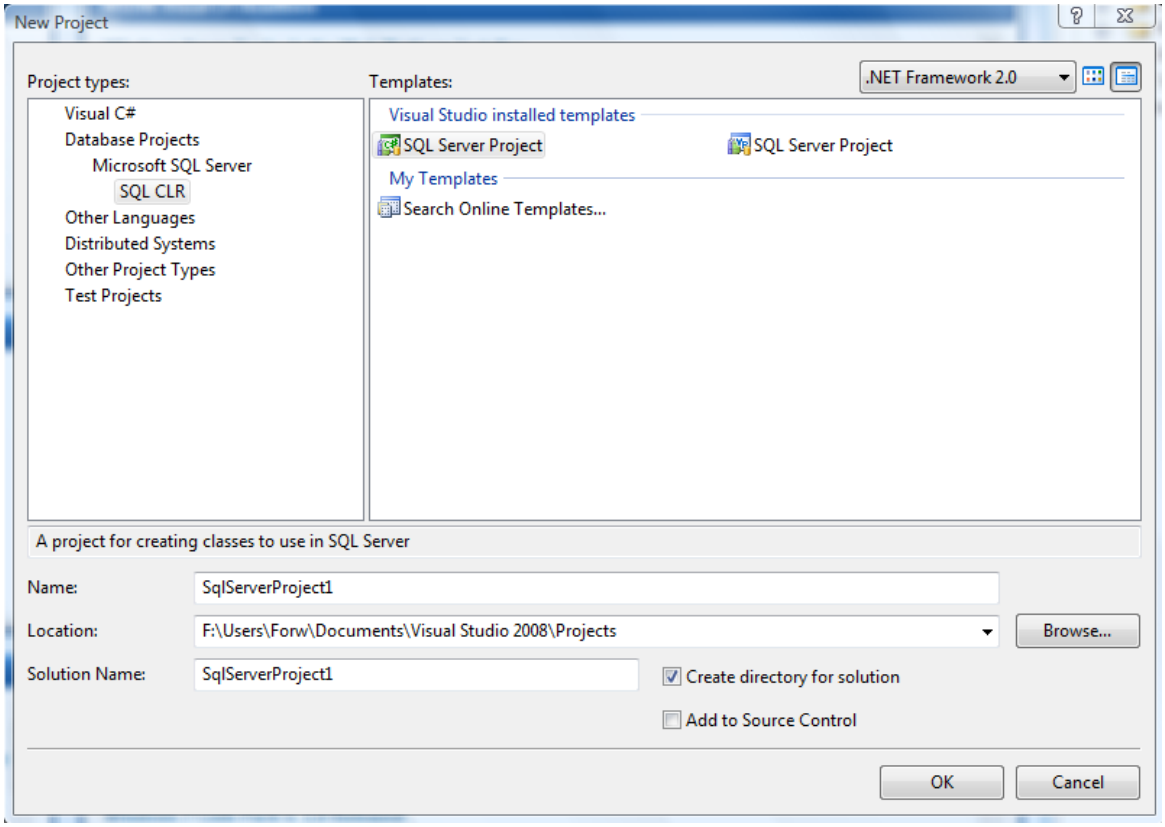


Рис. 12.1. Створення нового проекту в MS Visual Studio 2008

3. Далі можна вказати сервер і БД, які згодом можна буде використовувати для налагодження коду (див. рис. 12.2).

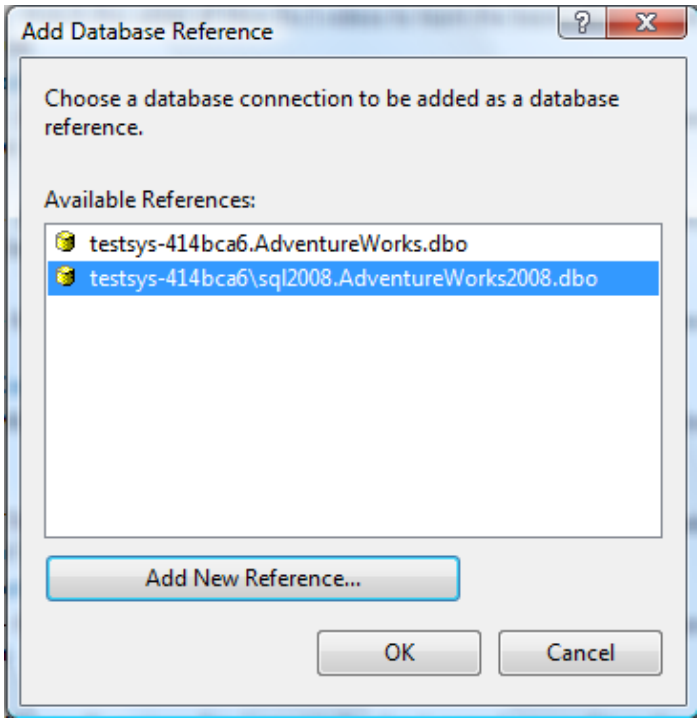


Рис. 12.2. Вибір сервера та БД

4. У створений проект додамо новий об’єкт – користувацьку функцію.

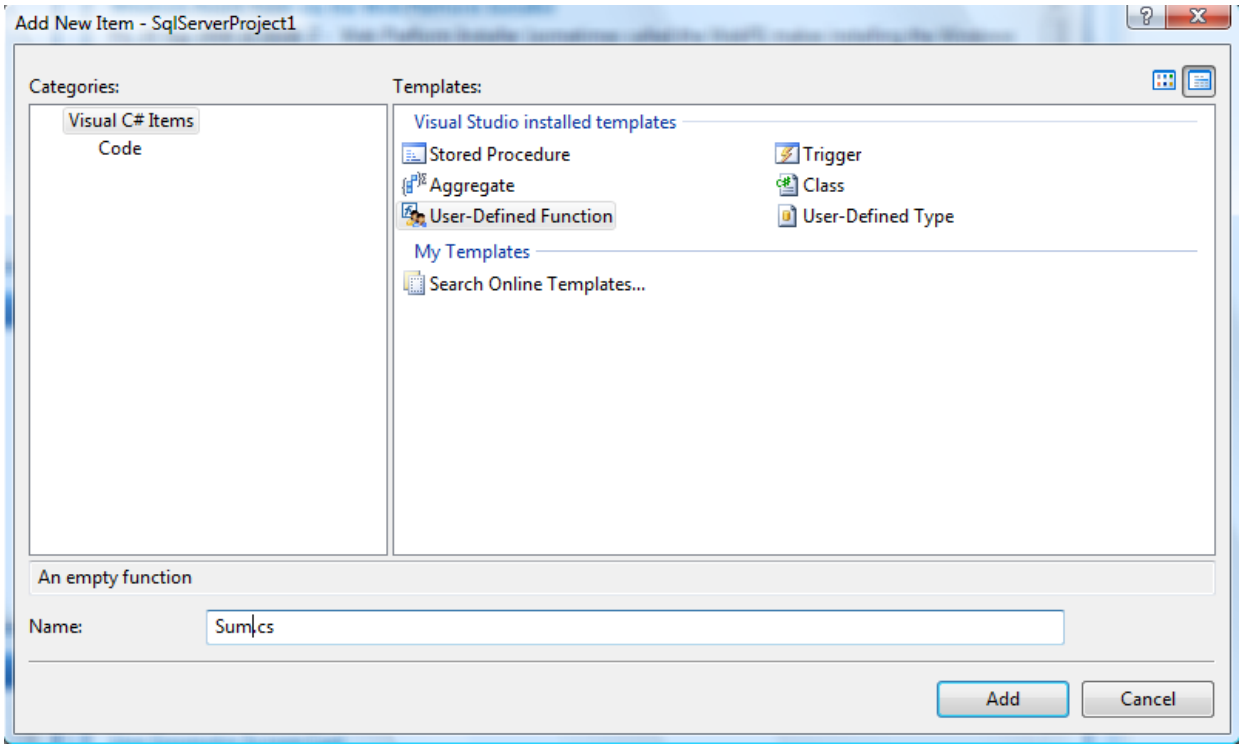


Рис. 12.3. Додавання функції в проект

5. Напишемо наступний код в цій функції:

```
using System;
using System.Data;
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 агрегатну 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#.

12.2. Технології доступу до даних: LINQ і ADO.NET

Технологія ADO.NET

ADO.NET є основною 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 упродовж усього сеансу роботи.

ADO.NET включає наступні ключові компоненти:

- *набори даних* (DataSet). Являють собою деяку частину реальної БД, включаючи в себе не тільки таблиці, але і зв'язки між таблицями і обмеження;
- *провайдери даних* (DataProvider). Завдяки наявності різних провайдерів, технологія ADO.NET може працювати з різними типами СУБД: MS SQL Server, MS Access, Oracle, а також з будь-якою БД, використовуючи технологію ODBC (Open DataBase Connectivity).

Загальна схема роботи з використанням ADO.NET:

1. Створити і встановити підключення до сервера. Наприклад, підключимося 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, отримаємо всі записи з таблиці *Orders*:

```
   SqlCommand cmd = new SqlCommand("SELECT * FROM
Orders", connection);
   SqlDataReader reader = cmd.ExecuteReader();
```

3. Обробити результати виконання команди. Наприклад, виведемо 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) – проект Microsoft по додаванню синтаксису мови запитів, що нагадує SQL, в мови програмування платформи .NET Framework.

LINQ дозволяє виконувати запити до об'єктів, які знаходяться в пам'яті, в типізованій базі даних і в XML документі. Для цього використовуються відповідні технології: LINQ, DLINQ, XLINQ.

Наприклад, створимо масив **arr** і заповнимо його числами від 0 до 10, а потім за допомогою LINQ виберемо елементи більші за 4 та менші за 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;
```

В результаті змінна **newArray** буде містити колекцію цілих чисел: 7, 5, 6.

Перед початком роботи LINQ з базами даних нам потрібно отримати описи 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***.

Розглянемо 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 ();
```

Розглянемо 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 ();
```

Стислі підсумки. 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.

ТЕСТИ ДО ТЕМ

ТЕМА 1. ВВЕДЕННЯ В ТЕОРІЮ БАЗ ДАНИХ

1. Що з перерахованого нижче є елементами реляційних баз даних:
   А. Поле
   Б. Рівень
   В. Запис
   Г. Вузол
2. Що з перерахованого нижче входить до складу СУБД:
   А. Апаратне забезпечення
   Б. Користувачі
   В. Дані
   Г. Програмне забезпечення
3. Які види моделей даних існують:
   А. Мережева
   Б. Ключова
   В. Реляційна
   Г. Доменна
4. Скільки рівнів виділяють в архітектурі СУБД:
   А. 1
   Б. 2
   В. 3
   Г. 4
5. Які рівні виділяють в архітектурі СУБД:
   А. Концептуальний
   Б. Логічний
   В. Зовнішній
   Г. Внутрішній
6. Що з перерахованого нижче є елементами мережевої моделі даних:
   А. Зв’язок
   Б. Рівень

В. Вузол

Г. Запис

7. Що з перерахованого нижче є елементами ієрархічної моделі даних:

А. Поле

Б. Рівень

В. Вузол

Г. Запис

8. Що з перерахованого нижче є основними елементами системи інвертованих списків:

А. Список зв’язків

Б. Список таблиць

В. Основний файл

Г. Інвертований список

ТЕМА 2. КОМПОНЕНТИ MICROSOFT SQL SERVER 2008

1. Які з наступних версій MS SQL Server 2008 існують:
   А. Enterprise Edition
   Б. Professional Edition
   В. Express Edition
   Г. Home Edition
2. Що з перерахованого нижче відноситься до завдань безпосередньо самої служби SQL Server, тобто до завдань ядра сервера:
   А. Управління файлами баз даних
   Б. Керування журналами транзакцій
   В. Повнотекстовий пошук
   Г. Виконання SQL запитів
3. Які з наступних служб існують в MS SQL Server 2008:
   А. SQL Server Agent
   Б. Full-Text Filter Daemon
   В. SQL Server Security

Г. SQL Server Browser

4. Які протоколи можна використовувати для підключення в MS SQL Server 2008:
   А. TCP/IP
   Б. HTTP
   В. HTTPS
   Г. Іменовані канали
5. Куди може виводити результати запиту Management Studio:
   А. На екран
   Б. У файл
   В. Management Studio не призначена для виконання запитів
   Г. Немає правильної відповіді
6. Для чого призначений додаток SQL Server Profiler:
   А. Для виконання запитів SQL
   Б. Для резервного копіювання БД
   В. Для зберігання профілів користувачів MS SQL Server
   Г. Для відстеження виконання команд на сервері
7. Яка максимальна кількість одночасних підключень може бути встановлено до MS SQL Server:
   А. 1
   Б. 32 767
   В. 2 147 483 647
   Г. Кількість підключень не обмежена
8. Скільки системних баз даних встановлено в MS SQL Server за замовчуванням:
   А. 1
   Б. 2
   В. 3
   Г. 4
9. Для чого призначена системна база даних master:

А. Для зберігання системної інформації про сервер і бази даних на сервері

Б. Для зберігання тимчасових даних

В. Як шаблон при створенні нових баз даних

Г. Для служби допоміжних служб, таких як SQL Server Agent

10. Для чого призначена системна база даних *msdb*:

А. Для зберігання системної інформації про сервер і бази даних на сервері

Б. Для зберігання тимчасових даних

В. Як шаблон при створенні нових баз даних

Г. Для служби допоміжних служб, таких як SQL Server Agent

11. Для чого призначена системна база даних *tempdb*:

А. Для зберігання системної інформації про сервер і бази даних на сервері

Б. Для зберігання тимчасових даних

В. Як шаблон при створенні нових баз даних

Г. Для служби допоміжних служб, таких як SQL Server Agent

12. Для чого призначена системна база даних *model*:

А. Для зберігання системної інформації про сервер і бази даних на сервері

Б. Для зберігання тимчасових даних

В. Як шаблон при створенні нових баз даних

Г. Для служби допоміжних служб, таких як SQL Server Agent

ТЕМА 3. ЗАГАЛЬНІ ВІДОМОСТІ ПРО TRANSACT-SQL

1. Що з переліченого нижче є рядками змінної довжини:

А. Varchar

Б. Nchar

В. Nvarchar

Г. Varbinary

2. Яка максимальна довжина імені змінної:
   А. 32
   Б. 128
   В. 256
   Г. Не обмежена
3. Які з наступних тверджень вірні:
   А. Ім’я локальної змінної може починатися з латинської букви
   Б. Вказувати тип даних для змінної не обов'язково
   В. При оголошенні змінної можна її проініціалізувати прямо в операторі DECLARE
4. Що виконує констукція WAITFOR?
   А. Відкладає виконання операції на невизначений час
   Б. Чекає настання деякої події (наприклад, завершення виконання іншої процедури)
   В. Такої конструкції не існує в MS SQL Server 2008
5. Яке з наступних тверджень правильне:
   А. Конструкція /*...*/ дозволяє створювати багаторядкові коментарі
   Б. Якщо в вбудовану функцію (наприклад, Radians) передати ціле число, то вона поверне ціле число, а якщо передати дійсне, то і поверне дійсне число в якості результату
   В. Агрегатні і скалярні функції – це одне і те ж
6. Які типи функцій виділяють в MS SQL Server:
   А. Агрегатні
   Б. Скалярні
   В. Додаткові
   Г. Функції наборів записів

7. Чи можлива налагодження (рос. отладка) коду SQL:
   А. Так, Management Studio надає таку можливість

Б. Можна, але для цього слід застосовувати MS Visual Studio 2008
В. Можна, але тільки з використанням сторонніх розробок
Г. Ні, не існує засобів для налагодження SQL кода

ТЕМА 4. ВИБІРКА ДАНИХ

1. Що таке DML:
   А. Data Meta-Language
   Б. Direct Macro Language
   В. Data Macro Language
   Г. Data Manipulation Language
2. Які з наступних розділів можуть зустрічатися в операторі SELECT:
   А. GROUP BY
   Б. SORT
   В. HAVING
   Г. REVERSE
3. Що означає ключове слово ALL в операторі SELECT:
   А. З таблиці будуть вибрані усі записи, навіть ті, які не задовольняють умові в розділі WHERE
   Б. З таблиці будуть вибрані усі записи, в тому числі і однакові
   В. Ключове слово ALL більш не застосовується в операторі SELECT
   Г. Немає правильної відповіді
4. Для чого застосовується ключове слово AS в операторі SELECT:
   А. Для перейменування атрибута
   Б. Для повернення усіх записів
   В. Для порівняння атрибутів
   Г. Для отримання набору унікальних записів, які не повторюються
5. Яке ключове слово потрібно вказати для отримання набору унікальних записів, які не повторюються:
   А. ALL

Б. PERCENT

В. DISTINCT

Г. TOP

6. Чи можливо додати до імені псевдоніму недопустимі символи:
   А. Ні, ім’я псевдоніму повинно задовольняти стандартним правилам іменування об’єктів
   Б. Так, якщо ім’я псевдоніму помістити в квадратні дужки
   В. Так, якщо ім.’я псевдоніма помістити в круглі дужки
   Г. Так, якщо в списку вибірки вказати зірочку
7. Які з перерахованих нижче тверджень є правильними для оператора SELECT:
   А. Можна використовувати символ * для виборки всіх полів
   Б. Розділи FROM, WHERE, GROUP BY, HAVING можна використовувати в довільному порядку
   В. Ключове слово TOP призначене для виводу перших *n* записів
   Г. За замовчуванням SELECT виводе набір записів, який відсортований за першим атрибутом
8. Які види з’єднань декількох таблиць існують:
   А. Зовнішнє
   Б. Внутрішнє
   В. Послідовне
   Г. Вибіркове
9. Який оператор використовується для об’єднання декількох наборів результатів:
   А. JOIN
   Б. UNION
   В. SELECT
   Г. UNIQUE
10. Яка агрегатна функція повертає кількість значень у списку, відмінних від NULL:

А. Sum ([all | distinct] вираз)

Б. Avg ([all | distinct] вираз)

В. Count ([all | distinct] вираз |)

Г. Max ([all | distinct] вираз)

11. Підзапити бувають:

А. Внутрішні

Б. Вкладені

В. Зовнішні

Г. Зв’язані

12. Який розділ застосовується для групування записів за полями або виразами:

А. SORT BY

Б. GROUP BY

В. GROUP ON

Г. SORT ON

ТЕМА 5. ДОПОМІЖНІ ОБ’ЄКТИ БАЗИ ДАНИХ

1. Збережена процедура – це …:

А. Іменований набір операторів Transact-SQL, що зберігається в клієнтській базі даних

Б. Іменований набір операторів Transact-SQL, що зберігається на сервері

В. Немає правильно відповіді

2. Які види збережених процедур підтримує MS SQL Server:

А. Серверні процедури

Б. Системні процедури

В. Користувальницькі процедури

Г. Програмні процедури

3. За допомогою якої команди можна заборонити виконання команд збереженої процедури:

А. GRANT
Б. DENY
В. RESTRICT
Г. FORBID

4. Яка з перерахованих нижче збережених процедур виводить список параметрів та їх типів даних для зазначеної процедури:
А. sp_helptext Им’яПроцедури
Б. sp_stored_procedures
В. sp_help Им’яПроцедури
Г. Немає правильної відповіді
5. Що з переліченого нижче вірно:
А. Процедура може містити необмежену кількість операторів, окрім операторів створення наступних об'єктів: процедури, уявлення, правила, умовчання
Б. Кількість параметрів не обмежена
В. Створення процедури може виконати будь-який користувач
Г. Немає правильної відповіді
6. Яка команда відповідає за створення збереженої процедури:
А. EXECUTE PROC
Б. CREATE PROC
В. ALLOW PROC
Г. ALTER PROC
7. Без якого ключового слова процедура не передає вихідне значення:
А. OUTPUT
Б. INPUT
В. RETURN
Г. Немає правильної відповіді
8. Які дії передбачає під собою управління збереженими процедурами:
А. Видалення

Б. Зміна

В. Переміщення

Г. Перейменування

9. У чому полягають переваги уявлень:

А. Забезпечують конфіденційність інформації

Б. Спрощують подання даних

В. Представляють дані

Г. Управляють правами доступу до даних

10. Яка команда використовується для створення уявлення:

А. CREATE PROCEDURE

Б. CREATE TABLE

В. CREATE VIEW

Г. Немає правильної відповіді

11. Члени яких ролей володіють правом створювати уявлення:

А. sysadmin

Б. db_user

В. db_dlladmin

Г. db_owner

12. Яку команду необхідно виконати для видалення уявлення:

А. DROP VIEW

Б. DELETE VIEW

В. REMOVE VIEW

Г. Немає правильної відповіді

ТЕМА 6. СИСТЕМА БЕЗПЕКИ В БАЗАХ ДАНИХ

1. Які твердження щодо системи безпеки вірні для MS SQL Server 2008?

А. Пароль користувача ніколи не може бути порожнім

Б. MS SQL Server використовує тільки аутентифікацію Windows

В. При установці сервера створюється тільки один обліковий запис системного адміністратора

Г. Управляти системою безпеки можна за допомогою SQL команд і за допомогою графічного інтерфейсу Management Studio

2. Скільки постійних серверних ролей існує на сервері:

А. 1

Б. 7

В. 8

Г. 9

3. Якими правами володіє роль dbcreator:

А. Створення та видалення БД

Б. Може створювати і керувати логіками

В. Може виконувати будь-які дії на SQL сервері

Г. Управляє файлами баз даних

4. Якими правами володіє роль *diskadmin*:

А. Створення та видалення БД

Б. Може створювати і керувати логіками

В. Може виконувати будь-які дії на SQL сервері

Г. Управляє файлами баз даних

5. Що з перерахованого нижче призначене для створення облікових записів у MS SQL Server:

А. sp_addlogin

Б. CREATE LOGIN

В. ADD LOGIN

Г. sp_adduser

6. Що з перерахованого нижче призначене для призначення серверної ролі облікового запису:

А. CREATE SERVERROLE

Б. sp_addsrvrolemember

В. sp_setserverrole

Г. Нічого із зазначеного

7. Що з перерахованого нижче призначене для створення користувача в БД:
   А. CREATE LOGIN
   Б. sp_adduser
   В. sp_grantdbaccess
   Г. CREATE USER
8. Які з наступних тверджень вірні:
   А. Оператор GRANT використовується для надання доступу до об’єктів БД
   Б. Пріоритет заборони нижче пріоритету дозволу
   В. Неявне відхилення доступу діє тільки на тому рівні, на якому воно задане
   Г. Оператор DENY використовується для завдання неявного відхилення доступу

ТЕМА 7. СТРУКТУРА БАЗ ДАННЫХ В MS SQL SERVER

1. Члени яких ролей володіють правами для створення БД:
   А. serveradmin
   Б. sysadmin
   В. setupadmin
   Г. dbcreator
2. Які з наступних тверджень вірні:
   А. Збережена процедура sp_helpdb показує інформацію про базу даних та її налагодженнях
   Б. Збережена процедура sp_spaceused показує вільне місце на диску
   В. Параметр FILEGROWTH в операторі CREATE DATABASE задає крок збільшення файлу

Г. Параметр NAME в операторі CREATE DATABASE задає ім’я бази даних

3. Яка команда використовується для управління файлами, файлами БД, що існують та для додавання нових:
   А. CHANGE DATABASE
   Б. MODIFY DATABASE
   В. ALTER DATABASE
   Г. ALTER FILES
4. Чи можливо перейменувати існуючу БД:
   А. Так, в будь-який момент часу
   Б. Так, але тільки якщо БД знаходиться в однокористувацькому режимі
   В. Так, але тільки якщо цю дію виконує член ролі *sysadmin*
   Г. Ні, так як після створення БД її ім’я змінити неможливо
5. Яка збережена процедура використовується для зміни власника БД:
   А. sp_changedbowner
   Б. sp_setdbowner
   В. sp_alterdbowner
   Г. Власником БД є той користувач, який її створив
6. Який мінімальний розмір може мати БД при створенні:
   А. 512Кб
   Б. 1Мб
   В. Розмір не може бути меншим, ніж поточний розмір БД model
   Г. Розмір не може бути менший, ніж поточний розмір БД master
7. Що з перерахованого застосовується для стиснення всієї БД:
   А. sp_shrinkdb
   Б. ALTER DATABASE
   В. DBCC SHRINKFILE
   Г. DBCC SHRINKDATABSE

8. Що з перерахованого застосовується для стиснення окремого файлу:
   А. sp_shrinkfile
   Б. ALTER DATABASE
   В. DBCC SHRINKFILE
   Г. DBCC SHRINKDATABSE
9. Які з наступних тверджень правильні:
   А. Параметр EMPTYFILE в операції DBCC SHRINKFILE видаляє файл з БД, якщо в ньому не міститься даних
   Б. Оператор BACKUP дозволяє здійснювати резервне копіювання як самої БД, та і її журналу
   В. При виконанні команди RESTORE можна дозволити доступ до відновленої БД тільки її власнику

ТЕМА 8. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ

1. Які з наступних тверджень правильні:
   А. Домен – це набір атрибутів у відношенні
   Б. Кортеж відповідає екземпляру запису
   В. Атрибути відповідають полям таблиці
2. Якими властивостями володіють відношення:
   А. Немає однакових кортежів
   Б. Всі кортежі і атрибути впорядковані
   В. Всі значення атрибутів є неподільними
   Г. Ніякі два атрибути не можуть бути визначені на одному домені
3. Якими властивостями повинен володіти потенційний ключ:
   А. Повинен складатися тільки з одного атрибута
   Б. Ненадмірність
   В. Унікальність
   Г. Ніякими з перерахованих вище
4. Що з переліченого нижче вірно:

А. Первинний ключ відповідає одному з потенційних ключів відношення

Б. Якщо існує зовнішній ключ FK у відношенні R2, то існує відношення R1 з потенційним ключем, всі значення якого присутні у FK

В. Зовнішній ключ у відношенні R2 може посилатися на потенційний ключ R2

5. Для яких операцій необхідно передбачити виконання компенсуючих дій для підтримки цілісності даних:

А. Вставка

Б. Видалення

В. Зміна

Г. Вибірка

6. Які компенсуючі операції можуть застосовуватися для збереження цілісності даних:

А. Обмеження виконання операції

Б. Каскадування

В. Жодні компенсуючі операції не потрібні, тому що якщо відношення нормалізовані, то цілісність даних буде забезпечуватися автоматично

7. Що таке NULL-значення:

А. Числовий нуль

Б. Порожній рядок

В. Спеціальний маркер для позначення відсутності інформації

Г. Будь-яке значення будь-якого типу прийняте в даній БД для позначення відсутності інформації

8. Які твердження щодо NULL-значень вірні:

А. Можна використовувати для первинних ключів

Б. Можна використовувати для альтернативних ключів

В. Можна використовувати для зовнішніх ключів

Г. Можна використовувати у всіх випадках крім, трьох вищеперелічених.

ТЕМА 9. ОПЕРАТОРИ РЕЛЯЦІЙНОЇ АЛГЕБРИ

1. Хто вперше визначив реляційну алгебру:
   А. Р. Бойс
   Б. Е. Кодд
   В. Р. Фейгін
   Г. К. Дейт
2. Виберіть традиційні операції з множинами, модифіковані для таблиць (відносин):
   А. Об’єднання
   Б. Віднімання
   В. Добуток
   Г. Ділення
3. Які реляційні операції є спеціальними:
   А. Вибірка
   Б. Множення
   В. Проекція
   Г. Ділення

4. Що з перерахованого є операціями поновлення:
   А. Вставка запису
   Б. Переміщення запису
   В. Видалення запису
   Г. Оновлення запису
5. Що буде результатом операції об'єднання:
   А. Об’єднання атрибутів таблиць за деяким полем
   Б. Об’єднання кортежів двох таблиць

В. Всі кортежі першого відношення, які також входять в друге відношення

Г. Нічого з наведеного

6. Що буде результатом операції перетину:

А. Об’єднання атрибутів таблиць за деяким полем

Б. Об’єднання кортежів двох таблиць

В. Всі кортежі першого відношення, які також входять у друге відношення

Г. Нічого з наведеного

7. Що буде результатом операції ділення:

А. Об’єднання атрибутів таблиць за деяким полем

Б. Об’єднання кортежів двох таблиць

В. Всі кортежі першого відношення, які також входять у друге відношення

Г. Нічого з наведеного

8. Які з наступних тверджень є вірними:

А. Операція проекції фактично є вибором деяких атрибутів з вихідного відношення

Б. Результатом операції вибірки будуть всі кортежі вихідного відношення, що відповідають заданій умові

В. Операція об’єднання може бути проведена над таблицями з різною кількістю атрибутів

Г. Операція з’єднання може бути проведена над таблицями з різною кількістю атрибутів

ТЕМА 10. ПЕРШІ НОРМАЛЬНІ ФОРМИ

1. Якими властивостями повинна володіти неприведена множина функціональних залежностей:

А. Жодна функціональна залежність з множини не може бути видалена без втрати інформації про зв’язки.

Б. Права частина функціональної залежності є підмножиною лівої частини

В. Залежна частина кожної функціональної залежності містить тільки один атрибут

Г. Залежна частина кожної функціональної залежності містить множину атрибутів

Д. Детермінант кожної функціональної залежності є неприведеним

2. Які правила сформулював Армстронг:

А. Правило тривіальності

Б. Правило рефлективності

В. Правило доповнення

Г. Правило агрегації

Д. Правило декомпозиції

3. Відношення знаходиться в першій нормальній формі тоді і тільки тоді, коли:

А. Всі використовувані домени містять тільки скалярні значення, тобто значення всіх полів відношення повинні бути неподільні

Б. Кожен неключовий атрибут неприводимо залежить від первинного ключа

В. Кожен неключовий атрибут не є транзитивно залежним від первинного ключа

Г. Немає правильної відповіді

4. Відношення знаходиться в другій нормальній формі тоді і тільки тоді, коли:

А. Всі використовувані домени містять тільки скалярні значення

Б. Воно знаходиться в 1НФ і кожен неключовий атрибут неприводимо залежить від первинного ключа

В. Воно знаходиться в 1НФ і кожна його нетривіальна і неприводима зліва функціональна залежність має в якості свого детермінанта деякий потенційний ключ

Г. Немає правильної відповіді

5. Відношення знаходиться в третій нормальній формі тоді і тільки тоді, коли:

А. Воно знаходиться в 2 НФ і кожна його нетривіальна і неприводима зліва функціональна залежність має в якості свого детермінанта деякий потенційний ключ

Б. Воно знаходиться у 2НФ і кожен неключових атрибут не є транзитивно залежним від первинного ключа

В. Воно знаходиться в 2 НФ і кожен неключових атрибут неприводимо залежить від первинного ключа

Г. Немає правильної відповіді

6. Відношення знаходиться в нормальній формі Бойса-Кодда тоді і тільки тоді, коли:

А. Кожна його нетривіальна і неприводима зліва функціональна залежність має в якості свого детермінанта деякий потенційний ключ

Б. Кожна його нетривіальна і неприводима зліва функціональна залежність має в якості свого детермінанта деякий первинний ключ

В. Кожен його неключових атрибут неприводимо залежить від деякого потенційного ключа

7. Які з наступних тверджень є правильними:

А. Відношення, що знаходиться в третій нормальній формі, автоматично знаходиться в першій і другій нормальних формах

Б. Головною операцією для нормалізації відношень є операція з’єднання

В. НФБК є уточненням третьої нормальної форми

Г. Якщо відношення знаходиться тільки в 1НФ, але не знаходиться у 2НФ і 3НФ, то можна говорити про надмірність інформації

8. Що означає X → Y:

А. X функціонально визначає Y

Б. X функціонально залежить від Y

В. Y є підмножиною X

Г. X є підмножиною Y

ТЕМА 11. ЧЕТВЕРТА ТА П'ЯТА НОРМАЛЬНІ ФОРМИ

1. Відношення R знаходиться в четвертій нормальній формі тоді і тільки тоді, коли:

А. У разі існування таких підмножин A і B атрибутів цього відношення R, для яких виконується нетривіальна багатозначна залежність A →→ B, всі атрибути відношення R також транзитивно залежать від атрибута A.

Б. У разі існування таких підмножин A і B атрибутів цього відношення R, для яких виконується нетривіальна багатозначна залежність A →→ B, всі атрибути відношення R також функціонально залежать від атрибута A.

В. Відношення R знаходиться в НФБК і всі багатозначні залежності в цьому відношенні представляють собою функціональні залежності від його зовнішніх ключів.

Г. Немає правильної відповіді

2. Відношення R знаходиться в п'ятій нормальній формі тоді і тільки тоді, коли:

А. Кожна нетривіальна залежність з'єднання в відношенні R визначається первинним ключем (або ключами) R.

Б. Кожна нетривіальна залежність з'єднання в відношенні R визначається потенційним ключем (або ключами) R.

В. Відношення R знаходиться в НФБК і всі багатозначні залежності в цьому відношенні представляють собою функціональні залежності від його потенційних ключів.

Г. Відношення R не містить аномалій.

3. Які з наступних нижче тверджень є правильними:

А. Будь-яке відношення можна розбити на два без втрат

Б. Існують відношення, для яких не можна виконати декомпозицію без втрат на дві проекції, але які можна розбити на більшу кількість відношень

В. П’ята нормальна форма вважається остаточною

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

4. Які з тверджень правильні відносно доменно-ключової нормальної форми:

А. Будь-яке відношення в ДКНФ знаходиться в 5НФ

Б. Будь-яке відношення можна привести до ДКНФ

В. ДКНФ заснована на обмеженнях доменів і ключів

Г. Будь-яке відношення в 5НФ автоматично знаходиться і в ДКНФ

5. Що з наступного правильно для опису процесу нормалізації:

А. Існують чіткі критерії, які забезпечують один єдиний спосіб приведення відношення в 5НФ

Б. Не будь-яка надмірність даних може бути усунена за допомогою декомпозиції

В. Існують ситуації, коли не слід виконувати нормалізацію до кінця

Г. У процесі нормалізації слід позбутися всіх функціональних і багатозначних залежностей, а також від залежностей з’єднання

6. Що означає запис A →→ B:

А. А функціонально залежить від В

Б. А функціонально визначає В

В. А багатозначно залежить від В

Г. А багатозначно визначає В

7. Яка операція лежит в основі процеса декомпозиції:
   А. Ділення
   Б. Проекції
   В. Вибірки
   Г. З’єднання
8. Які з наступних нижче тверджень є правильними:
   А. П’ята нормальна форма заснована на функціональних залежностях
   Б. Четверта нормальна форма заснована на багатозначних залежностях
   В. Знаходження в 4НФ передбачає обов’язкове знаходження в НФБК
   Г. Нормальна форма типу «вибірка–об’єднання» заснована на операції проекції

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Бен-Ган И. Microsoft SQL Server 2012. Высокопроизводительный код T-SQL. Оконные функции / И. Бен-Ган. – М.: Курс, 2012. – 430 с.
2. Бондарь А. Microsoft SQL Server 2014 / А. Бондарь – СПб: БХВ-Петербург, 2015. – 592 с.
3. Вийера Р. Программирование баз данных Microsoft SQL Server 2005 для профессионалов / Р. Вийера. – СПб.: Питер, 2008. – 640 с.
4. Вишневский А.В. Microsoft SQL Server 2008. Эффективная работа / А.В. Вишневский. – СПб.: Питер, 2009. – 541 с.
5. Гайджарги В. І. Основи проектування та використання баз даних: навчальний посібник / В.І. Гайджарги, О.А. Дацюк. – К.: ІВЦ «Видавництво «Політехніка», 2004. – 256 с.
6. Гайна Г. А. Основи проектування баз даних : навчальний посібник / Г. А. Гайна. – К.: ондор, 2008. – 200 с.
7. Дейт К. Дж. Введение в системы баз данных. – М.: Вильямс, 2008.
8. Дибетта П. Знакомство с Microsoft SQL Server 2005. – М.: Русская редакция, 2005.
9. Жилинский А. Самоучитель Microsoft SQL Server 2008. – СПб.: БХВ-Петербург, 2009.
10. Каленик А. И. Использование новых возможностей Microsoft SQL Server 2005. – М.: Русская редакция; СПб.: Питер, 2006.
11. Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг. – М.: Вильямс, 2003.
12. Пасічник В. В. Сховища даних : навчальний посібник / В. В. Пасічник, Н.Б. Шаховська ; За ред. В.В. Пасічника. - Львів : Магнолія 2006, 2008. – 496 с.
13. Петкович Д. Microsoft SQL Server 2008. Руководство для начинающих / Д. Петкович. – СПб: БХВ-Петербург, 2012. – 752 с.

14. Погромська Г.С. Побудова запитів на мові SQL: навчальний посібник / Г. С. Погромська. – Миколаїв: Іліон, 2015. – 136 с.
15. Хендерсон К. Профессиональное руководство по SQL Server: хранимые процедуры, XML, HTML / К. Хендерсон. - СПб. : Питер, 2005. – 620 с.
16. Lobel L., Brust A. J., Forte S. Programming Microsoft SQL Server 2008. – Microsoft Press, 2008.
17. Microsoft SQL Server 2008 Analysis Services. OLAP многомерный анализ данных / Под общ. ред. А.Б. Бергера, И.В. Горбач. – СПб. : БХВ-Петербург, 2007. – 928 с.
18. SQL Server 2008. Ускоренный курс для профессионалов / Р. Уолтерс, М. Коулс, Ф. Феррачати, Р. Рей, Д. Фармер. – М.: Вильямс, 2009 – 768 с.

Навчально-методичне видання

ПОГРОМСЬКА Ганна Сергіївна
МАХРОВСЬКА Наталя Анатоліївна

БАЗИ ДАНИХ: ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ

Навчально-методичний посібник

Підп. до друку . Формат 60x84 1/16. *Папір офсетний.*
Гарнітура «Times New Roman». Друк ризографічний.
Ум. друк. арк. . Обл.-вид. арк.
Тираж 100 пр. Зам. №

ВИГОТОВЛЮВАЧ
Поліграфічне підприємство

.