13

Т. О. Савчук

ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ

Частина 2

Міністерство освіти і науки України
Вінницький національний технічний університет

Т. О. Савчук

ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ

Частина 2
Лабораторний практикум

Вінниця
ВНТУ
2016

УДК 004.65
ББК 32.973.202
С13

Рекомендовано до друку Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 11 від 26.06.2014 р.).

Рецензенти
В. М. Лисогор, доктор технічних наук, професор, ВНАУ
Р. Н. Квєтний, доктор технічних наук, професор
О. О. Білик, кандидат технічних наук, доцент, Вінницький обласний інститут післядипломної освіти педагогічних працівників

Савчук, Т. О.
С13 Організація баз даних та знань. Частина 2 : лабораторний практикум / Т. О. Савчук. – Вінниця : ВНТУ, 2016. – 85 с.

У лабораторному практикумі наведено порядок виконання лабораторних досліджень з теоретичними рекомендаціями щодо їх виконання. При цьому виконання дослідницьких дій пояснено прикладами. Запропоновано перелік питань для перевірки набутих при виконанні лабораторних робіт теоретичних знань та практичних навичок.

УДК 004.65
ББК 32.973.202

ЗМІСТ

Вступ....4
Установлення Microsoft SQL Server....6
Лабораторна робота № 1.
Основи мови SQL. Основні типи даних СУБД MS SQL Server....16
Лабораторна робота № 2.
SQL MANAGEMENT STUDIO:
прості запити для вибирання даних. Арифметичні та
символьні функції....32
Лабораторна робота № 3.
SQL MANAGEMENT STUDIO:
Зв'язки між таблицями. Ключі та обмеження....39
Лабораторна робота № 4.
SQL MANAGEMENT STUDIO:
Вибирання даних за допомогою SQL запитів. Групові операції....51
Лабораторна робота № 5.
SQL MANAGEMENT STUDIO:
Вибирання даних за допомогою SQL запитів. Оператори об'єднання....60
Лабораторна робота № 6.
SQL MANAGEMENT STUDIO:
Вибирання даних за допомогою SQL запитів. Вкладені
та корельовані запити....69
Лабораторна робота № 7.
SQL MANAGEMENT STUDIO:
Вибирання даних за допомогою SQL запитів. Об'єднання
результатів кількох вибірок....74
Лабораторна робота № 8.
Створення індексів у середовищі SQL MANAGEMENT STUDIO....78

ВСТУП

Єдність законів опрацювання інформації в системах різної природи (фізичних, економічних, біологічних і т. п.) є фундаментальною основою теорії інформаційних процесів, що визначають її значимість і специфічність. Об'єктом вивчення цієї теорії є інформація – поняття, що існує "саме по собі" поза зв'язком із конкретною областю знань, у якій воно використовується.

Ця обставина накладає значний відбиток на всю інформатику як науку про організацію комп'ютерних інформаційних систем. Такі системи можуть використовуватися в найрізноманітніших предметних областях, додаючи до них "власні правила гри", свої закономірності, обмеження та водночас нові можливості організації бізнесу. У цьому плані неможливо переоцінити такі властивості інформації як доступність, своєчасність, комерційну цінність, захищеність від несанкціонованих втручань, надійність.

Інформаційні ресурси в сучасному суспільстві відіграють вагомішу роль, ніж ресурси матеріальні. Знання "кому", "коли" і "де" продати товар може цінуватися не менше, ніж сам товар, і в цьому плані динаміка розвитку суспільства свідчить про те, що на "вагах" матеріальних і інформаційних ресурсів, саме останні починають домінувати, причому тим сильніше, чим відкритішою є організація та чим розвинутішими є засоби комунікації, потужнішими є самі інформаційні потоки та обмін ними.

З позицій ринку, інформація давно вже стала товаром. Ця обставина потребує інтенсивного розвитку практики і теорії комп'ютеризації суспільства. Комп'ютер як інформаційне середовище не тільки дозволив здійснити якісний стрибок в організації промисловості, науки і ринку, але він визначив нові області промисловості: обчислювальну техніку, телекомунікації, програмні продукти.

Тенденції комп'ютеризації суспільства привели до появи нових професій, пов'язаних з обчислювальною технікою і різними категоріями користувачів електронно-обчислювальних машин (ЕОМ). Якщо в 60 – 70-і роки в цій сфері домінували фахівці з обчислювальної техніки (інженери-електроніки і програмісти), що створюють нові засоби обчислювальної техніки і нові пакети прикладних програм, на сьогодні інтенсивно розширюється категорія користувачів ЕОМ – представників найрізноманітніших областей знань, що не є фахівцями в прикладних комп'ютерних галузях, але які уміють використовувати IBM PC для розв'язання задач.

Користувач ЕОМ (або кінцевий користувач) повинен знати загальні принципи організації інформаційних процесів у комп'ютерному середовищі, вміти вибирати потрібні йому інформаційні системи й технічні засоби, швидко застосовувати їх до своєї предметної області з

огляду на інтенсивний розвиток обчислювальної техніки та ступінь насиченості ринку програмних продуктів.

Мінімум знань щодо організації комп'ютерних систем визначають комп'ютерну грамотність користувача. Не існує чітко визначених меж щодо пояснення цього поняття – кожний користувач визначає їх для себе сам, але, водночас, відсутність такої грамотності робить сьогодні неможливим доступ до багатьох вузькоспеціалізованих професій.

Основні ідеї сучасної інформаційної технології базуються на концепції баз даних (БД). Відповідно до цієї концепції, основою інформаційної технології є дані, що організовані у структурні одиниці з метою адекватного відображення реального світу і задоволення інформаційних потреб користувачів.

Збільшення обсягу і структурної складності збережених даних, розширення кола користувачів інформаційних систем сприяло формуванню вимог до створення зручних загальносистемних засобів інтеграції збережених даних і управління ними. Це привело до появи промислових систем управління базами даних (СУБД) – спеціалізованих програмних засобів, призначених для організації і маніпуляції БД. Систему, яка забезпечує створення, управління і застосування баз знань, можна розглядати як інструментальну або прикладну систему з конкретною прикладною базою знань. Існує тісний взаємозв'язок між технологією БД і системою БД, з одного боку, і технологією систем баз знань, з іншого. Виникла тенденція "інтелектуалізації" систем БД. На зовнішньому рівні їх архітектури реалізують різноманітні семантичні моделі знань, створюють "дружні" інтерфейси для користувачів, хоча традиційні СУБД є необхідною складовою частиною інструментарію управління даними в системах баз знань.

В лабораторному практикумі наведені основи організації баз даних та управління ними на концептуальному та даталогічному рівнях.

УСТАНОВЛЕННЯ MICROSOFT SQL SERVER

Мета: отримати навички зі встановлення Microsoft SQL Server, отримати інформацію про СУБД Microsoft SQL Server та про основні редакції випусків.

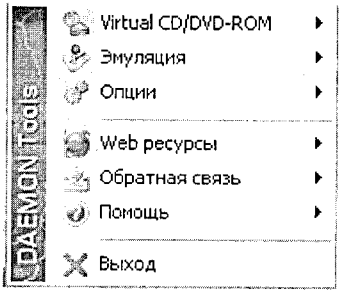
Порядок установлення

1. Підготовка до установлення SQL Server.

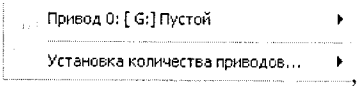
1.1 Для розпакування образу диска потрібно використати DAEMON Tools Lite.

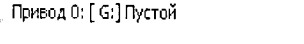
1.2 Натиснути правою кнопкою миші на 

, з'явиться низхідний список



обираємо 

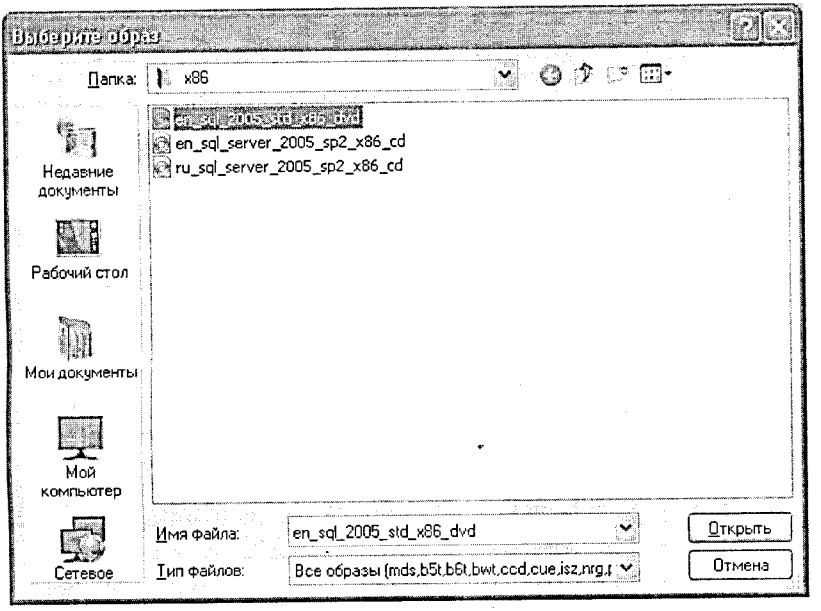
, після чого з'явиться низхідний список 

, де потрібно обрати 

. В результаті цього з'явиться низхідний список 

, де обирається

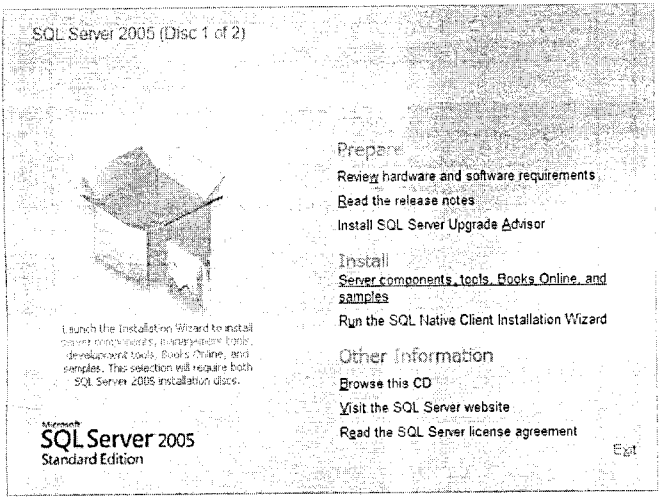
, після чого з'явиться вікно, де потрібно вказати шлях до потрібного образу.



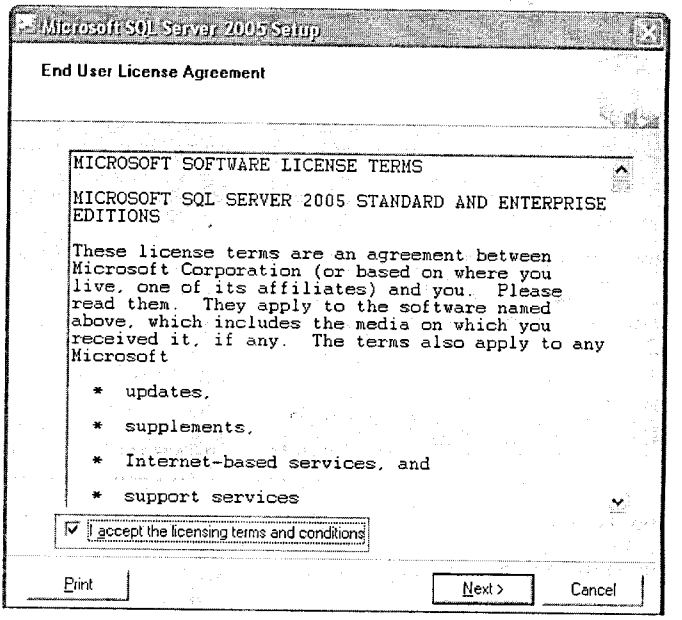
1.3. Обрати образ та натиснути [Открыть] – відкриється вікно початку установлення SQL Server 2005.

2. Установлення SQL Server 2005.

2.1. Після виконання дій попереднього пункту з'явиться вікно:

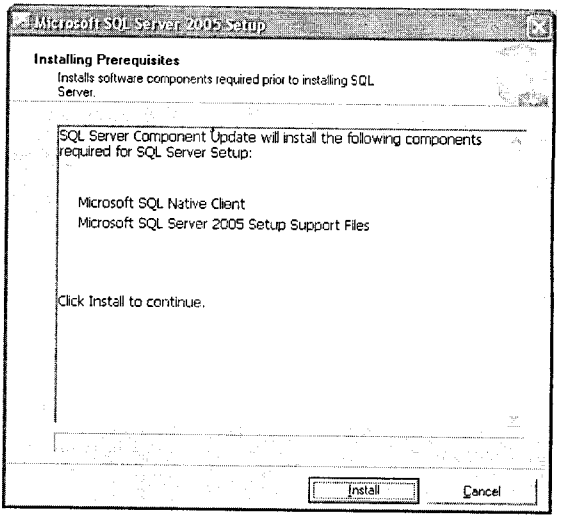


2.2. Після обрання посилання Server components, tools, Books Online, and samples з'явиться вікно, де потрібно погодитись із ліцензійними зобов'язаннями

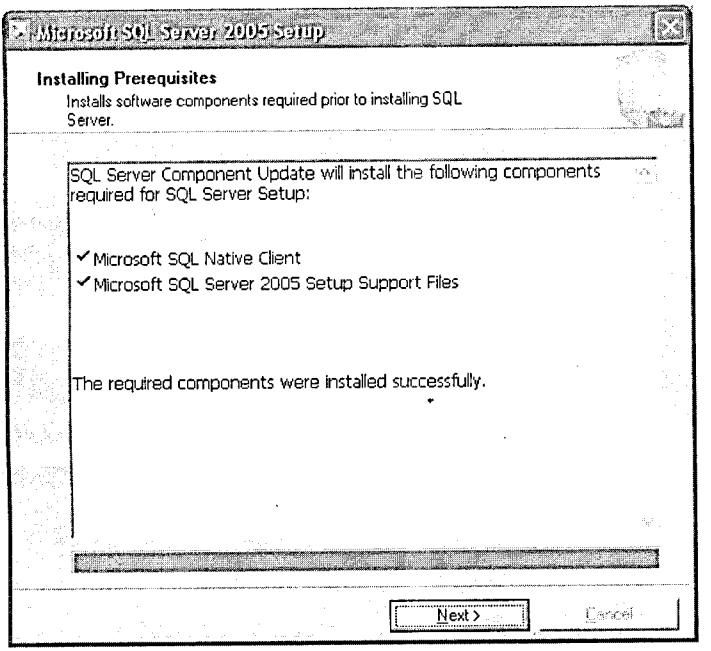


та перейти до наступного кроку установлення натисненням Next>.

З'явиться вікно:



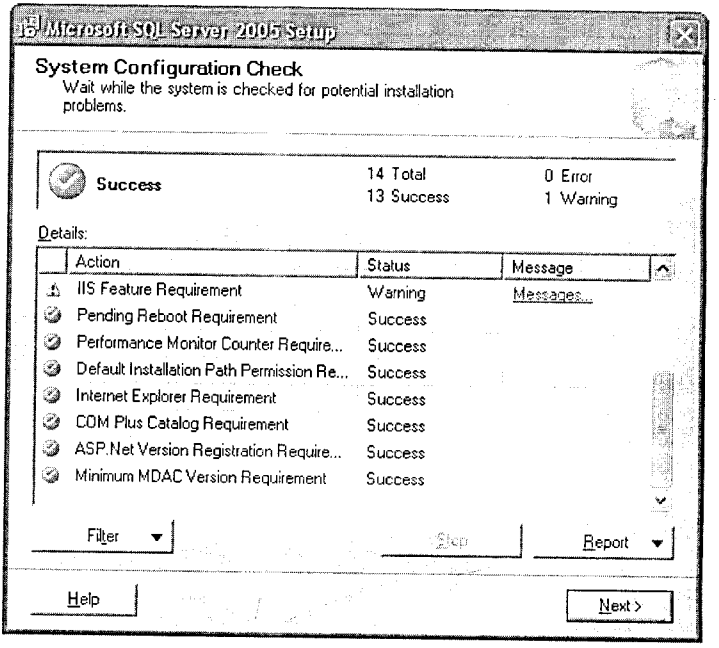
Слід натиснути кнопку Install. Через деякий час у вікні



натискаємо кнопку Next>.

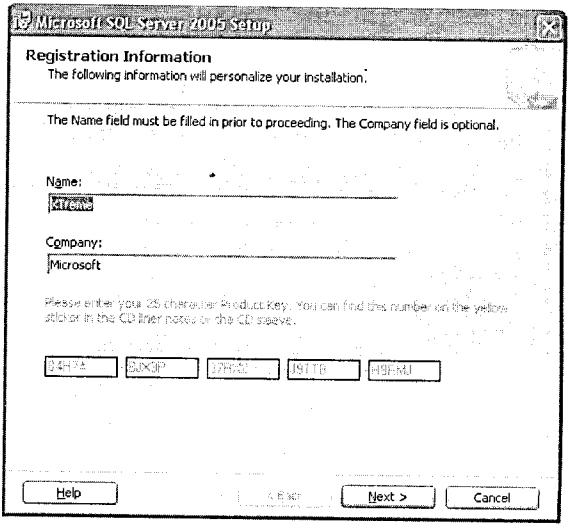
Далі на екрані з'явиться вікно:

Натискаємо Next>. На екрані з'явиться вікно:



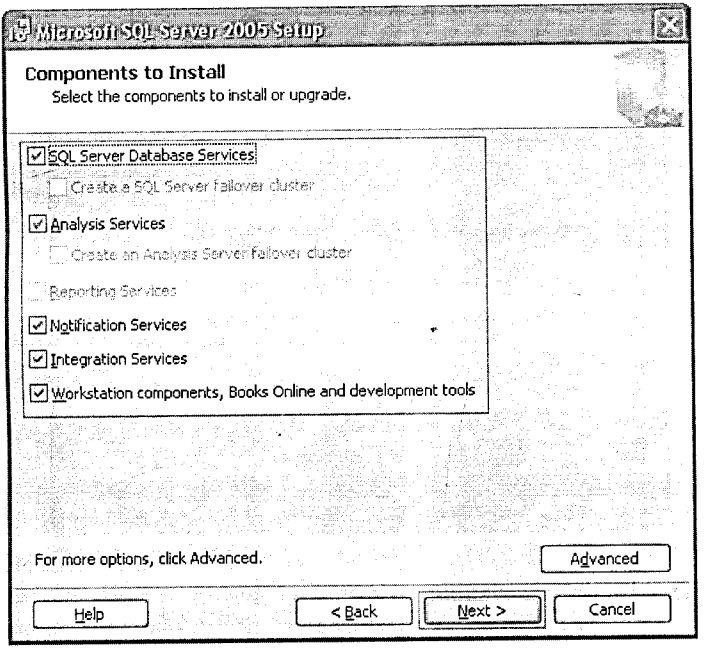
Обираємо Next>.

На екрані з'явиться вікно:

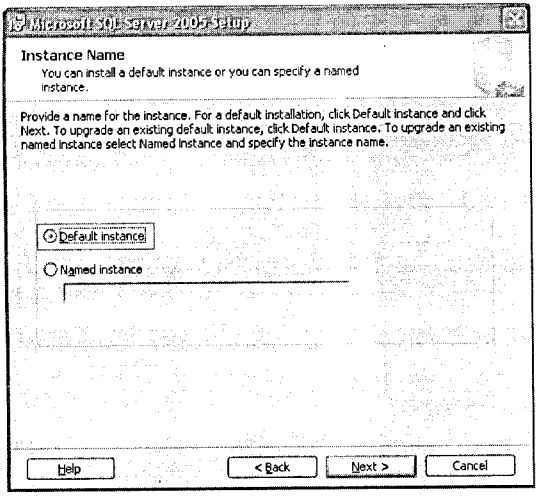


Натискаємо Next>.

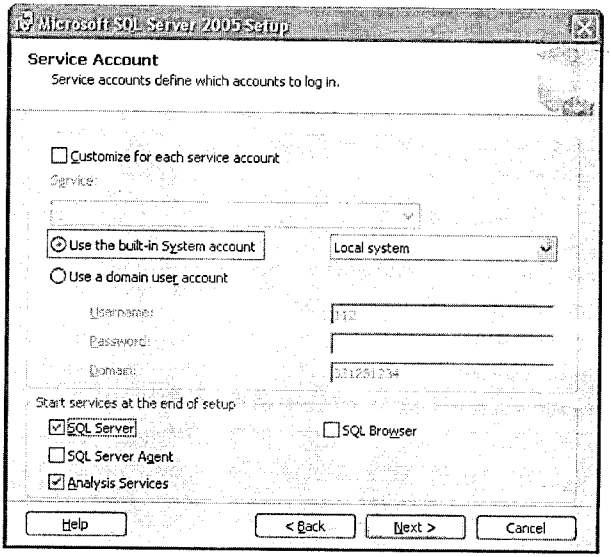
На екрані з'явиться вікно, в якому потрібно вибрати всі підпункти і натиснути Next.



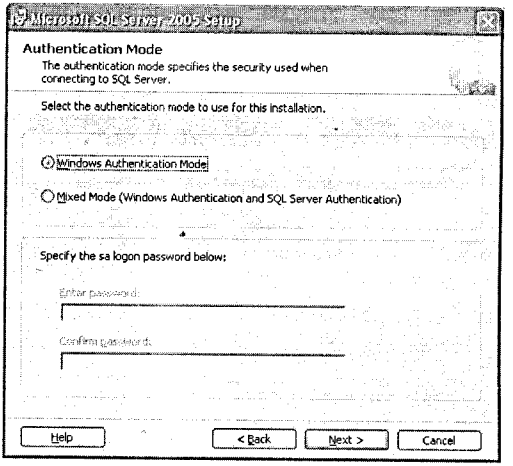
З'явиться вікно, в якому потрібно вибрати (•) Default instance натиснути Next.



На екрані з'явиться вікно, де потрібно обрати ○ Use the built-in System account і натиснути Next.

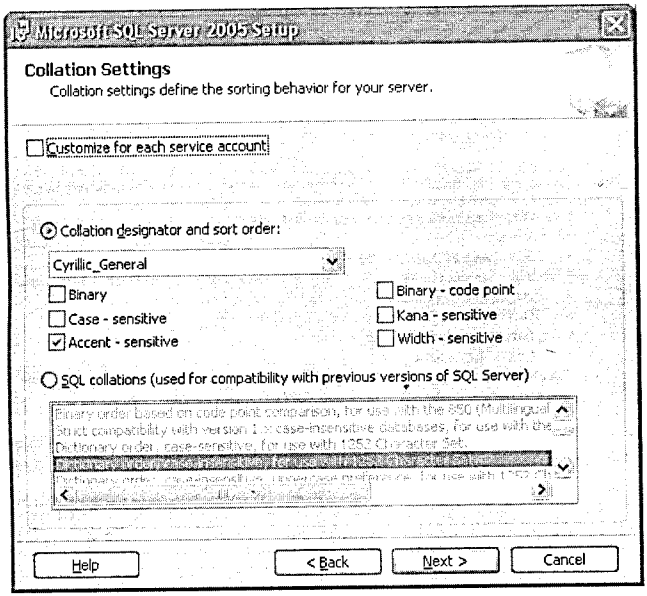


На екрані з'явиться вікно:



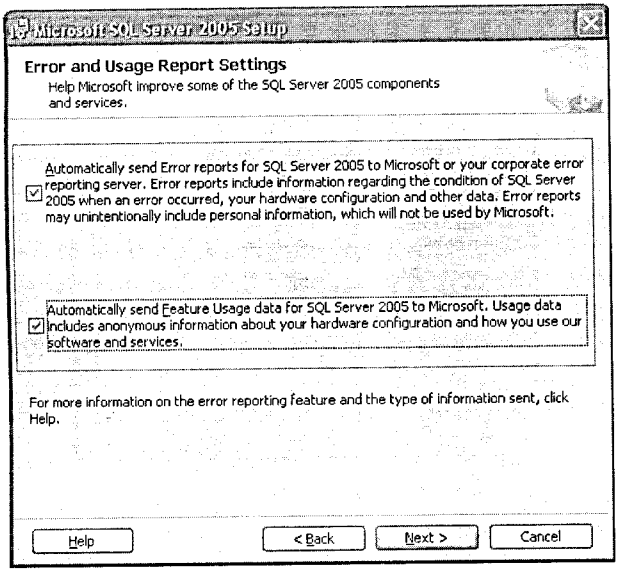
Слід натиснути Next>.

На екрані з'явиться вікно:

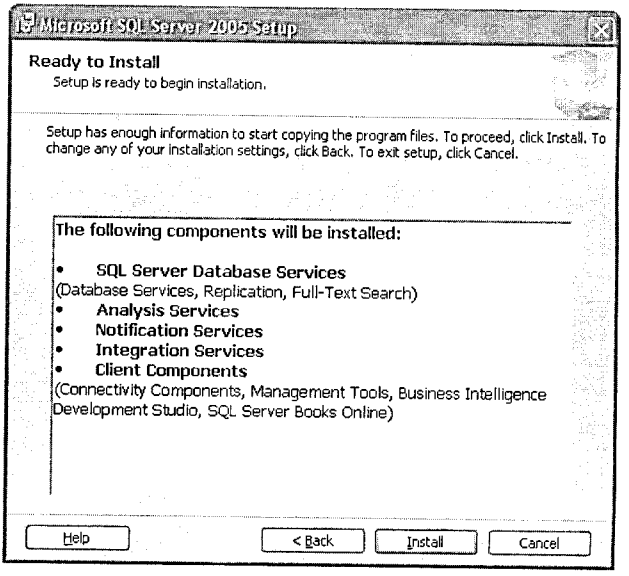


Натискаємо Next>.

На екрані з'явиться вікно, в якому потрібно вибрати всі підпункти і натиснути Next.



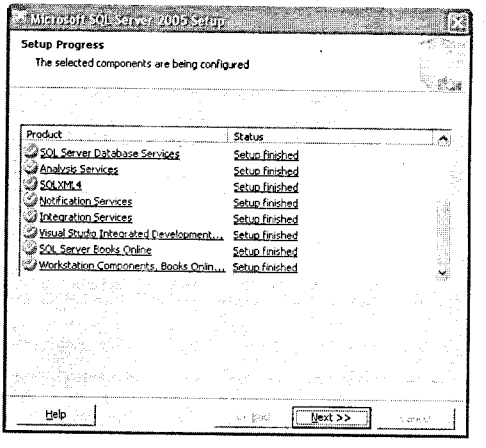
На екрані з'явиться вікно:



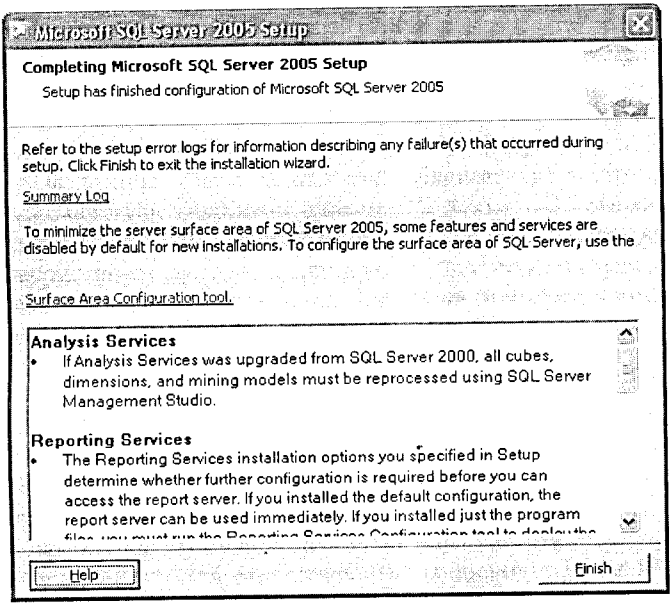
Слід обрати Install.

Установлення триватиме декілька хвилин. Якщо будуть з'являтись сервісні повідомлення, потрібно в кожному з них натискати кнопку "ОК".

Після закінчення процесу установлення з'явиться вікно, де слід обрати Next>>.



На екрані з'явиться вікно:



Установлення закінчено - натискаємо Finish.

Після перезавантаження системи SQL Server буде готовий до використання.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Дейт К. Дж. Введение в системы баз данных / Дейт К. Дж. – [8-е изд.]. – М. : Изд-во "Вильямс ", 2006. – 1328 с.
1. Дейт К. SQL и реляционная теория. Как грамотно писать код на SQL / Дейт К. – М. : Изд-во "Символ-Плюс", 2010. – 480 с.
2. Роберт Виейра Программирование баз данных Microsoft SQL Server 2008. Базовый курс / Роберт Виейра. – М. : Изд-во: "Диалектика-Вильямс", 2009. – 816 с.
3. Бен-Ган И. Microsoft SQL Server 2008. Основы T-SQL / Бен-Ган И. – Санкт-Петербург : Русская редакция, 2009. – 432 с.
4. Тернстрем Т. Microsoft SQL Server 2008. Разработка баз данных. Учебный курс Microsoft, экзамен 70-433 (+ CD) / Тернстрем. Т., Вебер Э., Хотек М. – М. : Изд-во "Русская Редакция", 2010. – 496 с.
5. Петкович Д. Microsoft SQL Server 2008. Руководство для начинающих / Петкович Д. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 752 с.

ЛАБОРАТОРНА РОБОТА № 1
ОСНОВИ МОВИ SQL. ОСНОВНІ ТИПИ ДАНИХ СУБД MS SQL SERVER

Мета: отримати загальні відомості про мову запитів SQL. Вивчити типи даних MS SQL Server. Отримати навички використання програми MS SQL Management Studio для виконання запитів.

Порядок виконання роботи

1. Відповідно до обраної предметної області визначити основні дані, що будуть зберігатись в базі даних. Спроектувати 2–3 таблиці не менш, ніж з 4 атрибутами кожна, що будуть вміщувати інформацію про об'єкти із предметної області. Обґрунтувати вибір типів даних кожного атрибуту.
2. Створити на сервері базу даних для виконання лабораторної роботи.
3. Відкрити вікно виконання запитів в програмі SQL Management Studio.
4. Створити запит на створення 2–3 таблиць за допомогою оператора CREATE TABLE відповідно до обраної предметної області та виконати його.
5. Переглянути результат виконання запиту у вікні "Обозреватель объектов" (мають бути створені декілька таблиць; інакше необхідно проаналізувати помилки при виконанні запиту та виправити їх).
6. Зберегти файл із запитом.
7. Видалити створені таблиці за допомогою оператора DROP TABLE.
8. Зберегти файл із запитом.
9. Оформити звіт.

Теоретичні відомості

1 Мова SQL

SQL (англ. Structured query language – мова структурованих запитів) – декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних і її модифікація, система контролю за доступом до бази даних. Сама по собі мова SQL не є ні системою керування базами даних, ні окремим програмним продуктом. SQL може формувати інтерактивні запити або, як вбудована в прикладні програми, виступати як інструкції для керування даними.

SQL – це діалогова мова програмування для здійснення запиту і внесення змін до бази даних, а також керування базами даних. Багато баз даних підтримує SQL з розширеннями до стандартної мови. SQL також включає CLI (Call Level Interface) для доступу і керування базами даних дистанційно.

1.1 Історія SQL

На початку 1970-х років в одній з дослідницьких лабораторій компанії IBM була розроблена експериментальна реляційна СУБД System/R, для якої була створена спеціальна мова SEQUEL, що дозволяла відносно просто управляти даними в цій СУБД. Абревіатура SEQUEL розшифровувалася як англ. Structured English QUEry Language - "структурована англійська мова запитів".

Метою розробки було створення простої непроцедурної мови, якою зміг би скористатися будь-який користувач, що може не мати навичок програмування. Розробкою мови запитів займалися Дональд Чемберлін, Рей Бойс, Пет Селінджер (вартісний оптимізатор (англ. cost-based optimizer), Реймонд Лорі (компілятор запитів).

Варто зазначити, що SEQUEL не була єдиною мовою подібного призначення. У Каліфорнійському університеті в Берклі була розроблена некомерційна СУБД Ingres (основа СУБД PostgreSQL), яка була реляційною СУБД, але використовувала свою власну мову QUEL.

Першими СУБД, що підтримують нову мову SQL, стали в 1979 році Oracle V2 для машин VAX від компанії Relational Software Inc., що згодом стала компанією Oracle, і System/38 від IBM, основана на System/R.

В 1986 році [2] перший стандарт мови SQL був прийнятий ANSI (American National Standards Institute), в 1987 році – ISO (Міжнародною організацією зі стандартизації). Подальший розвиток мови постачальниками СУБД привів до ухвалення в 1992 році нового розширеного стандарту. Наступним стандартом став SQL:1999. В даний час діє стандарт, прийнятий у 2003 році з модифікаціями, внесеними пізніше.

1.2 Стандарти SQL

Повна історія версій стандартів-ревізій SQL у таблиці 1.1:

Таблиця 1.1 – Історія введення стандартів SQL

Рік	Назва	Коментар
1986	SQL-86, SQL-87	Вперше оприлюднено ANSI. Ратифіковано ISO в 1987.
1989	SQL-89	Невеликі зміни.
1992	SQL-92, SQL-2	Вагомі зміни.
1999	SQL:1999, SQL-3	Додано регулярні вирази, рекурсивні запити, тригери та деякі об'єктно-орієнтовані нововведення.
2003	SQL:2003	XML-залежні нововведення.

1.3 Переваги

1.3.1 Незалежність від конкретної СУБД

Незважаючи на наявність діалектів і відмінностей в синтаксисі, тексти SQL-запитів, що містять DDL і DML, можуть бути досить легко перенесені з однієї СУБД в іншу. Існують системи, розробники яких покладаються на застосування кількох СУБД (наприклад: система електронного документообігу Documentum може працювати як з Oracle, так і з Microsoft SQL Server і IBM DB2).

1.3.2 Наявність стандартів

Наявність стандартів і набору тестів для виявлення сумісності і відповідності конкретній реалізації SQL загальноприйнятому стандарту тільки сприяє "стабілізації" мови.

1.3.3 Декларативність

За допомогою SQL програміст описує дані, які потрібно видалити або редагувати. Яким чином це зробити, вирішує СУБД безпосередньо при обробці SQL-запиту. Особливого значення це набуває при роботі з потужними базами даних і зі складними запитами – чим складніше сконструйований запит, тим більше він допускає варіантів написання різних за швидкістю виконання, але для опрацювання того ж самого набору даних.

1.4 Недоліки

1.4.1 Складність (використовується як інструмент програміста)

1.4.2 Відступи від стандартів

Незважаючи на наявність міжнародного стандарту ANSI SQL-92, провідні компанії (наприклад, Oracle, Sybase, Microsoft, MySQL) вносять зміни до мови SQL в розроблених ними СУБД, тим самим відступаючи від стандарту, що призводить до появи специфічних для кожної конкретної СУБД діалектів мови SQL.

1.4.3 Складність роботи з ієрархічними структурами

Раніше SQL не пропонував стандартного способу маніпуляції деревоподібними структурами, а тому постачальники СУБД пропонували свої рішення. Наприклад, Oracle використовує вираз CONNECT BY. В даний час за стандарт прийнята рекурсивна конструкція WITH.

1.5 Процедурні розширення

Оскільки SQL не є мовою програмування, тобто не надає засобів для автоматизації операцій з даними, введені різними виробниками розширення стосувалися в першу чергу процедурних розширень. Це збережені процедури (англ. stored procedures) і процедурні мови-

"надбудови". Практично в кожній СУБД застосовується своя процедурна мова. Подібні мови для найпопулярніших СУБД наведені в таблиці 1.2:

Таблиця 1.2 – Мови СУБД

СУБД	Коротка назва	Розшифровка
Borland InterBase/ Firebird	PSQL	Procedural SQL
IBM DB2	SQL PL	SQL Procedural Language (розширює SQL/PSM)
Microsoft SQL Server/ Sybase ASE	Transact-SQL	Transact-SQL
MySQL	SQL/PSM	SQL/Persistent Stored Module
Oracle	PL/SQL	Procedural Language/SQL (заснований на мові Ada)
PostgreSQL	PL/pgSQL	Procedural Language/PostgreSQL Structured Query Language (схожий на Oracle PL/SQL)

2. Створення запитів у середовищі SQL Management Studio

Для того, щоб виконати запит необхідно:

1) створити вікно редагування запиту, використовуючи контекстне меню (рис. 1.1);
2) набрати в редакторі текст запиту;
3) виконати запит, натиснувши на кнопку запуску в SQL Management Studio (рис. 1.2).

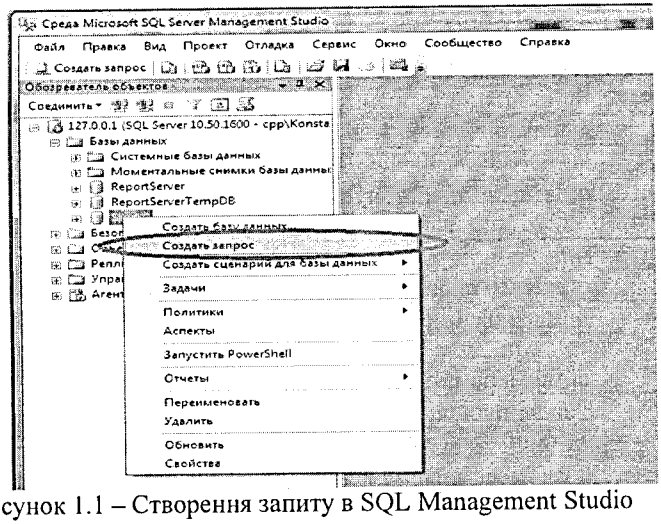


Рисунок 1.1 – Створення запиту в SQL Management Studio

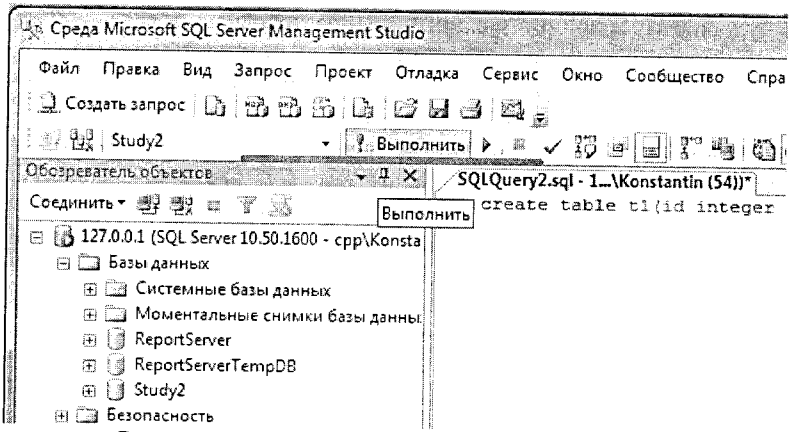


Рисунок 1.2 – Виконання запиту вікна редактора

3. Основні типи даних MS SQL Server

Одним із основних кроків у процесі створення таблиць є визначення типів даних для її полів. Тип даних поля таблиці визначає тип інформації, що буде розміщуватися в цьому полі. SQL-сервер підтримує різні типи даних: текстові, числові, двійкові і т. д.

Якщо є необхідність у розміщенні будь-яких специфічних даних у полі таблиці, то використовують так званий користувацький тип даних.

Основні типи даних, що використовуються в SQL-сервері, наведені у табл. 1.3 – 1.6.

Таблиця 1.3 – Числові типи даних

Назва	Опис
1	2
Int	Тип даних, що використовується для збереження додатних і від'ємних цілих чисел, що займає в пам'яті 4 байти. Діапазон: від (-2^{31}) до ($+2^{31}$).
Smallint	Займає в пам'яті 2 байти. Аналогічний типу даних Int, за винятком діапазону значень. Діапазон: від (–32768) до (+32767).
Tinyint	Тип даних, що використовується для збереження додатних цілих чисел. Займає в пам'яті 1 байт. Діапазон: від 0 до 255.
Real	Тип даних, що використовується для збереження додатних і від'ємних чисел із рухомою комою з точністю до 7 цифр; займає в пам'яті 4 байти. Діапазон: від -3,4e-38 до +3,4e+38.
Float	Тип даних, що використовується для збереження додатних і від'ємних чисел із рухомою комою з точністю до 15 цифр; займає в пам'яті до 8 байт. Діапазон: від -1,7e-38 до +1,7e+38.

Продовження таблиці 1.3

1	2
Decimal	Тип даних, що дозволяє точно визначати інтервал значень десяткових чисел, що вводяться, займає в пам'яті від 2 до 17 байт. Діапазон: від $-10e^{-38}$ до $+10e^{+38}$.
Numeric	Аналогічний типу даних Decimal.
Money	Тип даних, що використовується для збереження грошових значень; займає в пам'яті до 8 байт. Діапазон: від -922337203685477.5808 до +922337203685477.5807.
Small-money	Аналогічний типу даних Money; займає в пам'яті до 4 байт. Діапазон: від -214748.3648 до +214748.3647.

Таблиця 1.4 – Текстові типи даних

Назва	Опис
Char	Текстовий тип даних, при використанні якого задається його розмірність, причому для кожного символу виділяється один байт. Розмірність: до 8000 символів.
Nchar	Аналогічний типу даних Char, за винятком розмірності. Розмірність: до 4000 символів.
Varchar	Використовується для збереження текстової інформації змінної довжини. Розмірність: до 8000 символів.
Nvarchar	Аналогічний типу даних Varchar, за винятком розмірності. Розмірність: до 4000 символів.

Таблиця 1.5 – Типи дані дати і часу

Назва	Опис
Date time	Тип даних, що дозволяє зберігати комбінації дати і часу; займає в пам'яті 8 байт. Діапазон: від 01.01.1753 до 31.12.9999.
Small-date time	Аналогічний типу даних Datetime, що займає в пам'яті 4 байти Діапазон: від 01.01.1900 до 06.06.2079.

Таблиця 1.6 – Типи даних спеціального призначення

Назва	Опис
Bit	Тип даних, що дозволяє зберігати інформацію, яка набуває тільки двох значень: 0 чи 1; займає в пам'яті 1 біт. Діапазон: 0 чи 1.
Binary	Тип даних, що використовується для збереження бітових ланцюжків. Розмірність: до 8000 байт.

Продовження таблиці 1.6

Varbinary	Тип даних, що використовується для збереження бітових ланцюжків змінної довжини, аналогічно типу даних Binary. Розмірність: до 8000 байт.
Timestamp	Тип даних, що автоматично розміщає значення лічильника щоразу при вставці нового запису.
Uniqueidentifier	Розміщення унікального 16-розрядного ідентифікатора QUID (Globally unique identifier), що використовується для підтримки цілісності даних. Генерація нового ідентифікатора здійснюється з використанням команди SQL NEWID().

4. Створення таблиць

Таблиці створюються командою CREATE TABLE. Ця команда створює порожню таблицю без записів. Важливим в команді CREATE TABLE є визначення імені таблиці й опис набору імен полів, що вказуються у відповідному порядку. Крім того, цією командою також визначаються типи даних і розміри полів таблиці. В кожній таблиці має бути хоча б одне поле. Синтаксис команди CREATE TABLE такий:

CREATE TABLE <TABLE NAME> <COLUMN NAME><DATA TYPE> [(<SIZE>)], <COLUMN NAME><DATA TYPE> [(<SIZE>)]...);

Пропуски використовуються для розділення елементів команди SQL, вони не можуть бути частиною імені чи таблиці будь-якого іншого створюваного об'єкта. При цьому символ підкреслення (_), як правило, використовується для розділення слів в іменах таблиць.

Значення аргументу SIZE залежить від типу даних. Якщо його не вказати, то СУБД призначає значення автоматично. Для числових значень аргумент SIZE слід визначити обов'язково, інакше всі поля такого типу одержать однаковий розмір. Аргумент SIZE типу даних CHAR – це ціле число, яке визначає максимальну кількість символів, що може вмістити поле. Мінімальна кількість символів такого поля дорівнює нулю (якщо поле має значення NULL). За замовчуванням значення аргументу SIZE дорівнює 1, а це означає, що поле може містити тільки один символ.

Імена таблиць повинні відрізнятися. Однак різні таблиці можуть використовувати однакові імена полів. Наприклад, поле з ім'ям ДИСЦИПЛІНА може бути присутнім у таблицях СТУДЕНТ і ВИКЛАДАЧ.

Наведемо приклад команди, що створить структуру таблиці STUDENTS:

CREATE TABLE STUDENTS (SNUM INTEGER, SEAM CHAR (20), SIMA CHAR (10), SOTCH CHAR (15) STIP DECIMAL);

Порядок розташування полів у таблиці визначається тим, у якій послідовності вони зазначені в команді створення таблиці.

Записи в таблицю можна ввести за допомогою команди INSERT.

Команда ALTER TABLE використовується для внесення змін у структуру таблиці. Наприклад, синтаксис цієї команди для додання стовпця до таблиці такий:

ALTER TABLE <TABLE NAME> ADD <COLUMN NAME><DATA TYPE><SIZE>;

Нове поле стане останнім в таблиці та буде мати значення NULL для всіх записів таблиці.

Допускається додання кількох нових полів однією командою, відокремивши їх комами. Наприклад, для додання до таблиці STUDENTS двох полів для збереження інформації про курс і спеціальність студента, можна скористатися такою командою:

ALTER TABLE STUDENTS ADD COURS INTEGER, SPEC CHAR (10);

Для видалення таблиці SQL висувається вимога її очищення від даних, що дозволяє уникнути виникнення колізій. Таким чином, не порожня таблиця не може бути видаленою.

Синтаксис команди для видалення таблиці (за умови, що вона порожня) такий:

DROP TABLE <TABLE NAME>;

Наприклад, для видалення таблиці STUDENTS, у якій записи попередньо видалені, слід виконати команду:

DROP TABLE STUDENTS;

Після виконання цієї команди ім'я таблиці більше не розпізнається, а маніпуляції над нею неможливі. Перед видаленням таблиці варто переконатися в тому, що ця таблиця не посилається на іншу таблицю.

Таким чином, використання мови опису даних DDL (Data Definition Language) дозволяє створювати нові, змінювати структуру існуючих і видаляти порожні таблиці БД.

Приклад виконання лабораторної роботи

Предметна область: комп'ютерний магазин.

Визначимо основні запитання, на які в базі даних має міститись відповідь.

1. Скільки коштує деталь X? (де X – назва будь-якої деталі комп'ютера).
2. Чи є деталь X на складі?
3. Коли виготовлена деталь X?
4. Яка тактова частота процесора Y?
5. Процесор Y йде в комплекті із системою охолодження чи без неї?
6. Який тип сокета у процесора Y?

7. Які є материнські плати, що мають сокет Z?

8. Який максимальний обсяг оперативної пам'яті підтримує материнська плата M?

9. Скільки зовнішніх USB виходів є на материнській платі M?

10. Скільки всього USB виходів є на материнській платі M?

11. Який тип відеокарти підключається до материнської плати M?

Із вищенаведених запитань, можна визначити атрибути, що мають міститись в базі даних:

<Назва деталі>, <Вартість деталі>, <Наявність деталі на складі>, <Дата виготовлення>, <Назва процесора>, <Тактова частота процесора>, <Наявність системи охолодження>, <Тип сокета у процесора>, <Назва материнської плати>, <Тип сокета у материнської плати>, <Макс. обсяг оперативної пам'яті для материнської плати>, <Кількість зовнішніх USB виходів на материнській платі>, <Загальна кількість USB виходів на материнській платі>, <Тип відеокарти>.

Інтуїтивно можна виділити такі таблиці (кількість і структура таблиць визначається на власний розсуд без проведення детального аналізу):

1. Деталь (<Назва деталі>, <Вартість деталі>, <Наявність деталі на складі>, <Дата виготовлення>).

2. Процесор (<Назва процесора>, <Тактова частота процесора>, <Наявність системи охолодження>, <Тип сокета у процесора>).

3. Материнська плата (<Назва материнської плати>, <Тип сокета у материнської плати>, <Макс. обсяг оперативної пам'яті для материнської плати>, <Кількість зовнішніх USB виходів на материнській платі>, <Загальна кількість USB виходів на материнській платі>, <Тип відеокарти>).

Визначимо тип даних для кожного атрибута, враховуючи типи даних що підтримуються сервером баз даних.

<Назва деталі> – текстове поле, обираємо тип Varchar.

<Вартість деталі> – можна застосувати тип Decimal, однак сервер уже підтримує власний тип даних для грошової величини, тому обираємо тип Money.

<Наявність деталі на складі> – для даного поля є лише два допустимих значення, тому обираємо тип Bit.

<Дата виготовлення> – обираємо тип SmallDateTime.

<Назва процесора> – текстове поле, обираємо тип Varchar.

<Тактова частота процесора> – дана величина завжди є цілим додатним числом, верхню границю можна оцінити на рівні у 10×10^9, тому обираємо тип Decimal.

<Наявність системи охолодження> – для даного поля є лише два допустимих значення, тому обираємо тип Bit.

<Тип сокета у процесора> – текстове поле, обираємо тип Varchar.

<Назва материнської плати> – текстове поле, обираємо тип Varchar.

<Тип сокета у материнської плати> – текстове поле, обираємо тип Varchar.

<Макс. обсяг оперативної пам'яті для материнської плати> – величина завжди є цілим додатним числом, верхню границю можна оцінити на рівні у 100×10^9, тому обираємо тип Decimal.

<Кількість зовнішніх USB виходів на материнській платі> – кількість портів на материнській платі дорівнює 4÷8, тому обираємо тип TinyInt.

<Загальна кількість USB виходів на материнській платі> – кількість портів на материнській платі дорівнює 4÷8, тому обираємо тип TinyInt.

<Тип відеокарти> – поле набуває лише фіксованих текстових значень, довжина яких не перевищує 10 байт (розмір обирається з невеликим запасом в сторону збільшення), тому обираємо тип Varchar з обмеженням на довжину в 10 символів.

Після визначення структури і типів даних атрибутів таблиць, можна приступити до створення бази даних в середовищі SQL Management Studio.

Microsoft SQL Server 2008 повинен бути встановленим та запущеним (при інсталяції слід активувати запуск його основних служб при запуску системи).

Для створення та редагування структури бази даних використовується SQL Server Management Studio, що входить в пакет інсталяції Microsoft SQL Server. Запустити його можна із меню "Пуск" (рис. 1.3):

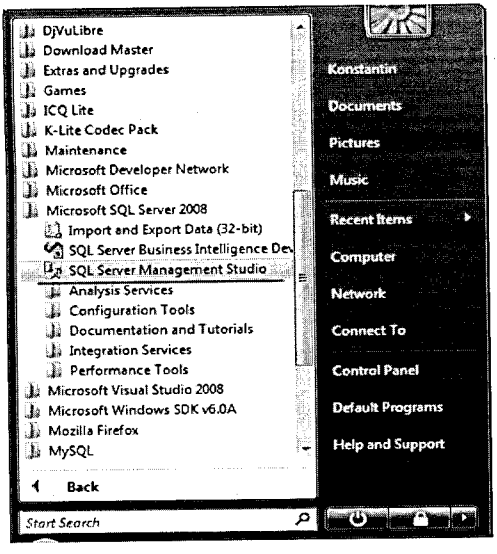


Рисунок 1.3 – Меню запуску SQL Server Management Studio

Якщо програму не видно у діалоговому вікні, то інсталяція сервера баз даних була невдалою, а тому слід повторити дії із установлення відсутніх компонентів або всього сервера баз даних. Якщо інсталяція успішна, то SQL Server Management Studio запуститься, а на екрані з'явиться вікно, в якому слід вказати параметри підключення до сервера баз даних (рис. 1.4, 1.5).

1. Тип підключення до сервера реляційних баз даних – Server type: Database Engine.
   Інші типи серверів використовуються в інших випадках (таких як аналіз даних, сервіс звітування та інтеграції).
2. Адреса сервера – Server name: 127.0.0.1. Автоматично записується IP адреса локального комп'ютера (127.0.0.1 є IP адресою локального комп'ютера в усіх операційних системах, що підтримують стек TCP/IP).
3. Authentication: Windows Authentication.

Даний параметр забезпечує можливість входження в систему керування базою даних під обліковим записом поточного користувача системи. В подальшому, при роботі логін та пароль у Вас запитуватись не буде.

Таким чином, основні кроки виконання лабораторної роботи такі.

1. Підключаємося до сервера баз даних, встановленого на локальному комп'ютері (рис. 1.4):

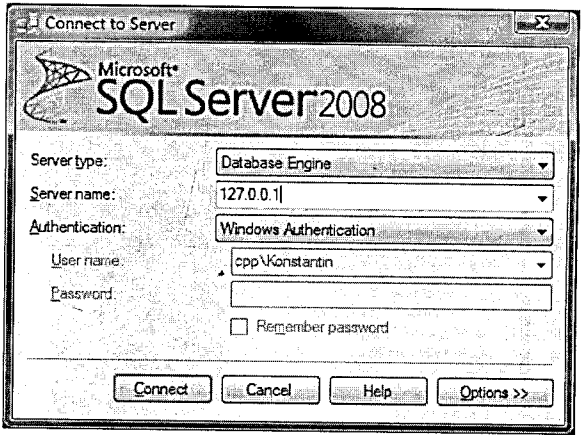


Рисунок 1.4 – Діалог підключення до сервера баз даних

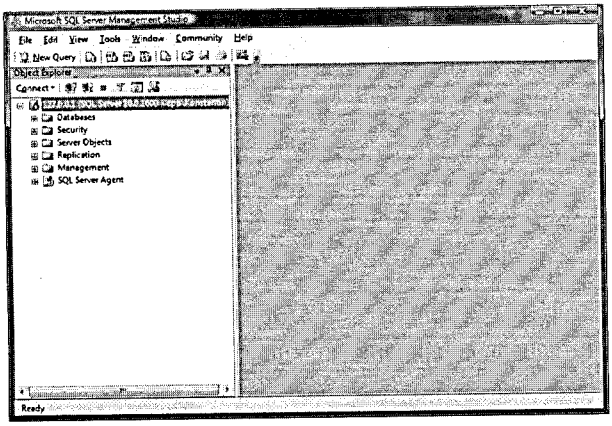


Рисунок 1.5 – Головне вікно SQL Server Management Studio

2. Створюємо базу даних з назвою “Lab2”:

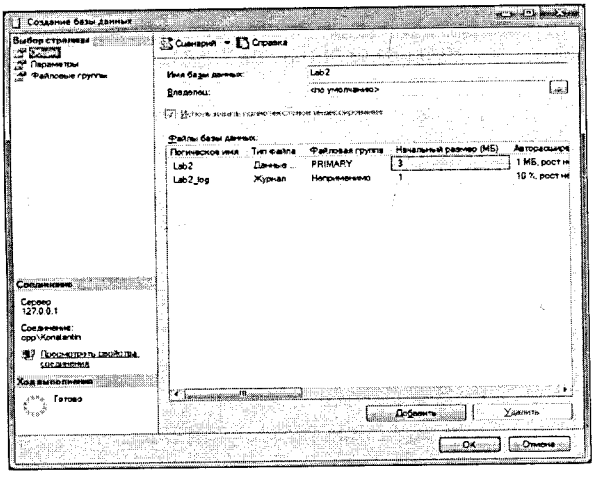


Рисунок 1.6 – Створення бази даних для виконання лабораторної роботи

3. Створимо таблиці бази даних. Це можна зробити як в графічному середовищі, так і за допомогою SQL. Використаємо SQL-команди для створення таблиць.

Відкриємо вікно для редагування запиту до бази даних:

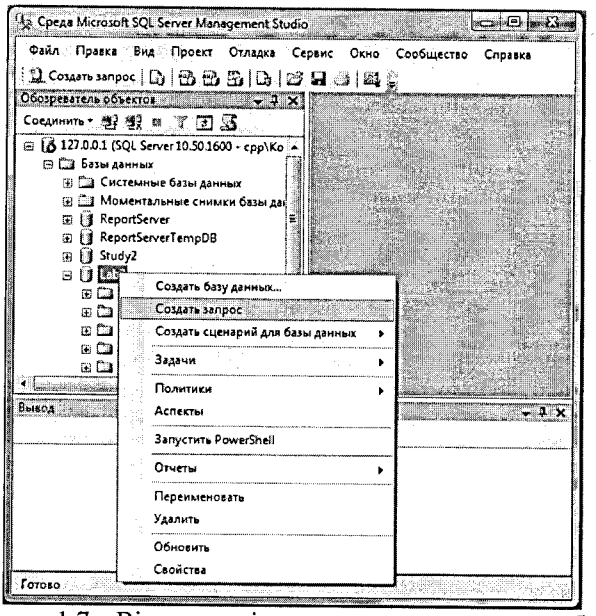


Рисунок 1.7 – Відкриття вікна для введення запиту до бази даних

4. Запит для створення таблиці “Деталь” матиме вигляд (загальновживаною є практика називати стовпці таблиць латинськими літерами).

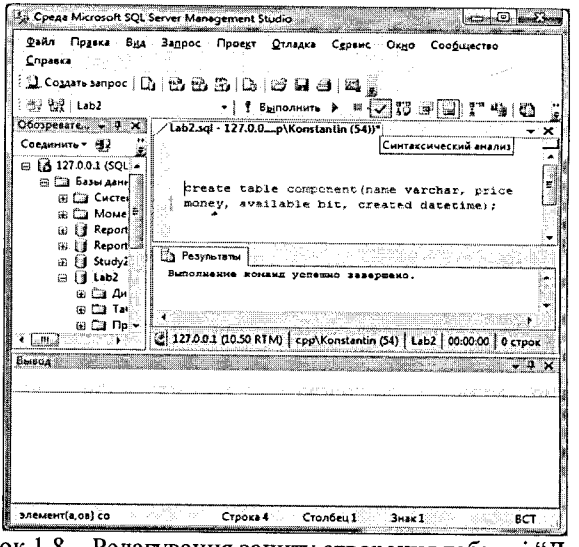


Рисунок 1.8 – Редагування запиту створення таблиці “Деталь”

Перевірити синтаксис запиту можна за допомогою кнопки “Синтаксический анализ” на панелі.

Запити для створення інших таблиць бази даних матимуть вигляд (кожен запит записується з нового рядка):

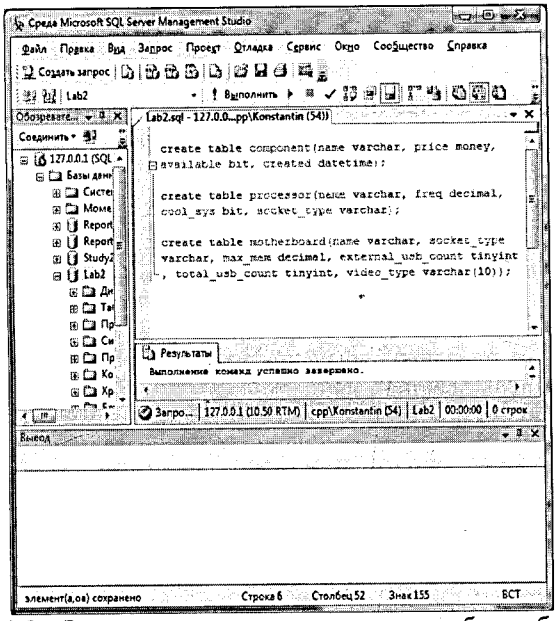


Рисунок 1.9 – Запити для створення структури таблиць бази даних

5. Після виконання запитів на створення таблиць у вікні “Обозреватель объектов” можна перевірити, чи були створені таблиці:

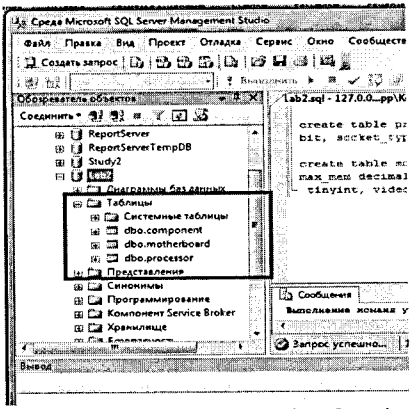


Рисунок 1.10 – Створені таблиці

6. Збережемо створені запити (Меню Файл → Сохранить або Ctrl+S):

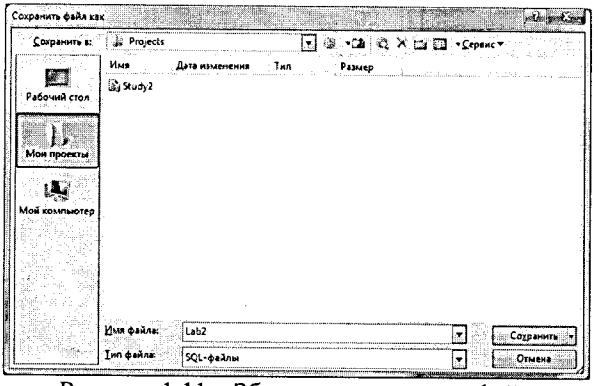


Рисунок 1.11 – Збереження запиту в файл

7. Виконаємо операцію видалення створених таблиць із бази даних, написавши відповідний запит:

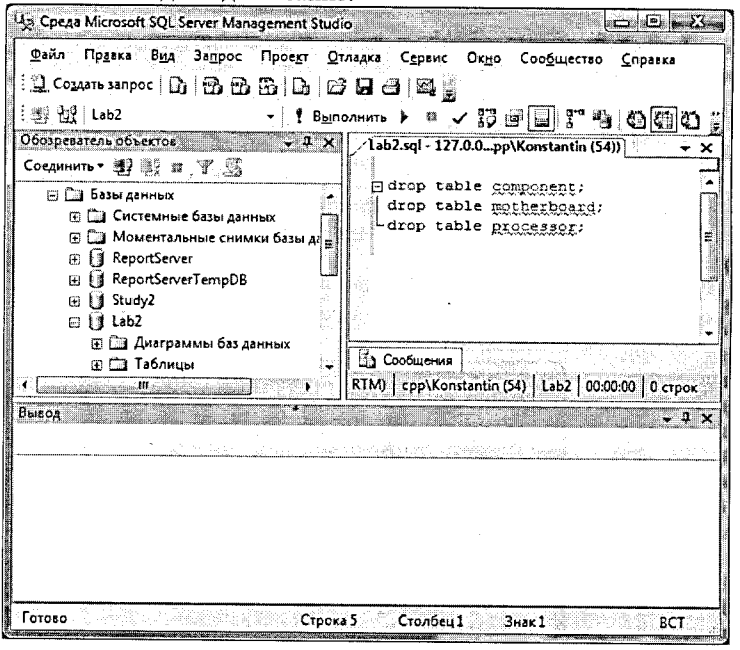


Рисунок 1.12 – Запит для видалення створених таблиць із бази даних

Після виконання даного запиту таблиці будуть видалені із бази даних.

КОНТРОЛЬНІ ПИТАННЯ

1. Що таке мова SQL?
2. Назвіть основні стандарти SQL.
3. Чому виникло багато різних діалектів стандарту SQL?
4. Переваги використання SQL порівняно із прямим доступом до даних.
5. Недоліки використання SQL у порівняно із прямим доступом до даних.
6. Для чого використовуються процедурні розширення SQL?
7. Назвати найбільш поширені в даний час процедурні розширення SQL.
8. Опишіть процес створення та виконання запиту в середовищі SQL Management Studio.
9. Назвіть основні типи даних, що використовуються у Microsoft SQL Server.
10. Способи створення баз даних та таблиць в середовищі SQL Management Studio.
11. За допомогою яких конструкцій мови SQL відбувається створення таблиць? Опишіть синтаксис даної конструкції.
12. За допомогою яких конструкцій мови SQL відбувається видалення таблиць? Опишіть синтаксис даної конструкції.
13. За допомогою яких конструкцій мови SQL відбувається модифікація структури таблиць? Опишіть синтаксис даної конструкції.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Дейт К. Дж. Введение в системы баз данных / Дейт К. Дж. – [8-е изд.]. – М. : Изд-во "Вильямс", 2006. – 1328 с.
2. Дейт К. SQL и реляционная теория. Как грамотно писать код на SQL / Дейт К. – М. : Изд-во "Символ-Плюс", 2010. – 480 с.
3. Роберт Виейра Программирование баз данных Microsoft SQL Server 2008. Базовый курс / Роберт Виейра. – М. : Изд-во: "Диалектика-Вильямс", 2009. – 816 с.
4. Бен-Ган И. Microsoft SQL Server 2008. Основы T-SQL / Бен-Ган И. – Санкт-Петербург : Изд-во "Русская редакция", 2009. – 432 с.
5. Тернстрем Т. Microsoft SQL Server 2008. Разработка баз данных. Учебный курс Microsoft, экзамен 70–433 (+ CD) / Тернстрем. Т., Вебер Э., Хотек М. – М. : Изд-во "Русская Редакция", 2010. – 496 с.
6. Петкович Д. Microsoft SQL Server 2008. Руководство для начинающих / Петкович Д. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 752 с.
7. Грофф Джеймс Р. SQL: полный справочник / Грофф Джеймс Р., Вайнберг Пол Н., Оппель Эндрю Дж. – Москва : Изд-во "Диалектика-Вильямс", 2010. – 960 с.
8. Дьюсон Р. SQL Server 2008 для начинающих разработчиков / Дьюсон Р. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 704 с.

ЛАБОРАТОРНА РОБОТА № 2
SQL MANAGEMENT STUDIO:
ПРОСТІ ЗАПИТИ ДЛЯ ВИБИРАННЯ ДАНИХ.
АРИФМЕТИЧНІ ТА СИМВОЛЬНІ ФУНКЦІЇ

Мета: отримати загальні відомості про вибирання даних в мові SQL. Вивчити базову структуру конструкції SELECT. Отримати навички використання арифметичних, символьних функцій при вибиранні даних, а також функцій для роботи з датою та часом. Навчитись формувати умови відбору при вибиранні даних.

Порядок виконання роботи

1. На основі запитів для створення таблиць, побудованих в лабораторній роботі № 1, створити на сервері базу даних для виконання лабораторної роботи.
2. Відкрити вікно виконання запитів в програмі SQL Management Studio.
3. Створити два запити для вибирання даних (із однієї довільної таблиці кожний) з використанням арифметичних функцій мови SQL та виконати їх.
4. Створити три запити для вибирання даних (із однієї довільної таблиці кожний) з використанням символьних функцій мови SQL та виконати їх.
5. Створити запит для вибирання даних, що буде використовувати функції опрацювання дати та часу, та виконати його.
6. Створити три запити з різними умовами відбору даних з однієї таблиці та виконати їх.
7. Зберегти файл із запитами.
8. Оформити звіт.

Теоретичні відомості

1. Загальна структура команди SELECT

Команда SELECT застосовується для отримання даних за заданими вимогами. Команда SELECT може бути простою або складною.

Синтаксично команда SELECT включає оператори SELECT і FROM. Вони задають відповідно стовпець/стовпці і таблицю/таблиці, з яких будуть вибиратися дані. Будь-який SQL-вираз починається з шаблону:

SELECT<Список стовпців>
FROM<Список таблиць>
[WHERE<Умови пошуку>]

Наприклад:
SELECT Name, Price
FROM Processor

WHERE Available = 1;

При цьому список стовпців описує стовпці, які потрібно вибрати, а список таблиць вказує таблиці, з яких вибираються дані.

Конструкція WHERE вказує умови вибору даних. Оператори SELECT і WHERE дозволяють містити обчислення, константи та інші висловлювання. Комбінація конструкцій SELECT, FROM і WHERE забезпечує гнучкі умови вибирання даних. Дані, які вибираються за допомогою команди SELECT – це записи, що відповідають заданим вимогам.

Нижче наводяться приклади команди SELECT.

```
SELECT [ALL DISTINCT] <Список стовпців в результаті>
[INTO [<Назва створюваної таблиці>] ]
[FROM{< Назва таблиці 1> | < Назва подання 1>}
[ (<Підказка оптимізатору>) ]
...
[,{<Назва таблиці M> | <Назва подання M>}
[(<Підказка оптимізатору>) ]]]
[WHERE<Вираз>]
[GROUP BY<Вираз>]
[HAVING<Вираз>]
[ORDER BY <Вираз>]
[COMPUTE <Опція>]
[FOR BROWSE]
```

Слід звернути увагу, що конструкція GROUP BY має йти до ORDER BY, інакше на екрані з'явиться повідомлення про синтаксичну помилку.

Якщо необхідно вивести інформацію з усіх стовпців, наявних в таблицях (зазначених у FROM), використовуйте зірочку (*). У цьому випадку стовпці на екран будуть виводитися в тому порядку та з такими ж іменами, як вони створювалися за допомогою команди CREATE TABLE.

Наприклад, результат виконання команди SELECT над таблицею Processor (Name, Price, Available)

```
SELECT *FROM Processor;
```

буде таким

Name	Price	Available
Intel Pentium-I	20	0
Intel Pentium-II	50	0
Intel Pentium-III	100	1
Intel Pentium-IV	300	1
Intel Core 2 Duo	800	1

Можна змінити заголовки стовпців, включивши їх до списку вибирання

перед їхніми іменами:
SELECT <Заголовок стовпця> = <Ім'я стовпця> {,<Ім'я стовпця>}
FROM <Ім'я таблиці>

Або вказати заголовок стовпця після його імені через пробіл:
SELECT <Ім'я стовпця><Заголовок стовпця> {, <Ім'я стовпця>]
FROM <Ім'я таблиці>

Приклад виконання команди SELECT над таблицею Processor (Name, Price, Available)
SELECT Name Назва, Price Ціна, Available Наявний **FROM** Processor;
Результат вибирання:

Name	Price	Available
Intel Pentium-I	20	0
Intel Pentium-II	50	0
Intel Pentium-III	100	1
Intel Pentium-IV	300	1
Intel Core 2 Duo	800	1

2. Арифметичні функції Microsoft SQL Server
Розглянемо вбудовані арифметичні функції Microsoft SQL Server.

Таблиця 2.1 – Арифметичні функції Microsoft SQL Server

Функція	Типи аргументів	Результат
1	2	3
ABS	Число	Абсолютне значення
ACOS, ASIN, ATAN, ATN2	Число з рухомою комою	Обернені косинус, синус і тангенс. Повертають кут в радіанах
COS, SIN, COT, TAN	Число з рухомою комою (в радіанах)	Косинус, синус, котангенс або тангенс кута
CEILING	Число	Найменше ціле число, яке більше або рівне заданому аргументу
DEGRESS	Число	Перетворення із радіан в градуси
EXP	Число з рухомою комою	Експонента від аргументу
FLOOR	Число	Найбільше ціле, яке менше або рівне вказаному аргументу
LOG	Число з рухомою комою	Натуральний логарифм аргументу
LOG10	Число з рухомою комою	Десятковий логарифм аргументу
PI	Задане число	Константа 3.141592653589793
POWER	Число, Y(число)	Повертає аргумент в степені Y
RADIANS	Число	Перетворення градусів в радіани

Продовження таблиці 2.1

1	2	3
RAND	Аргумент необов'язковий	Повертає випадкове число з плаваючою комою в діапазоні від 0 до 1
ROUND	Число, кількість цифр	Округляє число до вказаної кількості знаків після коми
SIGN	Число	Повертає знак числа (додатне, від'ємне або нуль)
SQRT	Число з плаваючою комою	Квадратний корінь від числа

Повний опис арифметичних функцій можна знайти на сайті MSDN за адресою http://msdn.microsoft.com/en-us/library/ms177516.aspx

3. Символьні функції Microsoft SQL Server

Більшість символьних функцій можуть бути використані лише з типами даних Char і Varchar (або з типами даних, які легко конвертуються в Char і Varchar). Якщо як аргументи символьних функцій виступають рядкові константи, то їх беруть у лапки.

Синтаксис рядкових функцій:

SELECT < Назва функції (параметри)>

У таблиці 2.2 наведено список символьних функцій.

Таблиця 2.2 – Символьні функції Microsoft SQL Server

Функція	**Типи аргументів**	**Результат**
1	2	3
+	Рядок плюс рядок	Об'єднання двох рядків (стовпців)
ASCII	Рядок	ASCII-код крайнього лівого символа
CHAR	Ціле число	Символьний еквівалент ASCII-кода переданого як аргумента
CHARINDEX	Шаблон, рядок	Повертає початкову позицію шаблона в середині рядка (другого параметра)
DIFFERENCE	Рядок 1, Рядок 2	Виявляє ступінь подібності рядків, повертаючи значення від 0 до 4. Число 4 означає повний збіг
LOWER	Рядок	Переведення виразу в нижній регістр
LTRIM	Рядок	Повертає вираз без початкових пропусків
PATINDEX	Шаблон, Рядок	Повертає шаблон рядка (не повертає, якщо шаблон не знайдено)

Продовження таблиці 2.2

1	2	3
REVERSE	Рядок	Повертає рядок навпаки
RIGHT	Рядок, ціле число	Частина рядка з вказаної позиції
RTRIM	Рядок	Рядок без пропусків на початку та в кінці
SOUNDEX	Рядок 1, Рядок 2	Повертає код з чотирьох цифр, що вказує на подібність двох рядків
SPACE	Ціле число	Повертає рядок з вказаною кількістю пропусків. Якщо число від'ємне, повертається нульовий рядок
STR	Число з плаваючою комою, довжина (ціле число), кількість десяткових знаків (ціле число)	Повертає символьне подання числа. Параметр "довжина" вказує загальну довжину отриманого виразу, включно з десятковим подільником, знаком, цифрами та пропусками. Аргумент «кількість десяткових знаків» вказує кількість знаків після коми
STUFF	Рядок 1, початкова позиція (ціле число), довжина (ціле число), рядок 2	Видаляє число знаків, що дорівнює параметру "довжина", з рядка 1, починаючи з початкової позиції, з подальшою заміною видалених знаків рядком 2
SUBSTRING	Рядок, початкова позиція (ціле число), довжина (ціле число)	Повертає частину рядка (заданої довжини) від початкової позиції
UPPER	Рядок	Переводить символи з нижнього регістра у верхній

Детальний опис символьних функцій можна знайти на сайті MSDN за адресою http://msdn.microsoft.com/en-us/library/ms181984.aspx.

4. Функції для роботи з датою

Особливої уваги до себе заслуговують функції для роботи з датою та часом, стовпці типу DATETIME. Для маніпуляцій з даним типом даних використовуються спеціальні функції.

Таблиця 2.3 – Функції Microsoft SQL Server для роботи з датами

Функція	**Типи аргументів**	**Результат**
1	2	3
DATEADD	Кількість днів, число, дата	Додає кількість днів до дати
DATEDIFF	Кількість днів, дата 1, дата 2	Визначає кількість днів між двома датами

Продовження таблиці 2.3

1	2	3
DATENAME	Кількість днів, дата	Повертає ASCI (значення частини дати для обраної дати (наприклад, May)
DATEPART	Частина дати, дати	Повертає числове значення (наприклад, 5) частини дати для обраної дати
GETDATE		Повертає поточний час і дату у внутрішньому форматі SQLServer

5. Формування умов відбору

При використанні технології "клієнт-сервер" кількість переданих по мережі даних суттєво впливає на продуктивність. Для обробки даних, як правило, використовують збережені процедури, коли додаток обмінюється з сервером лише параметрами і результатами.

При цьому найбільшого поширення набуває конструкція WHERE. Вона може містити в собі оператори, перераховані в таблиці 2.4.

Таблиця 2.4 – Оператори, що застосовуються в конструкції WHERE

Назва оператора	**Оператор**
Порівняння	(=, >, <, >=, <=, <>, !=, !<, !>)
Інтервал	BETWEEN, NOT BETWEEN
Список	IN, NOT IN
Порівняння рядків	LIKE, NOTLIKE
Перевірка значення	IS NULL, IS NOT NULL
Логічні	AND, OR
Інверсія	NOT

При використанні операторів порівняння необхідно дотримуватися двох простих правил.

1. Вирази можуть містити константи, імена стовпців, функції, вкладені запити і арифметичні оператори.
2. З даними типу char, varchar, text, datetime і smalldatetime використовувати одинарні лапки (подвійні лапки не заборонені, але одинарні лапки кращі для сумісності зі стандартом ANSI).

Часто потрібно переглянути дані в їх відношенні один до одного. Microsoft SQL Server забезпечує це за допомогою операторів порівняння, більшість з яких наведена у таблиці 2.5.

Оператори порівняння використовуються для роботи з числами, але в SQL можна їх використовувати і з типами char і varchar: знак "<" означає раніше за алфавітом, а ">" – пізніше. Крім того, оператори порівняння можна використовувати для дат.

В реченні WHERE доводиться використовувати кілька умов пошуку, які можна об'єднати логічними (булевими) операторами AND, OR, NOT.

Таблиця 2.5 – Оператори порівняння

Оператор	Значення
=	Дорівнює
>	Більше
<	Менше
>=	Не менше
<=	Не більше
!=	Не дорівнює
<>	Не дорівнює

У випадку, коли при формуванні запиту немає впевненості у значеннях поіменованих характеристик, використовують оператор LIKE. Синтаксис оператора LIKE:

WHERE <Ім'я стовпця> [NOT] LIKE <'Шаблон'> [ESCAPE <Символ>]

Шаблон повинен бути взятий у лапки та містити знаки підстановки. Опція ESCAPE використовується в тому випадку, коли пошукове значення містить в собі один із знаків підстановки. ANSI SQL забезпечує два знаки підстановки для LIKE: відсоток "%" (замінює собою рядок з нуля і більше символів) і підкреслення "_" (замінює один символ).

У деяких SQL-діалектах, у тому числі і Transact-SQL, підтримуються ще два види підстановки – дужки "[]" (символ повинен належати зазначеному діапазону, і "[^]" (символ не повинен належати зазначеному діапазону).

КОНТРОЛЬНІ ПИТАННЯ

1. З якою метою використовуються процедурні розширення SQL?
2. Наведіть процедурні розширення SQL.
3. Опишіть процес створення та виконання запиту в середовищі SQL Management Studio.
4. Назвіть основні типи даних, що використовуються у Microsoft SQL Server.
5. Обгрунтуйте необхідність правильного вибору типів даних при проектуванні бази даних.
6. Назвіть відомі вам способи створення баз даних та таблиць в середовищі SQL Management Studio.
7. За допомогою яких конструкцій мови SQL відбувається створення таблиць?
   Опишіть синтаксис даної конструкції.
8. За допомогою яких конструкцій мови SQL відбувається видалення таблиць?
   Опишіть синтаксис даної конструкції.
   За допомогою яких конструкцій мови SQL відбувається модифікація таблиць? Опишіть синтаксис даної конструкції.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. http://msdn.microsoft.com/en-us/library/ms174318.aspx – Вбудовані функції Microsoft SQL Server (T-SQL).
2. Дейт К. Дж. Введение в системы баз данных / Дейт К. Дж. – [8-е изд.]. – М. : Изд-во "Вильямс", 2006. – 1328 с.
3. Дейт К. SQL и реляционная теория. Как грамотно писать код на SQL / Дейт К. – М. : Изд-во "Символ-Плюс", 2010. – 480 с.
4. Роберт Виейра Программирование баз данных Microsoft SQL Server 2008. Базовый курс / Роберт Виейра. – М. : Изд-во: "Диалектика-Вильямс", 2009. – 816 с.
5. Бен-Ган И. Microsoft SQL Server 2008. Основы T-SQL / Бен-Ган И. – Санкт-Петербург : Изд-во "Русская редакция", 2009. – 432 с.
6. Тернстрем Т. Microsoft SQL Server 2008. Разработка баз данных. Учебный курс Microsoft, экзамен 70–433 (+ CD) / Тернстрем. Т., Вебер Э., Хотек М. – М. : Изд-во "Русская Редакция", 2010. – 496 с.
7. Петкович Д. Microsoft SQL Server 2008. Руководство для начинающих / Петкович Д. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 752 с.
8. Грофф Джеймс Р. SQL: полный справочник / Грофф Джеймс Р., Вайнберг Пол Н., Оппель Эндрю Дж. – Москва : Изд-во "Диалектика-Вильямс", 2010. – 960 с.
9. Дьюсон Р. SQL Server 2008 для начинающих разработчиков / Дьюсон Р. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 704 с.

ЛАБОРАТОРНА РОБОТА № 3
SQL MANAGEMENT STUDIO:
ЗВ'ЯЗКИ МІЖ ТАБЛИЦЯМИ. КЛЮЧІ ТА ОБМЕЖЕННЯ

Мета: ознайомитись з поняттям ключа та основними типами ключів. Ознайомитись зі структурою та призначенням обмежень, набути практичних навичок їх реалізації.

Порядок виконання роботи

1. Ознайомитись із теоретичними відомостями.
2. Створити на сервері базу даних для виконання лабораторної роботи та проаналізувати її з метою усунення наявних аномалій.
3. Додати первинний ключ до однієї з існуючих таблиць.
4. Створити таблицю із зовнішнім ключем.
5. Додати зовнішній ключ до однієї з існуючих таблиць.
6. Виконати додання обмеження типу CHECK до однієї із існуючих таблиць.
7. Додати обмеження типу DEFAULT в одну із існуючих таблиць.
8. Зробити висновки та оформити звіт.

Теоретичні відомості

1. Поняття ключа, основні типи ключів

Якщо кожному стовпцю таблиці присвоїти ім'я, то розміщення стовпців буде несуттєвим. Кожне відношення має своє ім'я, а список імен атрибутів одного відношення називається схемою заміщення. Формально схема відношення описується таким чином:

НАЗВА_ВІДНОШЕННЯ (A_1, A_2, A_3, ..., A_K),

де A_i – назва i-того атрибута, $i = \overline{1,K}$.

Множину поіменованих характеристик (атрибутів), значення яких однозначно ідентифікують кожний рядок (запис) таблиці, називають ключем, кожен з яких є можливим. Розрізняють такі основні типи ключів: первинний, можливий, простий, повністю складений ключ, напівскладений ключ, зовнішній.

Якщо відношення має більше одного можливого ключа, то вибирають один, який використовують при обробленні даних та називають первинним.

Простим називається ключ, що складається з одного атрибута, причому атрибут повинен бути атомарним, а екземпляри даного атрибута – унікальними. Атомарним є атрибут, значення якого сприймається як неподільний елемент даних.

Повністю складеним називається ключ, що містить декілька атрибутів, причому між цими атрибутами існує відображення (залежність) БАГАТО-ДО-БАГАТЬОХ. Атрибути, що складають такий ключ, не залежать один від одного. Можливі випадки, коли повністю складений ключ містить всі атрибути відношення.

Напівскладеним називається ключ, що містить декілька атрибутів, між якими на відміну від повністю складеного ключа існує залежність Б:1 (БАГАТО-ДО-ОДНОГО), а атрибути в ключі впорядковуються за принципом: кожний наступний уточнює попередній.

Взаємозв'язок таблиць є найважливішим елементом реляційної моделі даних, що забезпечується зовнішніми ключами (*foreign key*).

2. Обмеження

Під обмеженнями розуміють вимоги до організації даних. Обмеження встановлюються на рівні стовпця або таблиці та гарантують відповідність даних певним правилам забезпечення цілісності.

Серед інших особливої уваги заслуговують такі типи обмежень:

- обмеження сутностей;
- обмеження домену;
- обмеження посилальної цілісності;
- обмеження первинного ключа (PRIMARY KEY);
- обмеження зовнішнього ключа (FOREIGN KEY);
- обмеження унікальності (UNIQUE);

– обмеження перевірки (CHECK);
– обмеження заданих за замовчуванням значень (DEFAULT).

3. Обмеження сутностей

Обмеження сутності відносяться до окремих рядків. В обмеженнях цього типу у кожному рядку таблиці має бути наявним унікальне значення одного атрибута або множини атрибутів. При цьому не обмежується форматування та значення певного атрибута.

Обмеження такого роду будуть розглядатися в контексті опису обмеження PRIMARY KEY і UNIQUE.

4. Обмеження домену

Обмеження домену поширюються на один або кілька атрибутів. Під цими обмеженнями розуміють відповідність певним критеріям конкретного атрибута або множини атрибутів. Ці обмеження застосовуються при вставці або оновленні рядків. Таблицею домену називається таблиця, єдиним призначенням якої є надання обмеженого списку допустимих значень.

5. Обмеження посилальної цілісності

Обмеження посилальної цілісності створюються в тому випадку, якщо значення в одному стовпці повинні узгоджуватися зі значеннями в іншому стовпці.

Припустимо, що розробляється додаток, призначений для прийому замовлень на товари, який повинен забезпечувати можливість оплати за допомогою кредитних карток певної компанії, для чого необхідно укласти торговельну угоду з нею в певній формі. Саме в такому випадку набувають чинності обмеження посилальної цілісності.

6. Способи іменування обмежень

Всі можливі види обмежень повинні бути позначені ім'ям. Можна скористатися тим, що СУБД SQL Server надає ім'я для того обмеження, для якого ім'я не було передбачено розробником. Проте, імена, створювані СУБД SQL Server, не завжди прийнятні. Розглянемо приклад імені, що сформовано системою для таблиці Employees – PK__Employees__145C0A3F. В такому імені PK скорочено позначає PRIMARY KEY, Employees вказує на таблицю, до якої належить це обмеження, інша частина імені – це значення, сформоване системою випадковим чином для забезпечення унікальності обмеження. У випадку, якщо б для створення цієї таблиці використовувалася програма Management Studio, то обмеження отримало б ім'я PK_Employees. Але основний недолік імен, сформованих системою, полягає в тому, що ці імена не розкривають суті застосовуваних обмежень. Наприклад, при використанні обмеження CHECK для таблиці Customers, системою

формується ім'я на зразок CK__Customers__22AA2996. З цього імені можна зрозуміти, що воно відноситься до обмеження CHECK, але неможливо визначити в чому полягає характер відповідної перевірки CHECK. Крім того, на одній таблиці може бути задано декілька обмежень CHECK. Як приклад, їх імена можуть мати вигляд:

CK__Customers__22AA2996;
CK__Customers__258 69641;
CK__Customers__2 67ABA7A.

7. Обмеження ключів

Первинні ключі являють собою унікальні ідентифікатори для кожного рядка. Стовпець первинного ключа повинен містити унікальні значення (і тому в цьому стовпці не допускається наявність NULL-значення). Не слід плутати первинний ключ, який однозначно ідентифікує кожен рядок у таблиці, з ідентифікатором GUID, який є засобом, застосовуваним для ідентифікації будь-яких об'єктів у розподілених базах даних. Таблиця може мати не більше одного первинного ключа.

Первинний ключ гарантує унікальність поєднання значень атрибутів. Для створення первинного ключа, як правило, застосовуються два способи.

I. Створення первинного ключа за допомогою команди CREATE TABLE.

Виконується під час створення таблиці.

Розглянемо приклад з використанням оператора CREATE TABLE при створенні таблиці Customers:

```
CREATE TABLE Customers
(
CustomerNo int IDENTITY NOT NULL,
CustomerName varchar (30) NOT NULL,
Address1 varchar (30) NOT NULL,
Address2 varchar (30) NOT NULL,
City varchar (20) NOT NULL,
State char (2) NOT NULL,
Zip varchar (10) NOT NULL,
Contact varchar (25) NOT NULL,
Phone char (15) NOT NULL,
FedIDNo varchar (9) NOT NULL,
DateInSystem smalldatetime NOT NULL
)
```

Як первинний ключ позначимо стовпець CustomerNo. Щоб внести корективи в розглянутий оператор CREATE TABLE для задання в ньому обмеження PRIMARY KEY, потрібно ввести інформацію про обмеження відразу після визначення атрибутів, що входять до складу первинного ключа. У даному випадку досить задати ключове слово PRIMARY KEY :

```
CREATE TABLE Customers
```

```
(
CustomerNo int IDENTITY NOT NULL
PRIMARY KEY,
CustomerName varchar ( 30 ) NOT NOLL,
Addressl varchar ( 30 ) NOT NULL,
Address2 varchar ( 30 ) NOT NULL,
City varchar ( 20 ) NOT NULL,
State char ( 2 ) NOT NULL,
Zip varchar ( 10 ) NOT NULL,
Contact varchar ( 25 ) NOT NULL,
Phone char ( 15 ) NOT NULL,
FedlDNo varchar ( 9 ) NOT NULL,
DatelnSystem smalldatetime NOT NULL
)
```

II. Створення первинного ключа за допомогою команди ALTER TABLE для існуючої таблиці.

Розглянемо приклад створення первинного ключа на існуючій таблиці Employees, де зазначено тип обмеження PRIMARY KEY, а також атрибут, на який воно поширюється:

```
USE Accounting
ALTER TABLE Employees
ADD CONSTRAINT PK_EmployeeID
PRIMARY KEY ( EmployeelD )
```

Зовнішні ключі не тільки забезпечують цілісність даних, але і створюють зв'язки між таблицями. Після задання зовнішнього ключа на таблиці встановлюється залежність між таблицею, для якої визначається зовнішній ключ, і таблицею, на яку посилається зовнішній ключ. Після задання зовнішнього ключа таблиці будь-який рядок, що вставляється в цю таблицю, повинен відповідати одній з умов:

– мати узгоджений рядок, який відповідає зовнішньому ключу таблиці;
– мати значення атрибутів зовнішнього ключа, рівне NULL.

Розглянемо застосування визначення на прикладі.

Нехай створено базу даних Accounting. Створимо в ній ще одну таблицю і назвемо її Orders. У наведеному нижче сценарії з оператором CREATE в таблицях передбачено застосування як первинного, так і зовнішнього ключа. Синтаксична структура оголошення зовнішнього ключа передбачає необхідність зазначення атрибутів, на які поширюється обмеження FOREIGN KEY, і має вигляд:

```
<column name><data type><nullability>
FOREIGN KEY REFERENCES <table name> ( <column name> )
[ON DELETE { CASCADE | NO ACTION | SET NULL | SET DEFAULT }]
[ON UPDATE { CASCADEjNO ACTION | SET NULL I SET DEFAULT }]
```

Таким чином, для таблиці Orders, що вводиться до існуючої бази даних, матимемо:

```
USE Accounting CREATE TABLE Orders
(
OrderID int IDENTITY NOT NULL
PRIMARY KEY,
CustomerNo int NOT NULL
FOREIGN KEY REFERENCES Customers (CustomerNo),
OrderDate smalldatetime NOT NULL,
EmployeeID int NOT NULL
)
```

Слід враховувати, що стовпець, на який посилається зовнішній ключ, повинен мати обмеження PRIMARY KEY або UNIQUE.

Після успішного застосування наведеного вище коду потрібно виконати процедуру sp_help. Після цього в розділі constraints буде міститися інформація про нові обмеження. Для отримання детальних відомостей про наявні обмеження потрібно викликати на виконання процедуру sp_helpconstraint, яка також має простий синтаксис:

EXEC sp_helpconstraint <table name>

Таким чином, підтримання власної цілісності повинна забезпечувати сама база даних; кожен зовнішній ключ примусово вводить одне з обмежень, що поширюються на збережені дані, і тому гарантує забезпечення цілісності бази даних.

На відміну від первинних ключів, кількість зовнішніх ключів, заданих на таблиці, не повинно обмежуватися. Єдиною умовою є те, що кожен конкретний атрибут може згадуватися тільки в одному зовнішньому ключі. Проте, в кожному окремому зовнішньому ключі може бути задано декілька атрибутів.

8. Обмеження UNIQUE

Обмеження UNIQUE часто називають обмеженнями можливих ключів. На відміну від обмежень первинного ключа, в обмеженнях UNIQUE не використовуються унікальні ідентифікатори рядків таблиці. Крім того, на таблицю може бути задано множину обмежень UNIQUE та лише один первинний ключ.

Після введення обмеження UNIQUE, дотримується умова, щоб всі значення атрибутів, зазначені в обмеженні, були унікальними, а тому при спробі оновити або вставити рядок зі значеннями атрибутів, який вже є в таблиці, СУБД SQL Server виробляє повідомлення про помилку та відхиляє таку спробу). При цьому не виключається можливість вставити у відповідний стовпець NULL-значення.

9. Обмеження CHECK

Обмеження CHECK може бути застосованим як до окремого стовпця,

так і до таблиці бази даних. За допомогою цього обмеження виконується перевірка значень в одному стовпці на підставі значень другого стовпця (за умови, що стовпці належать до однієї таблиці бази даних), а також відповідності сполучення значень атрибутів заданому критерію.

Обмеження CHECK визначаються правилами, які поширюються на операції перевірки в конструкції WHERE.

Наприклад, для перевірки допустимості в стовпці DatelnSystem (поле системної дати, в якому не може знаходитись значення дати в майбутньому) бази даних Accounting, внесемо в таблицю Customers відповідне обмеження:

```
ALTER TABLE Customers
ADD CONSTRAINT CN_CustomerDateInSystem
CHECK
(DatelnSystem <= GETDATE ())
```

Внесемо в таблицю рядок із значенням, що порушує обмеження CHECK:

```
INSERT INTO Customers
(CustomerName, Addressl, Address2, City, State, Zip, Contact, Phone, FedlDNo, DatelnSystem) VALUES
CCustomerl', 'Addressl', 'Add2', 'MyCity', 'NY', '55555', 'No Contact', '553-1212', '930984954', '12-31-2049') Msg 547, Level 16, State 0, Line 1
```

СУБД SQL Server повідомить про помилку.

10. Обмеження DEFAULT

Обмеження DEFAULT належить до інструментальних засобів забезпечення цілісності даних в рамках таблиці бази даних. Такі обмеження вказують на значення атрибутів, що повинні бути присвоєні відповідним атрибутам при вставці (INSERT) нового рядка, що не містить значень даних. При цьому, слід обов'язково зазначити ці значення в операторі DEFAULT, навіть якщо це NULL-значення. Для введення обмеження DEFAULT достатньо ввести його в кінці визначення атрибута (стовпця). Приклад створення таблиці Shippers з обмеженням DEFAULT:

```
CREATE TABLE Shippers
(
ShipperlD        int              IDENTITY
PRIMARY KEY, ShipperName      varchar(30)
DatelnSystem     smalldatetime DEFAULT GETDATE ()
  )
```

Для перевірки дії заданого обмеження вставимо новий запис (рядок) в таблицю Shippers:

```
INSERT INTO Shippers
(ShipperName) VALUES
('United Parcel Service')
```

Виконання оператора SELECT по таблиці Shippers допоможе

впевнитись у дії обмеження DEFAULT:

SELECT * FROM Shippers

Отримаємо результат

ShipperID	ShipperName	DateInSystem
1	United Parcel Service	2000-07-13 23:26:00

NOT NULL

NOT NULL, NOT NULL

(1 row(s) affected)

Оператори ALTER та ADD, що орієнтовані на застосування до обмежень, повинні містити ключове слово FOR, яке вказує СУБД SQL Server цільовий для обмеження DEFAULT стовпець.

Приклад оператора модифікації таблиці Customers:

ALTER TABLE Customers

ADD CONSTRAINT CN_CustomerDefaultDateInSystem DEFAULT GETDATEO FOR DateInSystem

В таблиці може бути задано кілька обмежень DEFAULT, як і всіх інших обмежень, крім PRIMARY KEY.

Приклад виконання лабораторної роботи

Предметна область: комп'ютерний магазин.

Розглянемо базу даних, що описується таким чином:

CREATE TABLE Component(Name varchar, Price money, Available bit, Created datetime);

CREATE TABLE Processor(Name varchar, Freq decimal, Cool_sys bit, Socket_type varchar);

CREATE TABLE Motherboard(Name varchar, Socket_type varchar, Max_mem decimal, Extrnal_usb_count tinyint, Total_usb_count tinyint, Video_type varchar(10));

Діаграма бази даних зображена на рисунку 3.1 (її можна створити, використавши контекстне меню в "Обозреватель объектов" в SQL Management Studio):

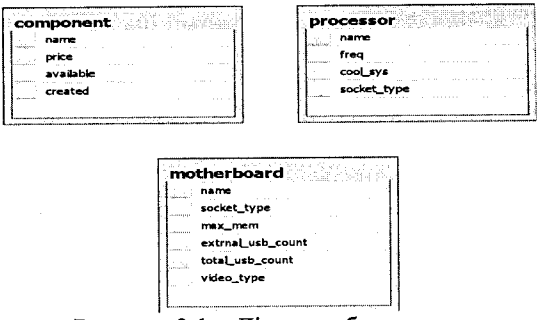


Рисунок 3.1 – Діаграма бази даних

Для таблиці Processor визначимо первинний ключ, використовуючи засоби SQL Management Studio.

1. Додамо новий стовпець до кожної з таблиць бази даних:

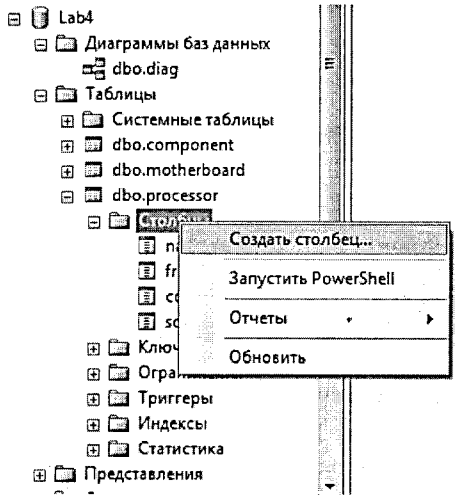


Рисунок 3.2 – Додання нового стовпця до таблиці в SQL Management Studio

2. Для створеного стовпця задамо тип *bigint* та назву *id,* а за допомогою контекстного меню при натисканні на початок рядка таблиці, що описує атрибут, установимо це поле як первинний ключ:

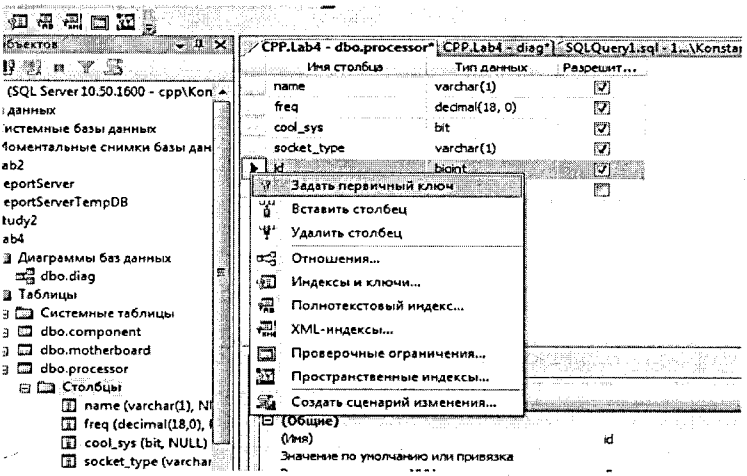


Рисунок 3.3 – Створення первинного ключа для таблиці

3. Забезпечимо автоматичне формування унікального значення ключового поля при доданні нових рядків у таблицю. Для цього необхідно встановити властивість "Спецификация идентификатора", а також задати "Начальное значение идентификатора" та "Шаг приращения идентификатора":

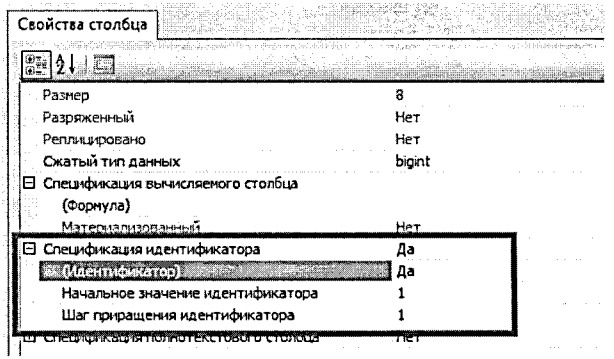


Рисунок 3.4 – Задання ідентифікатора

Після цього таблиця Processor набуде такого вигляду:

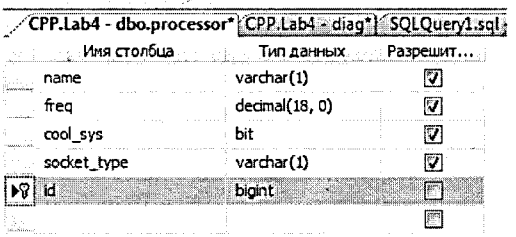


Рисунок 3.5 – Таблиця Processor після задання первинного ключа

Виконаємо аналогічні дії з використанням операторів мови запитів стосовно таблиці Motherboard. Послідовність запитів для цього має такий вигляд:

ALTER TABLE Motherboard ADD id bigint identity(1,1) notnull;

ALTER TABLE Motherboard ADD constraint pk_Motherboard_id PRIMARY KEY(id);

Створимо таблицю Computer, що буде містити інформацію про материнську плату та процесор. SQL-запит для створення таблиці створює первинний ключ, а також обмеження можливого ключа для одного з полів.

CREATE TABLE Computer (id bigint Primary key identity(1,1)notnull, Processor_id bigint FOREIGN KEY References Processor(id), Motherboard_id bigint);

Після створення даної таблиці зміниться схема бази даних – на ній з'явиться зв'язок між таблицями Processor та Computer:

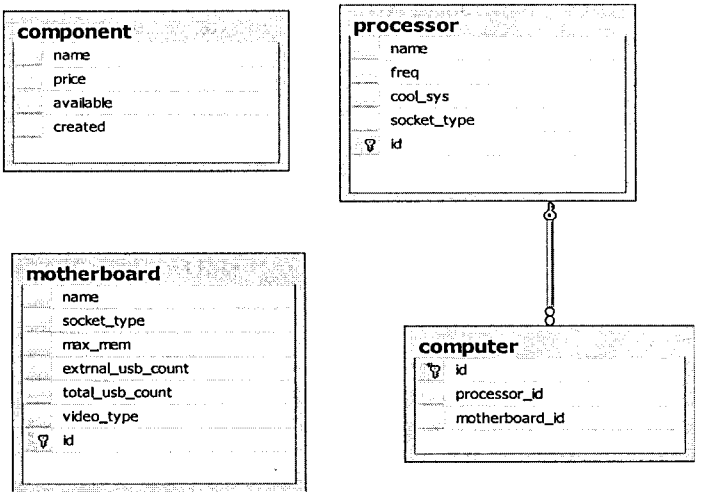


Рисунок 3.6 – Оновлена схема бази даних

Додамо обмеження зовнішнього ключа до таблиці Computer за допомогою запиту:

ALTER TABLE Computer ADD constraint fk_computer_has_motherboard FOREIGN KEY (motherboard_id) References motherboard(id);

Зовнішній ключ також можна задати в самій схемі бази даних, для чого необхідно "перетягнути" поле ключа таблиці Motherboard до відповідного поля таблиці Computer. На екрані з'явиться діалог для формування обмеження зовнішнього ключа:

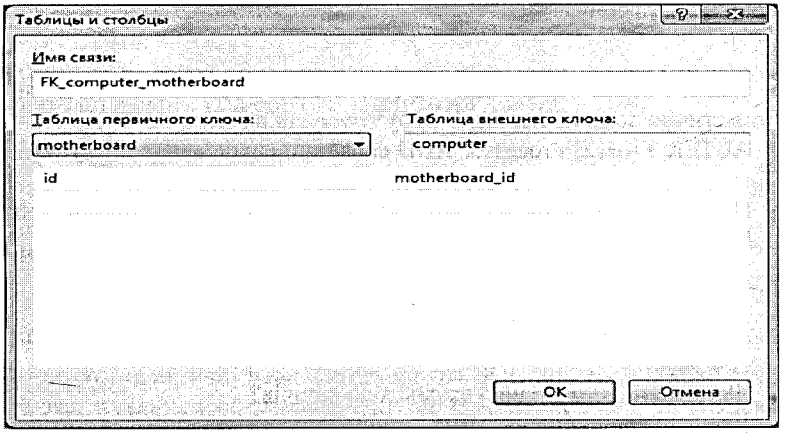


Рисунок 3.7 – Діалог для формування обмеження зовнішнього ключа

Схема бази даних набуде вигляду, поданого на рисунку 3.8.

Сформуємо обмеження типу CHECK до таблиці Motherboard:

ALTER TABLE Motherboard ADD constraint cn_external_usb_count CHECK(extrnal_usb_count<=total_usb_count);

В даному запиті встановлюється обмеження на кількість зовнішніх USB-портів материнської плати, що не може перевищувати загальної їх кількості.

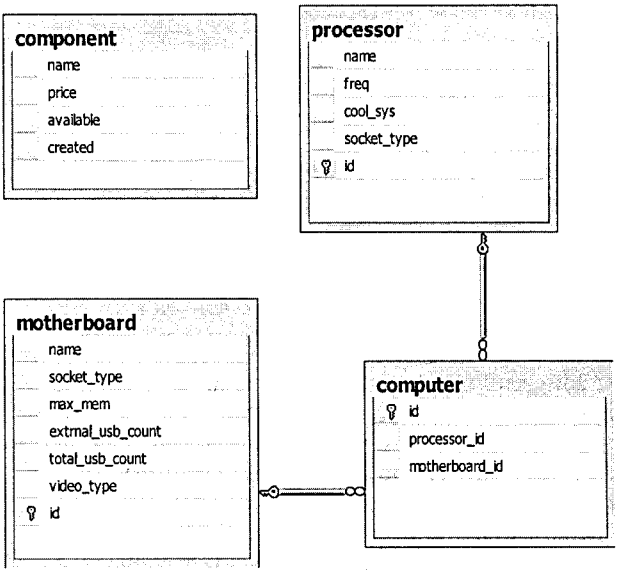


Рисунок 3.8 – Оновлена схема бази даних із сформованим зовнішнім ключем

Додамо обмеження типу Default до таблиці Processor, що буде автоматично встановлювати атрибут *cool_sys* в значення "1":

ALTER TABLE Processor ADD constraint cn_default_cool_sys DEFAULT 1 For cool_sys;

Після цього при доданні рядків в таблицю Processor значення атрибута *cool_sys* задавати не обов'язково.

КОНТРОЛЬНІ ПИТАННЯ

1. Дайте означення таких понять: ключ, первинний та зовнішній ключ, обмеження.
2. Які основні вимоги висуваються до ключів?
3. Які типи відношень між атрибутами характерні для різних типів ключів?

4. В яких випадках застосовуються зовнішні ключі?
5. Коли ключ містить всі атрибути відношення?
6. Які типи обмежень ви знаєте?
7. Що таке обмеження домену?
8. Дайте визначення обмеження сутності.
9. Наведіть приклад застосування обмеження Primary key.
10. Наведіть приклад застосування обмеження Foreign key.
11. Наведіть приклад таблиці, що посилається сама на себе.
12. Наведіть приклад застосування обмеження Unique.
13. Наведіть приклад застосування обмежень Check та Default.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Дейт К. Дж. Введение в системы баз данных / Дейт К. Дж. – [8-е изд.]. – М. : Изд-во "Вильямс", 2006. – 1328 с.
2. Дейт К. SQL и реляционная теория. Как грамотно писать код на SQL / Дейт К. – М. : Изд-во "Символ-Плюс", 2010. – 480 с.
3. Роберт Виейра Программирование баз данных Microsoft SQL Server 2008. Базовый курс / Роберт Виейра. – М. : Изд-во: "Диалектика-Вильямс", 2009. – 816 с.
4. Бен-Ган И. Microsoft SQL Server 2008. Основы T-SQL / Бен-Ган И. – Санкт-Петербург : Изд-во "Русская редакция", 2009. – 432 с.
5. Тернстрем Т. Microsoft SQL Server 2008. Разработка баз данных. Учебный курс Microsoft, экзамен 70–433 (+ CD) / Тернстрем. Т., Вебер Э., Хотек М. – М. : Изд-во "Русская Редакция", 2010. – 496 с.
6. Петкович Д. Microsoft SQL Server 2008. Руководство для начинающих / Петкович Д. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 752 с.
7. Гробф Джеймс Р. SQL: полный справочник / Гробф Джеймс Р., Вайнберг Пол Н., Оппель Эндрю Дж. – Москва : Изд-во "Диалектика-Вильямс", 2010. – 960 с.
8. Дьюсон Р. SQL Server 2008 для начинающих разработчиков / Дьюсон Р. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 704 с.

ЛАБОРАТОРНА РОБОТА № 4
SQL MANAGEMENT STUDIO:
ВИБИРАННЯ ДАНИХ ЗА ДОПОМОГОЮ SQL-ЗАПИТІВ.
ГРУПОВІ ОПЕРАЦІЇ

Мета: отримати практичні навички створення SQL-запитів для вибирання даних із застосуванням операторів ORDER BY, GROUP BY та групових операцій.

Порядок виконання роботи

1. Ознайомитись із теоретичною частиною.
2. Створити базу даних для виконання лабораторної роботи, яка має містити одну таблицю із 3–5 атрибутами числового типу (дата, час, числові типи даних) та двома атрибутами текстового типу.
3. Ввести в таблицю не менше 30 записів.
4. Створити запит, що буде виводити дані з таблиці, відсортовані за трьома атрибутами.
5. Створити запит для вибирання записів із таблиці, що мають задане викладачем значення одного довільного атрибута.
6. Створити запити для вибирання даних із таблиці з використанням групування за одним із атрибутів:
   а) для кожної групи вивести записи, що до неї входять, а також суму значень для одного із заданих викладачем числових атрибутів;
   б) для кожної групи вивести максимальне значення одного заданого викладачем атрибута числового типу;
   в) для кожної групи вивести середнє значення заданого викладачем числового атрибута.
7. Дати відповіді на контрольні питання.
8. Оформити звіт.

Теоретичні відомості

1. Сортування результатів вибирання

Для виведення даних, відсортованих за будь-яким атрибутом, використовується ключове слово ORDER BY. Результат вибирання можна відсортувати одночасно по 16 стовпцях. У Transact-SQL в речення ORDER BY можна містити атрибути або вирази, відсутні у списку вибирання. Сортувати можна за іменами атрибутів, за виразом або за номером, який вказує позицію стовпця в списку вибирання. Під час сортування за номером стовпця необхідно вказувати той номер, який реально присутній у списку вибирання.

Розглянемо приклад:

```
[ ORDER BY
   {
   order_by_expression
  [ COLLATE collation_name ]
  [ ASC | DESC ]
   } [ ,...n ]
]
```

ORDER_BY_EXPRESSION вказує стовпець, за яким має виконуватися сортування. COLLATE {*collation_name*} вказує, що операція ORDER BY повинна виконуватися відповідно до параметрів сортування, зазначених в аргументі collation_name, а згідно з параметрами сортування стовпця,

визначених у таблиці або поданні. Значення collation_name може бути ім'ям параметрів сортування Windows або ім'ям параметрів сортування SQL. Аргумент COLLATE застосовується тільки до стовпців даних типу Char, Varchar, Nchar і Nvarchar. ASC вказує, що значення в зазначеному стовпці повинні сортуватися за зростанням від менших значень до більших. Як правило, сортування за замовчуванням – ASC. DESC вказує, що значення в зазначеному стовпці повинні сортуватися за спаданням від більших значень до менших.

Значення NULL при сортуванні виводиться першим. Стовпці типу text і image використовувати в реченні ORDER BY не можна. Вкладені запити і подання не можуть містити оператор ORDER BY.

Приклад запиту на сортування в зворотному алфавітному порядку за допомогою оператора ORDER BY матиме вигляд:

SELECT top 5 * from Memory ORDER BY Frequency DESC;

Результат виконання запиту зображений на рисунку 4.1.

Результаты | Сообщения

	id	mtype	vendor	model	size	frequency	latency	price
1	29	DDR3	Corsair	PC3-19200	4000000000	2400000000	3	600.00
2	28	DDR3	Corsair	PC3-17000	4000000000	2133000000	5	500.00
3	18	DDR3	Kingston	PC3-16000	2000000000	2000000000	8	231.00
4	27	DDR3	Corsair	PC3-16000	2000000000	2000000000	5	400.00
5	26	DDR3	Corsair	PC3-14400	2000000000	1800000000	4	300.00

Рисунок 4.1 – Результат виконання запиту із сортуванням значення стовпця в зворотному алфавітному порядку

2. Групування даних

Функціями агрегування називаються функції, які використовуються для отримання сумарних значень. Вони застосовуються до:

- всіх записів у таблиці;
- тільки тих записів, які вибираються в операторі WHERE;
- груп записів, створених за допомогою оператора GROUP BY.

При заданні функцій агрегування потрібно визначити аргумент. Аргумент є виразом та береться в дужки. Загальний синтаксис функцій агрегування має вигляд:

<Функція агрегування> ([DISTINCT] <Вираз>)

Вираз, як правило, є стовпцем, але також може бути константою, функцією, комбінацією імен стовпців, констант, функцій, об'єднаних арифметичними операторами. У таблиці 4.1 перераховані всі функції агрегування, доступні в SQL Server.

Опцію DISTINCT можна використовувати з усіма функціями, крім COUNT (*).

Таблиця 4.1 – Функції агрегування SQL Server

Функція	Результат
AVG	Повертає середнє арифметичне для значень виразу; NULL-значення ігноруються
COUNT	Повертає кількість елементів у виразі (рівне кількості рядків)
COUNT_BIG	Те ж саме, що COUNT, але результат має тип даних *Bigint*, а не *Int*
GROUPING	Повертає спеціальний додатковий стовпець; застосовується, коли вираз GROUP BY містить операцію CUBE або ROLLUP
MAX	Повертає найбільше значення із значень виразу
MIN	Повертає найменше значення із значень виразу
STDEV	Повертає середнє квадратичне відхилення для всіх величин виразу. Ця функція передбачає, що вираз який використовується в розрахунку є зразком всієї сукупності даних
STDEVP	Повертає середнє квадратичне відхилення для всіх величин виразу. Ця функція передбачає, що вираз який використовується в розрахунку є всією сукупністю даних
SUM	Повертає суму всіх значень у виразі (або по всіх записах, або лише по унікальних)
VAR	Повертає значення дисперсії для всіх значень виразу. Ця функція передбачає, що вирази, які використовуються в розрахунку, є зразком всієї сукупності даних
VARP	Повертає значення дисперсії для всіх значень виразу. Ця функція передбачає, що вирази, які використовуються в розрахунку, є всією сукупністю даних

Функції SUM і AVG можуть використовуватися тільки з числовими виразами. Решту функцій можна застосовувати з виразами будь-якого типу.

Використання оператора DISTINCT дозволяє не враховувати дублювання значень з вибирання перед застосуванням функцій SUM, AVG, COUNT.

Використовувати функції агрегування в операторі WHERE не можна. Проте можна використовувати оператор WHERE для вибирання необхідних записів. Наприклад:

SELECT COUNT(*) From Memory WHERE Frequency > 800000000;

Для реалізації складних вибірок, що містять декілька стовпців, не обійтися без групування даних. Для групування даних в команді SELECT використовується оператор GROUP BY:

GROUP BY [ALL] <Умова групування 1> {, <Умова групування 2>]...

[, <Умова групування 3>] [WITH {CUBE | ROLLUP}]

Групування даних зазвичай супроводжується використанням функцій агрегування.

При використанні GROUP BY з оператором WHERE необхідно пам'ятати про послідовність виконання операцій. Спочатку оператор WHERE фільтрує дані, а потім, з решти записів, GROUP BY створює набори. Ті записи, які не будуть відібрані WHERE, не потраплять ні в одну групу.

При використанні GROUP BY ніякого сортування не відбувається. Для того, щоб відсортувати отримані дані в потрібному порядку, використовуйте ORDER BY. Оператор GROUP BY має обов'язково знаходитися перед оператором ORDER BY.

Стовпці, включені в оператор GROUP BY, утворюють перехресну таблицю, що дозволяє отримати додаткові групи. Якщо для цих стовпців використовуються функції агрегування (зазвичай підсумовування), то ми отримуємо додаткові розрахункові значення. Число додаткових груп визначається кількістю стовпців, включених в опцію GROUP BY.

Оператор CUBE використовується при підготовці даних для звітів, графіків або діаграм для отримання при групуванні даних додаткової аналітичної інформації, що забезпечується можливістю використання з усіма функціями агрегування, включно з AVG, SUM, MAX, MIN та COUNT. Він перебирає всі варіанти поєднань стовпців.

Оператор ROLLUP використовується для однонаправленого перебору всіх стовпців, які використовуються для агрегування даних.

Приклад виконання лабораторної роботи

Предметна область: комп'ютерний магазин.

Створимо таблицю Memory, яка буде описувати модулі оперативної пам'яті, що є в продажу у магазині. Нехай вона містить такі атрибути:

Id – код модуля пам'яті;

mType – тип пам'яті, текстовий тип, 20 символів;

Vendor – виробник модуля пам'яті, текстовий тип, 40 символів;

Model – модель модуля пам'яті, текстовий тип, 40 символів;

Size – об'єм модуля пам'яті, 64-бітове число;

Frequency – максимальна частота, при якій модель модуля пам'яті може працювати, 64-бітове число;

Latency – затримка при доступі до пам'яті, мілісекунди, 32-бітове число;

Price – вартість модуля пам'яті, грошовий тип.

Створимо таблицю Memory за допомогою запиту:

```
CREATE TABLE Memory(mtype varchar(20), vendor varchar(40), model varchar(40), size bigint, frequency bigint, latency integer, price money);
```

Нехай сформована таблиця 4.2 в результаті введення в неї інформації.

SQL Management Studio дозволяє скопіювати дані таблиці, щоб використовувати їх в інших офісних додатках:

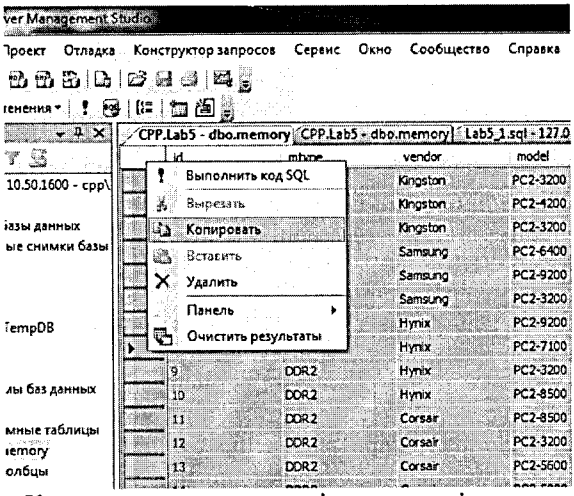


Рисунок 4.2 – Контекстне меню для копіювання виділених записів таблиці

Таблиця 4.2 – Дані, введені в таблицю бази даних

Id	Mtype	vendor	model	size	frequency	latency	price
1	2	3	4	5	6	7	8
1	DDR2	Kingston	PC2-3200	1000000000	400000000	4	100,0000
2	DDR2	Kingston	PC2-4200	1000000000	533000000	5	120,0000
3	DDR2	Kingston	PC2-3200	2000000000	400000000	6	200,0000
4	DDR2	Samsung	PC2-6400	2000000000	800000000	7	300,0000
5	DDR2	Samsung	PC2-9200	500000000	1150000000	4	60,0000
6	DDR2	Samsung	PC2-3200	5000000000	400000000	9	50,0000
7	DDR2	Hynix	PC2-9200	2000000000	1150000000	5	130,0000
8	DDR2	Hynix	PC2-7100	1000000000	888000000	6	98,0000
9	DDR2	Hynix	PC2-3200	1000000000	400000000	7	87,0000
10	DDR2	Hynix	PC2-8500	2000000000	1066000000	9	200,0000
11	DDR2	Corsair	PC2-8500	1000000000	1066000000	7	150,0000
12	DDR2	Corsair	PC2-3200	1000000000	400000000	6	91,5000
13	DDR2	Corsair	PC2-5600	2000000000	700000000	5	170,0000
14	DDR2	Corsair	PC2-6000	4000000000	750000000	7	250,0000
15	DDR2	Corsair	PC2-9600	4000000000	1200000000	5	600,0000
16	DDR3	Kingston	PC3-6400	2000000000	800000000	4	123,0000
17	DDR3	Kingston	PC3-10600	2000000000	1333000000	6	324,0000
18	DDR3	Kingston	PC3-16000	2000000000	2000000000	8	231,0000
19	DDR3	Samsung	PC3-12800	2000000000	1600000000	5	234,0000
20	DDR3	Samsung	PC3-6400	2000000000	800000000	6	444,0000
21	DDR3	Samsung	PC3-12800	4000000000	1600000000	7	555,0000
22	DDR3	Samsung	PC3-8500	4000000000	1066000000	4	666,0000
23	DDR3	Hynix	PC3-6400	2000000000	800000000	4	332,0000
24	DDR3	Hynix	PC3-8500	4000000000	1066000000	5	100,0000

Продовження таблиці 4.2

1	2	3	4	5	6	7	8
25	DDR3	Hynix	PC3-12800	4000000000	1600000000	6	200,0000
26	DDR3	Corsair	PC3-14400	2000000000	1800000000	4	300,0000
27	DDR3	Corsair	PC3-16000	2000000000	2000000000	5	400,0000
28	DDR3	Corsair	PC3-17000	4000000000	2133000000	5	500,0000
29	DDR3	Corsair	PC3-19200	4000000000	2400000000	3	600,0000
30	DDR	Corsair	PC-3200	1000000000	400000000	6	300,0000
31	DDR	Hynix	PC-3200	1000000000	400000000	7	250,0000
32	DDR	Kingston	PC-3200	1000000000	400000000	8	200,0000

Запит для вибирання даних із сортуванням за атрибутами Price, Frequency, Size має вигляд:

SELECT * From Memory ORDER BY Price, Frequency, Size;

Результат виконання запиту:

Результаты | Сообщения

	id	mtype	vendor	model	size	frequency	latency	price
1	6	DDR2	Samsung	PC2-3200	5000000000	400000000	9	50,00
2	5	DDR2	Samsung	PC2-9200	500000000	1150000000	4	60,00
3	9	DDR2	Hynix	PC2-3200	1000000000	400000000	7	87,00
4	12	DDR2	Corsair	PC2-3200	1000000000	400000000	6	91,50
5	8	DDR2	Hynix	PC2-7100	1000000000	888000000	6	98,00
6	1	DDR2	Kingston	PC2-3200	1000000000	400000000	4	100,00
7	24	DDR3	Hynix	PC3-8500	4000000000	1066000000	5	100,00
8	2	DDR2	Kingston	PC2-4200	1000000000	533000000	5	120,00
9	16	DDR3	Kingston	PC3-6400	2000000000	800000000	4	123,00
10	7	DDR2	Hynix	PC2-9200	2000000000	1150000000	5	130,00
11	11	DDR2	Corsair	PC2-8500	1000000000	1066000000	7	150,00
12	13	DDR2	Corsair	PC2-5600	2000000000	700000000	5	170,00

Рисунок 4.3 – Результат виконання запиту для вибирання даних, відсортованих за атрибутами Price, Frequency, Size

Запит для вибирання 5 модулів пам'яті, що мають найбільшу частоту від найбільшого до меншого за значенням частоти:

SELECT top 5 * From Memory ORDER BY Frequency DESC;

Результат виконання запиту зображений на рисунку 4.4:

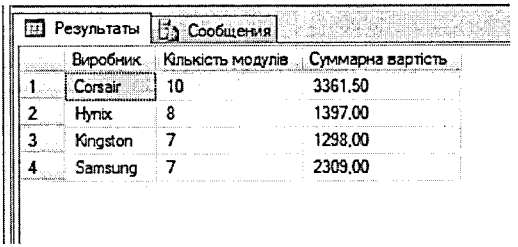
Результаты | Сообщения

	id	mtype	vendor	model	size	frequency	latency	price
1	29	DDR3	Corsair	PC3-19200	4000000000	2400000000	3	600,00
2	28	DDR3	Corsair	PC3-17000	4000000000	2133000000	5	500,00
3	18	DDR3	Kingston	PC3-16000	2000000000	2000000000	8	231,00
4	27	DDR3	Corsair	PC3-16000	2000000000	2000000000	5	400,00
5	26	DDR3	Corsair	PC3-14400	2000000000	1800000000	4	300,00

Рисунок 4.4 – Результат виконання запиту для вибирання 5 модулів пам'яті, що мають найбільшу частоту

Запит для вибирання модулів та підрахунку їх сумарної вартості за заданим виробником та кількістю, результат виконання якого подано на рис. 4.5:

```
SELECT vendor as 'Виробник',  COUNT(*) as 'Кількість модулів',
SUM(Price) as 'Суммарна вартість' From Memory GROUP BY vendor;
```



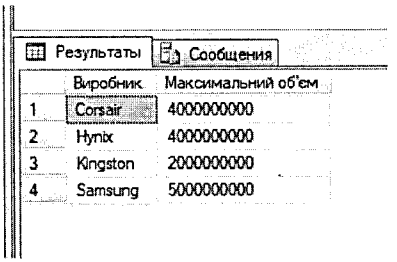
Результаты | Сообщения

	Виробник	Кількість модулів	Суммарна вартість
1	Corsair	10	3361.50
2	Hynix	8	1397.00
3	Kingston	7	1298.00
4	Samsung	7	2309.00

Рисунок 4.5 – Вибирання модулів та підрахунок їх сумарної вартості за заданим виробником та кількістю

Запит для вибирання модуля максимального обсягу для кожного виробника, результат виконання якого подано на рис. 4.6:

```
SELECT vendor as 'Виробник',  max(size) as 'Максимальний обсяг' From
Memory GROUP BY vendor;
```



Результаты | Сообщения

	Виробник	Максимальний об'єм
1	Corsair	4000000000
2	Hynix	4000000000
3	Kingston	2000000000
4	Samsung	5000000000

Рисунок 4.6 – Результат виконання запиту для вибирання максимального обсягу модуля для кожного виробника

Запит для виведення середніх значень усіх параметрів модулів пам'яті, згрупованих за атрибутом "Тип", результат виконання якого подано на рис. 4.7:

```
SELECT mtype, AVG(size) as 'Середній об'єм', AVG(frequency) as 'Середня
частота', AVG(latency) as 'Середня затримка доступу', AVG(price) as
'Середня ціна' From Memory GROUP BY mtype;
```

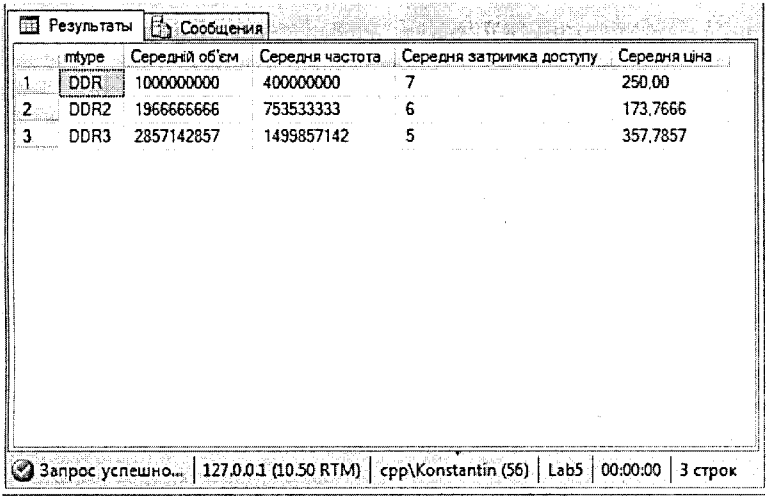


Рисунок 4.7 – Результат виконання запиту для виведення середніх значень усіх параметрів модулів пам'яті, згрупованих за атрибутом "Тип"

КОНТРОЛЬНІ ПИТАННЯ

1. Навести приклад запиту на сортування результатів вибирання.
2. Яка основна мета застосування оператора ORDER BY?
3. До чого можуть застосовуватись функції агрегування?
4. Наведіть приклади запитів із використанням відомих вам функцій агрегування.
5. Функції агрегування Transact-SQL.
6. Яке призначення оператора DISTINCT?
7. Для чого застосовується групування результатів вибирання?
8. Поясніть різницю між агрегуванням та групуванням результатів вибирання, наведіть приклад.
9. Наведіть приклад запиту із застосуванням оператора GROUP BY та функцій агрегування.
10. Для чого застосовуються оператори CUBE та ROLLUP?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Дейт К. Дж. Введение в системы баз данных / Дейт К. Дж. – [8-е изд.]. – М. : Изд-во "Вильямс", 2006. – 1328 с.
2. Дейт К. SQL и реляционная теория. Как грамотно писать код на SQL / Дейт К. – М. : Изд-во "Символ-Плюс", 2010. – 480 с.
3. Роберт Виейра Программирование баз данных Microsoft SQL Server 2008. Базовый курс / Роберт Виейра. – М. : Изд-во: "Диалектика-Вильямс", 2009. – 816 с.

4. Бен-Ган И. Microsoft SQL Server 2008. Основы T-SQL / Бен-Ган И. – Санкт-Петербург : Изд-во "Русская редакция", 2009. – 432 с.
5. Тернстрем Т. Microsoft SQL Server 2008. Разработка баз данных. Учебный курс Microsoft, экзамен 70-433 (+ CD) / Тернстрем. Т., Вебер Э., Хотек М. – М. : Изд-во "Русская Редакция", 2010. – 496 с.
6. Петкович Д. Microsoft SQL Server 2008. Руководство для начинающих / Петкович Д. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 752 с.
7. Грофф Джеймс Р. SQL: полный справочник / Грофф Джеймс Р., Вайнберг Пол Н., Оппель Эндрю Дж. – Москва : Изд-во "Диалектика-Вильямс", 2010. – 960 с.
8. Дьюсон Р. SQL Server 2008 для начинающих разработчиков / Дьюсон Р. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 704 с.

ЛАБОРАТОРНА РОБОТА № 5
SQL MANAGEMENT STUDIO:
ВИБИРАННЯ ДАНИХ ЗА ДОПОМОГОЮ SQL-ЗАПИТІВ.
ОПЕРАТОРИ ОБ'ЄДНАННЯ

Мета: отримати практичні навички створення SQL-запитів з використанням операторів об'єднання INNER JOIN, LEFT OUER JOINT, RIGHT OUTER JOIN та FULL OUTER JOIN.

Порядок виконання роботи

1. Ознайомитись із теоретичною частиною.
2. Розробити базу даних для виконання лабораторної роботи відповідно до обраної предметної області, яка має містити не менше трьох відношень з арністю не менше 10, форма нормалізації відношень – не менше третьої.
3. Написати за завданням викладача SQL-запити для створення бази даних та виконати їх.
4. Ввести в таблиці не менше, ніж по 5 записів.
5. Написати та виконати запит для вибирання даних із двох таблиць з використанням оператора INNER JOIN, проаналізувати та пояснити отримані результати.
6. Написати та виконати запит для вибирання даних із двох таблиць з використанням оператора LEFT OUTER JOIN, проаналізувати та пояснити отримані результати.
7. Написати та виконати запит для вибирання даних із двох таблиць з використанням оператора LEFT OUTER JOIN та групових функцій, проаналізувати та пояснити отримані результати.

8. Написати та виконати запит оновлення даних однієї із двох таблиць з використанням операторів UPDATE, LEFT OUTER JOIN та групових функцій, проаналізувати та пояснити отримані результати.
9. Оформити звіт.

Теоретичні відомості

Microsoft SQL Server спроектований для роботи із реляційними базами даних і забезпечує високу швидкість оброблення та цілісність даних, якщо дані організовані у вигляді багатьох взаємопов'язаних таблиць.

Нехай задана таблиця myTable (CustName, CastAddr, CustCity, Product1Name, Product1Price, Product2Name, Product2Price), в якій записана інформація про замовлення клієнта.

Таблиця може бути розбита на три взаємопов'язаних таблиці – Customers, Products і Sales, як показано на рисунку 5.1. Процес розбиття однієї таблиці на декілька називається нормалізацією.

Створимо тестову базу даних:

```
USE MASTER;
CREATE DATABASE JoinTest;
USE JoinTest;
CREATE TABLE Customers
    (CustomerNamevarchar(50),Addrvarchar(50),Cityvarchar(50) );
CREATE TABLE Products
    (ProductNamevarchar(50),Descrvarchar(50),Pricemoney);
CREATE TABLE Sales
    (SaleDatedatetime, ProductNamevarchar(50),
CustomerNamevarchar(50), Quantityint);
INSERT INTO Customers VALUES ('MrSmith', '123 ElmSt', 'Milwuakee');
INSERT INTO Customers VALUES ('MrJones', '456 SouthSt', 'Chicago');
INSERT INTO Customers VALUES ('MrBrown', '789 OakSt', 'StLouis');
INSERT INTO Products VALUES ('GreenWidgit', 'LagreGreen', 100);
INSERT INTO Products VALUES ('BlueWidgit', 'XLagreBlue', 130);
INSERT INTO SALES VALUES ('01/01/07', 'GreenWidgit', 'MrSmith', 1);
INSERT INTO SALES VALUES ('02/02/07', 'GreenWidgit', 'MrSmith', 2);
INSERT INTO SALES VALUES ('02/02/07', 'BlueWidgit', 'MrSmith', 3);
INSERT INTO SALES VALUES ('01/01/07', 'BlueWidgit', 'MrJones', 1);
```

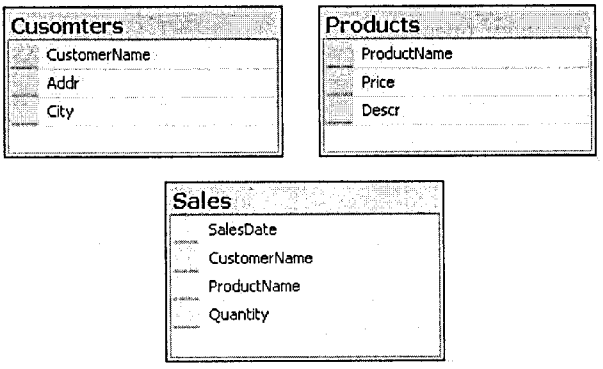


Рисунок 5.1 – Приклад зображення трьох взаємопов'язаних таблиць

Існує декілька типів операторів об'єднання. Оператор INNER JOIN (рисунок 5.2) використовують, щоб повернути дані із всіх таблиць, вказаних в операторі, відповідно до певної умови.

```
SELECT Customers.CustomerName, Sales.SaleDate
FROM Customers
INNER JOIN Sales ON Customers.CustomerName = Sales.CustomerName;
```

CustomerName	SaleDate
Mr Smith	2007-01-01 00:00:00.000
Mr Smith	2007-02-02 00:00:00.000
Mr Smith	2007-02-02 00:00:00.000
Mr Jones	2007-01-01 00:00:00.000

Рисунок 5.2 – Приклад використання оператора INNER JOIN

Запит повертає записи, коли атрибут в таблиці Customer збігається з атрибутом CustomerName в таблиці Sales. Синтаксис оператора в цьому випадку починається як в простому операторі SELECT, де вказуються стовпці, що мають бути повернуті в результаті виконання запиту. В умові FROM вказується об'єднання JOIN типу INNER і умова зв'язку таблиць. Розглянемо приклад, де умовою є збіг значень стовпця CustomerName для обох таблиць. Для об'єднання INNER JOIN порядок таблиць в запиті не має значення. Запит із прикладу може бути з вказанням таблиці Sales перед Customers:

```
SELECT Customers.CustomerName, Sales.SaleDate
FROM Sales
INNER JOIN Customers ON Customers.CustomerName =
Sales.CustomerName
```

Також слід звернути увагу на те, що об'єднання JOIN не висуває вимог до використання стовпців, які є первинним (PRIMARY) або зовнішнім (FOREIGN) ключами. В цьому прикладі об'єднання виконано за допомогою неключового стовпця типу Varchar. Проте, об'єднання з стовпцями, що є первинними ключами, підвищує продуктивність. Об'єднання з використанням стовпців типу Int виконується швидше, ніж об'єднання з використанням стовпців типу Text і тому є рекомендованим. Якщо поле CustomerName є первинним ключем, можна бути впевненим що всі його значення є унікальними. Це важливо тому, що кількість записів, яка повертається при використанні INNER JOIN, для двох таблиць визначається як добуток кількості рядків в першій таблиці та кількості рядків в другій таблиці. Тому в прикладі повертається один запис в таблиці Customer з трьома записами в таблиці Sales, тобто 3 записи. Відповідно, якщо в таблиці клієнтів буде дублюватись запис «Mr. Smit», буде повернуто 6 записів (два записи в таблиці Customers з трьома записами в таблиці Sales), як показано на рисунку 5.3.

CustomerName	SaleDate
Mr Smith	2007-01-01 00:00:00.000
Mr Smith	2007-02-02 00:00:00.000
Mr Smith	2007-02-02 00:00:00.000
Mr Jones	2007-01-01 00:00:00.000
Mr Smith	2007-01-01 00:00:00.000
Mr Smith	2007-02-02 00:00:00.000
Mr Smith	2007-02-02 00:00:00.000

Рисунок 5.3 – Приклад використання оператора INNER JOIN

Об'єднання INNER JOIN використовується в умові WHERE, наприклад, для пошуку всіх клієнтів, що робили замовлення:

```
SELECT Customers.CustomerName, Sales.SaleDate
FROM Sales, Customers
WHERE Customers.CustomerName = Sales.CustomerName
```

Для вивчення об'єднання LEFT OUTER JOIN створимо нову базу даних.

```
USE MASTER;
CREATE DATABASE JoinTest;
USE JoinTest;
CREATE TABLE Customers
(CustIDint, CustomerNamevarchar(50),Addrvarchar(50),Cityvarchar(50)
);
CREATE TABLE Products
(ProdIDint, ProductNamevarchar(50),Descrvarchar(50),Pricemoney
);
CREATE TABLE Sales
(SaleDatedatetime, ProdIDint, CustIDint, Quantityint);
```

```
INSERT INTO Customers VALUES (1, 'MrSmith', '123 ElmSt', 'Milwuakee');
INSERT INTO Customers VALUES (2, 'MrJones', '456 SouthSt', 'Chicago');
INSERT INTO Customers VALUES (3, 'MrBrown', '789 OakSt', 'StLouis');
INSERT INTO Products VALUES (4, 'GreenWidgit', 'LagreGreen', 100);
INSERT INTO Products VALUES (5, 'BlueWidgit', 'XLagreBlue', 130);
INSERT INTO SALES VALUES ('01/01/07', 4, 1, 1);
INSERT INTO SALES VALUES ('02/02/07', 4, 1, 2);
INSERT INTO SALES VALUES ('02/02/07', 5, 1, 3);
INSERT INTO SALES VALUES ('01/01/07', 5, 2, 1);
```

Дані запити створюють три таблиці, наведені на рисунку 5.4.

На відміну від INNER JOIN, де кожна таблиця, з якою виконується об'єднання, повинна мати хоча б один відповідний запис, об'єднання OUTER JOIN повертає всі записи із першої таблиці, і будь-які записи, що підходять під умову, із іншої таблиці.

Для пошуку всіх клієнтів (значень атрибута CustomerName) незалежно від того, чи робили вони замовлення слід сформувати запит:

```
SELECT Customers.CustomerName, Sales.SaleDate
FROM Customers
LEFT OUTER JOIN Sales ON Sales.CustID = Customers.CustID
```

Table - dbo.Customers

CustID	CustomerName	Addr	City
1	Mr Smith	123 Elm St	Milwuakee
2	Mr Jones	456 South St	Chicago
3	Mr Brown	789 Oak St	St Louis

1 of 3

Table - dbo.Products

ProdID	PrpductName	Descr	Price
4	Green Widgit	Lagre Green	100.0000
5	Blue Widgit	XLagre Blue	130.0000

1 of 2

Table - dbo.Sales

SaleDate	ProdID	CustID	Quantity
1/1/2007 12:00:...	4	1	1
2/2/2007 12:00:...	4	1	2
2/2/2007 12:00:...	5	1	3
1/1/2007 12:00:...	5	2	1

Рисунок 5.4 – Створення таблиць

При цьому порядок таблиць, що вказуються після ключового слова "ON", значення не має. Результат виконання запиту:

	CustomerName	SaleDate
1	Mr Smith	2007-01-01 00:00:00.000
2	Mr Smith	2007-02-02 00:00:00.000
3	Mr Smith	2007-02-02 00:00:00.000
4	Mr Jones	2007-01-01 00:00:00.000
5	Mr Brown	NULL

Рисунок 5.5 – Приклад використання оператора LEFT OUTER JOIN

Як видно, із виконання запиту в результаті наявні імена всіх клієнтів, включно із "MrBrown", для якого немає відповідних записів у таблиці замовлень (об'єднання повертає NULL як дату замовлення).

Об'єднання RIGHT JOIN виконує таку саму функцію, як і LEFT JOIN. За необхідності повернути всі записи із правої частини умови запит повинен містити оператор RIGHT JOIN:

```
SELECT Customers.CustomerName, Sales.SaleDate
FROM Sales
 RIGHT OUTER JOIN Customers ON Sales.CustID = Customers.CustID
```

Об'єднання типу FULL OUTER JOIN повертає дані із обох таблиць, незалежно від наявності значень атрибутів у кожній. Для демонстрації дії оператора введемо до таблиці SALES рядок, що не буде містити значення атрибуту CustomerName клієнта:

```
INSERT INTO SALES VALUES ('01/01/07', 0, 0, 1)
```

Запит із використанням FULL OUTER JOIN:

```
SELECT Customers.CustomerName, Sales.SaleDate
FROM Sales
FULL OUTER JOIN Customers ON Sales.CustID = Customers.CustID
```

Результат виконання запиту:

	CustomerName	SaleDate
1	Mr Smith	2007-01-01 00:00:00.000
2	Mr Smith	2007-02-02 00:00:00.000
3	Mr Smith	2007-02-02 00:00:00.000
4	Mr Jones	2007-01-01 00:00:00.000
5	NULL	2007-01-01 00:00:00.000
6	Mr Brown	NULL

Рисунок 5.6 – Результат використання операції FULL OUTER JOIN

З метою спрощення роботи з об'єктами (звертання до об'єктів бази даних під іншим іменем), використовують псевдоніми (Aliases). Наприклад, якщо є таблиця із довгою назвою "DivisonSalesResultsCube",

буде зручно її скоротити до псевдоніму "DivSales" та використовувати його в SQL-запитах.

Приклад використання псевдоніму для таблиці Sales:

```
SELECT s.CustID, s.CustName, s.Address FROM Sales s
```

Параметр "s" після назви таблиці Sales створює псевдонім для цієї таблиці. Псевдонім може бути символом або словом. Після цього псевдонім використовується перед кожною назвою стовпця, вказуючи який стовбець "s" (таблиці Sales) опрацьовується.

Розглянемо приклад (рис. 5.7):

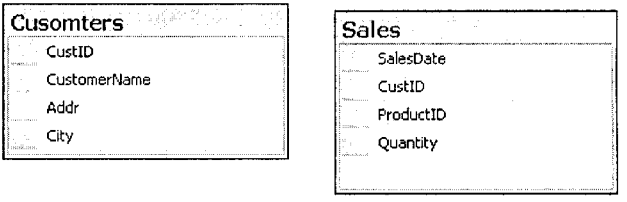


Рисунок 5.7 – Приклад обов'язкової назви таблиці

В обох таблицях бази даних є поле CustID. Якщо в запиті передбачено виконання дій над атрибутом "SELECT CustID", механізм SQL не зможе однозначно визначити, з якої саме таблиці слід вибрати значення. Тому для виконання запиту слід вказувати повну назву таблиць:

```
SELECT c.CustID
FROM Customers c INNER JOIN Sales ON c.CustID = Sales.CustID
```

В даному випадку використовується псевдонім лише для таблиці Customers, але, зазвичай, псевдоніми створюються для обох таблиць:

```
SELECT c.CustID
FROM Customers c INNER JOIN Sales s ON c.CustID = s.CustID
```

Оператор JOIN також підтримує об'єднання трьох або більше таблиць. Необхідно лише додати новий оператор JOIN для кожної таблиці:

```
SELECT Sales.*, Customers.*, Parts.*, Tax.*
FROM Sales
INNER JOIN Customers ON Sales.CustID = Customers.CustID
INNER JOIN Parts ON Parts.PartID = Sales.PartID
INNER JOIN Tax ON Tax.TaxID = Customers.TaxID
```

Об'єднання INNER JOIN можуть використовуватись сумісно із об'єднаннями OUTER JOIN в одному запиті. Останній запит із використанням псевдонімів має вигляд:

```
SELECT s.*, c.*, p.*, t.*
FROM Sales s
```

```
INNER JOIN Customers c ON s.CustID = c.CustID
INNER JOIN Parts p ON p.PartID = s.PartID
INNER JOIN Tax t ON t.TaxID = c.TaxID
```

Створимо нову базу даних, що складається із таблиць Customers і Sales:

```
CREATE TABLE [dbo].[Sales]
([SalesDate] [datetime] NULL,
[CustID] [int] NULL,
[ProductID] [int] NULL,
[Quantity] [int] NULL,
[CustName] [varchar](50) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL, [SalesAmt] [money] NULL) ON [PRIMARY]
CREATE TABLE [dbo].[Customers]
 ([CustID] [int] NULL,
 [CustomerName] [nchar](10) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL, [Addr] [nchar](10) COLLATE SQL
_Latin1_General_CP1_CI_AS NULL,
[City] [nchar](10) COLLATE SQL_Latin1_General_CP1_CI_AS NULL,
[TotalSales] [varchar](50) COLLATE SQL_Latin1_General_CP1_CI_AS
NULL) ON [PRIMARY]
```

Об'єднання також можна застосовувати в операторі UPDATE. Приклад нижче виконує об'єднання таблиць Sales та Customer по значеннях стовпців CustID та оновлює значення CustomerName в таблиці Sales.

```
UPDATE Sales
SET Sales.CustName = Customers.CustomerName
FROM Sales INNER JOIN Customers ON Sales.CustID = Customers.CustID
```

Інколи INNER JOIN описується в старому стилі - таблиці розділяються комою, умова ON замінюється на WHERE:

```
UPDATE Sales
SET Sales.CustName = Customers.CustomerName
FROM Sales, Customers
WHERE Sales.CustID = Customers.CustID
```

Для виконання функції підсумовування значень таблиці sales та запису агрегованих значень для кожного клієнта в поле TotalSales таблиці CUSTOMERS доцільно використати вкладений запит та об'єднання JOIN:

```
UPDATE Customers
SET TotalSales =
 (SELECT SUM(SalesAmt)
FROM Sales
WHERE Sales.CustID = Customer.CustID)
FROM Sales, Customers
UPDATE Customers
SET TotalSales = (
SELECT SUM(SalesAmt)
```

```
FROM Sales
WHERE Sales.CustID = Customers.CustID)
FROM Sales INNER JOIN Customers ON Sales.CustID = Customers.CustID
```

КОНТРОЛЬНІ ПИТАННЯ

1. Чому краще розбивати великі таблиці на декілька взаємопов'язаних таблиць меншого розміру?
2. Що таке нормалізація?
3. Які атрибути є неключовими?
4. Назвіть відомі Вам нормальні форми та їх властивості.
5. Як виконується нормалізація відношень?
6. Наведіть приклад запиту із використанням умови INNER JOIN, поясніть особливості його виконання.
7. Наведіть приклади запитів із використанням умов LEFT OUTER JOIN, RIGHT OUTER JOIN та FULL OUTER JOIN. Поясніть особливості їх виконання.
8. В чому різниця між LEFT OUTER JOIN та RIGHT OUTER JOIN? Наведіть приклади відповідних запитів.
9. Для чого використовуються псевдоніми?
10. Наведіть приклад запиту, в якому буде виконуватись об'єднання трьох або більше таблиць.
11. Наведіть приклад запиту із оператором UPDATE та використанням об'єднання таблиць.
12. Наведіть приклад запиту із застосуванням групових функцій та використанням об'єднання таблиць.
13. Оцініть кількість записів, що буде повертатись запитом із використанням операторів JOIN?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Дейт К. Дж. Введение в системы баз данных / Дейт К. Дж. – [8-е изд.]. – М. : Изд-во "Вильямс", 2006. – 1328 с.
2. Дейт К. SQL и реляционная теория. Как грамотно писать код на SQL / Дейт К. – М. : Изд-во "Символ-Плюс", 2010. – 480 с.
3. Роберт Виейра Программирование баз данных Microsoft SQL Server 2008. Базовый курс / Роберт Виейра. – М. : Изд-во: "Диалектика-Вильямс", 2009. – 816 с.
4. Бен-Ган И. Microsoft SQL Server 2008. Основы T-SQL / Бен-Ган И. – Санкт-Петербург : Изд-во "Русская редакция", 2009. – 432 с.
5. Тернстрем Т. Microsoft SQL Server 2008. Разработка баз данных. Учебный курс Microsoft, экзамен 70-433 (+ CD) / Тернстрем. Т., Вебер Э., Хотек М. – М. : Изд-во "Русская Редакция", 2010. – 496 с.

6. Петкович Д. Microsoft SQL Server 2008. Руководство для начинающих / Петкович Д. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 752 с.
7. Грофф Джеймс Р. SQL: полный справочник / Грофф Джеймс Р., Вайнберг Пол Н., Оппель Эндрю Дж. – Москва : Изд-во "Диалектика-Вильямс", 2010. – 960 с.
8. Дьюсон Р. SQL Server 2008 для начинающих разработчиков / Дьюсон Р. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 704 с.

ЛАБОРАТОРНА РОБОТА № 6
SQL MANAGEMENT STUDIO:
ВИБИРАННЯ ДАНИХ ЗА ДОПОМОГОЮ SQL-ЗАПИТІВ.
ВКЛАДЕНІ ТА КОРЕЛЬОВАНІ ЗАПИТИ

Мета: отримати практичні навички створення вкладених та корельованих SQL-запитів.

Порядок виконання роботи

1. Завантажити базу даних, яка використовувалась в лабораторній роботі № 5.
2. Розробити не менш як 2 вкладених запити для вибирання даних та виконати їх. Пояснити отримані результати.
3. Розробити не менш як 2 корельованих запити для вибирання даних та виконати їх. Пояснити отримані результати.
4. Розробити вкладений запит для вибирання даних з використанням оператора EXISTS та виконати його. Пояснити отримані результати.
5. Розробити вкладений запит для вибирання даних з використанням оператора NOT EXISTS та виконати його. Пояснити отримані результати.
6. Дати відповіді на контрольні питання.
7. Оформити звіт.

Теоретичні відомості

В деяких випадках неможливо за допомогою лише одного запиту виконати обробку даних. Тоді застосовуються вкладені запити, що дозволяють за одним запитом отримати інформацію, яку в іншому разі довелося б здобувати більшою кількістю запитів.

Нехай є база даних з відношеннями такої структури.

Таблиця 6.1 – Структура таблиці Orders (замовлення покупців)

Назва поля	Тип поля	Опис
customerID	Varchar	Код покупця
saleitemID	Varchar	Код товару
orderDate	Date	Дата заказа
saleitemsCount	Numeric	Кількість замовленого товару
soldDate	Date	Дата продажу
soldCount	Numeric	Кількість проданого товару
price	Numeric	Ціна одиниці товару

Таблиця 6.2 – Структура таблиці Customers (інформація про покупців)

Назва поля	Тип поля	Опис
customerID	Varchar	Код покупця
Name	Varchar	Найменування покупця
homeIndex	Varchar	Поштовий індекс
City	Varchar	Місто проживання
Street	Varchar	Вулиця
Phone	Character	Телефон покупця
Credit	Numeric	Кредит

Таблиця 6.3 – Структура таблиці Saleitems (інформація про товари)

Назва поля	Тип поля	Опис
SaleitemID	Varchar	Код товару
SaleitemName	Varchar	Назва товару
Price	Numeric	Ціна товару

Для визначення покупців, які зробили замовлення максимальної вартості без використання вкладених запитів, слід виконати два запити:

- визначити максимальну вартість зроблених замовлень
  SELECT Max(Price*SaleitemsCount) FROM Orders;
- визначити покупців, в яких значення добутку
  Price*SaleitemsCount дорівнює максимальній вартості зроблених замовлень.

Вирішити поставлену задачу з користанням вкладеного запиту можна так:

```
SELECT CustomerID, price*SaleitemsCount FROM Orders
    WHERE price*SaleitemsCount =
    (SELECT Max(Price*SaleitemsCount) FROM Orders)
```

У розглянутому прикладі дані з таблиці вибиралися на підставі результату додаткового запиту до цієї самої таблиці, причому результат цей являв собою таблицю з одним стовпцем і одним рядком, тобто одне число.

На практиці нерідко результатом додаткового запиту є таблиця з кількома рядками, значення полів якої використовуються як фільтри для вибору інформації з тієї самої таблиці, що була джерелом даних для додаткового запиту. Такі вибирання називають корельованими. Для їх виконання застосовують псевдоніми таблиць, які записують у запиті безпосередньо за відповідним ім'ям.

Розглянемо такий приклад. Нехай потрібно вибрати код, величину кредиту та місце проживання для кожного покупця, в якого кредит більший або дорівнює середньому для покупців з його міста.

Припустимо, що крім таблиці Customers (покупець) ми маємо її дублікат з ім'ям Cust2. У цьому разі потрібну інформацію можна було б дістати з допомогою вкладеного запиту до цієї таблиці, який обчислював би середнє значення кредиту покупців відповідного міста:

```
SELECT
Customers.CustomerID, Customers.credit, Customers.city
      FROM Customers WHERE Customers.credit >=
      (SELECT AVG (Cust2.credit) FROM Cust2
            WHERE Cust2.CITY = Customers.city)
```

Оскільки таблиця cust2 повністю ідентична таблиці Customers, то замість того, щоб створювати її, можна дати ще одне ім'я (псевдонім) таблиці Customers і звертатися до неї, як до іншої таблиці:

```
SELECT
      Customers.CustomerID, Customers.credit, Customers.city
            FROM Customers WHERE Customers.credit >=
            (SELECT AVG (Cust2.credit) FROM Customers Cust2
                  WHERE Cust2.city = Customers.city)
```

У цьому запиті можна, очевидно, використати і два псевдоніми таблиці Customers:

```
SELECT Cust1.Customerid, Cust1.kredit, Cust1.city
   FROM Customers Cust1WHERE Cust1.kredit>=
         (SELECT AVG (Cust2.kredit)FROM Customers Cust2
            WHERE Cust2.City = Cust1.city)
```

В підзапиті використовуються назви двох таблиць, але у фразі FROM вказано лише одну. Це пояснюється тим, що друга таблиця вказана в першій фразі FROM.

Розглянемо запит, який вже конструювався за допомогою групування і використання функції COUNT у фразі HAVING: визначити

коди товарів, які за період з 13.03.2011 р. по 20.03.2011 р. мали попит більш ніж у одного покупця. Отримати таку інформацію можна й за допомогою корельованих запитів. Наприклад:

```
SELECT DISTINCT Orders.SaleitemID FROM Orders
WHERE (Orders.soldDate Between '13.03.2011' And '20.03.2011') AND
(SELECT Count(*) FROM Orders ordr WHERE Orders.SaleitemID =
Orders.SaleitemID) > 1
```

Ще один варіант конструювання цього запиту - з використанням таблиці Saleitems:

```
SELECT Saleitems.SaleitemID FROM Saleitems
    WHERE (SELECT COUNT (Orders.CustomerID) FROM Orders
        WHERE Orders.SaleitemID=Saleitems.SaleitemID) > 1
```

При використанні вкладених запитів часто застосовується квантор існування. Це поняття, запозичене з формальної логіки, подається в мові SQL оператором EXISTS (існує). Він застосовується у виразах виду EXISTS (SELECT *FROM ...).

Вираз вважається істинним тоді і тільки тоді, коли результат обчислення підзапиту, поданого за допомогою SELECT * FROM ..., є непорожньою множиною. Іншими словами, вираз є істинним тоді і тільки тоді, коли існує будь-який рядок у вихідній таблиці підзапиту, яка задовольняє умову WHERE цього підзапиту.

Як приклад виберемо прізвища покупців, які замовили комп'ютер Core2Duo-3200 або Core2Duo-3600:

```
SELECT p.Name, FROM Customers p
    WHERE EXISTS (SELECT * FROM Orders z WHERE
                z.CustomerID = p.CustomerID AND
                (z.SaleitemID = 'Core2Duo-3200' OR
                z.SaleitemID='Core2Duo-3600'))
```

У даному прикладі послідовно розглядається кожне значення поля Name і перевіряється істинність умови існування. Припустимо, що перше значення поля Name є «Іваїов С. І.», а відповідне значення поля CustomerID – 751. Якщо у таблиці Orders існує хоча б один запис, в якому значення поля CustomerID дорівнює 751, а в полі SaleitemID записано Core2Duo-3200 або Core2Duo-3600, то тоді (і тільки тоді) рядок із значенням «Іванов С. І.» належатиме вихідній таблиці звіту. Аналогічно перевіряються й інші значення поля Name.

Оператор EXISTS реалізує одну з найважливіших можливостей мови SQL. Він може використовуватися разом з оператором NOT. Як приклад виберемо прізвища покупців, які не замовляли комп'ютер Core2Duo-3200 або Core2Duo-3600:

```
SELECT p.Name FROM Customers p
    WHERE NOT EXISTS
            SELECT * FROM Orders z
```

```
WHERE z.CustomerID = p.CustomerID AND
    ( z.SaleitemID = 'Core2Duo-3200' OR
    z.SaleitemID = 'Core2Duo-3600'))
```

Слід звернути увагу, що хоча в підзапиті використовуються назви двох таблиць, у фразі FROM вказано лише одну. Крім того, підзапит, що входить до виразу EXISTS, зовсім не обов'язково має використовувати команду SELECT у вигляді SELECT *, проте на практиці зазначений підзапит записується саме в такому вигляді.

КОНТРОЛЬНІ ПИТАННЯ

1. Для чого використовуються вкладені запити?
2. Наведіть приклад вибирання, яке неможливо виконати без використання вкладених запитів.
3. Які запити називаються корельованими?
4. Наведіть приклад корельованого запиту.
5. Що таке квантор існування?
6. Поясніть зміст операторів EXISTS та NOT EXISTS та наведіть приклади запиту з їх використанням.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Дейт К. Дж. Введение в системы баз данных / Дейт К. Дж. – [8-е изд.]. – М. : Изд-во "Вильямс", 2006. – 1328 с.
2. Дейт К. SQL и реляционная теория. Как грамотно писать код на SQL / Дейт К. – М. : Изд-во "Символ-Плюс", 2010. – 480 с.
3. Роберт Виейра Программирование баз данных Microsoft SQL Server 2008. Базовый курс / Роберт Виейра. – М. : Изд-во: "Диалектика-Вильямс", 2009. – 816 с.
4. Бен-Ган И. Microsoft SQL Server 2008. Основы T-SQL / Бен-Ган И. – Санкт-Петербург : Изд-во "Русская редакция", 2009. – 432 с.
5. Тернстрем Т. Microsoft SQL Server 2008. Разработка баз данных. Учебный курс Microsoft, экзамен 70-433 (+ CD) / Тернстрем. Т., Вебер Э., Хотек М. – М. : Изд-во "Русская Редакция", 2010. – 496 с.
6. Петкович Д. Microsoft SQL Server 2008. Руководство для начинающих / Петкович Д. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 752 с.
7. Грофф Джеймс Р. SQL: полный справочник / Грофф Джеймс Р., Вайнберг Пол Н., Оппель Эндрю Дж. – Москва : Изд-во "Диалектика-Вильямс", 2010. – 960 с.
8. Дьюсон Р. SQL Server 2008 для начинающих разработчиков / Дьюсон Р. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 704 с.

ЛАБОРАТОРНА РОБОТА № 7
SQL MANAGEMENT STUDIO:
ВИБИРАННЯ ДАНИХ ЗА ДОПОМОГОЮ SQL-ЗАПИТІВ.
ОБ'ЄДНАННЯ РЕЗУЛЬТАТІВ КІЛЬКОХ ВИБІРОК

Мета: отримати практичні навички створення складних запитів із застосуванням об'єднання результатів вибирання кількох простих запитів.

Порядок виконання роботи

1. Завантажити базу даних, яка використовувалась в лабораторній роботі № 6.
2. Розробити не менш як 2 запити для вибирання даних із застосуванням виразу UNION та виконати їх. Пояснити отримані результати.
3. Розробити не менш як 2 запити для вибирання даних із застосуванням виразу UNION ALL та виконати їх. Пояснити отримані результати.
4. Розробити запит для вибирання даних з однієї таблиці з використанням виразу UNION та виконати його. Пояснити отримані результати.
5. Розробити не менш як 2 запити для вибирання даних із застосуванням виразу UNION та операторів JOIN (для вибирання даних із кількох пов'язаних таблиць) і виконати їх. Пояснити отримані результати.
6. Дати відповіді на контрольні питання.
7. Оформити звіт.

Теоретичні відомості

Оператор UNION використовується як операція об'єднання записів в одну таблицю. Критерій відбору записів для об'єднання визначається виразом WHERE. Таблиці-операнди повинні мати однакову арність та сумісні типи даних у відповідних стовпцях.

Синтаксис

Оператор вказується між запитами. У спрощеному вигляді це виглядає так:

```
<запит1>
        UNION [ALL]
<запит2>
        UNION [ALL]
<запит3>
   .....
```

За замовчуванням будь-які дублюючі записи автоматично ігноруються, якщо не використано вираз UNION ALL.

Необхідно відзначити, що UNION ALL сам по собі не гарантує порядок рядків. Рядки з другого запиту можуть виявитися на початку, в кінці або взагалі змішатися з рядками з першого запиту. У випадках, коли потрібен певний порядок, необхідно використовувати оператор ORDER BY.

У Microsoft SQL Server стовпці з типом даних XML повинні бути еквівалентними. Всі стовпці повинні або мати тип, визначений у XML-схемі, або бути нетипізованими. Типізовані стовпці повинні ставитися до однієї і тієї ж колекції XML-схем.

Ще одне обмеження на сумісність – це заборона порожніх значень (NULL) у будь-якому стовпці об'єднання. Ці значення необхідно заборонити і для всіх відповідних стовпців в інших запитах об'єднання, оскільки порожні значення (NULL) заборонені з обмеженням NOT NULL. Крім того, не можна використовувати UNION ALL у підзапитах, а також не можна використовувати агрегатні функції в реченні SELECT-запиту в об'єднанні.

Розглянемо приклади використання виразу UNION.

Використання UNION при вибиранні з двох таблиць

Нехай дано дві таблиці:

Sales-2008

person	Amount
Петро	1000
Павло	2000
Василь	5000

Sales-2010

person	Amount
Петро	2000
Павло	2000
Семен	35000

При виконанні запиту

```
SELECT * FROM sales-2008
UNION
SELECT * FROM sales-2010;
```

отримаємо відношення, порядок рядків в якому може довільно змінюватися, оскільки ключовий вираз ORDER BY не був використаний:

person	Amount
Петро	1000
Павло	2000
Василь	5000
Петро	2000
Семен	35000

У відношенні-результаті наявні два рядки з person= "Петро", оскільки вони відрізняються значеннями Amount, проте один рядок з person= "Павло", оскільки значення Amount в них ідентичні.

Використання UNION ALL при вибиранні з двох таблиць

Використання UNION ALL дає інший результат, оскільки дублікати не ігноруються. Отже, виконання запиту:

```
SELECT * FROM Sales-2008
        UNION ALL
SELECT * FROM Sales-2010;
```

дасть результат:

person	Amount
Петро	1000
Петро	2000
Павло	2000
Павло	2000
Василь	5000
Семен	35000

Використання UNION при вибиранні з однієї таблиці

Аналогічним чином можна поєднувати два різних запити з однієї таблиці (хоча замість цього, як правило, необхідні параметри комбінують в одному запиті за допомогою ключових слів AND і OR в умові WHERE):

```
SELECT person, Amount FROM Sales-2008
            WHERE Amount=1000
UNION
SELECT person, Amount FROM Sales-2010
            WHERE person LIKE "Василь";
```

В результаті отримаємо:

person	Amount
Петро	1000
Василь	5000

Використання UNION як зовнішнє об'єднання

За допомогою UNION можна створювати також повні зовнішні об'єднання (іноді використовується у разі відсутності вбудованої прямої підтримки зовнішніх об'єднань):

```
SELECT * FROM employee LEFT JOIN department ON
    employee.DepartmentID=department.DepartmentID
SELECT * FROM employee RIGHT JOIN department ON
    employee.DepartmentID=department.DepartmentID
```

КОНТРОЛЬНІ ПИТАННЯ

1. Для чого використовується вираз UNION?
2. Наведіть приклад вибирання, яку неможливо отримати без використання виразу UNION.
3. В якому порядку виводяться записи, отримані в результаті виконання запиту із виразом UNION?
4. Назвіть основні правила застосування виразу UNION.
5. Наведіть приклад запиту із використанням виразу UNION для об'єднання вибірок з двох таблиць та поясніть, яким чином формується результуюча вибірка.
6. Наведіть приклад запиту із використанням виразу UNION ALL для об'єднання вибірок з двох таблиць та поясніть, яким чином формується результуюча вибірка.
7. Наведіть приклад запиту із використанням виразу UNION для об'єднання вибірок з однієї таблиці.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Дейт К. Дж. Введение в системы баз данных / Дейт К. Дж. – [8-е изд.]. – М. : Изд-во "Вильямс", 2006. – 1328 с.
2. Дейт К. SQL и реляционная теория. Как грамотно писать код на SQL / Дейт К. – М. : Изд-во "Символ-Плюс", 2010. – 480 с.
3. Роберт Виейра Программирование баз данных Microsoft SQL Server 2008. Базовый курс / Роберт Виейра. – М. : Изд-во: "Диалектика-Вильямс", 2009. – 816 с.
4. Бен-Ган И. Microsoft SQL Server 2008. Основы T-SQL / Бен-Ган И. – Санкт-Петербург : Изд-во "Русская редакция", 2009. – 432 с.
5. Тернстрем Т. Microsoft SQL Server 2008. Разработка баз данных. Учебный курс Microsoft, экзамен 70-433 (+ CD) / Тернстрем. Т., Вебер Э., Хотек М. – М. : Изд-во "Русская Редакция", 2010. – 496 с.
6. Петкович Д. Microsoft SQL Server 2008. Руководство для начинающих / Петкович Д. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 752 с.
7. Гробф Джеймс Р. SQL: полный справочник / Гробф Джеймс Р., Вайнберг Пол Н., Оппель Эндрю Дж. – Москва : Изд-во "Диалектика-Вильямс", 2010. – 960 с.
8. Дьюсон Р. SQL Server 2008 для начинающих разработчиков / Дьюсон Р. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 704 с.

ЛАБОРАТОРНА РОБОТА № 8
СТВОРЕННЯ ІНДЕКСІВ У СЕРЕДОВИЩІ SQL MANAGEMENT STUDIO

Мета: отримати практичні навички створення індексів для підвищення швидкості виконання запитів.

Порядок виконання роботи

1. Завантажити базу даних, яка використовувалась в лабораторній роботі № 6.
2. Визначити стовпці для індексації. Пояснити вибір (навести список запитів, швидкодія яких зросте після введення індексів).
3. Створити індекси для 2 стовпців у середовищі SQL Management Studio.
4. Створити SQL-запити для створення решти індексів, пояснити структуру одного із них.
5. Дати відповіді на контрольні питання.
6. Оформити звіт.

Теоретичні відомості

Індекс – це впорядкований вказівник на записи в таблиці.

Вказівник означає, що індекс містить значення одного або декількох полів в таблиці і адреси розміщення цих даних. Іншими словами, індекс складається з пар значень "значення поля – фізичне розміщення цього поля". Таким чином, за значенням поля (або полів), що входять до індексу, за допомогою індексу можна швидко знайти запис, що містить це значення. Впорядкований – означає, що значення полів, які зберігаються в індексі, впорядковані.

Індекси сприяють прискоренню пошуку запису за його індексованим полем (індексоване – те, що входить до індексу).

Індекс не є частиною таблиці – це окремий об'єкт, пов'язаний з таблицею і іншими об'єктами бази даних. Індекси використовуються в трьох основних випадках:

1. Прискорення виконання запитів. Індекси створюються для полів, які використовуються в умовах пошуку SQL-запитів;

2. Забезпечення унікальності значень в полях. Обмеження первинного ключа потребує, щоб в усій таблиці не знайшлося двох однакових значень полів, що входять до первинного ключа. Щоб виконати цю умову, необхідно при кожній вставці нового запису здійснювати пошук значення атрибута, яке буде вставлене;

3. Забезпечення цілісності посилань. Обмеження зовнішніх ключів FOREIGN KEY використовується для перевірки того, щоб значення, які вставляються до таблиці, обов'язково існували в іншій таблиці. При створенні зовнішнього ключа автоматично створюється індекс, який використовується як для прискорення запитів, що використовують

з'єднання таблиць, так і для перевірки умов зовнішнього ключа.

Спрощений формат команди, що створює індекси, має вигляд:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]
INDEX index_name
ON <object> ( column [ ASC | DESC ] [ ,...n ] ) [options]
```

Повний формат команди можна знайти в [1]. Мінімальним виразом, що створює індекс, є:

```
CREATE INDEX My_index ON Table_example(ID)
```

В цьому прикладі створюється індекс з іменем My_index для таблиці Table_example, причому індексованим полем є поле ID. Індекс є зростаючим, а також не унікальним. Як видно з опису синтаксису, індекс може містити не одне, а декілька полів.

Для того, щоб індекс дозволив заносити до таблиці лише унікальні значення, використовують оператор UNIQUE, який служить основою для реалізації унікальних ключів (UNIQUE KEY).

Для побудови індексів SQL Server використовує структуру B-дерева (B-tree, Balanced-tree), що складається з кореня, проміжних вузлів та кінцевих вузлів (листків). Проіндексованим може бути як окремий стовпець, так і сукупність вибраних стовпців.

За своєю структурою індекси поділяються на такі:

- кластерний індекс, що зберігає сторінки даних таблиці на рівні листків B-дерева, при цьому дані фізично впорядковані за ключем. Для кожної таблиці можна визначити лише один кластерний індекс. При створенні такого індексу відбувається фізичне сортування даних відповідно до індексу, а також перебудова всіх некластерних індексів. Як правило, кластерні індекси створюють для первинних ключів. Оптимальним варіантом вважають такий, коли індексовані значення для кластерного ключа унікальні. Якщо значення не унікальні, тоді SQL Server створює додаткові ключі сортування для рядків, що мають дублікати для основних ключів сортування;
- некластерний індекс, що на рівні листків B-дерева містить вказівник на рядок даних, який відповідає ключу в індексі. Якщо таблиця вже має кластерний індекс, тоді вказівник вказує на його ключ, а не на дані. Якщо ж кластерний індекс відсутній, тоді вказівник вказує на рядок даних. При створенні некластерного індексу SQL Server створює необхідні сторінки індексу, але не змінює фізичного розташування табличних даних, не видаляє інші індекси таблиці. Кожна таблиця може мати до 249 некластерних індексів;

Вибір стовпців для індексування

Застосування індексів підвищує швидкість пошуку необхідної інформації, але вимагає додаткових витрат оперативної пам'яті, зовнішньої пам'яті, машинного часу. При довільних змінах в індексованих стовпцях змінюється й індекс. У табл. 8.1 наводяться рекомендації щодо вибору стовпців для індексації, а у табл. 8.2 – щодо використання кластерних чи некластерних індексів.

Таблиця 8.1 – Рекомендації щодо створення індексів

Рекомендовано індексувати	Не рекомендовано індексувати
Таблиці з великою кількістю рядків	Таблиці з невеликою кількістю рядків
Стовпці, що часто використовуються в запитах	Стовпці, що рідко використовуються в запитах
Стовпці, що зберігають широкий діапазон значень і мають високу ймовірність бути вибраними в типовому запиті	Стовпці, що зберігають широкий діапазон значень і мають низьку ймовірність бути вибраними в типовому запиті
Стовпці, що використовуються в реченнях GROUP BY	Стовпці, що мають великий розмір у байтах
Стовпці, що використовуються в реченнях ORDER BY	Таблиці, де дані часто змінюються, але рідко зчитуються
Стовпці, що використовуються у з'єднанні таблиць	

Таблиця 8.2 – Рекомендації щодо кластеризації індексів

Кластеризувати індекс для	Не кластеризувати індекс для
Первинних ключів, що часто використовуються при пошуку, наприклад, номера рахунків	Первинних ключів, що зберігають послідовні значення ідентифікаторів, наприклад, ідентифікаційних стовпців
Запитів, що повертають численні результуючі набори	Запитів, що повертають невеликі результуючі набори
Стовпців, що використовуються в численних запитах	Зовнішніх ключів
Стовпців із високою селективністю	
Стовпців, що використовуються в реченнях GROUP BY чи ORDER BY	
Стовпці, що використовуються у з'єднанні таблиць	

Створення вторинного ключа або індексу таблиці (UNIQUE KEY/ INDEX)

Під вторинним (можливим) ключем розуміють обмеження UNIQUE, яке забезпечує унікальність даних. Таке обмеження відповідає обмеженню первинного ключа, оскільки потребує наявності унікальних значень у вказаному в ньому стовпці (чи комбінації стовпців) таблиці. При доданні рядка з дублюючим значенням для стовпця, для якого встановлено обмеження UNIQUE, SQL Server автоматично згенерує помилку та відхилить спробу введення такого рядка. При створенні вторинного ключа, аналогічно як і для первинного ключа, SQL Server автоматично створює унікальний некластерний індекс.

Створення вручну окремого унікального індексу теж забезпечує унікальність даних для певного стовпця (стовпців) таблиці. Для забезпечення унікальності даних немає різниці між створенням обмеження UNIQUE та створенням унікального індексу. Ця відмінність полягає в тому, що унікальний ключ призначений для забезпечення обмежень на дані, а індекси призначені для прискорення доступу до даних.

Основною суттєвою відмінністю унікального ключа від унікального індексу є те, що на нього можуть посилатися зовнішні ключі (FOREIGN KEY) для забезпечення цілісності даних. Особливістю ж індексів є те, що вони можуть бути як унікальними, так і не унікальними.

Для створення унікального ключа чи індексу необхідно (рис. 8.1):

1. Для певної таблиці увійти в режим конструктора таблиці, для чого в панелі Object Explorer (оглядач об'єктів) для вибраної таблиці бази даних в контекстному меню вузла цієї таблиці вибрати команду Design (проект);
2. В панелі інструментів обрати Manage Indexes and Keys (керування індексами та ключами);
3. У діалоговому вікні Indexes/Keys (індекси та ключі), у лівій частині вікна (рис. 8.2), обрати Add (Добавить);
4. Для новоствореного ключа/індексу у полі Columns (стовпці) задати стовпець (стовпці) та їхній порядок сортування, для чого необхідно обрати "...", що розташовано справа від поля (рис. 8.1). В результаті відобразиться діалогове вікно Index Columns (у стовпці індексу). У лівій частині вікна (рис. 8.3) за допомогою списків поля Column Name (ім'я стовпця), що випадають, слід задати кількість, вид та порядок входження стовпців у ключ/індекс. Для кожного стовпця справа від нього в полі Sort Order (порядок сортування) можна задати порядок сортування як Ascending (за зростанням), або як Descending (за спаданням);

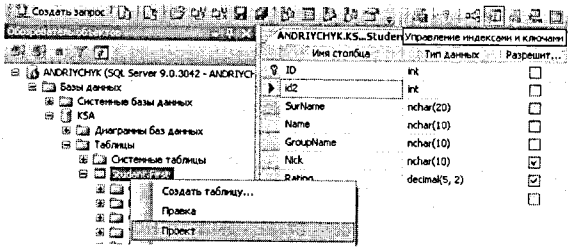


Рисунок 8.1 – Створення унікального ключа чи індексу таблиці

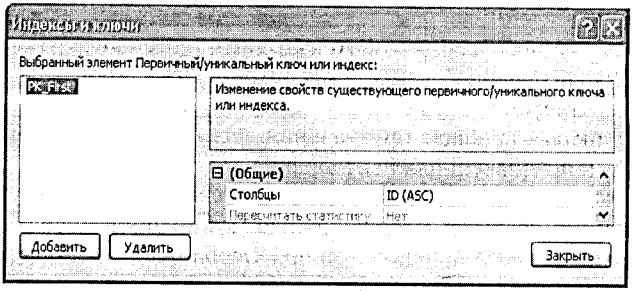


Рисунок 8.2 – Вибір параметрів унікального ключа чи індексу таблиці

5. У полі Type (Тип) вибрати (рис. 8.4) UNIQUE KEY (унікальний ключ) чи Index (Индекс) необхідно створити;
6. У полі Is Unique (Является уникальным) визначити буде індекс унікальним, чи ні (лише для індексу);
7. У полі (Name) (Имя) скоригувати назву ключа/індексу в зрозумілий контекст, наприклад, *IX_First*;
8. У полі Ignore Duplicate Keys (Игнорировать повторяющиеся ключи) задати режим генерації повідомлення помилки, у випадку додання повторюваного значення ключа. Параметр цього поля визначає дії сервера при операціях додання даних, що намагаються додати декілька рядків, у тому числі й з повторюваними значеннями. При значенні Yes (Да) рядки, що не вносять у таблицю повторюваних значень, будуть додані, дублюючі рядки будуть відалені, а транзакція загалом закінчиться успішно; при значенні No (Нет) виводиться повідомлення про помилку та виконується відкат усього пакета даних (лише для унікального індексу).

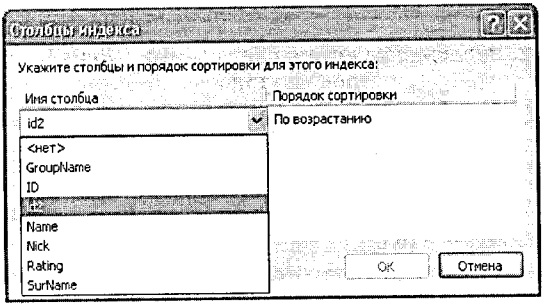


Рисунок 8.3 – Визначення параметрів унікального ключа чи індексу таблиці

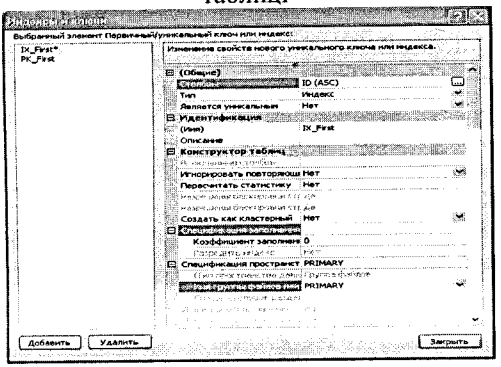


Рисунок 8.4 – Вибирання первинного елемента унікального ключа

9. У полі Create As Clustered (Создать как кластерный), якщо ще не створено для таблиці кластерний індекс, можна встановити цей ключ/індекс як кластерний;
10. У полі Fill Factor (Коэффициент заполнения) за необхідності можна задати значення параметра заповнення;
11. У полі Filegroup or Partition Scheme (Имя группы файлов или схемы разделов) вибрати групу файлів, у якій буде зберігатися ключ/індекс;
12. Закрити діалогове вікно та зберегти зміни в таблиці.

КОНТРОЛЬНІ ПИТАННЯ

1. Для чого використовуються індекси?
2. Наведіть приклад запиту для створення індексу.
3. Яка різниця між кластерним та некластерним індексом?
4. Які Ви знаєте рекомендації щодо створення індексів?
5. Що відбудеться при доданні рядка з дублюючим значенням для стовпця, для якого встановлено обмеження UNIQUE?
6. Які є відмінності між унікальним ключем та унікальним індексом?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Дейт К. Дж. Введение в системы баз данных / Дейт К. Дж. – [8-е изд.]. – М. : Изд-во "Вильямс", 2006. – 1328 с.
2. Дейт К. SQL и реляционная теория. Как грамотно писать код на SQL / Дейт К. – М. : Изд-во "Символ-Плюс", 2010. – 480 с.
3. Роберт Виейра Программирование баз данных Microsoft SQL Server 2008. Базовый курс / Роберт Виейра. – М. : Изд-во: "Диалектика-Вильямс", 2009. – 816 с.
4. Бен-Ган И. Microsoft SQL Server 2008. Основы T-SQL / Бен-Ган И. – Санкт-Петербург : Изд-во "Русская редакция", 2009. – 432 с.
5. Тернстрем Т. Microsoft SQL Server 2008. Разработка баз данных. Учебный курс Microsoft, экзамен 70–433 (+ CD) / Тернстрем. Т., Вебер Э., Хотек М. – М. : Изд-во "Русская Редакция", 2010. – 496 с.
6. Петкович Д. Microsoft SQL Server 2008. Руководство для начинающих / Петкович Д. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 752 с.
7. Гробф Джеймс Р. SQL: полный справочник / Гробф Джеймс Р., Вайнберг Пол Н., Оппель Эндрю Дж. – Москва : Изд-во "Диалектика-Вильямс", 2010. – 960 с.
8. Дьюсон Р. SQL Server 2008 для начинающих разработчиков / Дьюсон Р. – Санкт-Петербург : Изд-во "БХВ-Петербург", 2009. – 704 с.

Навчальне видання

Савчук Тамара Олександрівна

ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ
Частина 2

Лабораторний практикум

Редактор Т. Старічек

Оригінал-макет підготовлено Т. Савчук

Підписано до друку 04.05.2016 р.
Формат 29,7×42¼. Папір офсетний.
Гарнітура Times New Roman.
Друк різографічний. Ум. друк. арк. 5,4.
Наклад 75 пр. Зам. № 2016-069.

Вінницький національний технічний університет,
навчально-методичний відділ ВНТУ.
21021, м. Вінниця, Хмельницьке шосе, 95,
ВНТУ, к. 2201.
Тел. (0432) 59-87-36.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.

Віддруковано у Вінницькому національному технічному університеті
в комп'ютерному інформаційно-видавничому центрі
21021, м. Вінниця, Хмельницьке шосе, 95,
ВНТУ, ГНК, к. 114.
Тел. (0432) 59-87-38.
publish.vntu.edu.ua; email: kivc.vntu@gmail.com.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.