

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІНЖЕНЕРНО-ТЕХНІЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ОРГАНІЗАЦІЯ БАЗ ДАНИХ

Робота з базами даних.

Методичні вказівки і завдання

до лабораторних робіт

для студентів 2-го курсу інженерно-технічного факультету спеціальності

123 – «Комп'ютерна інженерія».

Ужгород – 2021

Зміст

Лабораторна робота № 1. Захист бази даних.	4
Лабораторна робота №2. Основні поняття мови <i>SQL</i> . Організація <i>SQL</i> -запитів.	9
Лабораторна робота № 3. Створення бази даних в СУБД <i>Mikrosoft SQL Server</i> .	21
Лабораторна робота № 4. Використання операторів маніпулювання даними в СУБД <i>Mikrosoft SQL Server</i> .	29
Лабораторна робота №5. Використання тригерів та переглядів в СУБД <i>Mikrosoft SQL Server</i> .	41
Лабораторна робота №6. Збережені процедури серверу.	59

Вступ

Лабораторні роботи є сполучною ланкою між лекційними заняттями і самостійною роботою студентів. В процесі виконання лабораторних робіт експериментально перевіряються ключові питання курсу «Організація баз даних», набуваються практичні навички проектування і розробки баз даних засобами сучасних систем управління базами даних, перевіряється рівень засвоєння основних положень предмету.

Метою методичних вказівок є надання допомоги студентам в отриманні практичних навичок роботи з системами керування базами даних.

У серії лабораторних робіт використовуються СУБД *Microsoft ACCESS*, СУБД *Microsoft SQL Server*, *Microsoft Visual Studio 2010*. Важливою складовою частиною робіт є освоєння *SQL* стандарту.

Під час виконання лабораторних робіт студенти отримають навички щодо:

- підходів до проектування баз даних;
- створення таблиць;
- організації взаємозв'язку між таблицями;
- створення запитів на отримання даних з таблиць;
- створення тригерів та переглядів;
- створення збережених процедур серверу.

Студент зобов'язаний до лабораторного заняття прочитати методичні вказівки до лабораторної роботи і спробувати виконати її самостійно. Під час лабораторного заняття студент показує викладачеві результати роботи, проводить консультації з питань, які виникли, та завершує роботу.

Захист роботи полягає в виконанні завдання до лабораторної роботи, відповіді на питання по темі лабораторної роботи і внесення деяких змін в базу даних, яка розроблялась, в присутності викладача.

Результати виконання робіт рекомендується зберігати в особистих папках, так як лабораторні роботи взаємопов'язані.

Лабораторна робота № 1

Тема роботи: Захист бази даних.

Мета роботи: навчитися організовувати захист даних в СУБД *Access*.

Основні теоретичні відомості.

Застосування паролів.

Захист за допомогою пароля представляє простий спосіб запобігання несанкціонованого доступу до даних. У разі його використання при кожній спробі відкрити файл бази даних (БД) з'являється діалогове вікно, в яке вимагається ввести пароль. Введення неправильного пароля приводить до припинення процедури відкриття файлу.

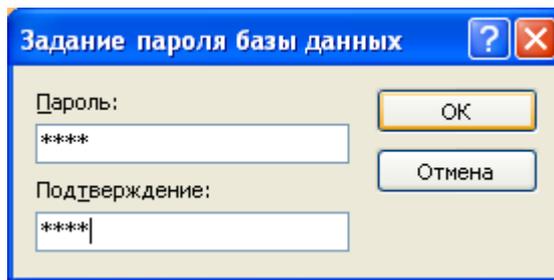
Цей спосіб достатньо надійний, так як *Access* шифрує пароль, що робить його недоступним при безпосередньому читанні файлу бази даних. Тому для забезпечення захисту БД, групою користувачів, що спільно використовується або існуючій на автономному комп'ютері, звичайно достатньо встановити пароль і виконати шифрування.

Для установки пароля слід:

1. Відкрити базу даних з монопольним доступом. Для цього виконати команду Файл→Відкрити (*File → Open*) меню *Access*, в діалоговому вікні Відкриття файлу бази даних (*Open*) виділити ім'я бази даних, клацнути по стрілці праворуч від кнопки Відкрити (*Open*) і в списку, що розкрився, вибрати пункт Монопольно (*Open Exclusive*).

2. Виконати команду меню Сервіс→Захист (*Tools→Security*). В підменю вибрати пункт Задати пароль бази даних (*Set Database Password*).

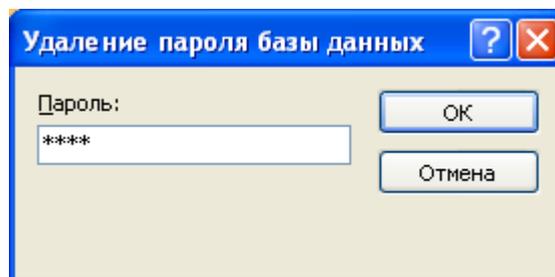
3. У вікні Визначення паролю бази даних (*Set Database Password*) задати пароль в полях Пароль (*Password*) і Підтвердження (*Verify*). Натиснути кнопку *OK*.



Введений пароль виглядає як послідовність зірочок, що забезпечує його приховування від інших користувачів у момент установки.

Одне і те ж поєднання символів, введене при різному стані регістра, буде сприйматися *Access*, як різні значення пароля. Тому при установці пароля слід звернути увагу на стан клавіші *<Caps Lock>*.

Щоб видалити існуючий пароль, слід відкрити базу даних з монопольним доступом, виконати команду меню Сервіс→Захист (*Tools→Security*). В підменю вибрати пункт Видалити пароль бази даних (*Unset Database Password*) — цей пункт стає видимим тільки в тому випадку, якщо пароль встановлений. Ввести пароль і натиснути кнопку *OK*.



Не слід використовувати захист бази даних за допомогою пароля, якщо передбачається виконувати її реплікацію, так як репліковані бази даних в цьому випадку не можуть бути синхронізовані. Реплікацією називається процес створення двох або більш копій (реплік) однієї бази даних. В репліках можлива синхронізація змін, що вносяться в дані однієї з копій.

Захист на рівні користувача.

Захист на рівні користувача є найбільш гнучким і поширеним способом захисту баз даних. Суть його полягає в наданні окремого користувачу або

групам користувачів привілею, який визначає можливість доступу до даних і об'єктів (таблицям, запитам, формам, звітам і макросам) бази даних. Цей привілей надається адміністратором БД або власником об'єкту.

Як правило, для спрощення організації доступу, привілеї привласнюються групам, а не окремим користувачам. Права на використання даних БД, призначені групі, автоматично одержує кожний користувач, що входить в неї. При появі нового користувача його додають в групу з відповідними привілеями. В *Access* визначено дві стандартні групи: *адміністратори (Admins)* і *користувачі (Users)*. При необхідності мати більш розгалужену структуру управління доступом існує можливість створення нових груп користувачів, привласнення групам різних наборів привілеїв на допуск і додавання нових користувачів у відповідні групи. Члени груп реєструються за допомогою облікових записів користувачів, що складаються з імені і особистого коду (*PID*), і облікових записів груп, що включають ім'я і код групи. Ця інформація зберігається у файлі робочої групи *Microsoft Access*.

Щоб скористатися базою даних, захищеною на рівні користувача, необхідно ввести ім'я і особистий код користувача при запуску *Microsoft Access*. СУБД аналізує файл робочої групи і визначає рівень доступу і об'єкти бази даних, до яких користувач допущений.

Організація захисту на рівні користувачів є складнішою задачею. Для її вирішення в *Microsoft Access* передбачений *Майстер захисту*, запуск якого здійснюється по команді меню Сервіс → Захист → Майстер (*Tools → Security → User-Level Security Wizard*). За допомогою Майстра захисту можна створити нові групи користувачів і визначити привілеї на роботу з базою даних і її об'єктами, а також встановити привілеї на доступ, які по замовчуванню присвоюються знову створюваним об'єктам.

Завдання для самостійної роботи:

1. Створити пароль (123456) для бази даних, згідно індивідуального варіанту.
2. Видалити пароль (123456) з своєї бази даних.
3. Оформити звіт.

Питання для самоконтролю:

1. Які існують способи захисту даних?
2. Вкажіть дії для задання паролю баз даних.
3. У чому полягає зміст захисту на рівні користувача?
4. Як організувати захист на рівні користувача?

Варіанти індивідуальних завдань:

Варіант 1. Спроекувати базу даних про студентів, для їхнього розподілу по місцях практики: прізвище, рік народження, група, факультет, середній бал, місце роботи, місто.

Варіант 2. Спроекувати базу даних про автомобілі: номер, рік випуску, марка, кольори, стан, прізвище власника, адреса.

Варіант 3. Спроекувати базу даних про квартири, призначені для продажу: район, поверх, площа, кількість кімнат, відомості про власника, ціна.

Варіант 4. Спроекувати базу даних про книги, які були куплені бібліотекою: назва, автор, рік видання, адреса автора, адреса видавництва, ціна, книготорговельна фірма.

Варіант 5. Спроекувати базу даних про співробітників, що мають комп'ютер: прізвище, номер кімнати, назва відділу, дані про комп'ютери.

Варіант 6. Спроекувати базу даних про замовлення, отриманих співробітниками фірми: прізвище, сума замовлення, найменування товару, назва фірми - клієнта, прізвище замовника.

Варіант 7. Спроекувати базу даних про оцінки, отриманих студентами на іспитах: прізвище, група, предмет, номер квитка, оцінка, викладач.

Варіант 8. Спроекувати базу даних про викладачів кафедри: прізвище, посада, ступінь, номер кімнати, курси, що читають.

Варіант 9. Спроекувати базу даних про авторів *web*-додатку та їхніх статей: ім'я, адреса, обліковий запис, пароль, тема, заголовок, текст статті, ілюстрації.

Варіант 10. Спроекувати базу даних про список розсилання й передплатників: тема й зміст листа, дата відправлення, імена й адреси передплатників, їхні облікові записи й паролі.

Варіант 11. Спроекувати базу даних «Мережа магазинів комп'ютерної техніки», яка містить інформацію про наступні об'єкти: магазини (назва, адреса), наявність товару, кількість, ціна, продавці (ПП, адреса, дата народження, номер паспорту, магазин, відділ).

Варіант 12. Спроекувати базу даних про працівників поліклініки: ПП, спеціальність, розклад прийому (день тижня, початок прийому, кінець прийому, кабінет, ділянка).

Варіант 13. Спроекувати базу даних «Картотека бібліотеки», яка містить наступну інформацію: картка книги, видавництво, місто, тема книги, дисципліна, список вибірки книги по дисципліні, позиція вибірки.

Варіант 14. Спроекувати базу даних для гуртожитку, яка містить наступну інформацію: корпус, кімнати, студенти, які там проживають, оплата, умови.

Варіант 15. Спроекувати базу даних про відвідування студентами занять. База даних містить наступну інформацію: кафедра, група, студент, дисципліна, викладач, заняття, відвідування заняття.

Лабораторна робота №2

Тема: Основні поняття мови *SQL*. Організація *SQL*-запитів.

Мета: вивчити реляційні можливості середовища СУБД *Access*, засвоїти основні навички роботи з конструктором візуальної побудови запитів *Query By Example (QBE)*, вміти створювати *SQL*-запити з простими та складними умовами відбору.

Основні теоретичні відомості

Запит – об'єкт бази даних, за допомогою якого можна виконати вибірку з таблиць на основі заданих критеріїв, модифікувати таблиці, виконати обчислення. Результатом виконання запиту є тимчасовий набір даних (НД), який називається *Recordset*. В *Access* запити діляться на *QBE*-запити (*Query By Example* – запит за зразком), параметри якого встановлюються за допомогою конструктора запитів, і *SQL*-запити (*Structured Query Language* – структурована мова запитів), при створенні яких використовуються оператори і функції мови *SQL*.

Таблична мова запитів *QBE* (Запити за зразком), разом з мовою *SQL*, використовується для створення різних запитів до реляційних баз даних (БД). Мова *QBE* є більш наглядною і простішою для розуміння в порівнянні з *SQL*, хоча і більш обмеженою в можливостях.

Для створення нового запиту необхідно перейти на закладку Запити і

натискувати кнопку Створити. В результаті з'явиться вікно з інструментами для створення запитів.

При виборі режиму конструктора або при натисненні кнопки Конструктор, з'являється вікно Додавання таблиці. В ньому необхідно вибрати таблицю або декілька таблиць, які будуть необхідні для побудови нового запиту. Їх додавання відбувається після натиснення кнопки Додати. Після додавання потрібних таблиць можна закрити це вікно.

Режим конструювання запитів має вигляд наступного вікна (Див. рис. 1):

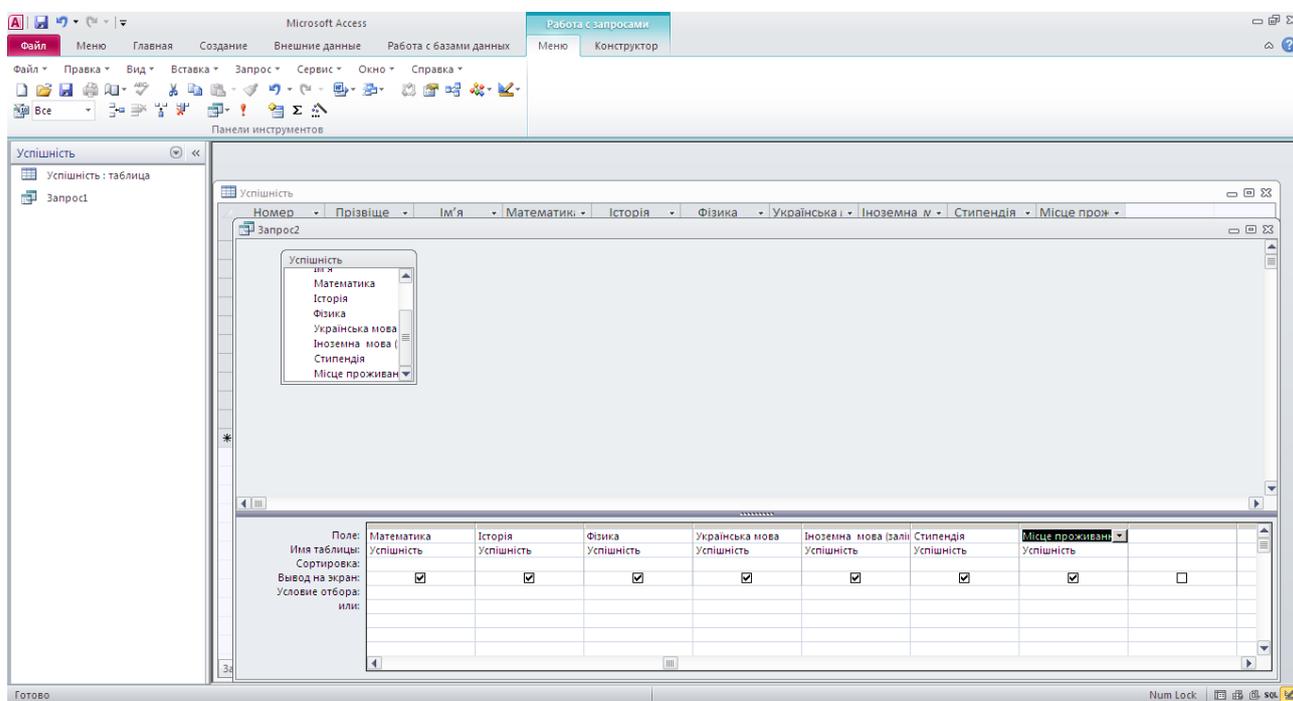


Рисунок 1 – Вікно створення запитів в режимі Конструктор

Вікно Запиту розбите горизонтально посередині. У верхній половині відображуються списки полів всіх вибраних таблиць, що беруть участь в запиті, а в нижній – власне специфікація запиту. Таблиці додаються в запит за допомогою кнопки панелі інструментів  (Додати таблицю) або за допомогою контекстного меню, викликаного для верхньої половини вікна. Запуск запиту на виконання здійснюється натисненням на кнопку панелі

інструментів  (Запуск).

У нижній частині вікна розташована сітка побудови запиту. Кожен стовпець відповідає полю, дані з якого включаються в запит. Сітка складається з наступних рядків:

Поле. Щоби вставити поле якоїсь таблиці в запит, його потрібно вибрати із списку полів випадного в цьому рядку. Символ «*» у списку полів позначає включення всіх полів з відповідної таблиці.

Ім'я таблиці. Вказується ім'я таблиці, поле якого було обране раніше

Групова операція. Використовується для підрахунку ряду обчислень. Якщо такий рядок відсутній в сітці, то вона додається при натисненні кнопки панелі інструментів .

Сортування. В цьому рядку визначається, по яких полях будуть відсортовані результати виконання запиту. При сортуванні по декількох полях *Access* сортує дані в порядку появи полів в сітці побудови запиту зліва направо. Положення стовпця поля можна змінити, виділивши його клацанням миші на заголовку і перетягнувши його на нове місце.

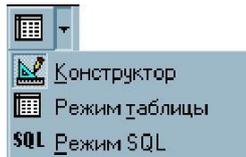
Вивід на екран. Встановлюється, якщо дані з поля відповідного стовпця мають бути включені в результат запиту.

Умова відбору. В цьому рядку вводиться критерій відбору даних. Найбільш розповсюджений запит на вибірку, який виконує відбір даних з однієї або кількох таблиць відповідно до заданих критеріїв.

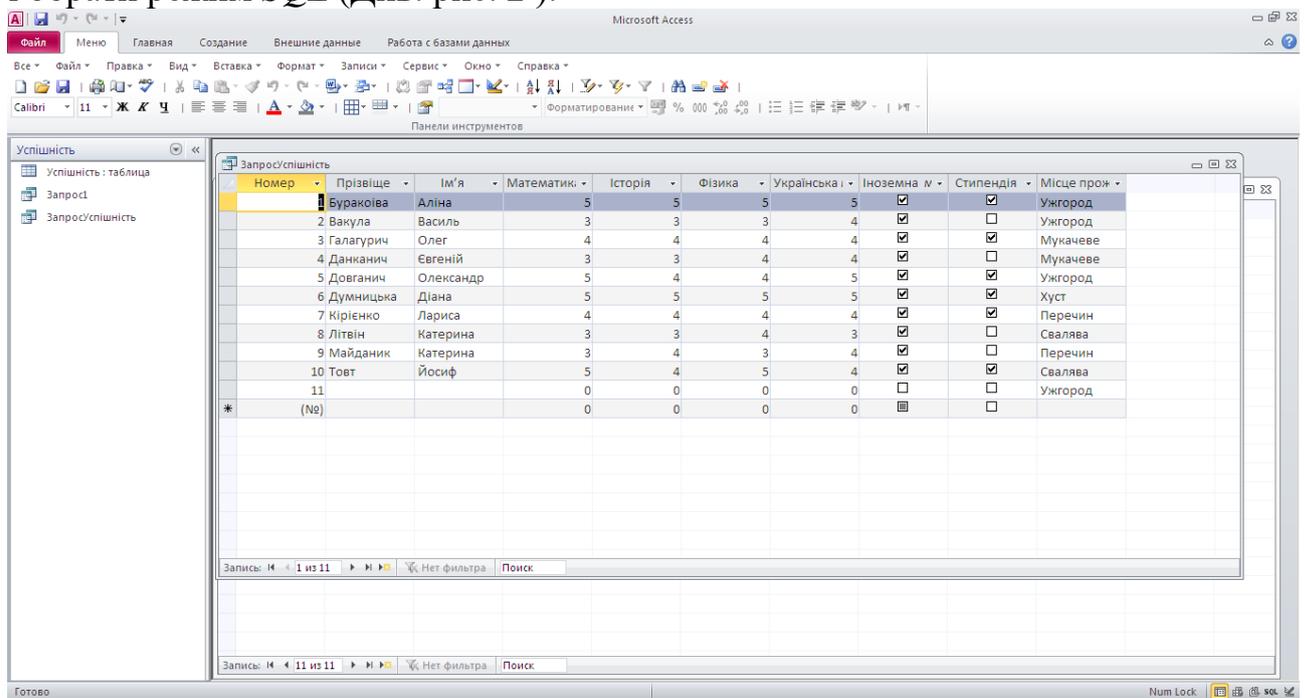
Мова *SQL (Structured Query Language)* використовується при створенні запитів, а також для оновлення і управління реляційними БД, у тому числі і БД *Microsoft Access*. Коли користувач створює запит в режимі конструктора запиту, *Access* автоматично створює еквівалентну інструкцію *SQL*. Користувач має можливість переглядати і змінювати інструкції *SQL* в режимі

SQL. Зміни, внесені до запиту в режимі *SQL*, приведуть до відповідних змін в бланку запиту в режимі конструктора. Деякі запити не можуть бути визначені в бланку запиту конструктора. Для створення таких запитів потрібно ввести інструкцію *SQL* безпосередньо у вікно запиту в режимі *SQL*.

Для перегляду та зміни інструкції *SQL* необхідно створити або відкрити існуючий запит, натиснувши на панелі інструментів кнопку Вигляд



і обрати режим *SQL* (Див. рис. 2):



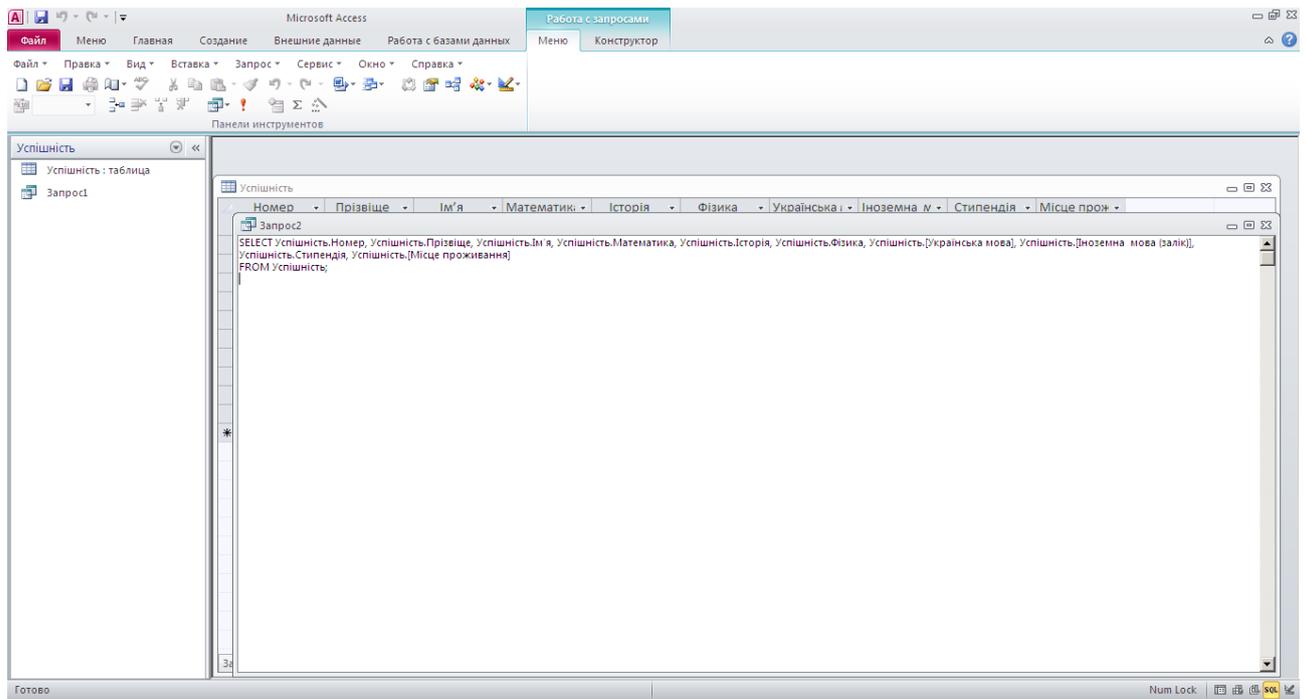


Рисунок 2 – Перегляд запиту в режимі *SQL*

Основні поняття мови *SQL*. Мова *SQL* надає для використання наступні функції:

- статистичні:
 - *AVG* (середнє значення);
 - *MAX* (максимальне значення);
 - *MIN* (мінімальне значення);
 - *SUM* (сума);
 - *COUNT* (кількість значень);
 - *COUNT (*)* (кількість ненульових значень);
- функції роботи з рядками:
 - *UPPER (Str)* (перетворення символів рядка *Str* до верхнього регістру);
 - *LOWER (Str)* (перетворення символів рядка *Str* до нижнього регістру);
 - *TRIM (Str)* (знищення пропусків на початку та в кінці рядка *Str*);

➤ *SUBSTRING (Str FROM n1 TO n2)* (виділення із рядка *Str* підрядка, який включає в себе символи, починаючи з позиції *n1* і закінчуючи позицією *n2*);

➤ *CAST (<Expression> AS <Type>)* (зведення виразу *Expression* до типу *Type*);

▪ функції декодування дати та часу:

➤ *EXTRACT (<Елемент> FROM <Вираз>)* (із виразу, який містить значення дати або часу, забирається значення, що відповідає вказаному елементу).

Вибір даних з таблиць полягає в отриманні з них полів і записів, які задовольняють заданим умовам. Результат виконання запиту, на основі якого обираються записи, називають *вибіркою*. Дані можна обирати із однієї або декількох таблиць за допомогою інструкції *SELECT*, яка має наступний формат:

SELECT [DISTINCT] {|<Список_полів>} FROM <Список_таблиць>*

[WHERE<Умова вибору>]

[ORDER BY <Список полів для сортування>]

[GROUP BY<Список полів для групування>]

[HAVING <Умови групування>]

[UNION< Вкладена інструкція SELECT >]

Список полів має містити хоча б одне поле. Якщо в результуючий НД (набір даних) треба включити всі поля таблиці, то замість перерахування імен полів можна вказати символ «*». При явному вказуванні імен полів ми можемо керувати порядком їх слідування в НД. Оператор *DISTINCT* дозволяє або забороняє повторення записів в результуючому НД. Якщо він відсутній, то в НД дозволено повторення записів (тобто ці записи мають однакові значення полів). Оператор *WHERE* задає умови відбору, яким мають задовольняти записи в результуючому НД. Оператор *ORDER BY* містить

список полів, які визначають порядок сортування записів результуючого НД. За замовченням сортування виконується в порядку зростання. Якщо вимагається для поля сортування в порядку спадання, то після імені відповідного поля вказується оператор *DESC*. Оператор *GROUP BY* дозволяє виділити групи записів в результуючому НД. Групою називають записи з однаковими значеннями в полях, перерахованих за оператором *GROUP BY*. Оператор *HAVING* діє сумісно з оператором *GROUP BY* і використовується для вибору записів у самих групах.

Інструкції *SELECT* можуть мати складну структуру і бути вкладені одна в одну. Для об'єднання інструкцій використовують оператор *UNION*, в якому розміщується вкладена інструкція *SELECT*. Результуючий НД – це записи, які відібрано із врахуванням виконання умов вибору, заданих операторами *WHERE* обох інструкцій.

Простий критерій вибору. Критерій вибору – логічний вираз, в якому можна використовувати операції, перераховані нижче.

1. Порівняння. Можливі оператори :
 - = (дорівнює);
 - > (більше)
 - < (менше)
 - >= (більше або дорівнює)
 - <= (менше або дорівнює)
 - <> або != (не дорівнює)
 - !> (не більше)
 - !< (не менше)

Приклад

SELECT Прізвище, Ім'я, Проживання *DISTINCT FROM* Успішність

WHERE Математика >=4

ORDER BY Прізвище DESC

2. *LIKE* (порівняння за шаблоном). У виразах операції *LIKE* допускається використання шаблону, в якому дозволені всі алфавітно-цифрові символи (із врахуванням регістру). Два символи мають спеціальне призначення:

«*» – заміна будь-якої кількості символів, в тому числі і нульового;

«?» – заміна одного символу;

«#» – заміна однієї цифри;

«[A–K]» – діапазон символів.

Приклад

SELECT Прізвище, Ім'я FROM Успішність

*WHERE Прізвище LIKE '*ан*'*

GROUP BY Інформатика

3. *IS NULL* (перевірка на нульове значення), яка має формат
<Вираз> *IS [NOT] NULL*

Приклад

SELECT Прізвище FROM Успішність

WHERE Історія IS NULL

4. *IN* (перевірка на входження значення у список), яка має формат
<Вираз> *[NOT] IN (<Список значень>)*, де значення вводяться як значення параметрів.

Приклад

SELECT Прізвище, Ім'я, Місце проживання FROM Успішність

WHERE Місце проживання *IN* (*Param1*,*Param2*)

GROUP BY Стипендія

HAVING Стипендія = 'True'

Значення *Param1* – Ужгород, *Param2* – Хуст.

5. *BETWEEN* (перевірка на входження значення в діапазон). Її формат

<Вираз> [*NOT*] *BETWEEN*

Приклад

*SELECT *FROM* Успішність

WHERE Історія *BETWEEN* '3' *AND* '5'

Складний критерій відбору складається з:

- простих умов;
- логічних операцій:
 - *AND* (логічне І);
 - *OR* (логічне АБО);
 - *NOT* (логічне НІ);
- круглих дужок.

За допомогою секції *WHERE* можна не тільки зв'язувати таблиці, але і створювати досить складні критерії відбору даних. Для цього в секції вказується довільна кількість операторів вигляду ПОЛЕ ~ ЗНАЧЕННЯ, які пов'язані логічними операціями *NOT*, *AND*, *OR*, де знак (~) означає операцію відношення:

*SELECT * FROM* Успішність

WHERE (Математика >=4) *AND* (Історія *BETWEEN* '3' *AND* '5')

ORDER BY Прізвище

Зауваження. В мові *SQL* пріоритет операцій порівняння вище за пріоритет логічних операцій. Тому відсутність дужок у виразі трактує його зміст зовсім інакше.

Модифікація записів. Модифікація записів полягає в редагуванні записів, вставленні в НД та знищення з НД записів. Редагування записів – зміна значень полів в групі записів. Воно виконується інструкцією наступного формату:

```
UPDATE <Ім'я таблиці>  
  
SET <Ім'я поля 1> = <Вираз 1>  
  
...  
  
< Ім'я поля N> = <Вираз N>  
  
[WHERE <Умови вибору >]
```

Приклад

```
UPDATE Успішність SET Історія = Історія + 1  
  
WHERE Історія < 5
```

В одній інструкції *UPDATE* можна змінювати значення декількох полів. В цьому випадку для кожного з них вказується відповідне значення. Якщо оператор *WHERE* не заданий, то змінюються значення усіх вказаних полів.

Вставка записів виконується за допомогою інструкції *INSERT*, яка дозволяє додавати до таблиці одну або декілька записів. Для одного запису:

```
INSERT INTO <Ім'я таблиці>  
  
[( <Список полів> )]
```

VALUES (<Список значень>)

В результаті виконання цієї інструкції до таблиці, ім'я якої вказане після слова *INTO*, додається один запис. Для доданого запису заповнюються поля, перераховані в списку. Значення полів беруться зі списку, розміщеного після слова *VALUES*.

Приклад

*INSERT INTO Успішність (Прізвище, Ім'я, Місце проживання
проживання)*

VALUES ('Товт', 'Олександр', 'Ужгород')

При додаванні до таблиці декількох значень інструкція *INSERT* має формат:

INSERT INTO <Ім'я таблиці>

(<Список полів>)

ІНСТРУКЦІЯ SELECT

Значення полів нових записів визначаються через значення полів записів, відібраних за допомогою інструкції *SELECT*. Кількість доданих записів дорівнює кількості відібраних записів. Список значень полів, що повертається інструкцією *SELECT*, має відповідати списку інструкцій *INSERT* за кількістю і типу полів.

Таблиця може містити обчислювальні поля. При їх визначенні вказуються не їх імена, а вираз, за яким обчислюються їх значення.

Приклад

*SELECT *, CINA*0,2+CINA AS NEWCINA FROM SKLAD*

NEWCINA – обчислювальне поле.

Завдання для самостійної роботи.

Згідно індивідуального завдання з Лабораторної роботи №1, виконати наступні дії:

1. Створити найпростіший додаток, який дозволяє виконувати та створювати *SQL*-запити.
2. Створити наступні види *SQL*-запитів:
 - a) простий запит на вибірку;
 - b) запит на вибірку з використанням спеціальних функцій: *LIKE*, *IS NULL*, *IN*, *BETWEEN*;
 - c) запит зі складним критерієм;
 - d) запит з унікальними значеннями;
 - e) запит з використанням обчислювального поля;
 - f) запит з групуванням по заданому полю, використовуючи умову групування;
 - g) запит із сортування по заданому полю в порядку зростання та спадання значень;
 - h) запит з використанням дій по модифікації записів.
3. Оформити короткий звіт про виконану роботу, внести в нього основні теоретичні відомості.

Питання для самоконтролю:

1. Що таке запит?
2. Які бувають типи запитів (за способом створення)?
3. Що таке запит на вибірку?
4. Як створити запит у режимі конструктора?
5. Як створити запит за допомогою майстра?
6. Як за допомогою запиту відсортувати дані?
7. Яке призначення функцій *LIKE*, *IS NULL*, *IN*, *BETWEEN*?
8. Як створити запит за шаблоном?

9. Що таке простий критерій вибору?
- 10.Що таке складний критерій вибору?
- 11.Що таке модифікація даних?
- 12.Як створити запит для редагування даних?
- 13.Як виконати вставку запису?

Лабораторна робота № 3

Тема: Створення бази даних в СУБД *Microsoft SQL Server*.

Мета: за допомогою операторів мови *Transact SQL* навчитися створювати бази даних, спроектувати її базові таблиці і визначати відношення між ними.

Основні теоретичні відомості.

Мова *SQL*.

Мова *SQL* є найбільш поширеною мовою для роботи з БД (базами даних). На даний час існують такі міжнародні стандарти на мову *SQL*: *SQL1*, *SQL2*, *SQL3*.

Мова *SQL* не володіє функціями повноцінної мови розробки і орієнтована на доступ до БД. Використання мови *SQL* може бути самостійним і вона може включатися в склад засобів розробки програм. В цьому випадку її називають вбудованою *SQL*. Розрізняють два головних методи використання вбудованої *SQL*: статичний і динамічний.

Статичне використання передбачає застосування в програмі функцій викликів мови *SQL*, які включаються в програмний модуль і виконуються після компіляції програми.

Динамічне використання передбачає динамічну побудову викликів функцій мови *SQL* та інтерпретацію цих викликів у ході виконання програми. Динамічний метод застосовується тоді, коли вид *SQL* запиту заздалегідь невідомий і будується у діалозі з користувачем.

Будь-яке *SQL*-застосування реляційної БД складається з трьох частин: інтерфейсу користувача, набору таблиць в БД і *SQL*-машини.

Microsoft SQL сервер підтримує доступ до БД в багатокористувацькому режимі за допомогою локальної мережі, та мережі Інтернет. Доступ до бази реалізованої на *Microsoft SQL* сервері, може здійснюватись за допомогою *Web*-додатків, мов програмування *Java*, та *C#*. А також підтримується робота з мовою розмітки *XML*.

Microsoft SQL Server – це реляційна СУБД. У реляційних БД дані зберігаються в таблицях. Взаємопов'язані дані можуть групуватися в таблиці, крім того, можуть бути встановлені також і відношення між таблицями. Користувачі отримують доступ до даних на сервері через програми, а

адміністратори, виконуючи завдання конфігурування, адміністрування та підтримки БД, виробляють безпосередній доступ до сервера. *SQL Server* є масштабованою БД, це означає, що вона може зберігати значні обсяги даних і підтримувати роботу багатьох користувачів, які здійснюють одночасний доступ до БД.

Під час створення БД необхідно визначити її ім'я, розмір, а також файли і групи файлів, у яких вона буде зберігатися.

***SQL Server* створює нову БД у два етапи:**

1. Використовуючи копію бази *Model*, *SQL Server* ініціалізує нову та її метадані.

2. Після цього *SQL Server* заповнює частину, що залишилася, БД (крім сторінок із внутрішніми даними, що відбивають використання дискового простору, зайнятого БД) порожніми сторінками.

Приклад створення БД.

Як приклад БД, яка буде створена програмно за допомогою операторів мови *Transact SQL*, виберемо БД «Книжкова справа» (Див. рис. 3).

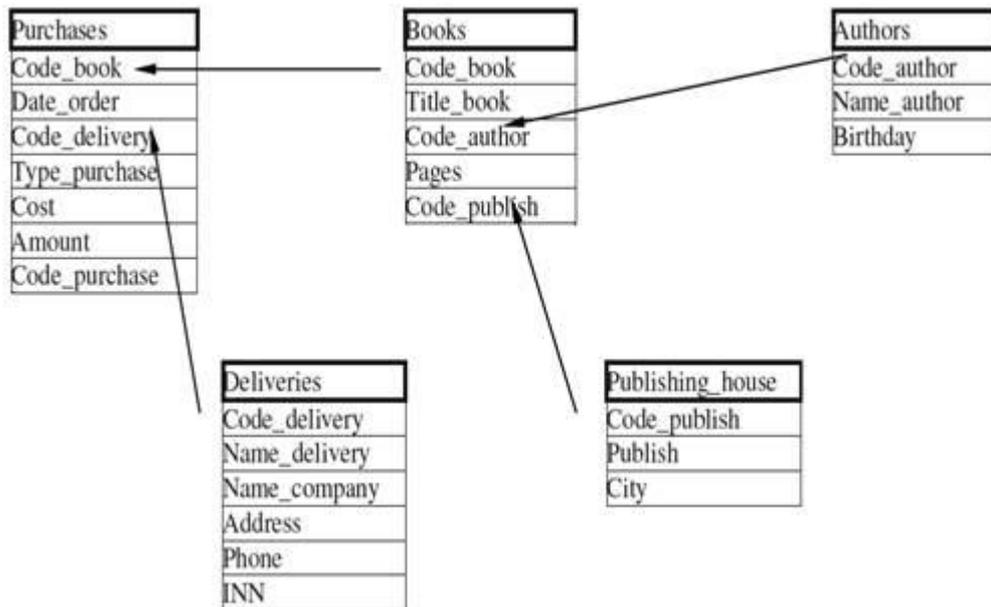


Рисунок 3 – Фрагмент БД «Книжкова справа»

Структура таблиць даної БД наведена в Таблицях 1 – 5.

Таблиця 1 – Покупки (назва таблиці *Purchases*)

Назва поля	Тип поля	Опис поля
<i>Code_book</i>	<i>Int</i>	Код книги, яку купують
<i>Date_order</i>	<i>DateTime</i>	Дата замовлення книги
<i>Code_delivery</i>	<i>Int</i>	Код постачальника
<i>Type_purchase</i>	<i>Bit</i>	Тип закупівлі (опт/роздріб)
<i>Cost</i>	<i>Money</i>	Вартість одиниці товару
<i>Amount</i>	<i>Int</i>	Кількість екземплярів
<i>Code_purchase</i>	<i>Int</i>	Код покупки

Таблиця 2 – Довідник книг (Назва таблиці *Books*)

Назва поля	Тип поля	Опис поля
<i>Code_book</i>	<i>Int</i>	Код книги
<i>Title_book</i>	<i>Char</i>	Назва книги
<i>Code_autho</i>	<i>Int</i>	Код автора
<i>Pages</i>	<i>Int</i>	Кількість сторінок
<i>Code_publish</i>	<i>Int</i>	Код видавництва

Таблиця 3 – Довідник авторів (назва таблиці <i>Authors</i>)		
Назва поля	Тип поля	Опис поля
<i>Code_author</i>	<i>Int</i>	Код автора
<i>Name_author</i>	<i>Char</i>	ПІБ автора
<i>Birthday</i>	<i>DateTime</i>	Дата народження

Таблиця 4 – Довідник постачальників (назва таблиці <i>Deliveries</i>)		
Назва поля	Тип поля	Опис поля
<i>Code_delivery</i>	<i>Int</i>	Код постачальника
<i>Name_delivery</i>	<i>Char</i>	ПІБ відповідальної особи
<i>Name_company</i>	<i>Char</i>	Назва компанії-постачальника
<i>Address</i>	<i>Char</i>	Юридична адреса
<i>Phone</i>	<i>Numeric</i>	Контактний телефон
<i>INN</i>	<i>Char</i>	ПІН

Таблиця 5 – Довідник видавництв (назва таблиці <i>Publishing_house</i>)		
Назва поля	Тип поля	Опис поля
<i>Code_publish</i>	<i>Int</i>	Код видавництва
<i>Publish</i>	<i>Char</i>	Видавництво
<i>City</i>	<i>Char</i>	Місто

Запустити *SQL Server Management Studio*, перевірити включення сервера. Для запуску *MS SQL Server 2005* оберіть утиліту *SQL Server Management Studio* та запустіть її. Для написання програмного коду в *SQL Server Management Studio* потрібно натиснути кнопку «Створити запит» («*New query*») на панелі інструментів «Стандартна» («*Standart*»).

Створити нову БД з назвою *DB_Books* за допомогою команди: *CREATE DATABASE DB_BOOKS*.

Для виконання команди натиснути *F5* або виконати команду *Execute* розділу *Query* головного меню.

Відкрити утиліту *SQL Server Management Studio*. Перевірити наявність БД *DB_Books*, якщо її не бачите в розділі *DataBases*, то натисніть *F5* для оновлення. Створена БД матиме наступний вигляд (Див. рис. 4):

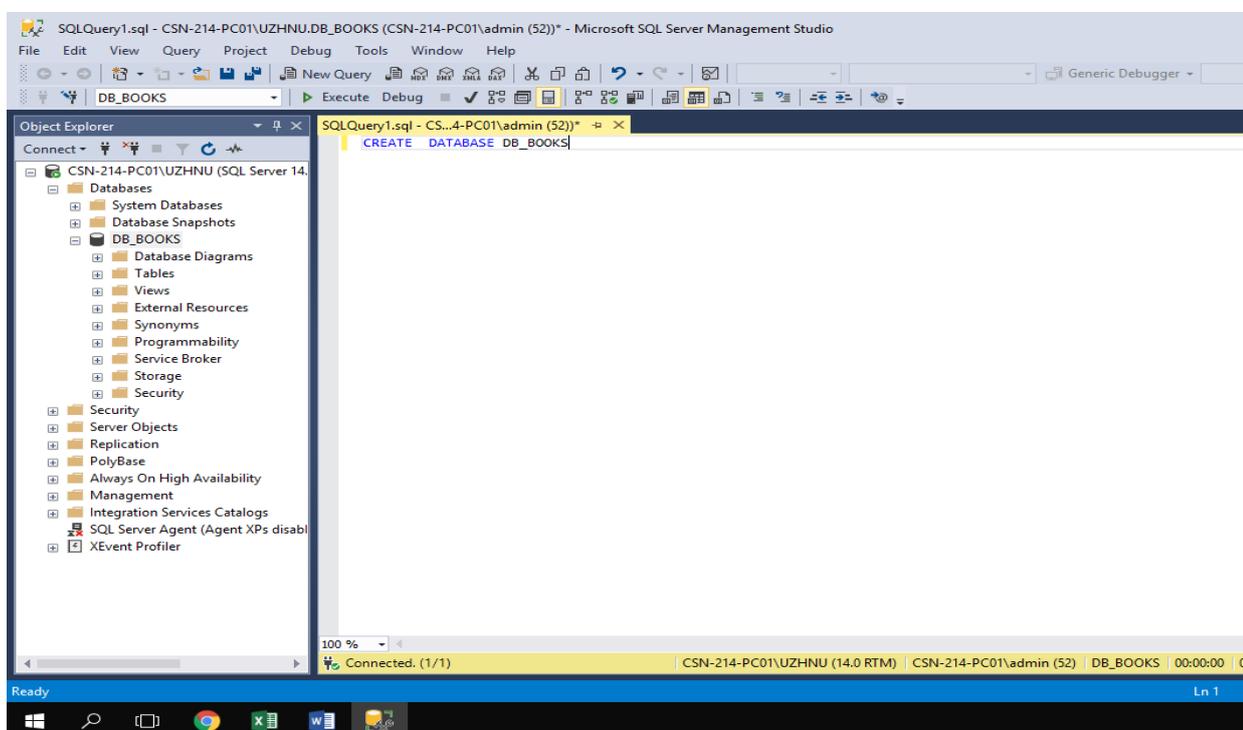


Рисунок 4 – Результат створення БД

Створити в ній перераховані таблиці за допомогою нижченаведених команд (для створення нової сторінки для коду в *SQL Server Management Studio* натиснути кнопку «*New query*»).

```
use DB_BOOKS
```

```
CREATE TABLE Authors(Code_author INT PRIMARY KEY, name_author CHAR(30), Birthday DATETIME)
```

```
CREATE TABLE Publishing_house(Code_publish INT PRIMARY KEY, Publish CHAR(30), City CHAR(20))
```

```
CREATE TABLE Books(Code_book INT PRIMARY KEY, Title_book CHAR(40), Code_author INT FOREIGN KEY REFERENCES
```

Authors(Code_author), Pages INT, Code_publish INT FOREIGN KEY REFERENCES Publishing_house(Code_publish))

CREATE TABLE Deliveries(Code_delivery INT PRIMARY KEY, Name_delivery CHAR(30), Name_company CHAR(20), Address VARCHAR(100), Phone BIGINT, INN CHAR(13))

CREATE TABLE Purchases(Code_purchase INT PRIMARY KEY, Code_book INT FOREIGN KEY REFERENCES Books(Code_book), Date_order SMALLDATETIME, Code_delivery INT FOREIGN KEY REFERENCES Deliveries(Code_delivery), Type_purchase BIT, Cost FLOAT, Amount INT)

Запустіть команду клавішею *F5*.

В утиліті *SQL Server Management Studio* перевірити наявність БД *DB_Books* і таблиць в ній.

У розділі діаграм в контекстному меню обрати «*New diagram*» створити нову діаграму, в яку додати зі списку п'ять наших таблиць, перевірити зв'язки між таблицями (Див. рис. 5).

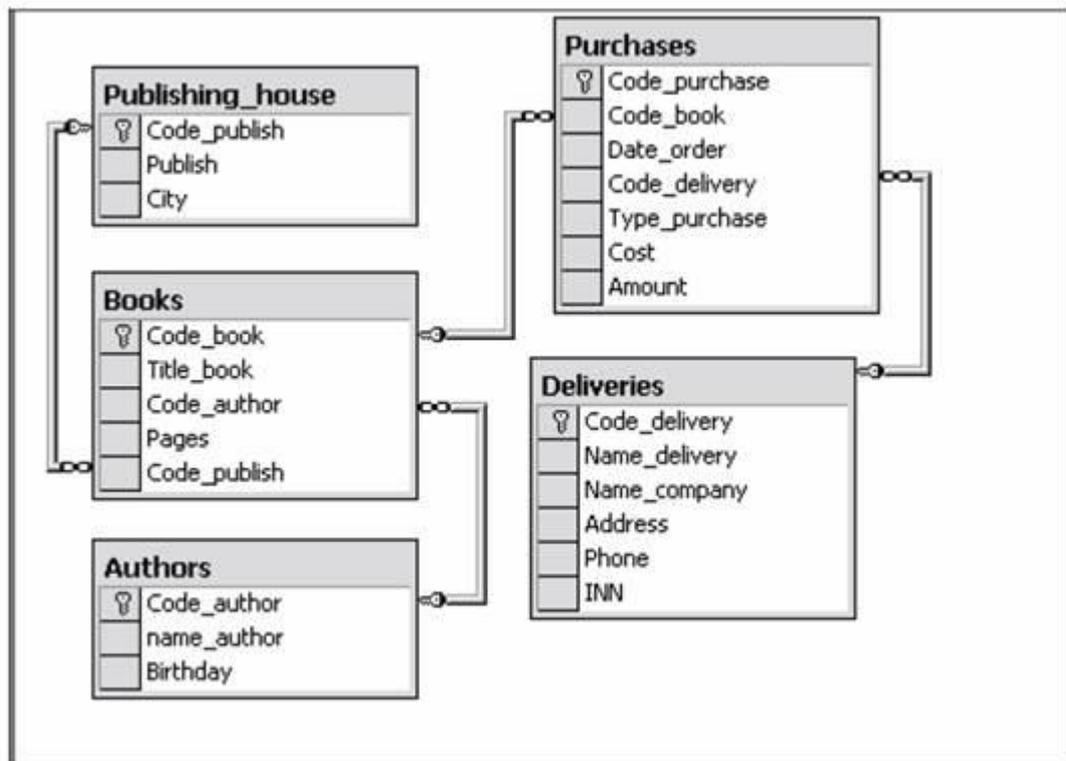


Рисунок 5 – Результат створення діаграми

Використані оператори:

PRIMARY KEY – ознака створення ключового поля.

FOREIGN KEY ... REFERENCES ... – ознака створення поля зв'язку з іншою таблицею.

CREATE TABLE – команда створення таблиці в поточній БД.

USE – зробити активною конкретну БД.

CREATE DATABASE – команда створення нової БД.

Завдання до лабораторної роботи:

В утиліті *SQL Server Management Studio* створити нову БД за допомогою оператора *Create Database*, назву БД визначити, виходячи з предметної області. Закоментувати оператор (-- – однорядковий коментар, /* */ – багаторядковий коментар). Програмно зробити активною створену БД за допомогою оператора *Use*. Створити перераховані таблиці за допомогою операторів *Create table*, причому самостійно визначити типи таблиць

(батьківська чи підпорядкована), типи полів і їх розміри, знайти поля типу *Primary key* і *Foreign key*. Зберегти файл програми з назвою *ПрізвищеСтудента_Лаб_3_№варіанта*. У *SQL Server Management Studio* в розділі діаграм створеної БД згенерувати нову діаграму, перевірити зв'язки між таблицями.

Питання для самоконтролю:

1. Яке призначення і склад оператора *SELECT*?
2. Вимоги до порядку розміщення стовпців в операторі *SELECT*.
3. Яка особливість використання символу (*) в операторі *SELECT*?
4. Призначення пропозиції оператора *SELECT – FROM*.
5. Яке призначення пропозиції оператора *SELECT – WHERE*?
6. Яка суть пошуку за шаблоном?
7. Які особливості використання ключових слів *AND* і *OR*?
8. Яке призначення пропозиції оператора *SELECT – ORDER BY*?
9. Яке призначення і склад оператора *INSERT*?
10. Яке призначення і структура оператора *UPDATE*?

Лабораторна робота № 4

Тема: Використання операторів маніпулювання даними в СУБД *Mikrosoft SQL Server*.

Мета: навчитися використовувати оператори маніпулювання даними *Select, Insert, Update, Delete*.

Основні теоретичні відомості.

Оператори маніпулювання даними.

SQL займає центральне місце у проектуванні та використанні реляційних БД (баз даних). Будь-який програмний додаток, що взаємодіє з реляційною базою даних, незалежно від його користувальницького інтерфейсу, посилає серверу баз даних оператори *SQL*.

Оператор *SQL* складається з набору команд, що виконують певні дії над об'єктами бази даних або даними, що зберігаються в ній. Реляційні БД підтримують три типи операторів *SQL*: мову визначення даних (*Data Definition Language, DDL*), мову маніпулювання даними (*Data Manipulation Language, DML*) і мову управління даними (*Data Control Language, DCL*).

Мова маніпулювання даними використовується для отримання, занесення, редагування та видалення даних, що містяться в об'єктах, визначених за допомогою *DDL*. Як основні оператори маніпулювання даними використовуються оператори: *SELECT, INSERT, UPDATE* і *DELETE*.

Оператор *SELECT* здійснює вибірку інформації, що зберігається в БД і дозволяє вибрати один або декілька записів з однієї або декількох таблиць. Оператор *INSERT* додає в таблицю новий запис (рядок), оператор *UPDATE* служить для редагування даних, оператор *DELETE* видаляє записи з таблиці.

Оператор *SELECT* є фактично найважливішим для користувача й самим складним оператором *SQL*. Він призначений для вибірки даних із таблиць, тобто він і реалізує одне з основних призначень БД – надавати інформацію користувачеві.

Оператор *SELECT* завжди виконується над деякими таблицями, що входять у БД. Насправді в БД можуть бути не тільки постійно збережені таблиці, а також тимчасові таблиці й так звані перегляди. Перегляд – це *SELECT* - вираз, що зберігається в БД. З погляду користувачів перегляд – це таблиця, яка не зберігається постійно в БД, а «виникає» у момент звертання до неї. З погляду оператора *SELECT* і постійно збережені таблиці, і тимчасові таблиці, й перегляди виглядають зовсім однаково. Звичайно при реальному виконанні оператора *SELECT* системою враховуються розбіжності між збереженими таблицями й переглядами, але ці розбіжності приховані від користувача. Результатом виконання оператора *SELECT* завжди є таблиця.

Оператор *INSERT* додає в таблицю новий запис.. Якщо користувач буде вводити всі значення в новий рядок в порядку, який був визначений при створенні таблиці, то можна вказувати лише ім'я таблиці та список значень, що вводяться.

Якщо список вводу неповний, або порядок значень відрізняється від того, що закладався при створенні таблиці, після назви таблиці вказується потрібний список полів. Полям, що не вказані в списку, присвоюється значення по замовчуванню, якщо воно вказувалося при створенні таблиці або ж значення *NULL*.

За допомогою оператора *INSERT* можна переміщати значення від однієї таблиці до іншої, якщо структура згаданих таблиць ідентична.

Оператор *UPDATE* дозволяє змінювати значення деяких або всіх полів таблиці.

Наприклад, нехай необхідно замінити назву дисципліни «Математика» (*subj_id* = 43) на назву «Вища математика», при цьому ідентифікаційний номер необхідно зберегти:

```
UPDATE subject1
```

```
SET subj_name = 'Вища математика', HOUR = 36, SEMESTER= 1
```

```
WHERE subj_id = 43
```

Видалення записів із таблиці здійснюється за допомогою команди *DELETE*. Можна усунути всі записи таблиці. В результаті таблиця стає порожньою, після чого вона може бути видаленою командою *DROP TABLE*.

Для усунення декількох записів застосовується умова *WHERE*:

```
DELETE FROM STUDENT1
```

```
WHERE CITY = 'Київ'
```

В умові *WHERE* команди *DELETE* можна застосовувати підзапити. В полях умови *FROM* підзапиту не можна посилатися на таблицю, з якої здійснюється видалення. Однак можна посилатися на поточний запис, що є кандидатом на видалення, тобто на запис, який на даний час перевіряється в умові основного запиту.

Завдання для самостійної роботи:

Створити нову БД з назвою *DB_Books* за допомогою оператора *Create Database*, створити в ній перераховані таблиці с допомогою операторів *Create table*. Зберегти файл програми з назвою *ПрізвищеСтудента_Лаб_4_DB_Books*. В утиліті *SQL Server Management Studio* за допомогою кнопки «Створити запит» створити окремі програми по кожному запиту, які зберігаються на диску з назвою:

ПрізвищеСтудента_Лаб_4_№_завдання. Можна зберігати всі виконані запити в одному файлі. Для перевірки роботи операторів *SELECT* попередньо створити програму, яка за допомогою операторів *INSERT* заповнить всі таблиці БД *DB_Books* декількома записами, збережіть програми з назвою ПрізвищеСтудента_Лаб_4_Insert.

Сортування.

1. Вибрати всі відомості про книги з таблиці *Books* і впорядкувати результат за кодом книги (поле *Code_book*).
2. Вибрати з таблиці *Books* коди книг, назви і кількість сторінок (поля *Code_book*, *Title_book* і *Pages*), впорядкувати результат за назвами книг (поле *Title_book* по зростанню) і по полю *Pages* (по спаданню).
3. Вибрати з таблиці *Deliveries* список постачальників (поля *Name_delivery*, *Phone* і *INN*), впорядкувати результат по полю *INN* (по спаданню).

Зміна порядку полів.

4. Вибрати всі поля з таблиці *Deliveries* таким чином, щоб в результаті порядок стовпців був наступним: *Name_delivery*, *INN*, *Phone*, *Address*, *Code_delivery*.
5. Вибрати всі поля з таблиці *Publishing_house* таким чином, щоб в результаті порядок стовпців був наступним: *Publish*, *City*, *Code_publish*.

Вибір деяких полів з двох таблиць.

6. Вибрати з таблиці *Books* назви книг і кількість сторінок (поля *Title_book* і *Pages*), а з таблиці *Authors* вибрати ім'я відповідного автора книги (поле *Name_author*).

7. Вибрати з таблиці *Books* назви книг і кількість сторінок (поля *Title_book* і *Pages*), а з таблиці *Deliveries* вибрати ім'я відповідного постачальника книги (поле *Name_delivery*).

8. Вибрати з таблиці *Books* назви книг і кількість сторінок (Поля *Title_book* і *Pages*), а з таблиці *Publishing_house* вибрати назву відповідного видавництва і місця видання (поля *Publish* і *City*).

Умова неточного збігу.

9. Вибрати з довідника постачальників (таблиця *Deliveries*) назви компаній, телефони та ПІН (поля *Name_company*, *Phone* і *INN*), у яких назва компанії (поле *Name_company*) починається з «ВАТ».

10. Вибрати з таблиці *Books* назви книг і кількість сторінок (поля *Title_book* і *Pages*), а з таблиці *Authors* вибрати ім'я відповідного автора книги (поле *Name_author*), у яких назва книги починається зі слова «Мемуари».

11. Вибрати з таблиці *Authors* прізвища, імена, по батькові авторів (поле *Name_author*), значення яких починаються з «Бойко».

Відсутність точного співпадання значень одного з полів.

12. Вивести список назв видавництв (поле *Publish*) з таблиці *Publishing_house*, які не перебувають в місті «Харків» (умова по полю *City*).

13. Вивести список назв книг (поле *Title_book*) з таблиці *Books*, які випущені будь-якими видавництвами, крім видавництва «ПітерСофт» (поле *Publish* з таблиці *Publishing_house*).

Вибір записів відповідно до діапазону значень (*Between*).

14. Вивести прізвища, імена, по батькові авторів (поле *Name_author*) з таблиці *Authors*, у яких дата народження (поле *Birthday*) знаходиться в діапазоні 01.01.1840 - 01.06.1860.

15. Вивести список назв книг (поле *Title_book* з таблиці *Books*) і кількість примірників (поле *Amount* з таблиці *Purchases*), які були закуплені в період з 12.03.2016 по 15.06.2016 (умова по полю *Date_order* з таблиці *Purchases*).

16. Вивести список назв книг (поле *Title_book*) і кількість сторінок (поле *Pages*) з таблиці *Books*, у яких обсяг в сторінках належить діапазону 200–300 (умова по полю *Pages*).

17. Вивести список прізвищ, імен, по батькові авторів (поле *Name_author*) з таблиці *Authors*, у яких прізвище починається на одну з букв діапазону «В»–«Г» (умова по полю *Name_author*).

Вибір записів відповідно до діапазону значень (*In*).

18. Вивести список назв книг (поле *Title_book* з таблиці *Books*) і кількість (поле *Amount* з таблиці *Purchases*), які були надані постачальниками з кодами 3, 7, 9, 11 (умова по полю *Code_delivery* з таблиці *Purchases*).

19. Вивести список назв книг (поле *Title_book*) з таблиці *Books*, які випущені такими видавництвами: «Київ-Софт», «Альфа», «Наука» (умова по полю *Publish* з таблиці *Publishing_house*).

20. Вивести список назв книг (поле *Title_book*) з таблиці *Books*, які написані наступними авторами: «Леся Українка», «Іван Франко», «Тарас Шевченко» (умова по полю *Name_author* з таблиці *Authors*).

Вибір записів з використанням *Like*.

21. Вивести список авторів (поле *Name_author*) з таблиці *Authors*, які починаються на букву «К».

22. Вивести назви видавництв (поле *Publish*) з таблиці *Publishing_house*, які містять в назві поєднання «софт».

23. Вибрати назви компаній (поле *Name_company*) з таблиці *Deliveries*, у яких значення закінчується на «ський».

Вибір записів відповідно до декількох умов.

24. Вибрати коди постачальників (поле *Code_delivery*), дати замовлень (поле *Date_order*) і назви книг (поле *Title_book*), якщо кількість книг (поле *Amount*) в замовленні більше 100 або ціна (поле *Cost*) за книгу знаходиться в діапазоні від 200 до 500.

25. Вибрати коди авторів (поле *Code_author*), імена авторів (поле *Name_author*), назви відповідних книг (поле *Title_book*), якщо код видавництва (поле *Code_Publish*) знаходиться в діапазоні від 10 до 25 і кількість сторінок (поле *Pages*) в книзі більше 120.

26. Вивести список видавництв (поле *Publish*) з таблиці *Publish-ing_house*, в яких випущені книги, назви яких (поле *Title_book*) починаються зі слова «Праці» і місто видання (поле *City*) – «Харків».

Багатотабличні запити (вибірка з двох таблиць, вибірка з трьох таблиць з використанням *JOIN*).

27. Вивести список назв компаній-постачальників (поле *Name_company*) і назви книг (поле *Title_book*), які вони постачали в період з 01.01.2016 по 31.12.2016 (умова по полю *Date_order*).

28. Вивести список авторів (поле *Name_author*), книги яких були випущені у видавництві «Мир» (умова по полю *Publish*).

29. Вивести список постачальників (поле *Name_company*), які постачають книги видавництва «Махаон» (умова по полю *Publish*).

30. Вивести список авторів (поле *Name_author*) і назви книг (поле *Title_book*), які були поставлені постачальником «ВАТ Книготорг» (умова по полю *Name_company*).

Обчислення.

31. Вивести сумарну вартість партії однойменних книг (використовуючи поля *Amount* і *Cost*) і назву книги (поле *Title_book*) в кожній поставці.

32. Вивести вартість однієї друкованої сторінки кожної книги (використовувати поля *Cost* і *Pages*) і назви відповідних книг (поле *Title_book*).

33. Вивести кількість років з моменту народження авторів (використовувати поле *Birthday*) і імена відповідних авторів (поле *Name_author*).

Обчислення підсумкових значень з використанням агрегатних функцій.

34. Вивести загальну суму поставок книг (використовувати поле *Cost*), виконаних «ЗАТ Оптторг» (умова по полю *Name_company*).

35. Вивести загальну кількість всіх поставок (використовувати будь-яке поле з таблиці *Purchases*), виконаних в період з 01.01.2016 по 01.02.2016 (умова по полю *Date_order*).

36. Вивести середню вартість (використати поле *Cost*) і середню кількість екземплярів книг (використати поле *Amount*) в одній поставці, де автором книги є «Олесь Гончар» (поле *Name_author*).

37. Вивести всі відомості про постачання (всі поля таблиці *Purchases*), а також назву книги (поле *Title_book*) з мінімальною загальною вартістю (використовувати поля *Cost* і *Amount*).

38. Вивести всі відомості про постачання (всі поля таблиці *Purchases*), а також назву книги (поле *Title_book*) з максимальною загальною вартістю (використовувати поля *Cost* і *Amount*).

Зміна імен полів (використання оператора AS).

39. Вивести назву книги (поле *Title_book*), сумарну вартість партії однойменних книг (використовувати поля *Amount* і *Cost*), помістивши результат в поле з назвою *Itoigo*, поставки за період з 01.01.2016 по 01.06.2016 (умова по полю *Date_order*).

40. Вивести вартість однієї друкованої сторінки кожної книги (використати поля *Cost* і *Pages*), помістивши результат в поле з назвою *One_page*, і назви відповідних книг (поле *Title_book*).

41. Вивести загальну суму поставок книг (використовувати поле *Cost*) і помістити результат в поле з назвою *Sum_cost*, виконаних «ЗАТ Промінь» (умова по полю *Name_company*).

Використання змінних в умові.

42. Вивести список угод (всі поля з таблиці *Purchases*) за останній місяць (умова з використанням поля *Date_order*).

43. Вивести список авторів (поле *Name_author*), вік яких менше заданого користувачем (умова з використанням поля *Birthday*).

44. Вивести список книг (поле *Title_book*), яких закуплено менше, ніж зазначено в запиті користувача (умова з використанням поля *Amount*).

Використання змінних замість назв таблиць.

45. Вивести список назв компаній-постачальників (поле *Name_company*) і назви книг (поле *Title_book*), які вони поставили.

46. Вивести список авторів (поле *Name_author*), книги яких були випущені у видавництвах «СВІТ», «НАУКА» (умова по полю *Publish*).

47. Вивести список видавництв (поле *Name_company*), книги, що були продані за ціною 350 грн. (поле *Cost*).

Вибір результату в курсор.

48. Вивести список назв книг (поле *Title_book*) і кількості сторінок (поле *Pages*) в кожній книзі і помістити результат в курсор з назвою *Temp1*.

49. Вивести список назв компаній-постачальників (поле *Name_company*) і помістити результат в курсор з назвою *Temp2*.

50. Вивести список авторів (поле *Name_author*) і помістити результат в курсор з назвою *Temp3*.

Використання функцій спільно з підзапитом.

51. Вивести список книг (поле *Title_book*), у яких кількість сторінок (поле *Pages*) більше середньої кількості сторінок усіх книг в таблиці.

52. Вивести список авторів (поле *Name_author*), вік яких менше середнього віку всіх авторів в таблиці (умова по полю *Birthday*).

53. Вивести список книг (поле *Title_book*), у яких кількість сторінок (поле *Pages*) дорівнює мінімальній кількості сторінок книг, представлених в таблиці.

Використання квантора існування в запитах.

54. Вивести список видавництв (поле *Publish*), книги яких були придбані оптом («опт» з поля *Type_Purchase*).

55. Вивести список авторів (поле *Name_author*), книг яких немає в таблиці *Books*.

56. Вивести список книг (поле *Title_book*), які були поставлені постачальником «ЗАТ Квантор» (умова по полю *Name_company*).

Оператор обробки даних *Update*.

57. Змінити в таблиці *Books* вміст поля *Pages* на 300, якщо код автора рівний 56 (поле *Code_author*) і назва книги (поле *Title_book*) – «Мемуари».

58. Змінити в таблиці *Deliveries* вміст поля *Address* на «немає відомостей», якщо значення поля є порожнім.

59. Збільшити в таблиці *Purchases* ціну (поле *Cost*) на 20 відсотків, якщо замовлення було оформлено протягом останнього місяця (умова по полю *Date_order*).

Оператор обробки даних *Insert*.

60. Додати в таблицю *Purchases* новий запис, причому так, щоб код покупки (поле *Code_purchase*) було автоматично збільшено на одиницю, а в тип закупівлі (поле *Type_purchase*) внести значення «ОПТ».

61. Додати в таблицю *Books* новий запис, причому замість ключового поля поставити код (поле *Code_book*), автоматично збільшений на одиницю від максимального коду в таблиці, замість назви книги (поле *Title_book*) написати «Наука. Техніка. Інновації».

62. Додати в таблицю *Publish_house* новий запис, причому замість ключового поля поставити код (поле *Code_publish*), автоматично збільшений на одиницю від максимального коду в таблиці, замість назви міста – «Київ» (поле *City*), замість видавництва – «Наука» (поле *Publish*).

Оператор обробки даних *Delete*.

63. Видалити з таблиці *Purchases* всі записи, у яких кількість книг в замовленні (поле *Amount*) = 0.

64. Видалити з таблиці *Authors* всі записи, у яких немає імені автора в полі *Name_Author*.

65. Видалити з таблиці *Deliveries* всі записи, у яких не зазначено ПІН (поле *INN* порожнє).

Питання для самоконтролю:

1. Яке основне призначення оператора *SELECT*?
2. Які головні властивості результуючого набору описує більшість операторів *SELECT*?
3. Загальну структуру конструкцій оператора *SELECT*.
4. Які конструкції оператора *SELECT* є обов'язковими та з якою метою вони використовуються?
5. Що таке підзапити і для чого вони використовуються?
6. Визначення курсору.
7. Підтримку яких функцій забезпечують курсори?
8. Методи додавання інформації в БД.
9. Призначення, структура та порядок застосування оператора *INSERT*.
10. Призначення, структура та порядок застосування оператора *UPDATE*.
11. Призначення, структура та порядок застосування операторів, що видаляють дані з базових таблиць.

Лабораторна робота №5

Тема: Використання тригерів та переглядів в СУБД *Mikrosoft SQL Server*.

Мета: Навчитися програмно реалізовувати тригери та представлення в середовищі *MS SQL Server*.

Тригери представляють спеціальний тип збереженої процедури, яка викликається автоматично при виконанні певних дій над таблицею або переглядом, зокрема, при додаванні, зміні або видаленні даних, тобто при виконанні команд *INSERT, UPDATE, DELETE*.

Основні теоретичні відомості.

Формальне визначення тригера:

```
1 CREATE TRIGGER і'мя_тригера
2 ON {і'мя_таблиці | і'мя_перегляду}
3 {AFTER | INSTEAD OF} [INSERT | UPDATE | DELETE]
4 AS вираз_sql
```

Для створення тригера застосовується вираз *CREATE TRIGGER*, після якого йде ім'я тригера. Як правило, ім'я тригера відображає тип операцій і ім'я таблиці, над якою проводиться операція. Кожен тригер асоціюється з певною таблицею або поданням, ім'я яких вказується після слова *ON*.

Потім встановлюється параметр тригера. Ми можемо використовувати один з двох параметрів:

AFTER: виконується після виконання дії. Визначено тільки для таблиці.

INSTEAD OF: виконується замість дії (тобто по суті дії – додавання, зміна або видалення – взагалі не виконуються). Визначається для таблиць та переглядів.

Після цього тригера вказують операції, для яких визначається тригер: *INSERT*, *UPDATE* або *DELETE*.

Для тригера *AFTER* можна застосовувати відразу для декількох дій, наприклад, *UPDATE* і *INSERT*. У цьому випадку операції вказуються через кому. Для тригера *INSTEAD OF* можна визначити тільки одну дію.

І потім після слова *AS* йде набір виразів *SQL*, які власне і складають тіло тригера.

Створимо тригер. Допустимо, у нас є база даних *productdb* з наступним визначенням:

```
1 CREATE DATABASE productdb;
2 GO
3
4 USE productdb;
5 CREATE TABLE Products
6 (
7     Id INT IDENTITY PRIMARY KEY,
8     ProductName NVARCHAR(30) NOT NULL,
9     Manufacturer NVARCHAR(20) NOT NULL,
10    ProductCount INT DEFAULT 0,
11    Price MONEY NOT NULL
12 );
```

Визначимо тригер, який буде спрацьовувати при додаванні та оновленні даних:

```
1 USE productdb;
2 GO
3 CREATE TRIGGER Products_INSERT_UPDATE
```

```
4 ON Products
5 AFTER INSERT, UPDATE
6 AS
7 UPDATE Products
8 SET Price = Price + Price * 0.38
9 WHERE Id = (SELECT Id FROM inserted)
```

Припустимо, в таблиці *Products* зберігаються дані про товари. Але ціна товару нерідко містить різні надбавки типу податку на додану вартість, податку на додану корупцію і так далі. Людина, що додає дані, може не знати усі ці тонкощі з податковою базою, і він визначає чисту ціну. За допомогою тригера ми можемо виправити ціну товару на деяку величину. Таким чином, тригер спрацьовуватиме при будь-якій операції *INSERT* або *UPDATE* над таблицею *Products*. Сам тригер змінюватиме ціну товару, а для отримання того товару, який був доданий або змінений, знаходимо цей товар по *Id*. Але яке значення повинен мати *Id* такого товару? Річ у тому, що при додаванні або зміні дані зберігаються в проміжну таблицю *inserted*. Вона створюється автоматично. І з неї ми можемо отримати дані про додані/змінені товари.

І після додавання товару в таблицю *Products* в реальності товар буде мати трохи більшу ціну, ніж та, яка була визначена при додаванні:

Тригери є однією з різновидів процедур. Їх виконання відбувається при виконанні для таблиці будь-якого оператора мови маніпулювання даними (*DML*). Тригери використовуються для перевірки цілісності даних, а також для відкату транзакцій.

Тригер – це відкомпільована *SQL*-процедура, виконання якої обумовлено настанням певних подій всередині реляційної бази даних. Застосування тригерів здебільшого вельми зручно для користувачів бази даних. І все ж їх використання часто пов'язано з додатковими витратами ресурсів на операції введення / виведення. У тому випадку, коли тих же результатів (з набагато меншими непродуктивними витратами ресурсів)

можна домогтися за допомогою збережених процедур або прикладних програм, застосування тригерів недоцільно.

Тригери – особливий інструмент *SQL*-сервера, який використовується для підтримки цілісності даних в базі даних. За допомогою обмежень цілісності, правил і значень за замовчуванням не завжди можна домогтися потрібного рівня функціональності. Часто потрібно реалізувати складні алгоритми перевірки даних, що гарантують їх достовірність і реальність. Крім того, іноді необхідно відстежувати зміни значень таблиці, щоб потрібним чином змінити пов'язані дані. Тригери можна розглядати як свого роду фільтри, які вступають в дію після виконання всіх операцій відповідно до правил, стандартних значень і т.д.

Тригер являє собою спеціальний тип збережених процедур, що запускаються сервером автоматично при спробі зміни даних в таблицях, з якими тригери пов'язані. Кожен тригер прив'язується до конкретної таблиці. Всі вироблені їм модифікації даних розглядаються як одна транзакція. У разі виявлення помилки або порушення цілісності даних відбувається відкат цієї транзакції. Тим самим внесення змін забороняється. Скасовуються також всі зміни, вже зроблені тригером.

Створює тригер тільки власник бази даних. Це обмеження дозволяє уникнути випадкового зміни структури таблиць, способів зв'язку з ними інших об'єктів і т.д.

Тригер являє собою дуже корисне і в той же час небезпечний засіб. Так, при неправильній логіці його роботи можна легко знищити цілу базу даних, тому тригери необхідно дуже ретельно налагоджувати.

На відміну від звичайної підпрограми, тригер виконується неявно в кожному випадку виникнення тригерної події, до того ж він не має аргументів.

Реалізація тригерів в середовищі *MS SQL Server*.

У реалізації СУБД *MS SQL Server* використовується наступний оператор створення або зміни тригера:

```
<Визначення_тригера> ::=
{CREATE | ALTER} TRIGGER ім'я_тригера
ON { ім'я_таблиці | ім'я_перегляду }
[WITH ENCRYPTION ]
{
  { { FOR | AFTER | INSTEAD OF }
  { [ DELETE] [,] [ INSERT] [,] [ UPDATE] }
  [ WITH APPEND ]
  [ NOT FOR REPLICATION ]
  AS
    sql_оператор[...n]
  } /
  { {FOR | AFTER | INSTEAD OF } { [INSERT] [,]
  [UPDATE] }
  [ WITH APPEND]
  [ NOT FOR REPLICATION]
  AS
    { IF UPDATE(ім'я_поля)
    [ {AND | OR} UPDATE(ім'я_поля)] [...n]
  /
  IF (COLUMNS_UPDATED){оператор_біт_обробки
  {оператор_біт_порівняння }біт_маска [...n]}
  sql_оператор [...n]
  }
}
```

У *Mikrosoft SQL Server* існує два параметри, що визначають поведінку тригерів:

AFTER. Тригер виконується після успішного виконання викликом його команд. Якщо ж команди з якої-небудь причини не можуть бути успішно завершені, тригер не виконується. Слід зазначити, що зміни даних в результаті виконання запиту користувача і виконання тригера здійснюється в тілі однієї транзакції: Якщо станеться відкат тригера, то будуть відхилені і призначені для користувача зміни. Можна визначити кілька *AFTER*-тригерів для кожної операції (*INSERT*, *UPDATE*, *DELETE*). Якщо для таблиці передбачено виконання кількох *AFTER*-тригерів, то за допомогою системної збереженої процедури *sp_settriggerorder* можна вказати, який з них буде виконуватися першим, а який останнім. За замовчуванням в *SQL Server* всі тригери є *AFTER*-тригерами.

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

Тригери розрізняють за типом команд, на які вони реагують.

Існує три типи тригерів:

INSERT TRIGGER – запускаються при спробі вставки даних за допомогою команди *INSERT*.

UPDATE TRIGGER – запускаються при спробі зміни даних за допомогою команди *UPDATE*.

DELETE TRIGGER – запускаються при спробі видалення даних за допомогою команди *DELETE*.

Конструкції *[DELETE] [,] [INSERT] [,] [UPDATE]* і *FOR / AFTER / INSTEAD OF* *{[INSERT] [,] [UPDATE]}* визначають, на яку команду буде реагувати тригер. При його створенні має бути вказана хоча б одна команда. Допускається створення тригера, що реагує на дві або на все три команди.

Аргумент *WITH APPEND* дозволяє створювати кілька тригерів кожного типу.

При створенні тригера з аргументом *NOT FOR REPLICATION* забороняється його запуск під час виконання модифікації таблиць механізмами реплікації.

Конструкція *AS sql_оператор [... n]* визначає набір *SQL*- операторів і команд, які будуть виконані при запуску тригера.

Всередині тригера не допускається виконання ряду операцій, таких, наприклад, як:

- створення, зміна та видалення бази даних;
- відновлення резервної копії бази даних або журналу транзакцій.

Виконання цих команд не дозволено, так як вони не можуть бути скасовані у разі відкату транзакції, в якій виконується тригер. Ця заборона навряд чи може якимось чином позначитися на функціональності створених тригерів. Важко знайти таку ситуацію, коли, наприклад, після зміни рядка таблиці потрібно виконати відновлення резервної копії журналу транзакцій.

Програмування тригера.

При виконанні команд додавання, зміни і видалення записів сервер створює дві спеціальні таблиці: *inserted* і *deleted*. У них містяться списки рядків, які будуть вставлені або видалені після закінчення транзакції. Структура таблиць *inserted* і *deleted* ідентична структурі таблиць, для якої визначається тригер. Для кожного тригера створюється свій комплект таблиць *inserted* і *deleted*, тому ніякої інший тригер не зможе отримати до них доступ. Залежно від типу операції, що забезпечує виконання тригера, вміст таблиць *inserted* і *deleted* може бути різним:

команда *INSERT* – в таблиці *inserted* містяться всі рядки, які користувач намагається вставити в таблицю; в таблиці *deleted* не буде ні одного рядка; після завершення тригера всі рядки з таблиці *inserted* перемістяться в вихідну таблицю;

команда *DELETE* – в таблиці *deleted* будуть міститися всі рядки, які користувач спробує видалити; тригер може перевірити кожен рядок і визначити, чи дозволено його видалення; в таблиці *inserted* не буде жодного рядка;

команда *UPDATE* – при її виконанні в таблиці *deleted* знаходяться старі значення рядків, які будуть видалені при успішному завершенні тригера. Нові значення рядків містяться в таблиці *inserted*. Ці рядки добавляться в вихідну таблицю після успішного виконання тригера.

Для отримання інформації про кількість рядків, які будуть змінені при успішному завершенні тригера, можна використовувати функцію *@@ROWCOUNT*. Вона повертає кількість рядків, оброблених останньою командою. Слід підкреслити, що тригер запускається ні до спроби змінити конкретний рядок, а в момент виконання команди зміни. Одна така команда впливає на безліч рядків, тому тригер повинен обробляти всі ці рядки.

Якщо тригер виявив, що зі 100 рядків, які вставляються або змінюються, тільки один не задовольняє ті чи інші умови, то ніякий рядок не буде вставлений, змінений або видалений. Така поведінка обумовлена вимогами транзакції – повинні бути виконані або всі модифікації, або жодної.

Тригер виконується неявно, як певна транзакція, тому всередині тригера допускається застосування команд управління транзакціями. Зокрема, при виявленні порушення обмежень цілісності для переривання виконання тригера і скасування всіх змін, які намагався виконати користувач, необхідно використовувати команду *ROLLBACK TRANSACTION*.

Для отримання списку стовпців, змінених при виконанні команд *INSERT* або *UPDATE*, що викликали виконання тригера, можна використовувати функцію *COLUMNS_UPDATED* (). Вона повертає двійкове число, кожний біт якого, починаючи з молодшого, відповідає одному стовпцю таблиці (в порядку проходження стовпців при створенні таблиці). Якщо біт встановлено у значення "1", то відповідний стовпець був змінений. Крім того, факт зміни стовпчика визначає і функція *UPDATE* (ім'я_стовпця).

Для видалення тригера використовується команда

```
DROP TRIGGER {ім'я_тригера} [... n]
```

Буває, що ми хочемо зупинити дію тригера, але видаляти його повністю не хочемо. В цьому випадку його можна тимчасово відключити за допомогою команди *DISABLE TRIGGER*:

```
1 DISABLE TRIGGER Products_INSERT_UPDATE ON Products
```

А коли тригер знадобиться, його можна включити за допомогою команди *ENABLE TRIGGER*:

```
1 ENABLE TRIGGER Products_INSERT_UPDATE ON Products
```

Для розгляду операцій з тригерами визначимо наступну базу даних *productsdb* (Див. рис. 6) та заповнимо її даними (Див. рис. 7):

```
1 CREATE DATABASE productsdb;
2 GO
3 USE productsdb;
4 CREATE TABLE Products
5 (
6     Id INT IDENTITY PRIMARY KEY,
7     ProductName NVARCHAR(30) NOT NULL,
8     Manufacturer NVARCHAR(20) NOT NULL,
9     ProductCount INT DEFAULT 0,
10    Price MONEY NOT NULL
11 );
12 CREATE TABLE History
13 (
```

```

14     Id INT IDENTITY PRIMARY KEY,
15     ProductId INT NOT NULL,
16     Operation NVARCHAR(200) NOT NULL,
17     CreateAt DATETIME NOT NULL DEFAULT GETDATE(),
18 );

```

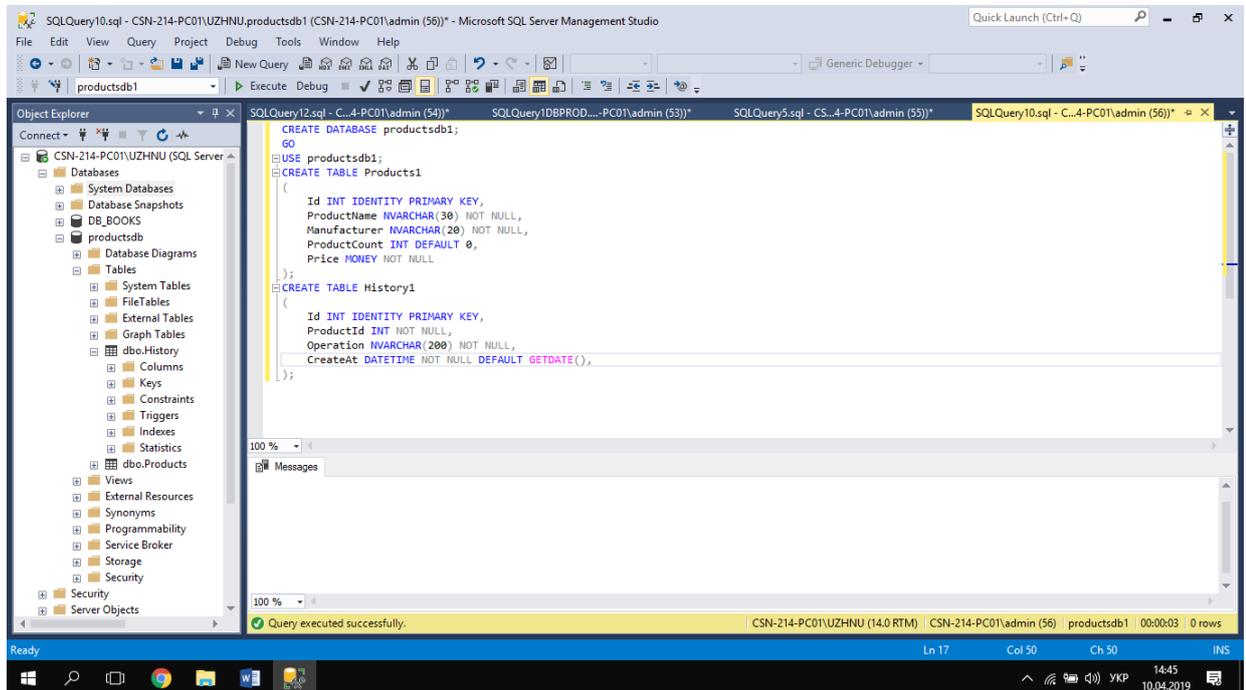


Рисунок 6 – Визначення БД *productsdb*

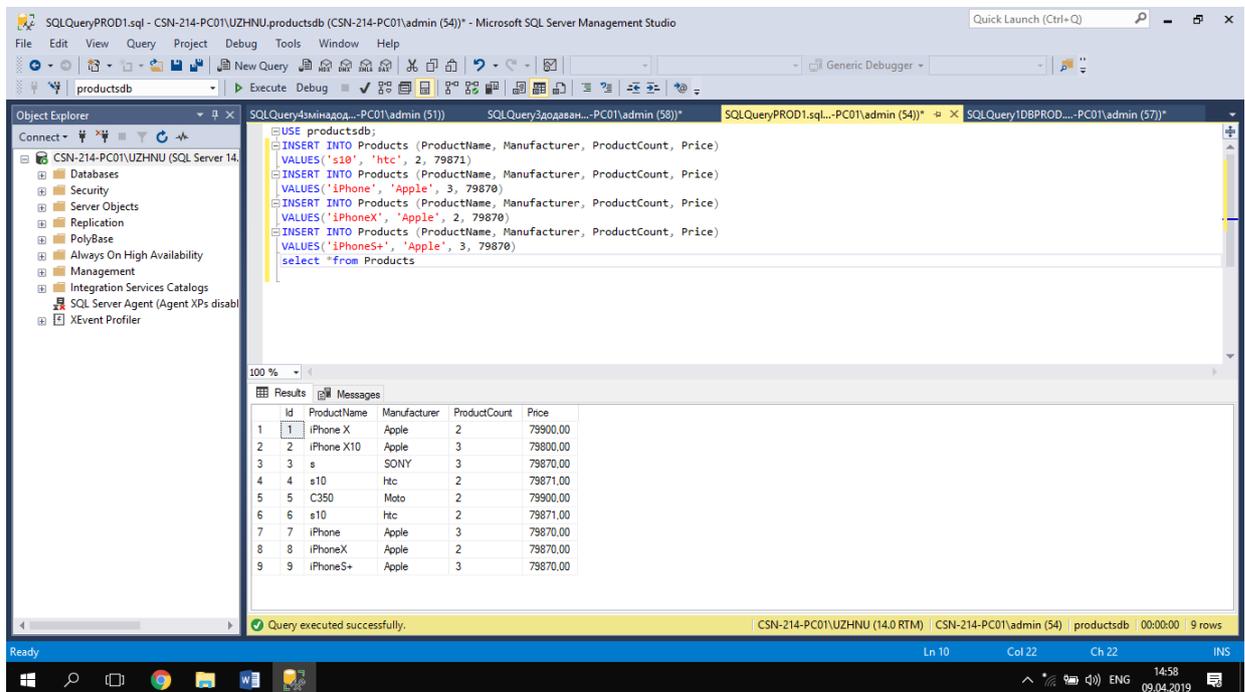


Рисунок 7 – Заповнення БД даними

Додавання.

При додаванні даних (при виконанні команди *INSERT*) в тригері ми можемо отримати додані дані з віртуальної таблиці *INSERTED*.

Визначимо тригер, який буде спрацьовувати після додавання:

```
USE productsdb
1 GO
2 CREATE TRIGGER Products_INSERT
3 ON Products
4 AFTER INSERT
5 AS
6 INSERT INTO History (ProductId, Operation)
7 SELECT Id, 'Доданий товар ' + ProductName + ' фірма ' + Manufacturer
8 FROM INSERTED
9
```

Цей тригер буде додавати в таблицю *History* дані про додавання товару, які беруться з віртуальної таблиці *INSERTED* (Див. рис. 8):

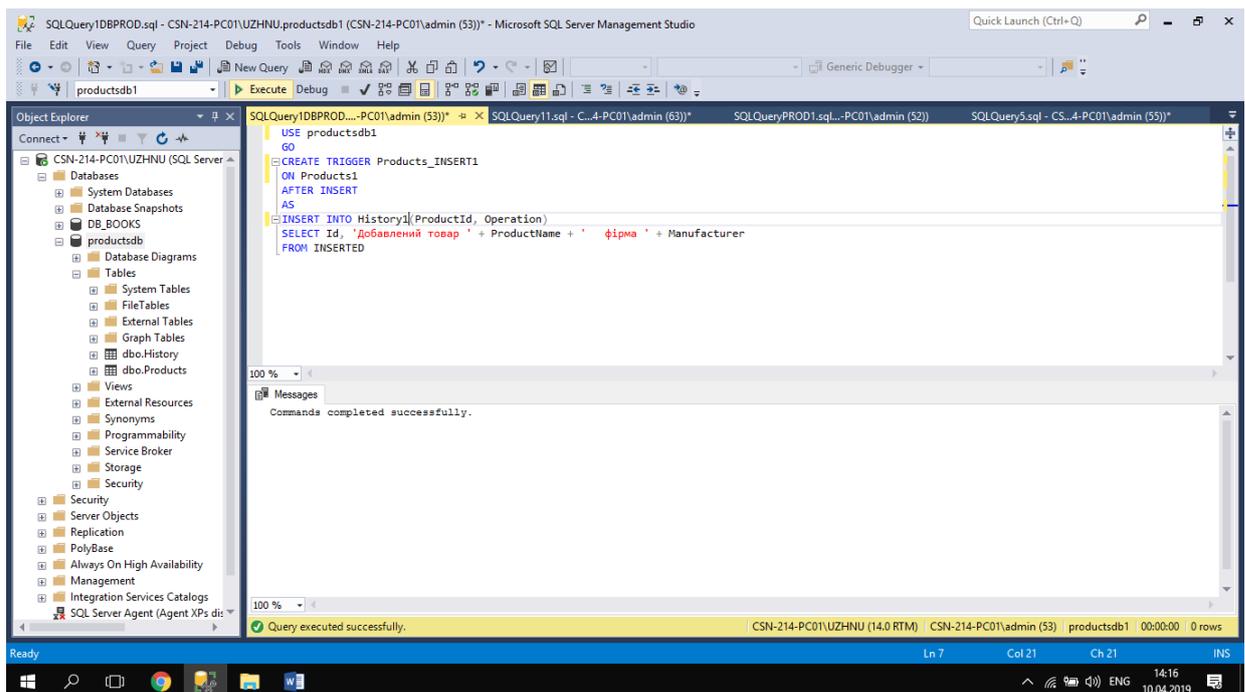


Рисунок 8 – Тригер додавання даних

Виконаємо додавання даних в *Products* і отримаємо дані з таблиці *History* (Див. рис. 9):

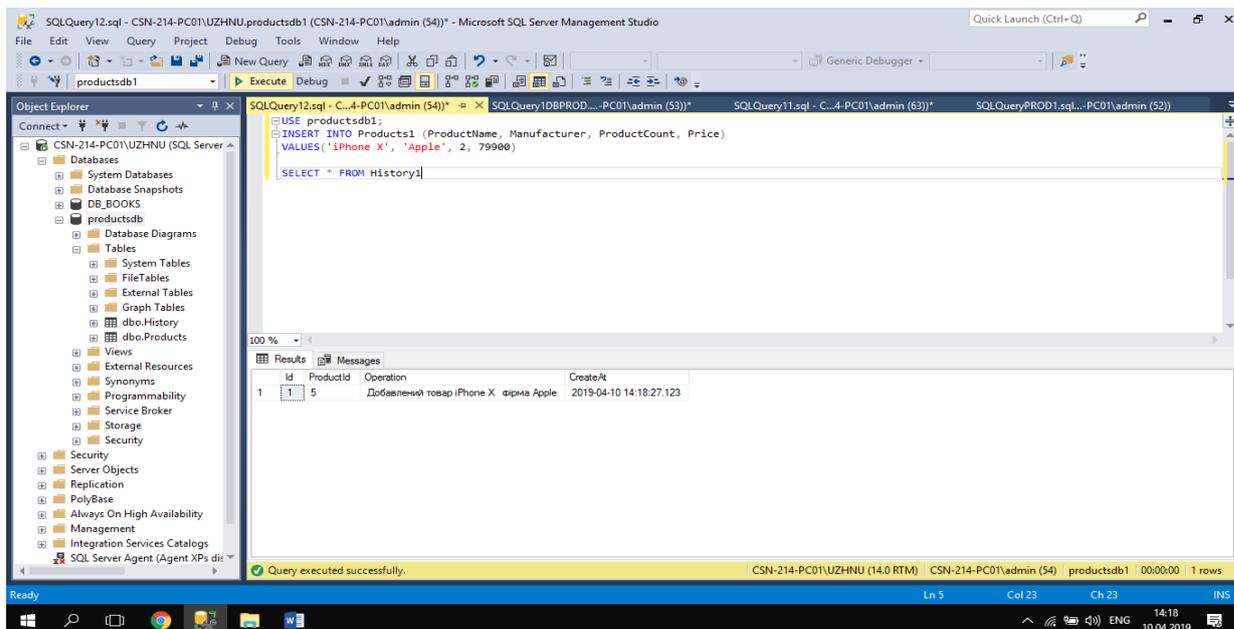


Рисунок 9 – Результат виконання додавання

Видалення даних.

При видаленні всі видалені дані містяться в віртуальну таблицю *DELETED*:

```
USE productsdb
1 GO
2 CREATE TRIGGER Products_DELETE
3 ON Products
4 AFTER DELETE
5 AS
6 INSERT INTO History (ProductId, Operation)
7 SELECT Id, 'Видалений товар ' + ProductName + ' фірма ' +
Manufacturer
8 FROM DELETED
```

Тут, як і в випадку з попереднім тригером, розмістимо інформацію про видалені товари в таблицю *History*.

Виконаємо команду на видалення (Див. рис. 10):

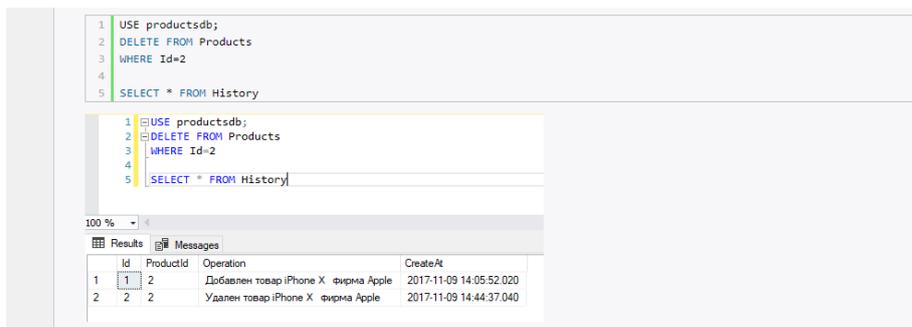


Рисунок 10 – Результат виконання видалення даних

Зміна даних.

Тригер поновлення даних спрацьовує при виконанні операції *UPDATE*. І в такому тригері ми можемо використовувати дві віртуальних таблиці. Таблиця *INSERTED* зберігає значення рядків після поновлення, а таблиця *DELETED* зберігає ті ж рядки, але до оновлення.

Створимо тригер поновлення:

```

1 USE productsdb
2 GO
3 CREATE TRIGGER Products_UPDATE
4 ON Products
5 AFTER UPDATE
6 AS
7 INSERT INTO History (ProductId, Operation)
8 SELECT Id, 'Оновлений товар ' + ProductName + ' фірма ' + Manufact
9 FROM INSERTED

```

І при оновленні даних спрацює даний тригер (Див. рис. 11):

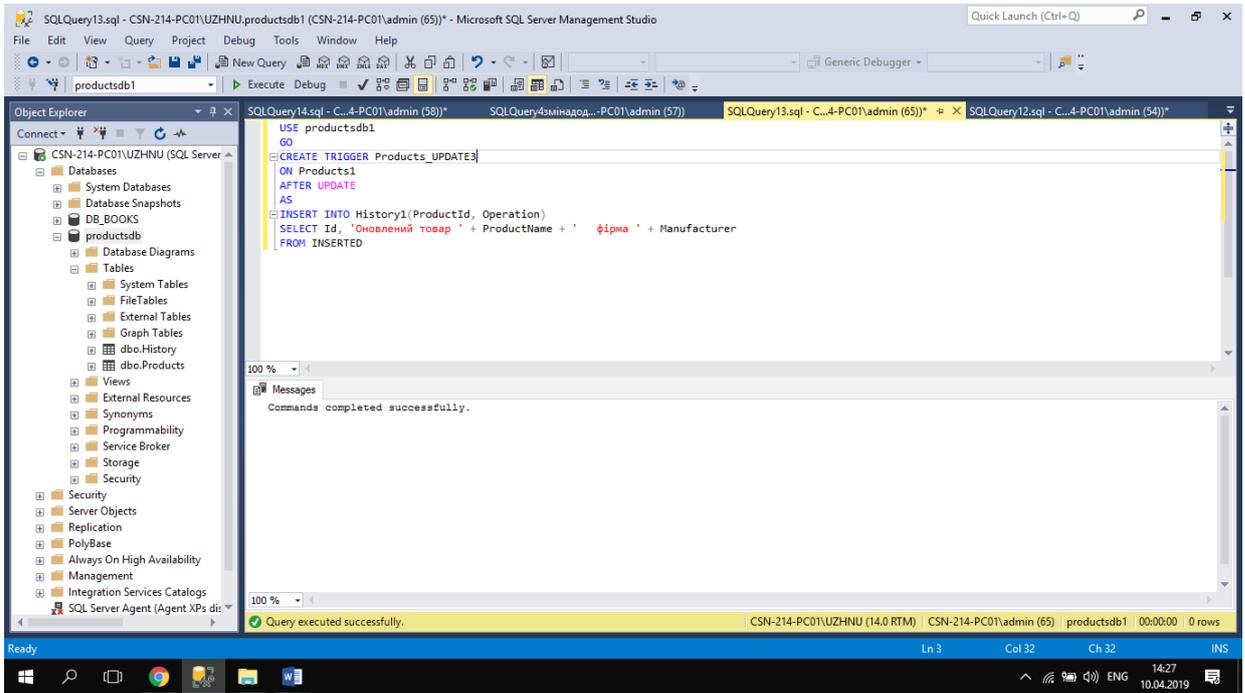


Рисунок 11 – Тригер поновлення даних

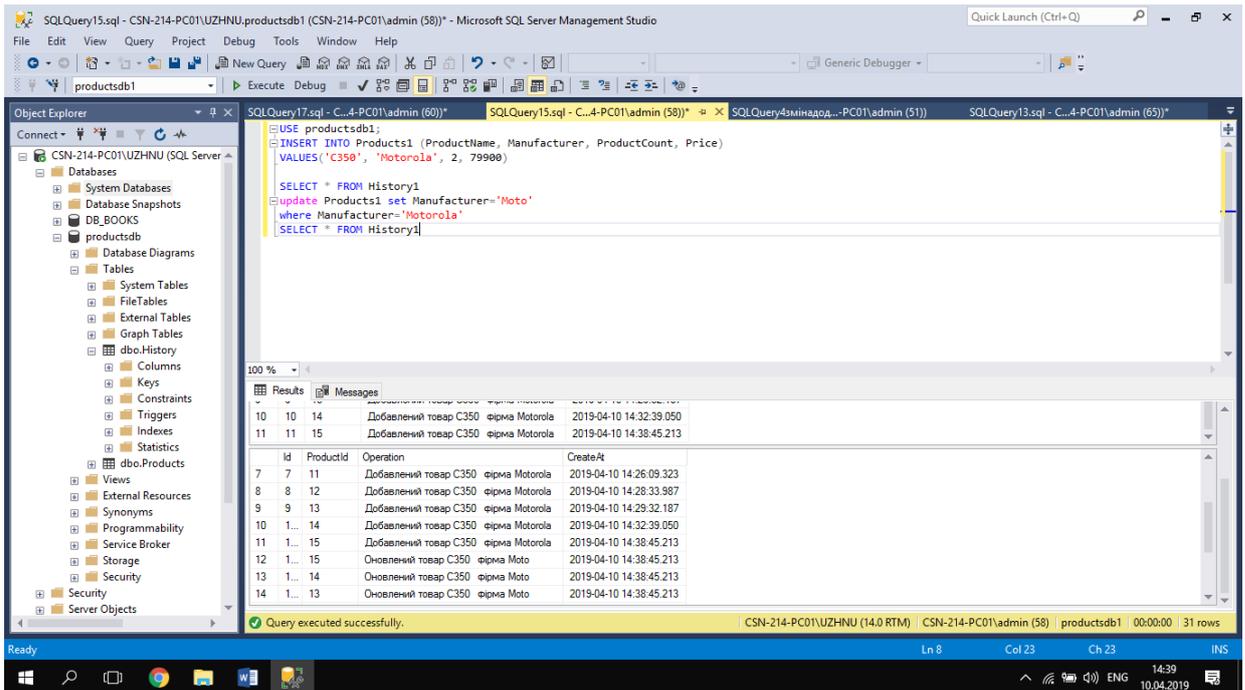


Рисунок 12 – Результат виконання тригера

Тригер *INSTEAD OF*.

Тригер *INSTEAD OF* спрацьовує замість операції з даними. Він визначається в принципі також, як тригер *AFTER*, за тим винятком, що він

може визначатися тільки для однієї операції – *INSERT*, *DELETE* або *UPDATE*. І також він може застосовуватися як для таблиць, так і для переглядів (тригер *AFTER* застосовується тільки для таблиць). Наприклад, створимо такі базу даних і таблицю:

```
1 CREATE DATABASE prods;
2 GO
3 USE prods;
4 CREATE TABLE Products
5 (
6     Id INT IDENTITY PRIMARY KEY,
7     ProductName NVARCHAR(30) NOT NULL,
8     Manufacturer NVARCHAR(20) NOT NULL,
9     Price MONEY NOT NULL,
10    IsDeleted BIT NULL
11 );
```

Тут таблиця містить стовпець *IsDeleted*, який вказує, чи повністю видалений запис. Тобто замість жорсткого видалення повністю з бази даних ми хочемо виконати м'яке видалення, при якому запис залишається в базі даних.

Визначимо тригер для видалення запису:

```
1 USE prods
2 GO
3 CREATE TRIGGER products_delete
4 ON Products
5 INSTEAD OF DELETE
6 AS
7 UPDATE Products
8 SET IsDeleted = 1
9 WHERE ID =(SELECT Id FROM deleted)
```

Додамо деякі дані в таблицю і виконаємо видалення з неї:

```
1 USE prods;
2
3 INSERT INTO Products(ProductName, Manufacturer, Price)
4 VALUES ('iPhone X', 'Apple', 79000),
5 ('Pixel 2', 'Google', 60000);
6
7 DELETE FROM Products
```

```

8 WHERE ProductName='Pixel 2';
9
10 SELECT * FROM Products;

```

Таким чином, записи, що видаляються, насправді не будуть видалятися, просто у них буде встановлюватися значення для стовпця *IsDeleted* (Див.рис. 13):

The screenshot shows a SQL query window with the following code:

```

1 USE prods;
2
3 INSERT INTO Products(ProductName, Manufacturer, Price)
4 VALUES ('iPhone X', 'Apple', 79000),
5 ('Pixel 2', 'Google', 60000);
6
7 DELETE FROM Products
8 WHERE ProductName='Pixel 2';
9
10 SELECT * FROM Products;

```

Below the query window, the 'Results' tab is active, displaying a table with the following data:

	Id	ProductName	Manufacturer	Price	IsDeleted
1	1	iPhone X	Apple	79000.00	NULL
2	2	Pixel 2	Google	60000.00	1

Рисунок 13 – Результат виконання запиту

Реалізація переглядів.

Перегляди або *Views* представляють віртуальні таблиці. Але на відміну від звичайних стандартних таблиць в базі даних перегляди містять запити, які динамічно витягають використовувані дані.

Перегляди дають нам ряд переваг. Вони спрощують комплексні *SQL*-операції. Вони захищають дані, так як перегляди можуть дати доступ до частини таблиці, а не до всієї таблиці. Перегляди також дозволяють повертати відформатовані значення з таблиць в потрібній і зручній формі.

Для створення перегляду використовується команда *CREATE VIEW*, яка має наступну форму:

```

1 CREATE VIEW назва_перегляду [(поле_1, поле_2, ....)]
2 AS вираз_SELECT

```

Визначимо три зв'язані таблиці:

```
1 CREATE TABLE Products
2 (
3     Id INT IDENTITY PRIMARY KEY,
4     ProductName NVARCHAR(30) NOT NULL,
5     Manufacturer NVARCHAR(20) NOT NULL,
6     ProductCount INT DEFAULT 0,
7     Price MONEY NOT NULL
8 );
9 CREATE TABLE Customers
10 (
11     Id INT IDENTITY PRIMARY KEY,
12     FirstName NVARCHAR(30) NOT NULL
13 );
14 CREATE TABLE Orders
15 (
16     Id INT IDENTITY PRIMARY KEY,
17     ProductId INT NOT NULL REFERENCES Products(Id) ON DELETE CASCADE
18     CustomerId INT NOT NULL REFERENCES Customers(Id) ON DELETE CASCADE
19     CreatedAt DATE NOT NULL,
20     ProductCount INT DEFAULT 1,
21     Price MONEY NOT NULL
22 );
```

Тепер додамо в базу даних, в якій містяться дані таблиці, такий перегляд:

```
1 CREATE VIEW OrdersProductsCustomers AS
2 SELECT Orders.CreatedAt AS OrderDate,
3         Customers.FirstName AS Customer,
4         Products.ProductName As Product
5 FROM Orders INNER JOIN Products ON Orders.ProductId = Products.Id
6 INNER JOIN Customers ON Orders.CustomerId = Customers.Id
```

Тобто даний перегляд фактично буде повертати зведені дані з трьох таблиць. І після його створення ми зможемо його побачити в вузлі *Views* у обраної бази даних в *SQL Server Management Studio* (Див. рис. 14):

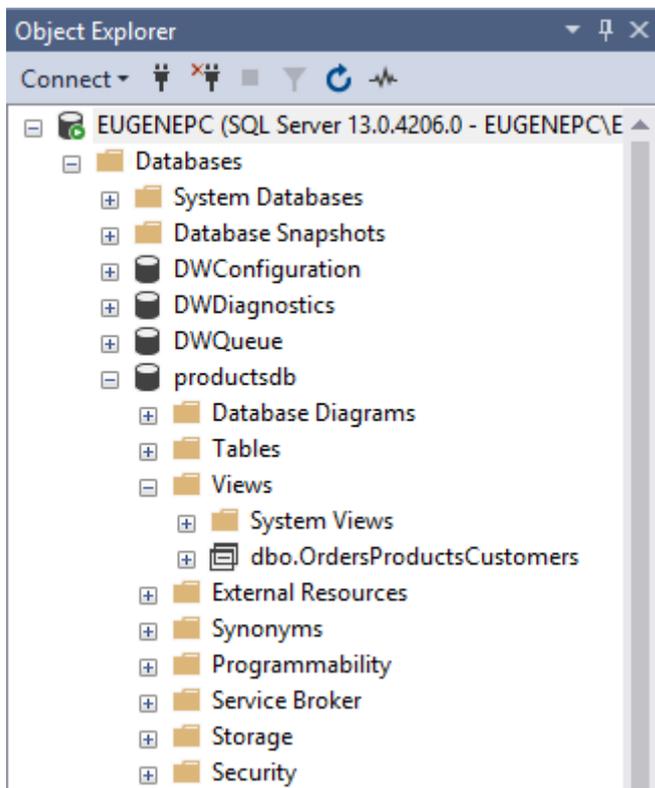


Рисунок 14 – Вікно *SQL Server Management Studio*

Тепер використовуємо створений вище перегляд для отримання даних (Див. рис. 15):

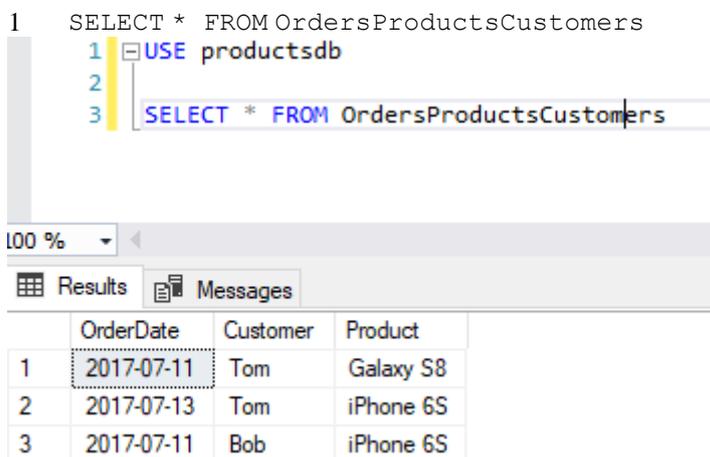


Рисунок 15 – Використання створеного перегляду

При створенні переглядів слід враховувати, що перегляди, як і таблиці, повинні мати унікальні імена в рамках тієї ж бази даних.

Перегляди можуть мати не більше 1024 стовпців і можуть звертатися не більше, ніж до 256 таблиць.

Також можна створювати перегляди на основі інших переглядів. Такі перегляди ще називають вкладеними (*nested views*). Однак рівень вкладеності не може бути більше 32-х.

Команда *SELECT*, яка використовується в перегляді, не може включати вирази *INTO* або *ORDER BY* (за винятком тих випадків, коли також застосовується вираз *TOP* або *OFFSET*). Якщо ж необхідне сортування даних в перегляді, то вираз *ORDER BY* застосовується в команді *SELECT*, яка отримує дані з переглядів.

Також при створенні перегляду можна визначити набір його стовпців:

```
1 CREATE VIEW OrdersProductsCustomers2 (OrderDate, Customer, Product)
2 AS SELECT Orders.CreatedAt,
3           Customers.FirstName,
4           Products.ProductName
5 FROM Orders INNER JOIN Products ON Orders.ProductId = Products.Id
6 INNER JOIN Customers ON Orders.CustomerId = Customers.Id
```

Питання для самоконтролю:

1. Що таке тригер, яке його призначення?
2. Параметри тригеру.
3. Для яких операцій визначається тригер?
4. Які операції не допускаються всередині тригеру?
5. Коли застосування тригеру недоцільно?
6. Яке призначення оператору *@@ROWCOUNT*?
7. Яке призначення оператору *ROLLBACK TRANSACTION*?
8. Яке призначення функції *COLUMNS_UPDATED* ()?
9. Як видалити тригер?
10. Що таке перегляд, яке його призначення?
11. Які переваги дають перегляди?
12. Як створити та видалити перегляд?

Лабораторна робота №6

Тема: Збережені процедури серверу.

Мета: засвоїти основні операції із збереженими процедурами серверу.

Основні теоретичні відомості.

Створення та виконання процедур.

Нерідко операція з даними представляє набір інструкцій, які необхідно виконати в певній послідовності. Наприклад, при додаванні купівлі товару необхідно внести дані в таблицю замовлень. Однак перед цим треба перевірити, а чи є товар, що купується, в наявності. Можливо, при цьому знадобиться перевірити ще ряд додаткових умов. Тобто фактично процес покупки товару охоплює кілька дій, які повинні виконуватися в певній послідовності. І в цьому випадку більш оптимально буде додати всі ці дії в один об'єкт – збережену процедуру (*stored procedure*).

Тобто по суті збережена процедура – це набір інструкцій, які виконуються як єдине ціле. Тим самим збережені процедури дозволяють спростити комплексні операції і винести їх в єдиний об'єкт. Чи зміниться процес покупки товару, відповідно досить буде змінити код процедури. Тобто процедура також спрощує управління кодом.

Також збережені процедури дозволяють обмежити доступ до даних в таблицях і тим самим зменшити ймовірність навмисних або неусвідомлених небажаних дій щодо цих даних.

І ще один важливий аспект – продуктивність. Збережені процедури зазвичай виконуються швидше, ніж звичайні *SQL*-інструкції. Все тому, що код процедур компілюється один раз при першому її запуску, а потім зберігається в компільованій формі.

Для створення збереженої процедури застосовується команда *CREATE PROCEDURE* або *CREATE PROC*.

Таким чином, процедура, що зберігається, має три ключові особливості: спрощення коду, безпеку і продуктивність.

Наприклад, нехай в базі даних є таблиця, яка зберігає дані про товари:

```
1 CREATE TABLE Products
2 (
3     Id INT IDENTITY PRIMARY KEY,
4     ProductName NVARCHAR(30) NOT NULL,
5     Manufacturer NVARCHAR(20) NOT NULL,
6     ProductCount INT DEFAULT 0,
7     Price MONEY NOT NULL
8 );
```

Створимо збережену процедуру для отримання даних з цієї таблиці:

```
1 USE productsdb;
2 GO
3 CREATE PROCEDURE ProductSummary AS
4 SELECT ProductName AS Product, Manufacturer, Price
5 FROM Products
```

Оскільки команда *CREATE PROCEDURE* повинна викликатися в окремому пакеті, то після команди *USE*, яка встановлює поточну базу даних, використовується команда *GO* для визначення нового пакету.

Після імені процедури повинно йти ключове слово *AS*.

Для відділення тіла процедури від іншої частини скрипту код процедури нерідко розміщується в блок *BEGIN ... END*:

```
1 USE productsdb;
```

```
2 GO
3 CREATE PROCEDURE ProductSummary AS
4 BEGIN
5     SELECT ProductName AS Product, Manufacturer, Price
6     FROM Products
7 END;
```

Після додавання процедури ми її можемо побачити в вузлі бази даних в *SQL Server Management Studio* в підвузлі *Programmability* -> *Stored Procedures*.

Скористаємося створеною процедурою, застосовуючи команду *EXEC* (Див. рис. 16):

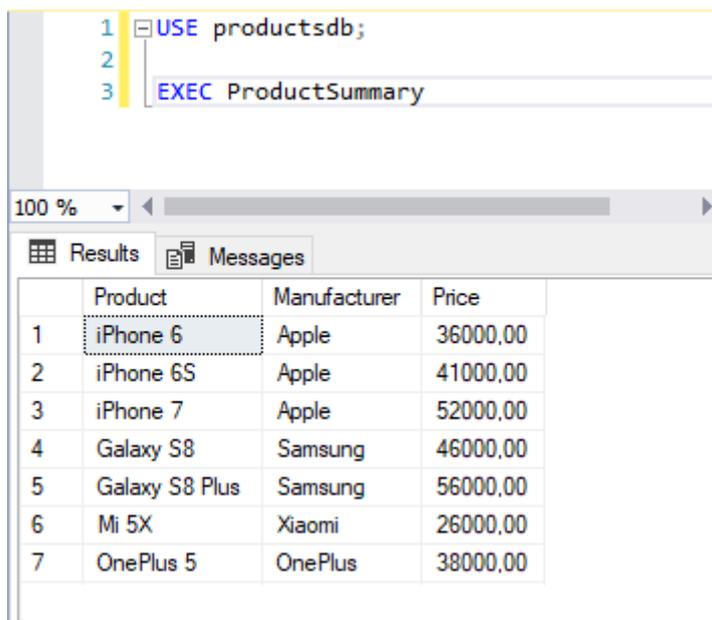


Рисунок 16 – Результат використання процедури

І ми зможемо керувати процедурою також і через візуальний інтерфейс.

Виконання процедури.

Для виконання процедури викликається команда *EXEC* або *EXECUTE*:

```
1 EXEC ProductSummary
```

Видалення процедури.

Для видалення процедури застосовується команда *DROP PROCEDURE*:

```
1 DROP PROCEDURE ProductSummary
```

Питання для самоконтролю:

1. Що таке збережена процедура сервера, її призначення та особливості?
2. Як створити, виконати та видалити збережену процедуру серверу?

Список використаних літературних джерел:

1. Пасічник В.В., Резніченко В. А. Організація баз даних та знань.—К. : Видавнича група «ВНУ», 2006. —384 с.
2. Базы даних в інформаційних системах : підруч. / В. І. Гайдаржи, І. В. Ізварін. - К. : Ун-т Україна, 2018. - 418 с.
3. Балик Н.Р., Мандзюк В.І. Базы даних MySQL: Навчальний посібник. — Тернопіль: «Навчальна книга – Богдан», 2010.— 160 с.
4. Берко А.Ю., Верес О.М., Пасічник В.В. Системи баз даних та знань. Книга 1. Організація баз даних та знань. – «Комп'ютинг», 2006. – 460с.
5. Берко А.Ю., Верес О.М., Пасічник В.В. Системи баз даних та знань. Книга 2. Організація баз даних та знань. – «Комп'ютинг», 2006. – 590с.
6. Коннолли Т., Бегг К., Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание. : Пер. с англ. — М.: Издательский дом «Вильямс», 2016. — 1440с.

Гарнітура Times New Roman. Папір офсетний.
Формат видання 60x84/16.
Умов. друк. арк.3.54 Зам. № 22. Наклад 50 прим.

Видруковано ПП «АУТДОР - ШАРК»
88000, м. Ужгород, пл. Жупанатська, 15/1. Тел.: 3-51-25. Е-mail: office@shark.com.ua

*Свідоцтво про внесення суб'єкта видавничої справи
до державного реєстру видавців,
виготівників і розповсюджувачів
видавничої продукції*

Серія 3т № 40 від 29 жовтня 2012 року