МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий Інститут телекомунікаційних систем

Затверджено
вченою радою НН ІТС
протокол № _4___
від “ 24” квітня 2023 року

Бази даних

Методичні вказівки
до виконання комп‘ютерного практикуму
для студентів спеціальності
"172 - Електронні комунікації та радіотехніка"

Затверджено Вченою радою НН ІТС КПІ ім. Ігоря Сікорського

Київ – 2023

Бази даних [Текст]: метод. вказівки до виконання комп‘ютерного практикуму для студентів спеціальності "Електронні комунікації та радіотехніка " / Уклад.: Суліма С.В., Глоба Л.С., Скулиш М.А.. – К.: КПІ ім. Ігоря Сікорського, 2023. – 54 с.

Гриф надано Вченою радою НН ІТС КПІ ім. Ігоря Сікорського
(Протокол № 4 від 24.04.2023)

Навчальне видання

Бази даних

Методичні вказівки
до виконання комп‘ютерного практикуму
для студентів спеціальності
"Електронні комунікації та радіотехніка"

Укладачі: *Суліма Світлана Валеріївна*, канд. техн. наук.
Глоба Лариса Сергіївна, докт. техн. наук, проф.
Скулиш Марія Анатоліївна, докт. техн. наук, проф.

Відповідальний редактор: *С.В. Суліма*, канд. техн. наук

Рецензенти: *С.О. Кравчук*, докт. техн. наук, проф.

За редакцією укладачів
Надруковано з оригінал-макета замовника

Зміст

Вступ

Сучасні вимоги до інженерів з телекомунікацій ґрунтуються на глибоких науково-технічних знаннях та вміннях розв’язувати практичні задачі, в тому числі задачі щодо створення та роботи з базами даних, а саме підходів до розробки схем баз даних, створення таблиць, роботи з засобами занесення та отримання знань такими як форми, запити, звіти.

Метою методичних вказівок є надання допомоги студентам в отриманні практичних навичок роботи з системами керування базами даних. Виконання лабораторних робіт забезпечить закріплення теоретичного та лекційного матеріалу.

Під час виконання лабораторних робіт студенти отримають навички щодо:

- підходів до проектування баз даних;
- створення таблиць;
- створення запитів на отримання даних з таблиць
- .

Загальні методичні вказівки

1. У кожній лабораторній роботі визначено: мету роботи, рекомендації з підготовки до роботи, програму і порядок її виконання.

2. Напередодні кожної лабораторної роботи необхідно:

– вивчити теоретичний матеріал зазначений у розділі "Підготовка до лабораторної роботи";

– усвідомити мету, зміст і порядок виконання;

– у лабораторії перевірити наявне лабораторне устаткування.

3. До виконання лабораторної роботи допускаються тільки підготовлені студенти після тестування (усного чи письмового), що проводиться викладачем перед початком виконання роботи.

Лабораторні роботи виконуються самостійно кожним студентом.

Звіт з роботи акуратно оформлюється і подається під час захисту лабораторної роботи. Схеми, зображені в звіті, мають відповідати вимогам стандартів.

4. Студенти, відсутні на заняттях, виконують роботу у час, погоджений з викладачем і інженером лабораторії після тестування.

5. Перед початком робіт кожному студенту необхідно вивчити правила техніки безпеки, здати залік, за що розписатися в журналі.

Вимоги до оформлення звіту з лабораторної роботи

Звіт оформлюється на аркушах формату А4 і повинен містити:

1. Титульну сторінку з номером та назвою лабораторної роботи, прізвищем студента, номером групи.
2. Особливості завдання.
3. Опис структури бази даних та виконаного завдання.
4. Висновки по роботі.

Інструкція з техніки безпеки

1. Вимоги з техніки безпеки перед початком роботи
   1.1 Провести огляд з зовні електророзеток, шнурів, вилок підключення до мережі живлення та заземлення (занулення).
   1.2 Забороняється працювати на несправному устаткуванні.
   1.3 За необхідності отримати додаткове устаткування у викладача та перевірити його справність.

2. Вимоги з техніки безпеки під час виконання робіт
   2.1 Необхідно виконувати лише ту роботу, з якої був проведений інструктаж, забороняється передоручати свою роботу іншим особам.
   2.2 Забороняється:
   - експлуатація кабелів та проводів з пошкодженою ізоляцією або такою, що втратила захисні властивості за час експлуатації;
   - залишати під напругою кабелі та проводи з неізольованими провідниками;
   - застосовувати саморобні подовжувачі, що не відповідають вимогам ПВЕ для переносних електропровідників;
   - користуватися пошкодженими розетками, розгалужувальними та з'єднувальними коробками, вимикачами та іншими електровиробами, а також лампами, скло яких має слід затемнення або випинання;
   - використовувати електроустаткування та прилади в умовах, що не відповідають інструкції з експлуатації підприємств-виробників;
   - залишати пристрої, що працюють без нагляду на тривалий час;
   - переносити пристрої, що підключені до електромережі;
   - забороняється самостійно ремонтувати апаратуру;
   - класти будь-які предмети на апаратуру комп'ютера, напої на клавіатуру або поруч з нею - це може вивести їх з ладу.

3. Вимоги з техніки безпеки після закінчення роботи

3.1. Зберегти необхідні файли на жорсткий диск комп'ютера або на переносний носій.

3.2 Виключити комп'ютер.

3.3. Виключити додаткове устаткування та віддати його викладачу.

3.4 Прибрати робоче місце.

Лабораторна робота №1
Тема: Знайомство з мовою SQL. Створення, редагування і видалення баз даних та таблиць

I. Підготовка до лабораторної роботи

Ознайомитись з теоретичною та практичною частиною наведеною нижче для виконання даної лабораторної роботи

II. Теоретичні відомості

1. Запис SQL-операторів

Для успішного вивчення мови SQL необхідно привести короткий опис структури SQL-операторів і нотації, які використовуються для визначення формату різних конструкцій мови. Оператор SQL складається із зарезервованих слів, а також зі слів, визначених користувачем. Зарезервовані слова є постійною частиною мови SQL і мають фіксоване значення. Їх слід записувати в точності так, як це встановлено, не можна розбивати на частини для переносу з одного рядка на іншу. Слова, що визначаються користувачем, задаються їм самим (відповідно з синтаксичними правилами) і являють собою ідентифікатори або імена різних об'єктів бази даних. Слова в операторі розміщуються також відповідно до встановлених синтаксичними правилами.

Ідентифікатори мови SQL призначені для позначення об'єктів в базі даних і є іменами таблиць, представлень, стовпців та інших об'єктів бази даних. Символи, які можуть використовуватися в створюваних користувачем ідентифікаторах мови SQL, повинні бути визначені як набір символів. Стандарт SQL задає набір символів, який використовується за умовчанням, - він включає великі та малі літери латинського алфавіту (A-Z, a-z), цифри (0-9) і символ підкреслення (_). На формат ідентифікатора накладаються наступні обмеження:

- ідентифікатор може мати довжину до 128 символів;
- ідентифікатор повинен починатися з літери;
- ідентифікатор не може містити пробіли.

Більшість компонентів мови не чутливі до регістру. Оскільки у мови SQL вільний формат, окремі SQL-оператори та їх послідовності будуть мати більш зручний вигляд при використанні відступів та вирівнювання.

Мова, в термінах якого дається опис мови SQL, називається метамовою.

Синтаксичні визначення зазвичай задають за допомогою спеціальної символіки, формулами Бекуса-Науер (БНФ). Великі літери використовуються для запису зарезервованих слів і повинні вказуватися в операторах точно так, як це буде показано. Малі букви вживаються для запису слів, що визначаються користувачем. Застосовувані в нотації БНФ символи та їх позначення показані в таблиці.

Символ	Позначення
::=	Дорівнює по визначенню
|	Необхідність вибору одного з декількох наведених значень
<…>	Описана за допомогою метамови структура мови
{…}	Обов'язковий вибір деякої конструкції зі списку
[…]	Необов'язковий вибір деякої конструкції зі списку
[,…n]	Необов'язкова можливість повторення конструкції від нуля до декількохразів

2. Створення бази даних

Процес створення бази даних у системі SQL-сервера складається з двох етапів: спочатку організується сама база даних, а потім журнал транзакцій. Інформація розміщується у відповідних файлах, що мають розширення *.mdf (для бази даних) та *.ldf. (для журналу транзакцій). У файлі бази даних записуються відомості про основні об'єкти (таблицях, індексах, переглядах і т.д.), а у файлі журналу транзакцій - про процес роботи з транзакціями (контроль цілісності даних, стану бази даних до і після виконання транзакцій).

Створення бази даних у системі SQL-сервер здійснюється командою CREATE DATABASE. Слід зазначити, що процедура створення бази даних в SQL-сервері вимагає наявності прав адміністратора сервера.

Синтаксис:

<визначення бази даних> ::=
CREATE DATABASE ім’я_бази_даних
[ON [PRIMARY]
[<визначення_файлу> [,…n]]
[,<визначення_файлу> [,…n]]]
[LOG ON {< визначення_файлу>[,…n] }]
[FOR LOAD | FOR ATTACH]

При виборі імені бази даних слід керуватися загальними правилами іменування об'єктів. Якщо ім'я бази даних містить пробіли або будь-які інші неприпустимі символи, вони заключаються в обмежувачі (подвійні лапки або квадратні дужки). Ім'я бази даних повинне бути унікальним у межах сервера і не може перевищувати 128 символів.

При створенні і зміні бази даних можна вказати ім'я файлу, який буде для неї створено,змінити ім'я, шлях і вихідний розмір цього файлу. Якщо в процесі використання бази даних планується її розміщення на кількох дисках, то можна створити так звані вторинні файли бази даних з розширенням *.ndf. У цьомувипадку основна інформація про базу даних розташовується в первинному (PRIMARY) файлі, а при нестачі для нього вільного місця додається інформація буде розміщуватися у вторинному файлі. Підхід, який використовується в SQL-сервері, дозволяє розподіляти вміст бази даних по декількох дискових томах.

Параметр ON визначає список файлів на диску для розміщення інформації, що зберігається в базі даних.

Параметр PRIMARY визначає первинний файл. Якщо він пропущений, то первинним єперший файл у списку.

Параметр LOG ON визначає список файлів на диску для розміщення журналу транзакцій. Файл для журналу транзакцій генерується на основі імені бази даних, і врешті до нього додаються символи _log.

При створенні бази даних можна визначити набір файлів, з яких вона складатиметься. Файл визначається за допомогою наступної конструкції:

<визначення_файлу>::=
([NAME=логічне_ім’я_файлу,]
FILENAME=’фізичне_ім’я_файлу’
[,SIZE=розмір_файлу]
[,MAXSIZE={max_розмір_файлу |UNLIMITED}]
[, FILEGROWTH=величина_приросту])[,...n]

Тут логічне ім'я файлу - це ім'я файлу при виконанні різних SQL-команд.

Фізичне ім’я файлу призначено для вказівки повного шляху і назви відповідного фізичного файлу, який буде створений на свій диск. Це ім'я залишиться за файлом на рівні операційної системи.

Параметр SIZE визначає початковий розмір файлу; мінімальний розмір параметра - 512Кб, якщо він не вказаний, за замовчуванням приймається 1 Мб.

Параметр MAXSIZE визначає максимальний розмір файлу бази даних. При значенні параметра UNLIMITED максимальний розмір бази даних обмежується вільним місцем на диску.

При створенні бази даних можна дозволити або заборонити автоматичне зростання їїрозміру (це визначається параметром FILEGROWTH) і вказати прирощення за допомогою абсолютної величини в Мб або процентним співвідношенням. Значення може бути вказано в кілобайтах, мегабайтах, гігабайтах, терабайт або відсотках (%). Якщо вказано число без суфікса МБ, КБ або %, то за умовчанням використовується значення МБ. Якщо розмір кроку зростання зазначений у відсотках (%), розмір збільшується на задану частину у відсотках від розміру файлу. Зазначений розмір округлюється до найближчих 64 КБ.

Приклад 1. Створити базу даних, причому для даних визначити два файли на диску D,для журналу транзакцій - один файл на диску D.

```
CREATE DATABASE Archive
ON PRIMARY ( NAME=Arch1,
  FILENAME='d:\archdat1.mdf',
SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Arch2,
  FILENAME='d:\archdat2.mdf',
  SIZE=100MB, MAXSIZE=200, FILEGROWTH=20)
 LOG ON
(NAME=Archlog1,
  FILENAME='d:\archlog1.ldf',
    SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
 (NAME=Archlog2,
   FILENAME='d:\archlog2.ldf',
     SIZE=100MB, MAXSIZE=200, FILEGROWTH=20)
```

3. Зміна бази даних

Більшість дій щодо зміни конфігурації бази даних виконується за допомогою наступної конструкції:

```
        <зміна_бази_даних> ::=
 ALTER DATABASE ім’я_бази_даних
{ ADD FILE <визначення_файла>[,...n]
[TO FILEGROUP ім’я_групи_файлів ]
| ADD LOG FILE <визначення_файла>[,...n]
| REMOVE FILE логічне_ім’я_файла
| ADD FILEGROUP ім’я_групи_файлів
| REMOVE FILEGROUP ім’я_групи_файлів
| MODIFY FILE <визначення_файла>
| MODIFY FILEGROUP ім’я_групи_файлів <властивості_групи_файлів>}
```

Як видно з синтаксису, за один виклик команди може бути змінено не більше одного параметра конфігурації бази даних. Якщо необхідно виконати декілька змін, доведеться розбити процес на ряд окремих кроків.

В базу даних можна додати (ADD) нові файли даних (у зазначену групу файлів абов групу, прийняту за замовчуванням) або файли журналу транзакцій.

Параметри файлів та груп файлів можна змінювати (MODIFY).

Для видалення з бази даних файлів або груп файлів використовується параметр REMOVE. Проте видалення файлу можливе лише за умовийого звільнення від даних. В іншому випадку сервер не дозволить видалення.

Як властивостей групи файлів використовуються наступні:

READONLY – група файлів використовується тільки для читання; READWRITE – в групі файлів дозволяються зміни; DEFAULT – зазначена група файлів приймається за замовчуванням.

4. Видалення бази даних

Видалення бази даних здійснюється командою:

```
DROP DATABASE ім’я_бази_даних [,...n]
```

Приклад:

```
DROP DATABASE Archive
```

5. Створення таблиці

Після створення загальної структури бази даних можна приступити до створення таблиць, які представляють собою відношення, що входять до складу проекту бази даних. Таблиця - основний об'єкт для зберігання інформації в реляційної базі даних. Вона складається з рядків і стовпців, займає в базі даних фізичний простір і може бути постійною або тимчасовою.

Поле, також зване в реляційної базі даних стовпцем, є частиною таблиці, за якою закріплений певний тип даних. Кожна таблиця бази даних повинна містити хоча б один стовпець. Рядок даних - це запис у таблиці бази даних, вона включає поля, що містять дані з одного запису таблиці.

Приступаючи до створення таблиці, необхідно мати відповіді на ряд питань:

- Як буде називатися таблиця?
- Як будуть називатися стовпці (поля) таблиці?
- Які типи даних будуть закріплені за кожним стовпцем?
- Який розмір пам'яті повинен бути виділений для зберігання кожного стовпця?
- Які стовпці таблиці вимагають обов'язкового введення?
- З яких стовпців складатиметься первинний ключ?

Базовий синтаксис оператора створення таблиці має наступний вигляд:

```
<визначення_таблиці> ::=
 [USE ім’я_бази_даних ]
  CREATE TABLE  ім’я_таблиці
   (ім’я_стовбця тип_даних
    [NULL | NOT NULL ] [,...n])
```

Головне в команді створення таблиці - визначення імені таблиці та опис набору імен полів, які зазначаються у відповідному порядку. Крім того, цією командою обумовлюються типи даних і розміри полів таблиці. Ключове слово USE вказує в якій базі даних буде створена таблиця.

Ключове слово NULL використовується для вказівки того, що в даному стовпці можуть міститися значення NULL. Значення NULL відрізняється від пробілу або нуля – до нього вдаються, коли необхідно вказати, що дані недоступні, опущені або неприпустимі. Якщо вказано ключове слово NOT NULL, то будуть відхилені будь-які спроби помістити значення NULL в даний стовпець. Якщо вказаний параметр NULL, розміщення значень NULL в стовпець дозволено. За замовчуванням стандарт SQL передбачає наявність ключового слова NULL.

Приклад: Створити таблицю для зберігання даних про товари, що надходять у продаж в деякій торговій фірмі. Необхідно врахувати такі відомості, як назва і тип товару, його ціна, сорт і місто, де товар виробляється.

```
USE Archive
CREATE TABLE Товар
(Название       VARCHAR(50) NOT NULL,
    Цена        MONEY NOT     NULL,
    Тип         VARCHAR(50) NOT NULL,
    Сорт        VARCHAR(50),
    ГородТовара VARCHAR(50))
```

6. Зміна таблиці

Структура існуючої таблиці може бути модифікована за допомогою команди ALTER TABLE

```
    [USE ім'я_бази_даних ]
ALTER TABLE ім'я_таблиці
  {[ALTER COLUMN ім'я_стовпця
  {новий_тип_даних [(точність[,масштаб])]
  [ NULL | NOT NULL ]}]
  | ADD { [ім'я_стовпця тип_даних]
  | ім'я_стовпця AS вираз } [,...n]
  | DROP {COLUMN ім'я_стовпця }[,...n]
}
```

Команда дозволяє додавати і видаляти стовпці, змінювати їх визначення.

Одне з основних правил при додаванні стовпців в існуючу таблицю говорить: коли в таблиці вже містяться дані, стовпець, що додається, не може бути визначений з атрибутом NOT NULL. Цей атрибут означає, що для кожного рядка даних відповідний стовпець повинен містити деякий значення, тому додавання стовпця з атрибутом NOT NULL призводить до появи протиріччя - вже існуючі рядки даних таблиці не будуть мати на новому стовпці ненульових значень.

Проте існує спосіб додавання обов'язкових полів в існуючу таблицю. Для цього необхідно:

- додати до таблиці новий стовпець, визначивши його з атрибутом NULL (тобто стовпець не зобов'язаний утримувати будь-яких значень);
- ввести в новий стовпець які-небудь значення для кожного рядка даних таблиці;
- переконавшись, що новий стовпець містить ненульові значення для кожного рядка даних, змінити структуру таблиці, замінивши атрибут цього стовпця на NOT NULL.

При зміні визначень стовпців слід брати до уваги деякі загальноприйняті правила:

- розмір стовпця може бути збільшений до максимального значення, що допускається відповідним типом даних;
- розмір стовпця може бути зменшений тільки в тому випадку, якщо міститься в ньому найбільше значення не буде перевершувати його нового розміру;
- кількість розрядів числового типу даних завжди може бути збільшено;
- кількість розрядів числового типу даних може бути зменшено лише в тому випадку, якщо кількість розрядів найбільшого значення в відповідному стовпці не буде перевершувати нового числа розрядів, визначеного для цього стовпця;
- кількість десяткових знаків числового типу даних може бути зменшено або збільшено;
- тип даних стовпця, як правило, може бути змінений.

Деякі реалізації фактично можуть обмежити розробника у використанні деяких опцій команди ALTER TABLE. Наприклад, може виявитися неприпустимим видалення стовпців з існуючої таблиці. Щоб добитися цього, спочатку потрібно видалити саму таблицю і тільки потім заново її побудувати з потрібними стовпцями. Причому вже внесені в таблицю дані будуть втрачені.

Можливі труднощі, пов'язані з видаленням з таблиці стовпця, який залежить від деякогостовпця іншої таблиці. У такому випадку спочатку доведеться видалити обмеження стовпця, а потім сам стовпець.

Приклад: Додати стовпець Дата Виготовлення с типом DateTime в таблицю Товар

```
USE Archive
ALTER TABLE Товар
ADD ДатаВиготовлення DateTime
```

7. Видалення таблиці

З плином часу структура бази даних змінюється: створюються нові таблиці, а колишністають непотрібними і видаляються з бази даних за допомогою оператора:

```
[USE імя_бази_даних]
DROP TABLE ім'я_таблиці [RESTRICT | CASCADE]
```

Слід зазначити, що ця команда видалить не тільки зазначену таблицю, а й усі вхідні в неїрядки даних. Якщо потрібно видалити з таблиці лише дані, зберігши структуру таблиці, слід скористатися командою DELETE.

Оператор DROP TABLE додатково дозволяє зазначено, чи операцію видалення виконувати каскадно. Якщо в операторі вказано ключове слово RESTRICT, то при наявності в базі даних хоча б одного об'єкта, існування якого залежить від таблиці, що видаляється, виконання оператора DROP TABLE буде скасовано. Якщо вказано ключове слово CASCADE, автоматично видаляються і всі інші об'єкти бази даних, чиє існування залежить від таблиці, що видаляється, а також інші об'єкти, що залежать від об'єктів, що видаляються. Загальний ефект від виконання оператора DROP TABLE з ключовим словом CASCADE може виявитися досить відчутним, тому подібні оператори слід використовувати з максимальною обережністю.

Найчастіше оператор DROP TABLE використовується для виправлення помилок, допущених при створенні таблиці. Якщо таблиця була створена з некоректною структурою, можна скористатися оператором DROP TABLE для її видалення, після чого створити таблицю заново.

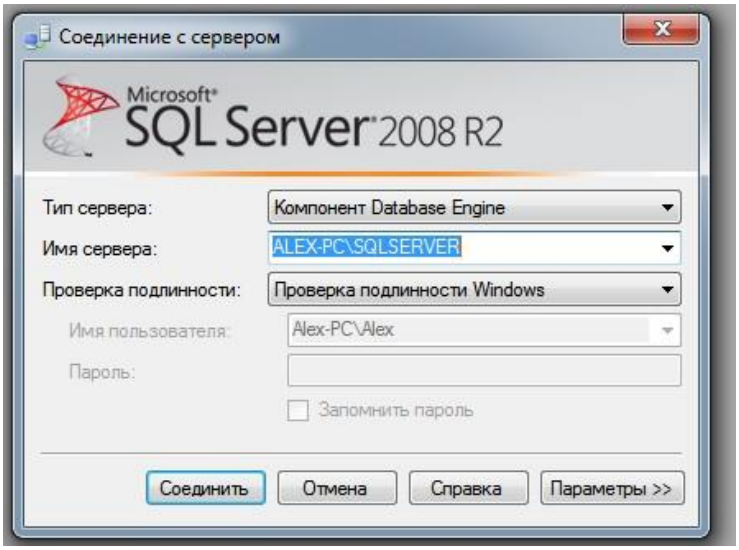
Приклад:

Видалити таблицю Товар

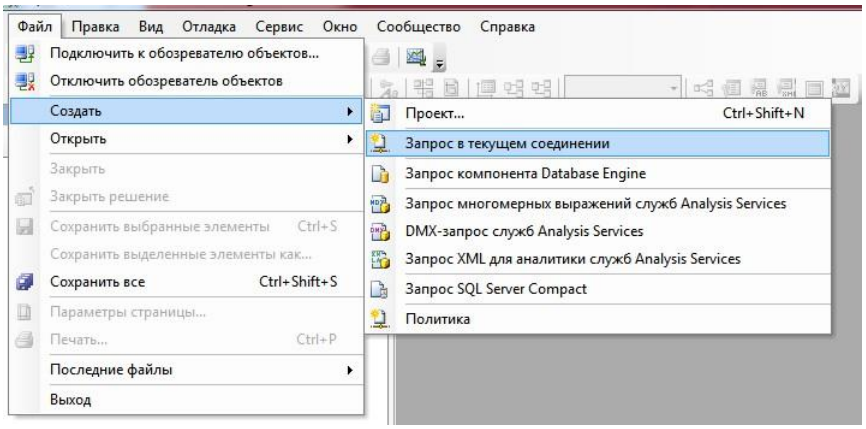
```
DROP TABLE Товар
```

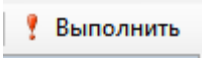
8. Початок роботи з MS SQL Server

1. Установіть Microsoft SQL Server 2005 або Microsoft SQL Server 2008.
2. Відкрийте середовище Microsoft SQL Server Management Studio.
3. Зайдіть під своїм іменем. наприклад:



4. Створіть новий запит. Для цього виберіть в меню Файл> Створити> Запит в поточному з'єднанні

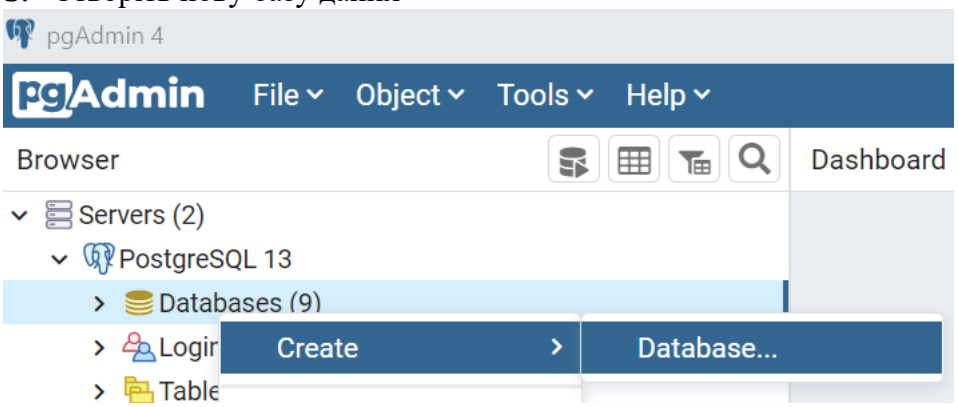


5. Напишіть і виконайте запит. Для цього виберіть на панелі інструментів Виконати 

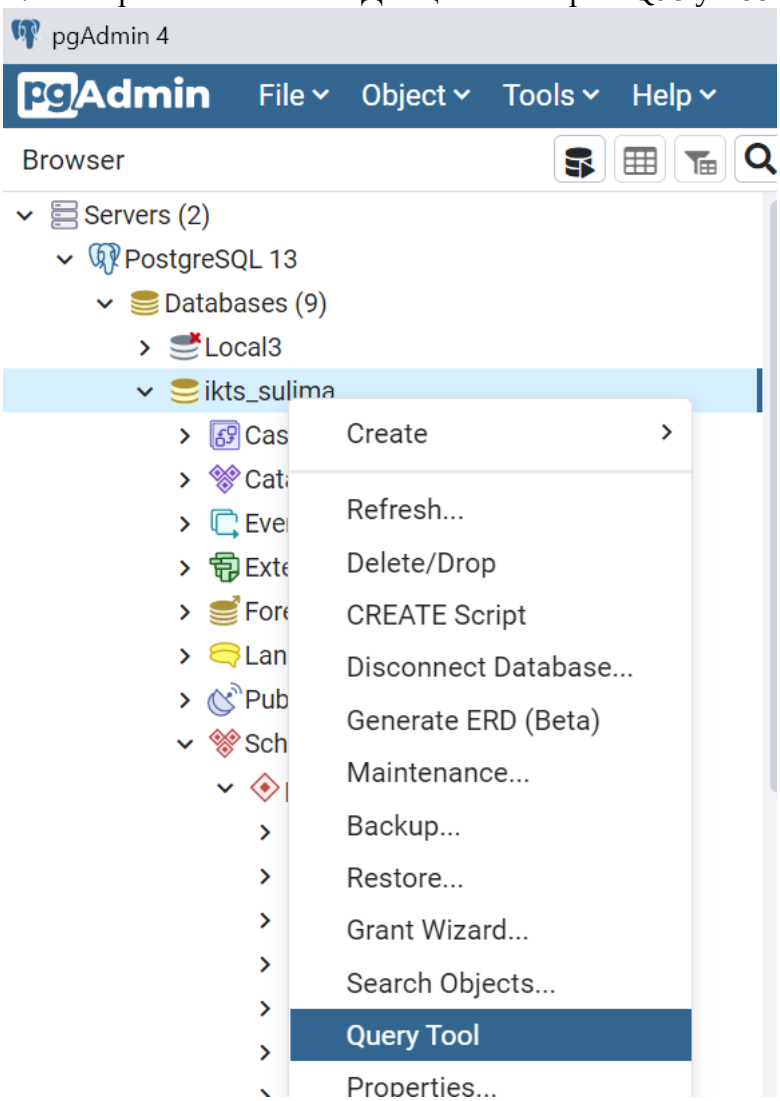
,або натисніть F5

9. Початок роботи з PostgreSQL

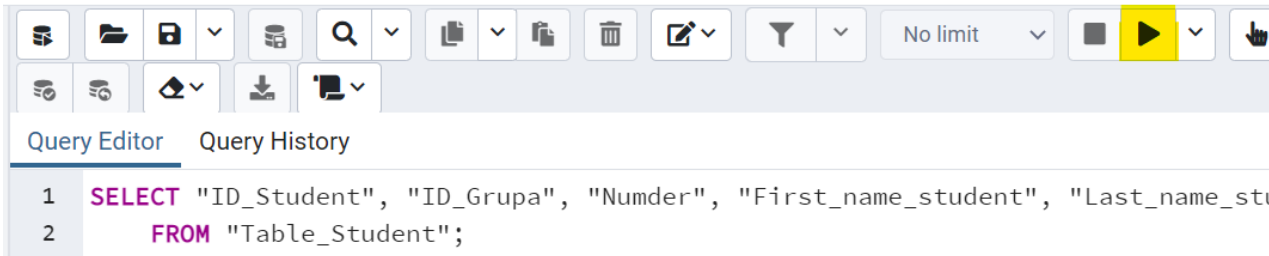
1. Установіть PostgreSQL https://www.enterprisedb.com/downloads/postgres-postgresql-downloads.
2. Відкрийте середовище pgAdmin.
3. Створіть нову базу даних



4. Створіть новий запит. Для цього виберіть Query Tool



5. Напишіть і виконайте запит. Для цього виберіть на панелі інструментів Виконати



10. Генерація рандомних даних

Як вставити випадкові дані для тестування в PostgreSQL?

Функція repeat()

Функція, яка приймає два параметри repeat (рядок для повторення, кількість разів для повторення) для повторення рядка.

Приклад:

INSERT INTO test (stringColumn) VALUES (repeat("Wow ", 3));

Буде вставлено рядок із stringColumn зі значенням Wow Wow Wow

Функція generate_series()

Функція, яка приймає два обов'язкові параметри та один необов'язковий generate_series(start, stop, increment by [default is 1]) для створення серії чисел, кожне в рядку.

Приклад:

INSERT INTO test (id) VALUES (generate_series(1, 1000));

Це вставить 1000 рядків кожен з ідентифікатором, що дорівнює попередньому ідентифікатору, збільшеному на одиницю.

Функція random()

Функція, яка генерує випадкове число з плаваючою точкою від 0 до 1.

приклад:

INSERT INTO test (id) VALUES (trunc(random()*100));

Це створить ідентифікатор із трьох цифр від 0 до 100 (усікайте, щоб зробити число з плаваючою точкою цілим).

ВСЕ РАЗОМ:

Для прикладу створимо таблицю гравців:

CREATE TABLE players (id INT, about TEXT, age INT);

Тоді давайте вставимо 1000 рядків з деякими випадковими даними:

INSERT INTO players (id, about, age) VALUES (generate_series(1, 1000), repeat('A cool player. ', 2) || 'My number is ' || trunc(random()*1000), trunc(random()*10 * 2 + 10));

Ще приклади:

```
discuss=> select random(), random(), trunc(random()*100);
      random       |      random       | trunc
-------------------+-------------------+-------
 0.19253553216125816 | 0.751528221182525 |    91

discuss=> select repeat('Neon ', 5);
          repeat
---------------------------
 Neon Neon Neon Neon Neon

discuss=> select generate_series(1,5);
 generate_series
-----------------
               1
               2
               3
               4
               5
```

Висновки

Поняття	*Значення*
Стовпець	Список значень, котрі характеризуються певним типом.
Рядок	Об'єкт, що складається з набору значень стовпців; іноді називається також «кортежем».
Таблиця	Множина рядків з відповідними значеннями стовпців.
Первинний ключ	Унікальне значення, що визначає конкретний рядок.
CRUD	Create, Read, Update, Delete.
Індекс	Структура даних для оптимізації пошуку екземплярів даних таблиці за умовами до значень стовпців.

III. Завдання на лабораторну роботу

- Створити базу даних з назвою «Група_Прізвище».
- Створити таблиці бази даних, які характеризують навчальний процес, згідно з детальним описом таблиць див. Додаток А, Додаток Б.
- Освоїти засоби і правила коректування даних у таблицях. Наповнити всі таблиці даними. Мінімум 5 – 10 елементів.

IV. Контрольні питання

1. Опишіть процес створення нової БД.
2. Опишіть процес створення нової та редагування структури існуючої таблиці.
3. Що таке БД?
4. Як виникла реляційна модель даних?
5. Які переваги та недоліки реляційної моделі?
6. Які бувають інші типи БД?
7. Які переваги БД перед файловою системою?
8. Що таке СУБД? Які функції виконує СУБД?
9. У чому різниця між DML та DDL ?
10. Архітектура Client – Server. Які функції виконують Client та Server ?
11. Перелічіть основні цілі створення реляційної моделі БД.
12. Що таке відношення?
13. Назвіть та дайте визначення основним елементам відношення.
14. Перелічіть властивості відношення.
15. Назвіть типи ключів в реляційній моделі даних. Дайте визначення кожному з них. Наведіть приклади.
16. Що означає визначник NULL?
17. Що таке цілісність? Які існують види цілісності?

Лабораторна робота №2
Тема: Створення запитів у MS SQL Server

I. Підготовка до лабораторної роботи

Для підготовки до лабораторної роботи слід проробити відповідний теоретичний матеріал, що вказаний в цьому розділі.

Також для виконання цієї лабораторної роботи необхідною умовою є виконана лабораторна робота №1.

II. Теоретичні відомості

1. Запис операторів SQL

Оператори SQL складаються із зарезервованих слів, а також із слів, що визначаються користувачем. Зарезервовані слова є постійною частиною мови SQL і мають певне значення. Їх необхідно записувати лише так як вказано в стандарті і не можна розбивати на частини для переносу з одної строки в іншу. Слова, що визначаються користувачем, задаються самим користувачем (в відповідності до певного синтаксису) і представляють собою імена різних об'єктів баз даних – таблиць, представлень, індексів і т.д.

Оскільки мова SQL має вільний формат, певні оператори SQL і їх послідовності будуть мати більш зручний для читання вигляд, якщо використовувати відступи і вирівнювання. Рекомендується також притримуватись наступних правил:

- Кожна конструкція в операторі повинна починатись з нової строки
- Початок кожної конструкції необхідно позначати таким ж відступом як і інші конструкції оператора
- Якщо конструкція складається з декількох частин, то кожна з них повинна починатись з нової строки з деяким відступом відносно початку конструкції, що вказуватиме на їх підлеглість
- Прописні букви будуть використовуватись для запису зарезервованих слів і повинні вказуватись в операторах так само як це буде показано
- Строчні букви будуть використовуватись для запису слів, що використовуватимуться користувачем
- Вертикальна лінія (|) вказуватиме на необхідність вибору одного з декількох приведених значень
- Фігурні лапки позначатимуть обов'язкові елементи
- Квадратні лапки позначатимуть необов'язкові елементи
- (…) позначатимуть для того щоб вказати необов'язкової можливості повторення конструкції від 0 до декількох раз наприклад {a,b}[,c…] Такий запис означає що після a або b може бути від 0 додекількох повторень c, розділених комами.

2. Синтаксична організація інструкції запиту

В своїй базовій формі інструкція SELECT повідомляє серверу, які данні необхідно видати, а саме, які стовбці і строки із яких таблиць отримати і як від сортирувати данні

Нижче наведений прикладу прощеного синтаксису SELECT

```
SELECT * , стовбці і вирази
[FROM таблиця
JOIN таблиця ON умова]
[WHERE умова]
[ GROUP BY стовбці]
[HAVING умова]
```

[ORDER BY стовбці];

Інструкція SELECT починається зі списку стовбців або виразів. Обов'язкова присутність як мінімум хоча б 1 виразу, все решта є не обов'язковим.

Найпростіша із можливих інструкцій SELECT має вигляд:

```
SELECT * ;
```

Блок FROM в інструкції SELECT збирає всі потрібні данні в один набір, над яким буде працювати решта частини інструкції. В блоці FROM може брати участь багато таблиць які посилаються одна на одну за допомогою різних типів з'єднань.

Блок WHERE фільтрує набір даних, що їх зібрав FROM, на основі деяких умов. Агрегатні функції виконують в наборі даних групові розрахунки. Блок GROUP BY перетворює велику множину на маленькі підмножини на основі значень стовбців, що згадані в цьому виразі. Після цього агрегатні функції приміняться до цих невеликих підмножин. Після чого результати агрегатних функцій фільтруються застосуванням HAVING.

І нарешті ORDER BY упорядковує результуючий набір даних.

3. Іменовані діапазони

Будь-якій таблиці в блоці FROM може бути присвоєний іменований діапазон, або псевдонім. Як тільки у таблиці появляється псевдонім, по ньому до неї можна звертатись в інших блоках інструкції SELECT. Ключове слово AS є необов'язковим, і часто про нього забувають. В наступному прикладі доступ здійснюється до таблиці Guide, але звертання до неї проходить по псевдоніму G:

```
FROM таблиця [AS] псевдонім\\
USE Cha2
SELECT G.FirstName , G.LastName
FROM GUIDE AS G
```

4. Умова WHERE

Умова WHERE фільтрує набір даних, який сформований блоком FROM, і вибирає з цього набору лише ті строки, які задовольняють умові. Умови можуть посилатись на дані в таблицях, вирази, вбудовані скалярні функції SQL і користувацькі функції. В умові WHERE можуть також використовуватись оператори порівняння і символи макропідстановки.

5. Використання умови BETWEEN

Умова BETWEEN перевіряє значення на його приналежність деякому діапазону. В даному випадку діапазон включає граничні значення. Наприклад: умова between 1 and 10 буде істинною для чисел 1 і 10

Найчастіше умову BETWEEN застосовують разом з датою.

```
USE Cha2
SELECT  EventCode, DateBeginFROM dbo.Event
WHERE DateBegin BETWEEN '01/01/11' AND '05/05/200
```

6. Визначення порядку сортування за допомогою імен стовбців

Найпростішим способом сортування результуючого набору даних є явно заданий порядок слідування стовбців, по яким відбувається упорядкування

```
USE Cha2
SELECT FirstName, LastName
```

FROM dbo.Customer
ORDER BY LastName, FirstName;
Результат буде мати вигляд:

First Name	LastName
Joe	Adams
Missy	Anderson
Debbie	Bettys

…

7. SELECT, DISTINCT

Першим предикатом, що використовується в поєднанні з ключовим словом SELECT, є DISTINCT. Він виключає дублювання даних запиту. Ці дублювання оцінюються на рівні стовбців результуючого набору даних а не початкових таблиць. Протилежну функцію виконує предикат ALL. Так як він використовується за замовчуванням, в запитах його зазвичай ігнорують.

8. Підзапити

Підзапитом називають вставлену у зовнішній запит інструкцію SELECT. Підзапити дають відповідь на запит зовнішнього запиту, у вигляді скалярного значення, списку значень , або набору даних. Підзапити можуть заміняти собою вирази, списки і таблиці в структурі зовнішнього запиту.

9. Основні підсумкові функції

Мова SQL містить велику множину підсумкових функцій, які можна використовувати в якості виразів в інструкції SELECT для отримання підсумкових даних

Функції, що найчастіше використовуються:

SUM()
COUNT()
AVG()
MIN()
MAX()

10. Запити на оновлення Update

Інструкція UPDATE мови SQL надзвичайно проста. Вона здатна замінити значення всього однієї комірки, а також всіх стовпців всіх строк таблиць.

Структура інструкції UPDATE:

UPDATE dbo.[Назва таблиці]
SET [Назва стовпця 1]= значення / вираз / стовпець[Назва стовпця 2] = значеня…
FROM [Джерело даних]WHERE [Умови];

11. Запити на вставку Insert

Форма вставки	Опис
INSERT/VALUES	Вставляє одну строку значень. Зазвичай використовується для

	інтерактивного вводу даних користувачем
INSERT/SELECT	Вставляє в таблицю результуючий набір даних підзапиту
INSERT/EXEC	Вставляє в таблицю результат виконання збереженої процедури. Зазвичай використовується для складних маніпулювань даними
INSERT DEFAULT	Створює нову строку з застосуванням всіх значень за замовчуванням
SELECT INTO	Створює нову таблицю з результуючого набору даних інструкції SELECT

12. Вставка одного рядку значень

Форма вставки має один набір значень, вона обмежена вставкою в таблицю лише однієїстроки, і використовується переважно для вводу даних в базу користувачем.

Структура запиту має вигляд:

INSERT INTO[] [Назва таблиці] [(стовпець1, стовпець2, …)]VALUES (значення1,значення2,…)

Не всі стовбці таблиці повинні бути перераховані в інструкції INSERT, у випадку, якщо якийсь стовпець не буде перерахований в цій інструкції, йому автоматично присвояться значення NULL.

13. Вставка результуючого набору даних інструкції SELECT

Дані можна перемістити із результуючого набору даних в таблицю за допомогою інструкції INSERT … SELECT

Структура такого запиту має вигляд :

INSERT INTO [Назва таблиці в яку буде проходити вставка]
SELECT *
FROM [Назва таблиці джерела даних]
WHERE [Умови для вибірки]

14. Вставка результуючого набору даних з збереженої процедури

Форма INSERT…EXEC інструкції вставки використовує результат виконання збереженої процедури, як джерела даних для вставки в таблицю.

Структура такого запиту має вигляд :

INSERT INTO [Назва таблиці] [(стовбці)]
EXEC [Назва збереженої процедури] параметри

Варто зазначити, що збережена процедура може вернути декілька наборів даних і інструкція INSERT вставить в таблицю кожен з цих наборів.

15. Запити на видалення Delete

Інструкція DELETE в своїй найпростіші формі видаляє всі строки з таблиці. Для того щоб видалялись лише потрібні строки застосовують інструкцію WHERE:

Такий запит матиме вигляд:

```
DELETE FROM [Назва таблиці з якої потрібне видалення]
WHERE [Умова відбору потрібних строк]
```

Можливе також видалення з декількох таблиць. Для цього необхідне застосування інструкції JOIN.

III. Завдання на лабораторну роботу

1. Вибрати з таблиці Student всі строки і стовпці: FirstName, LastName, GroupId.
2. З таблиці Student вибрати записи про студентів, прізвище яких закінчується на «ко»
3. Підрахувати середній прохідний бал в кожній групі.
4. Визначити фактичне число студентів у кожній групі кафедри ТК, все посортувати за кількістю студентів в групі.
5. Підрахувати в кожній з груп число студентів, що мають прохідний бал більше 4.5.
6. Підрахувати кількість двієчників (Pball_student<2) на кожній із кафедр. Результат повинен містити назву кафедри та кількість двієчників.
7. Вивести ID предметів, по яким є лекції, але немає лабораторних.
8. Вивести прізвища студентів, які вивчають дисципліну «Бази даних» і також вивчають «Програмування». (назви дисциплін можуть бути інакшими відповідно до наповнення Вашої бази даних)
9. Розглянути 3 способи з'єднання на прикладі запиту на вибірку з таблиць "student" і "subject_success". Визначити як впливає обраний спосіб з'єднання на результат запиту.
10. Вибрати інформацію про оцінки, отримані студентами з усіх предметів. Результат повинен містити прізвище студента, найменування предмету та оцінку.
11. Вибрати предмети, в яких загальна кількість годин не дорівнює сумі годин лекцій та практик.
12. Визначити середнє навантаження викладача кафедри в поточному семестрі. Середнє навантаження розраховується як загальна кількість годин, проведених кафедрою, розділена на число викладачів кафедри.
13. Створити запит на створення нової таблиці «Навантаження викладача», в якій буде записане навантаження викладачів всіх кафедр. Нова таблиця повинна містити поля: викладач, предмет, загальна кількість годин.
14. Створити запит на оновлення, який оновить інформацію в таблиці «Група» (поле MaxCountStudents) згідно з фактичною кількістю студентів в базі даних.

IV. Контрольні питання

1. Дайте визначення реляційної алгебри й реляційного числення. В чому відмінність цих понять?
2. Назвіть основні операції реляційної алгебри.
3. Назвіть додаткові операції реляційної алгебри.
4. Дайте визначення операції вибірки. Наведіть приклад.
5. Дайте визначення операції проекції. Наведіть приклад.
6. Дайте визначення операції різниці. Наведіть приклад.
7. Дайте визначення операції об'єднання. Наведіть приклад.
8. Дайте визначення операції перетину. Наведіть приклад.
9. Дайте визначення операції декартового добутку. Наведіть приклад.
10. Дайте визначення операції ділення. Наведіть приклад.
11. Дайте визначення операції з'єднання? Назвіть типи операції з'єднання. Наведіть приклади.

Лабораторна робота №3
Створення і використання збережених процедур і тригерів

I. Підготовка до лабораторної роботи

Збережені процедури представляють собою групи пов'язаних між собою операторів SQL, застосування яких робить роботу програміста легшою і гнучкою, оскільки виконати збережену процедуру часто виявляється набагато простіше, ніж послідовність окремих операторів SQL. Збережені процедури представляють собою набір команд, що складається з одного або кількох операторів SQL або функцій та зберігається в базі даних в відкомпільованому вигляді. Виконання в базі даних збережених процедур замість окремих операторів SQL дає користувачеві наступні переваги:

1) необхідні оператори вже містяться в базі даних;
2) всі вони пройшли етап синтаксичного аналізу і знаходяться в виконуваному форматі;
3) перед виконанням процедури SQL Server генерує для неї план виконання, виконує її оптимізацію та компіляцію;
4) збережені процедури підтримують модульне програмування, так як дозволяють розбивати великі завдання на самостійні, більш дрібні та зручні в управлінні частини;
5) збережені процедури можуть викликати інші процедури і функції;
6) збережені процедури можуть бути викликані з прикладних програм інших типів;
7) як правило, збережені процедури виконуються швидше, ніж послідовність окремих операторів;
8) збережені процедури простіше використовувати: вони можуть складатися з десятків і сотень команд, але для їх запуску достатньо вказати лише ім'я потрібної процедури. Це дозволяє зменшити розмір запиту, що посилається від клієнта на сервер, а значить, і навантаження на мережу.

II.Теоретичні відомості

1. Збережені процедури

Зберігання процедур в тому ж місці, де вони виконуються, забезпечує зменшення обсягу переданих по мережі даних і підвищує загальну продуктивність системи. Застосування процедур спрощує супровід програмних комплексів та внесення змін до них. Зазвичай всі обмеження цілісності у вигляді правил і алгоритмів обробки даних реалізуються на сервері баз даних і доступні кінцевому додатку у вигляді набору процедур, які й становлять інтерфейс обробки даних. Для забезпечення цілісності даних, а також в цілях безпеки, додаток зазвичай не отримує прямого доступу до даних - вся робота з ними ведеться шляхом виклику тих чи інших процедур.

Подібний підхід робить досить простою модифікацію алгоритмів обробки даних, які негайно ж стають доступними для всіх користувачів мережі, і забезпечує можливість розширення системи без внесення змін до самого додатку: досить змінити процедуру, що зберігається на сервері баз даних. Розробнику не потрібно перекомпільовувати додаток, створювати його копії, а також інструктувати користувачів про необхідність роботи з новою версією. Користувачі взагалі можуть не підозрювати про те, що в систему внесені зміни.

Збережені процедури існують незалежно від таблиць або будь-яких інших об'єктів баз даних. Вони викликаються клієнтською програмою, іншою збереженою процедурою або тригером. Розробник може керувати правами доступу до збереженої процедури,

дозволяючи чи забороняючи її виконання. Змінювати код процедури дозволяється тільки її власнику або члену фіксованою ролі бази даних. При необхідності можна передати права володіння нею від одного користувача до іншого.

Збережені процедури в середовищі MS SQL Server

При роботі з SQL Server користувачі можуть створювати власні процедури, що реалізують ті чи інші дії. Збережені процедури є повноцінними об'єктами бази даних, а тому кожна з них зберігається в конкретній базі даних. Безпосередній виклик збереженої процедури можливий тільки якщо він здійснюється в контексті тієї бази даних, де знаходиться процедура.

Типи збережених процедур

У SQL Server є кілька типів збережених процедур.

1) Системні процедури призначені для виконання різних адміністративних дій. Практично всі дії з адміністрування сервера виконуються з їх допомогою. Можна сказати, що системні збережені процедури є інтерфейсом, що забезпечує роботу з системними таблицями, яка, в кінцевому рахунку, зводиться до зміни, додавання, видалення і вибірці даних із системних таблиць як для користувача, так і системних баз даних. Системні процедури мають префікс sp_, зберігаються в системній базі даних і можуть бути викликані в контексті будь-якої іншої бази даних.

2) Користувальницькі процедури реалізують ті чи інші дії. Збережені процедури - повноцінний об'єкт бази даних. Внаслідок цього кожна збережена процедура розташовується в конкретній базі даних, де і виконується.

3) Тимчасові процедури існують лише деякий час, після чого автоматично знищуються сервером. Вони діляться на локальні і глобальні. Локальні тимчасові процедури можуть бути викликані тільки з того з'єднання, в якому створені. При створенні такої процедури їй необхідно дати ім'я, що починається з одного символу #. Як і всі тимчасові об'єкти, збережені процедури цього типу автоматично видаляються при відключенні користувача, перезапуску або зупинці сервера. Глобальні тимчасові процедури доступні для будь-яких з'єднань сервера, на якому є така ж процедура. Для її визначення досить дати їй ім'я, що починається з символів # #. Видаляються ці процедури при перезапуску або зупинці сервера, а також при закритті з'єднання, в контексті якого вони були створені.

Створення, зміна або видалення збережених процедур

Створення процедури передбачає вирішення наступних завдань:

1) визначення типу створюваної збереженої процедури: тимчасова або користувацька. Крім цього, можна створити свою власну системну збережену процедуру, призначивши їй ім'я з префіксом sp_ і помістивши її в системну базу даних. Така процедура буде доступна в контексті будь-якої бази даних локального сервера;

2) планування прав доступу. При створенні збереженої процедури слід враховувати, що вона буде мати ті ж права доступу до об'єктів бази даних, що й користувач, який створив її;

3) визначення параметрів процедури. Подібно процедурам, які входять до складу більшості мов програмування, збережені процедури можуть мати вхідні і вихідні параметри;

4)розробка коду збереженої процедури. Код процедури може містити послідовність будь-яких команд SQL, включаючи виклик інших процедур.

Створення нової і зміна наявної процедури здійснюється за допомогою наступної команди:

```
<визначення_процедури>::=
{CREATE | ALTER } [PROCEDURE]
  ім'я_процедури [;номер]
[{@ім’я_параметра тип_даних }
 [VARYING ]
```

```
 [=default][OUTPUT] ][,...n]
[WITH { RECOMPILE | ENCRYPTION |
RECOMPILE, ENCRYPTION }]
[FOR
REPLICATION]
AS
sql_оператор [...n]
```

Розглянемо параметри даної команди.

Використовуючи префікси sp_, #, # #, створювану процедуру можна визначити як системну або тимчасову. Як видно з синтаксису команди, не допускається вказувати ім'я власника, якому належатиме створювана процедура, а також ім'я бази даних, де вона повинна бути розміщена. Таким чином, щоб розмістити створювану збережену процедуру в конкретній базі даних, необхідно виконати команду CREATE PROCEDURE в контексті цієї бази даних. При зверненні з тіла процедури до об'єктів тієї ж бази даних можна використовувати укорочені імена, тобто без вказівки імені бази даних. Коли ж потрібно звернутися до об'єктів, розташованих в інших базах даних, вказівка імені бази даних обов'язково.

Номер в імені - це ідентифікаційний номер процедури, однозначно визначає її в групі процедур. Для зручності управління процедурами логічно однотипні процедури можна групувати, привласнюючи їм однакові імена, але різні ідентифікаційні номери.

Для передачі вхідних і вихідних даних в створюваній збереженій процедурі можуть використовуватися параметри, імена яких, як і імена локальних змінних, повинні починатися з символу @. В одній збереженій процедурі можна поставити безліч параметрів, розділених комами. У тілі процедури не повинні застосовуватися локальні змінні, чиї імена збігаються з іменами параметрів цієї процедури.

Для визначення типу даних, який буде мати відповідний параметр процедури, годяться будь-які типи даних SQL, включаючи визначені користувачем. Однак тип даних CURSOR може бути використаний тільки як вихідний параметр процедури, тобто із зазначенням ключового слова OUTPUT.

Наявність ключового слова OUTPUT означає, що відповідний параметр призначений для повернення даних з збереженої процедури. Однак це зовсім не означає, що параметр не підходить для передачі значень в збережену процедуру. Вказівка ключового слова OUTPUT наказує серверу при виході з процедури привласнити поточне значення параметра локальної змінної, яка була вказана при виклику процедури як значення параметра. Зазначимо, що за умов згадування ключового слова OUTPUT значення відповідного параметра при виклику процедури може бути поставлено лише за допомогою локальної змінної. Не дозволяється використання будь-яких виразів чи констант, допустимих для звичайних параметрів.

Ключове слово VARYING застосовується спільно з параметром OUTPUT, що мають тип CURSOR. Воно визначає, що вихідним параметром буде результуюча множина.

Ключове слово DEFAULT є значенням, яке буде приймати відповідний параметр за замовчуванням. Таким чином, при виклику процедури можна не вказувати явно значення відповідного параметра.

Так як сервер кешує план виконання запиту і компільований код, при наступному виклику процедури будуть використовуватися вже готові значення. Проте в деяких випадках все ж потрібно виконувати перекомпіляцію коду процедури. Вказівка ключового слова RECOMPILE наказує системі створювати план виконання збереженої процедури при кожному її виклику.

Параметр FOR REPLICATION потребується при реплікації даних і включенні створюваної збереженої процедури в публікацію.

Ключове слово ENCRYPTION наказує серверу виконати шифрування коду процедури, що може забезпечити захист від використання авторських алгоритмів, що реалізують роботу збереженої процедури.

Ключове слово AS розміщується на початку власне тіла збереженої процедури, тобто набору команд SQL, за допомогою яких і буде реалізовуватися ту чи іншу дію. У тілі процедури можуть застосовуватися практично всі команди SQL, оголошуватися транзакції, встановлюватися блокування і викликатися інші процедури. Вихід з збереженої процедури можна здійснити за допомогою команди RETURN.

Видалення збереженої процедури здійснюється командою:

```
DROP PROCEDURE {імя_процедури} [,... n]
```

Для виконання процедури використовується команда:

```
[[EXECUTE] імя_процедури [; номер]
[[@ Імя_параметра =] {значення | @
 ім'я_змінної} [OUTPUT] | [DEFAULT ]][,... n]
```

Якщо виклик збереженої процедури не є єдиною командою в блоці, то присутність команди EXECUTE обов'язкова. Більше того, ця команда потрібна для виклику процедури з тіла іншої процедури або тригера.

Використання ключового слова OUTPUT при виклику процедури дозволяється тільки для параметрів, які були оголошені під час створення процедури з ключовим словом OUTPUT.

Коли ж при виклику процедури для параметра вказується ключове слово DEFAULT, то буде використано значення за замовчуванням. Природно, вказане слово DEFAULT дозволяється тільки для тих параметрів, для яких визначено значення за замовчуванням.

З синтаксису команди EXECUTE видно, що імена параметрів можуть бути опущені при виклику процедури. Однак у цьому випадку користувач повинен вказувати значення для параметрів у тому ж порядку, в якому вони перераховувалися при створенні процедури. Присвоїти параметру значення за замовчуванням, просто пропустивши його при перерахуванні можна. Якщо ж потрібно опустити параметри, для яких визначено значення за замовчуванням, досить явного вказівки імен параметрів при виклику процедури, що зберігається. Більш того, таким способом можна перераховувати параметри та їх значення в довільному порядку.

Відзначимо, що при виклику процедури вказуються або імена параметрів зі значеннями, або лише значення без імені параметра. Їх комбінування не допускається.

Приклад 3.1. Процедура без параметрів. Розробити процедуру для отримання назв і вартості товарів, придбаних Івановим.

```
CREATE PROC
my_proc1 AS
SELECT Товар.Найменування,
Товар.Ціна * Угода.Кількість
AS Вартість, Клієнт.
FROM Клієнт INNER JOIN
(Товар INNER JOIN Угода
ON Товар.КодТовара = Угода.КодТовара)
ON Клієнт.КодКлієнта = Угода.КодКлієнта
WHERE Клієнт.Прізвище = 'Іванов'
```

Приклад 3.2. Процедура для отримання назв і вартості товарів, придбаних Івановим. Для звернення до процедури можна використовувати команди:

EXEC my_proc1 або my_proc1 Процедура повертає набір даних.

Приклад 3.3. Процедура без параметрів. Створити процедуру для зменшення ціни товару першого сорту на 10%.

```
CREATE PROC my_proc2
 AS
UPDATE Товар SET Ціна = Ціна * 0.9
 WHERE Сорт = 'перший'
```

Приклад 3.3. Процедура для зменшення ціни товару першого сорту на 10%. Для звернення до процедури можна використовувати команди:

EXEC my_proc2 або my_proc2 Процедура не повертає ніяких даних.

Приклад 3.4. Процедура з вхідним параметром. Створити процедуру для отримання назв і вартості товарів, які придбав заданий клієнт.

```
CREATE PROC my_proc3
@ K VARCHAR (20)
AS
SELECT Товар.Назва,
Товар.Ціна * Угода.Кількість
AS Вартість, Клієнт.
FROM Клієнт INNER JOIN
(Товар INNER JOIN Угода
ON Товар.КодТовара = Угода.КодТовара)
ON Клієнт.КодКлієнта = Угода.КодКлієнта
WHERE Клієнт. Прізвище = @ k
```

Приклад 3.5. Процедура для отримання назв і вартості товарів, які придбав заданий клієнт.

Для звернення до процедури можна використовувати команди:

```
EXEC my_proc3 'Іванов' або
my_proc3 @ k =
'Іванов'
```

Приклад 3.6. Процедура з вхідними параметрами. Створити процедуру для зменшення ціни товару заданого типу відповідно до зазначеного %.

```
CREATE PROC my_proc4
@ T VARCHAR (20), @ p FLOAT
 AS
UPDATE Товар SET Ціна = Ціна * (1 - @ p)
WHERE Тип = @ t
```

Приклад 3.7. Процедура для зменшення ціни товару заданого типу відповідно до зазначеного %.

Для звернення до процедури можна використовувати команди:

```
EXEC my_proc4 'Вафлі', 0.05 або
EXEC my_proc4 @ t = 'Вафлі', @ p = 0.05
```

Приклад 3.8. Процедура з вхідними параметрами і значеннями за замовчуванням. Створити процедуру для зменшення ціни товару заданого типу відповідно до зазначеного %.

```
CREATE PROC my_proc5
@ T VARCHAR (20) = 'Цукерки',
 @ P FLOAT = 0.1
```

```
AS
UPDATE Товар SET Ціна = Ціна * (1 - @ p)
WHERE Тип = @ t
```

Приклад 3.9. Процедура з вхідними параметрами і значеннями за замовчуванням. Створити процедуру для зменшення ціни товару заданого типу відповідно до зазначеного %. Для звернення до процедури можна використовувати команди:

```
EXEC my_proc5 'Вафлі', 0.05 або
EXEC my_proc5 @ t = 'Вафлі', @ p = 0.05 або
EXEC my_proc5 @ p = 0.05
```

У цьому випадку зменшується ціна цукерок (значення типу не вказано при виклику процедури і береться за замовчуванням).

```
EXEC my_proc5
```

В останньому випадку обидва параметри (і тип, і відсотки) не вказані при виклику процедури, їх значення беруться за замовчуванням.

Приклад 3.10. Процедура з вхідними і вихідними параметрами. Створити процедуру для визначення загальної вартості товарів, проданих за конкретний місяць.

```
CREATE PROC my_proc6
@ M INT,
@S FLOAT OUTPUT
  AS
SELECT @ s = Sum (Товар.Ціна * Угода.Кількість)
FROM Товар INNER JOIN Угода
ON Товар.КодТовара = Угода.КодТовара
GROUP BY Month (Угода.Дата)
HAVING Month (Угода.Дата) = @m
```

Приклад 3.11. Процедура з вхідними і вихідними параметрами. Створити процедуру для визначення загальної вартості товарів, проданих за конкретний місяць.

Для звернення до процедури можна використовувати команди:

```
DECLARE @ st FLOAT
EXEC my_proc6 1, @ st OUTPUT
SELECT @ st
```

Цей блок команд дозволяє визначити вартість товарів, проданих у січні (вхідний параметр місяць вказаний рівним 1.

Створити процедуру для визначення загальної кількості товарів, придбаних фірмою, в якій працює заданий співробітник.

Спочатку розробимо процедуру для визначення фірми, де працює співробітник.

```
CREATE PROC my_proc7
 @ N VARCHAR (20),
@ F VARCHAR (20) OUTPUT
 AS
SELECT @ f = Фірма
FROM Клієнт
WHERE Прізвище = @ n
```

Приклад 3.12. Використання вкладених процедур. Створити процедуру для визначення загальної кількості товарів, придбаних фірмою, в якій працює заданий співробітник.

Потім створимо процедуру, підраховують загальну кількість товару, який закуплений фірмою, що цікавить нас.

```
CREATE PROC my_proc8
@ Fam VARCHAR (20),
@ Kol INT OUTPUT
AS
DECLARE @ firm VARCHAR (20)
EXEC my_proc7 @ fam, @ firm OUTPUT
SELECT @ kol = Sum (Угода.Кількість)
FROM Клієнт INNER JOIN Угода
ON Клієнт.КодКлієнта = Угода.КодКлієнта
GROUP BY Клієнт.Фірма
HAVING Клієнт.Фірма = @firm
```

Приклад 3.13. Створення процедури для визначення загальної кількості товарів, придбаних фірмою, в якій працює заданий співробітник.

Виклик процедури здійснюється за допомогою команди:

```
DECLARE @ k INT
EXEC my_proc8 'Іванов', @ k OUTPUT
SELECT @ k
```

2. Визначення тригера в стандарті мови SQL

Тригери є одним з різновидів збережених 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 скомпільована SQL-процедура, виконання якої обумовлено настанням певних подій всередині реляційної бази даних. Застосування тригерів здебільшого дуже зручно для користувачів бази даних. І все ж їх використання часто пов'язано з додатковими витратами ресурсів на операції введення/виводу. 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 та реальність. Крім того, іноді необхідно відстежувати зміни значень таблиці, щоб потрібним чином змінити пов'язані дані. Тригери можна розглядати як свого роду фільтри, що вступають в дію після виконання всіх операцій відповідно до правил, стандартними значеннями і т.д.

Тригер є спеціальний тип збережених процедур, що запускаються сервером автоматично при спробі зміни даних у таблицях, з якими тригери пов'язані. Кожен тригер прив'язується до конкретної таблиці. Всі вироблювані 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.п.

Тригер є вельми корисним і в той же час небезпечним засобом. 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 тригерній події, до того ж він не має аргументів. Приведення його в дію іноді називають запуском тригера. За допомогою тригерів досягаються наступні цілі:

1) перевірка коректності введених даних і виконання складних обмежень цілісності даних, які важко, якщо взагалі можливо, підтримувати з допомогою обмежень цілісності, встановлених для таблиці;

2) видача попереджень, що нагадують про необхідність виконання деяких дій при оновленні таблиці, реалізованому певним чином;

3) накопичення аудиторської інформації за допомогою фіксації відомостей про внесені зміни і тих осіб, які їх виконали;

4) підтримка реплікації.

Основний формат команди CREATE TRIGGER показаний нижче:

<Визначення_тригера>::=
CREATE TRIGGER ім'я_тригера
BEFORE | AFTER <тригерна подія>
ON <ім'я_таблиці>
[REFERENCING
<список_старих_або_нових_псевдонімів>]
[FOR EACH {ROW | STATEMENT}]
[WHEN(умова_тригера)]
<тіло_тригера>

Тригерні події складаються з вставки, видалення і оновлення рядків в таблиці. В останньому випадку для тригерній події можна вказати конкретні імена стовпців таблиці. Час запуску тригера визначається за допомогою ключових слів BEFORE (тригер запускається до виконання пов'язаних з ним подій) або AFTER (після їх виконання).

Виконувані тригером дії задаються для кожного рядка (FOR EACH ROW), охопленої даними подією, або тільки один раз для кожної події (FOR EACH STATEMENT).

Позначення <список_старих_або_нових_псевдонімів> відноситься до таких компонентів, як старий чи новий рядок (OLD/NEW) або стара чи нова таблиця (OLD TABLE/NEW TABLE). Ясно, що старі значення не застосовні для подій вставки, а нові - для подій видалення.

За умови правильного використання тригери можуть стати дуже потужним механізмом. Основна їх перевага полягає в тому, що стандартні функції зберігаються всередині бази даних та злагоджено активізуються при кожному її оновлення. Це може істотно спростити програми. Проте слід згадати і про властиві тригеру недоліки:

1) складність: при переміщенні деяких функцій в базу даних ускладнюються завдання її проектування, реалізації та адміністрування;

2) прихована функціональність: перенесення частини функцій в базу даних і збереження їх у вигляді одного або декількох тригерів іноді призводить до приховування від користувача деяких функціональних можливостей. Хоча це певною мірою спрощує його роботу, але, на жаль, може стати причиною незапланованих, потенційно небажаних і шкідливих побічних ефектів, оскільки в цьому випадку користувач не в змозі контролювати всі процеси, що відбуваються в базі даних;

3) вплив на продуктивність: перед виконанням кожної команди по зміні стану бази даних СУБД повинна перевірити тригерну умову з метою з'ясування необхідності запуску тригера для цієї команди. Виконання подібних обчислень позначається на загальній продуктивності СУБД, а в моменти пікового навантаження її зниження може стати особливо помітним. Очевидно, що при зростанні кількості тригерів збільшуються і накладні витрати, пов'язані з такими операціями.

4) Неправильно написані тригери можуть привести до серйозних проблем, таких, наприклад, як поява «мертвих» блокувань. Тригери здатні тривалий час блокувати

безліч ресурсів, тому слід звернути особливу увагу на зведення до мінімуму конфліктів доступу.

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_стовпця)
[{AND | OR} UPDATE(ім'я_стовпця)] [...n]
|
IF (COLUMNS_UPDATES(){оператор_біт_обробки}
біт_маска_зміни)
{оператор_біт_порівняння }біт_маска [...n]}
sql_оператор [...n]
}
}
```

Тригер може бути створений лише в поточній базі даних, але допускається звернення всередині тригера до інших баз даних, у тому числі і розташованим на віддаленому сервері.

Розглянемо призначення аргументів з команди CREATE | ALTER TRIGGER.

Ім'я тригера повинно бути унікальним в межах бази даних. Додатково можна вказати ім'я власника.

При вказівці аргументу WITH ENCRYPTION сервер виконує шифрування коду тригера, щоб ніхто, включаючи адміністратора, не міг отримати до нього доступ і прочитати його. Шифрування часто використовується для приховання авторських алгоритмів обробки даних, що є інтелектуальною власністю програміста або комерційною таємницею.

Типи тригерів

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

1) AFTER. Тригер виконується після успішного виконання команд, що викликали його. 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 відбудеться відкат тригера, то будуть відхилені і зміни користувача. 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 умовчанням в SQL Server всі тригери є AFTER- тригерами.

2) 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:

1) INSERT TRIGGER - запускаються при спробі вставки даних за допомогою команди INSERT.

2) UPDATE TRIGGER - запускаються при спробі зміни даних за допомогою команди UPDATE.

3) 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 повинна бути вказана хоча б одна команда. Допускається створення тригера, що реагує на дві або на всі 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.

Зазначимо, що 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, тому що вони не можуть бути скасовані у разі відкату транзакції, в якій виконується тригер. Ця заборона навряд чи може якимось чином позначитися на функціональності створюваних 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 та видалення записів сервер створює дві спеціальні таблиці: inserted і deleted. У них містяться списки рядків, які будуть вставлені або видалені по завершенні 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 іншої тригер не зможе отримати до них доступ. Залежно від типу операції, що викликала виконання тригера, вміст таблиць inserted і deleted може бути різним:

1) команда INSERT - у таблиці inserted містяться всі рядки, які користувач намагається вкласти в таблицю; в таблиці deleted не буде жодного рядка; після завершення тригера всі рядки з таблиці inserted перемістяться у вихідну таблицю;

2) команда DELETE - у таблиці deleted будуть міститися всі рядки, які користувач спробує видалити; тригер може перевірити кожен рядок і визначити, чи дозволено її видалення; в таблиці inserted не виявиться ні одного рядка;

3) команда UPDATE - при її виконанні в таблиці deleted знаходяться старі значення рядків, які будуть вилучені при успішному завершенні тригера. Нові значення рядків містяться в таблиці inserted. Ці рядки додадуться у вихідну таблицю після успішного виконання тригера.

Для отримання інформації про кількість рядків, яке буде змінено при успішному завершенні тригера, можна використовувати функцію @@ROWCOUNT, вона повертає кількість рядків, оброблених останньою командою. Слід підкреслити, що тригер

запускається не при спробі 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 з 100 рядків, які вставляються, змінюються або видаляються, тільки один не задовольняє тих чи інших умов, то ніякий рядок не буде вставлений, змінений або видалений. Така поведінка обумовлена вимогами транзакції - повинні бути виконані або всі модифікації, або жодної.

Тригер виконується як неявно 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 (у порядку проходження стовпців при створенні таблиці). Якщо біт встановлено у значення "1", то відповідний стовпець був змінений. Крім того, факт зміни стовпця визначає і функція UPDATE (ім'я_стовпця).

Для видалення тригера використовується команда

```
DROP TRIGGER {ім’я_тріггера} [,... n]
```

Приклад. Використання тригера для реалізації обмежень на значення. Операція запису кількості проданого товару має бути не більше, ніж його залишок з таблиці Склад.

Команда вставки запису в таблицю Угода може бути, наприклад, такий:

```
INSERT INTO Угода
VALUES (3,1, -299, '01 / 08/2002 ')
```

Створюваний тригер повинен відреагувати на її виконання наступним чином: необхідно скасувати команду, якщо в таблиці Склад величина залишку товару виявилася меншою продаваного кількості товару з введеним кодом (у прикладі код товару = 3). Кількість товару вказується із знаком "+", якщо товар поставляється, і зі знаком "-", якщо він продається. Представлений тригер налаштований на обробку тільки одного запису.

```
CREATE TRIGGER Тригер_ins
ON Угода FOR INSERT
AS
IF @@ROWCOUNT=1
BEGIN
IF NOT EXISTS(SELECT *
FROM inserted
WHERE -inserted.кількість<=ALL(SELECT
  Склад.
FROM Склад,Угода
WHERE Склад.КодТовара = Угода.КодТовара))
BEGIN
ROLLBACK TRAN PRINT
'Скасування поставки: товару на складі немає'
END
  END
```

3. Транзакції

Транза́кція (англ. transaction) — група послідовних операцій з базою даних, яка є логічною одиницею роботи з даними. Транзакція може бути виконана або цілком і успішно, дотримуючись цілісності даних і незалежно від інших транзакцій, що йдуть паралельно, або

не виконана зовсім, і тоді вона не може справити ніякого ефекту. Транзакції обробляються транзакційними системами, в процесі роботи яких створюється історія транзакцій.

Транзакції з базою даних використовуються для досягнення наступних цілей:

1. Забезпечення надійних робочих елементів, які дозволяють правильно відновити роботу у випадку збоїв та зберігати цілісність бази даних у випадку системних відмов, коли виконання операцій зупиняється (повністю або частково) і більшість операцій над базою даних залишаються незавершеними з нез'ясованим статусом.
2. Для забезпечення роздільного доступу для процесів, що одночасно звертаються до бази даних. При відсутності ізоляції операцій, результати отримані процесами можуть бути помилковими.

Властивості транзакцій

Одним з найбільш розповсюджених наборів вимог до транзакцій і транзакційних систем є набір ACID (англ. Atomicity, Consistency, Isolation, Durability). Вимоги ACID були в головному сформульовані наприкінці 70-х років Джимом Ґреєм. Разом з тим, існують спеціалізовані системи з ослабленими транзакційними властивостями.

Atomicity — Атомарність

Атомарність гарантує, що ніяка транзакція не буде зафіксована в системі частково. Будуть або виконані всі її складові операції, або не виконано жодної. Оскільки на практиці неможливо одночасно і атомарно виконати всю послідовність операцій усередині транзакції, вводиться поняття «відкат» (англ. rollback): якщо транзакцію не вдається повністю завершити, результати всіх її до сих пір виконаних дій будуть відмінені, і система повернеться до стану на початок транзакції.

Consistency — Узгодженість

Одна з найскладніших і неоднозначних властивостей з четвірки ACID. У відповідності до цієї вимоги, система знаходиться в узгодженому стані до початку транзакції і повинна залишитись в узгодженому стані після завершення транзакції. Не можна плутати вимогу узгодженості з вимогами цілісності (англ. integrity). Останні правила є більш вузькими і, багато в чому, специфічні для реляційних СКБД: є вимоги цілісності типів (англ. domain integrity), цілісності посилань (англ. referential integrity), цілісності сутностей (англ. entity integrity), які не можуть бути порушені фізично в силу особливостей реалізації системи.

Узгодженість є більш широким поняттям. Наприклад, в банківській системі може існувати вимога рівності суми, що списується з одного рахунка, сумі, що зараховується на інший. Це бізнес-правило і воно не може бути гарантовано тільки перевірками цілісності, його повинні дотриматись програмісти при написанні коду транзакцій. Якщо будь-яка транзакція виконає списання, але не виконає зачислення, то система залишиться в некоректному стані і властивість узгодженості буде порушена.

Нарешті, ще одне зауваження стосується того, що під час виконання транзакції узгодженість не потребується. В нашому прикладі, списання і зачислення будуть, скоріш за все, двома різними підопераціями и між їх виконанням всередині транзакції буде видно неузгоджений стан системи. Однак не треба забувати, що при виконанні вимоги ізоляції, жодним іншим транзакціям ця неузгодженість не буде видна. А атомарність гарантує, що транзакція або буде повністю завершена, або жодна з операцій транзакції не буде виконана. Тим самим ця проміжна неузгодженість є прихованою.

Isolation — Ізольованість

Під час виконання певної транзакції, транзакції, які відбуваються паралельно, не повинні впливати на її результат. Ця властивість не дотримується на рівні ізольованості Repeatable Read та нижче.

Durability — Надійність

Незалежно від проблем на нижніх рівнях (наприклад, при знеструмленні системи чи збоях в обладнанні), зміни, зроблені транзакцією, яка успішно завершена, повинні залишитись збереженими після повернення системи до роботи. Іншими словами, якщо

користувач отримав підтвердження від системи, що транзакція виконана, він може бути впевнений, що зміни, які він зробив, не будуть відмінені через будь-який збій.

Рівні ізоляції транзакції

В ідеалі транзакції різних користувачів повинні виконуватись так, щоб створювалась ілюзія, що користувач поточної транзакції — єдиний. Однак в реальності, з міркувань продуктивності і для виконания деяких спеціальних задач, СКБД забезпечують певні рівні ізоляції транзакцій.

Рівні є описаними в порядку збільшення ізольованості транзакцій і, відповідно, надійності роботи з даними.

- 0 — Читання непідтверджених даних (брудне читання) (Read Uncommitted, Dirty Read) — читання незафіксованих змін як своєї транзакції, так і паралельних транзакцій. Немає гарантії, що дані, змінені іншими транзакціями, не будуть в будь-який момент змінені в результаті їх відката, тому таке читання є потенційним джерелом помилок. Неможливими є втрачені зміни (lost changes), можливими є неповторювані читання і фантоми.
- 1 — Читання підтверджених даних (Read Committed) — читання всіх змін своєї транзакції і зафіксованих змін паралельних транзакцій. Втрачені зміни и брудне читання не дозволяються, можливими є неповторюване читання і фантоми.
- 2 — Повторюване читання (Repeatable Read, Snapshot) — читання всіх змін своєї транзакції, будь-які зміни, внесені паралельними транзакціями після початку своєї, є недоступними. Втрачені зміни, брудне і неповторюване читання не є можливими; можливі фантоми.
- 3 — Впорядкований — (Serializable) — впорядковані транзакції. Результат паралельного виконання впорядкованої транзакції з іншими транзакціями повинен бути логічно еквівалентний результату їх будь-якого послідовного виконання. Проблеми синхронізації не виникають.

Чим вище рівень ізоляції, тим більше потребується ресурсів, щоб його забезпечити. Відповідно, підвищення ізольованості може приводити до зниження швидкості виконання паралельних транзакцій, що є «платою» за підвищення надійності.

В СКБД рівень ізоляції транзакцій можна вибрати як для всіх транзакцій разом, так і для одної конкретної транзакції. За умовчанням в більшості баз даних використовується рівень 1 (Read Committed). Рівень 0 використовується в основному для відстеження змін тривалих транзакцій або для читання даних, що рідко змінюються. Рівні 2 и 3 використовуються при підвищених вимогах до ізольованості транзакцій.

Управління транзакціями

Для управління транзакціями використовуються наступні команди:

- COMMIT - зберегти зміни.
- ROLLBACK - скасувати зміни.
- SAVEPOINT - створює точки збереження в групах транзакцій.
- SET TRANSACTION - поміщає ім'я в транзакцію.

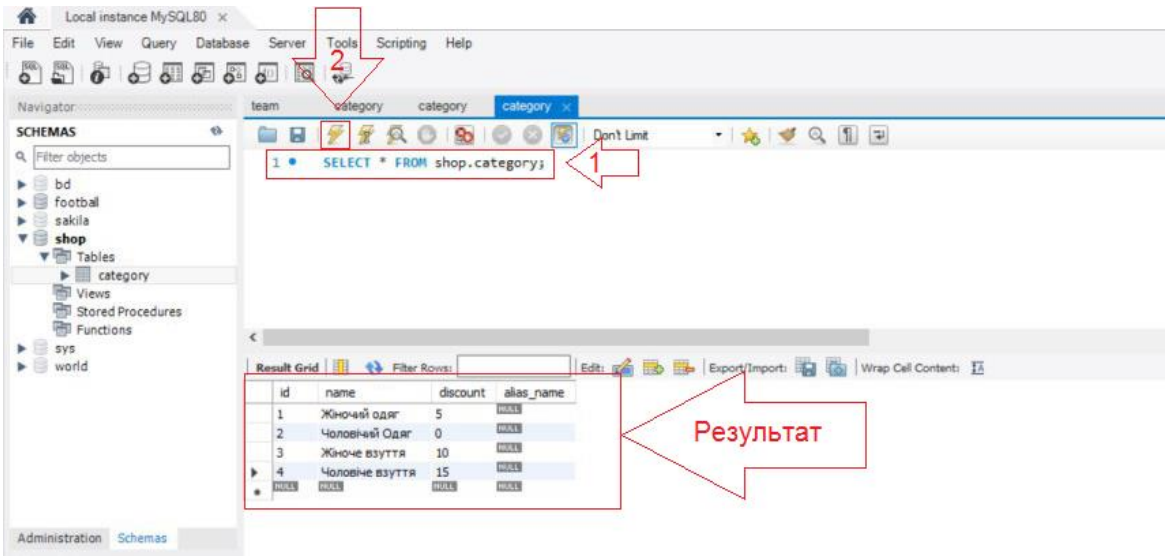
Команди управління транзакціями

Команди управління транзакціями використовуються тільки з командами DML, такими як - INSERT, UPDATE і DELETE. Вони не можуть використовуватися при створенні таблиць або їх видаленні, оскільки ці операції автоматично фіксуються в базі даних.

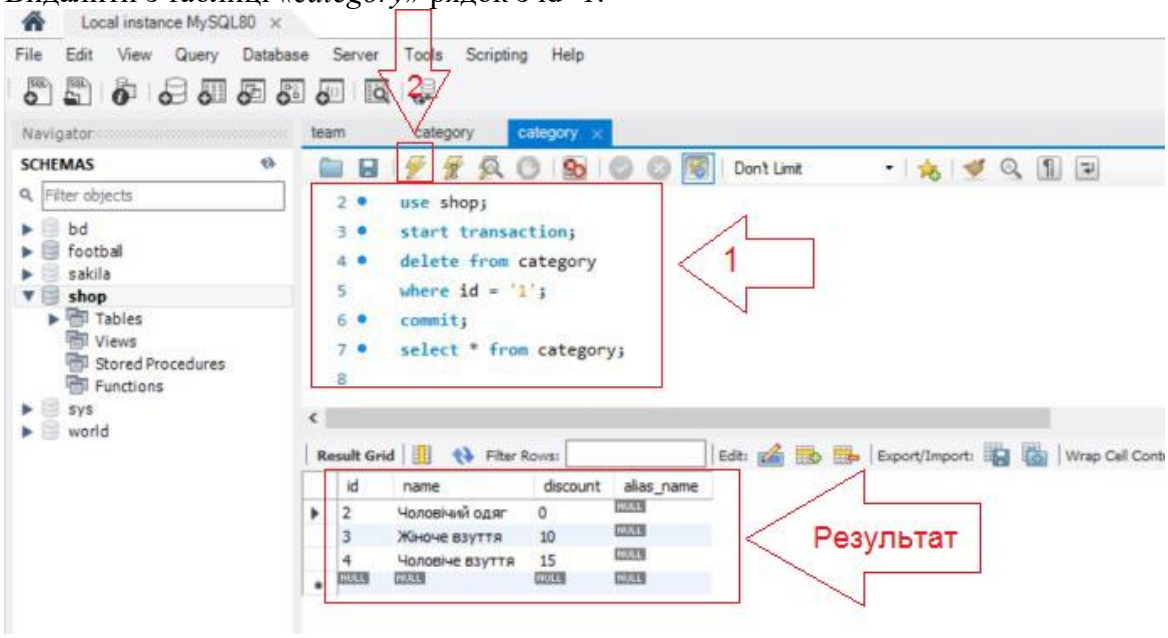
Команда COMMIT

Команда COMMIT – це команда транзакції, яка використовується для збереження змін внесених транзакцією в базу даних. Команда COMMIT зберігає всі транзакції в базі даних з моменту виконання останньої команди COMMIT або ROLLBACK.

Вивід значення таблиці *«category»*:

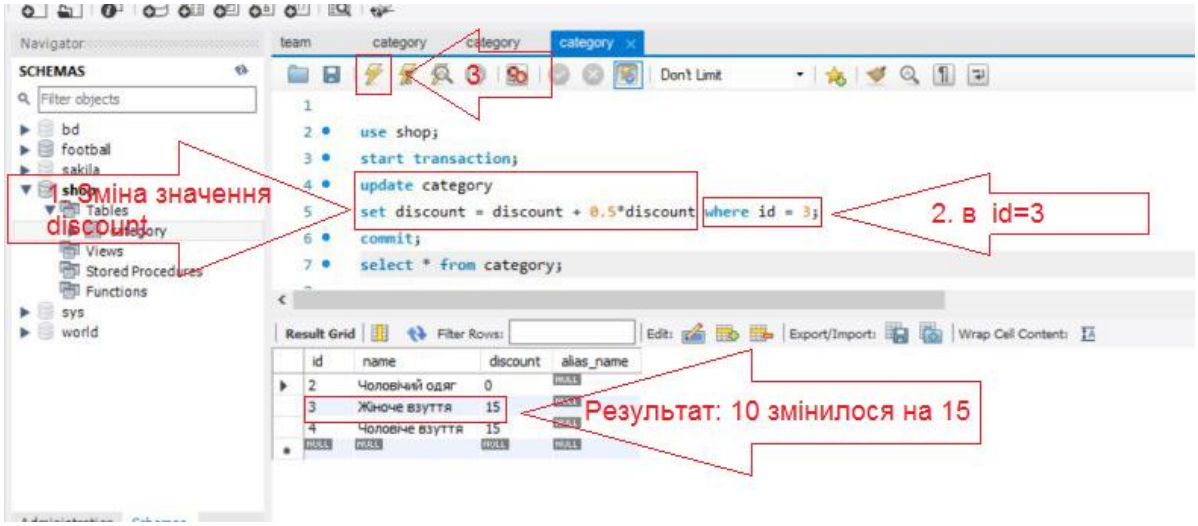


Видалити з таблиці *«category»* рядок з id=1:



Команда DELETE видаляє рядок №1 в таблиці, а команда COMMIT зберігає зміни в таблиці.

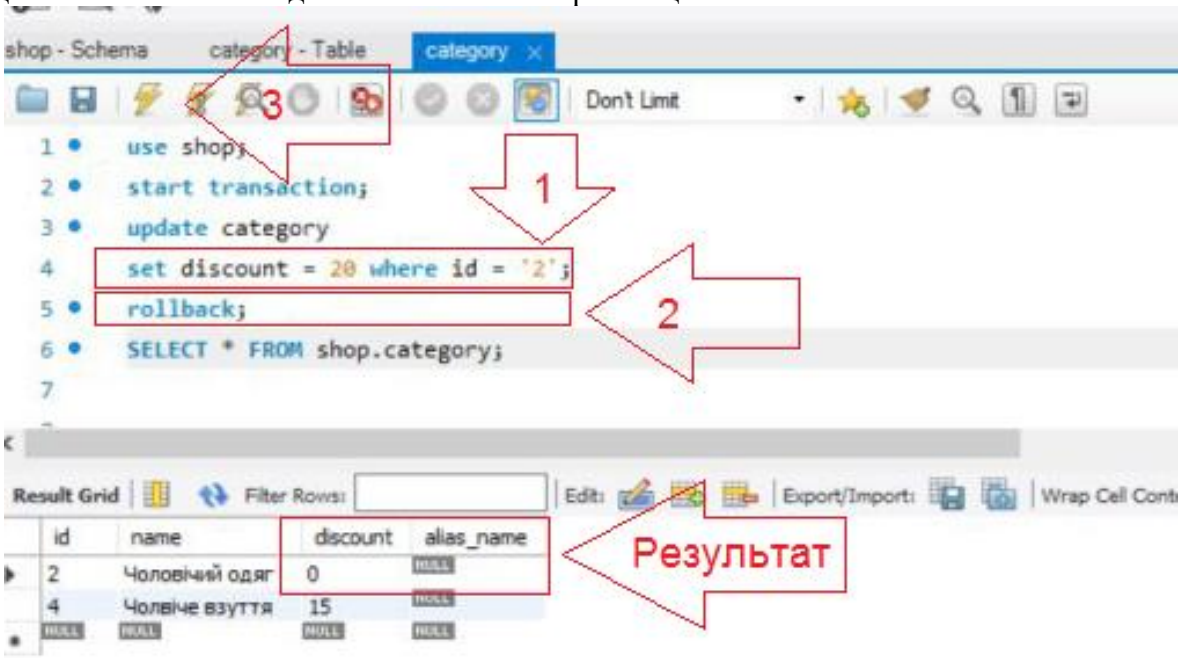
Оновити значення *«discount»* для рядка з id=3 на «дане значення» + «половина даного значення»:



Команда ROLLBACK

Команда ROLLBACK - це транзакційна команда, яка використовується для скасування транзакцій, які ще не були збережені в базі даних. Ця команда може використовуватися тільки для скасування транзакцій з моменту виконання останньої команди COMMIT або ROLLBACK.

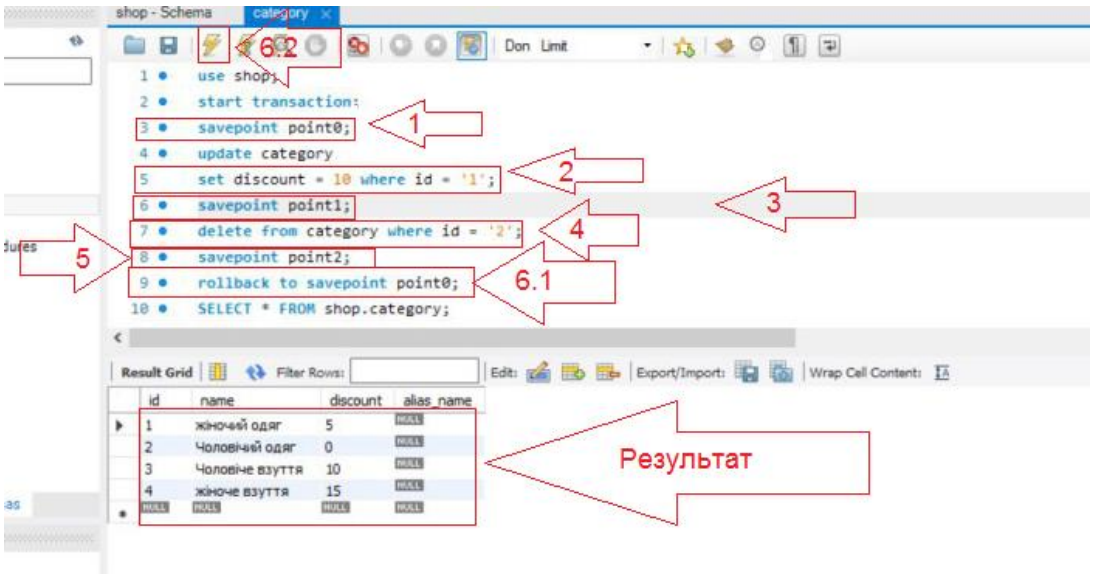
Повинно було встановитись в id=2 в «*discount*» значення 20, але була використана команда ROLLBACK яка відмінила виконання транзакції.



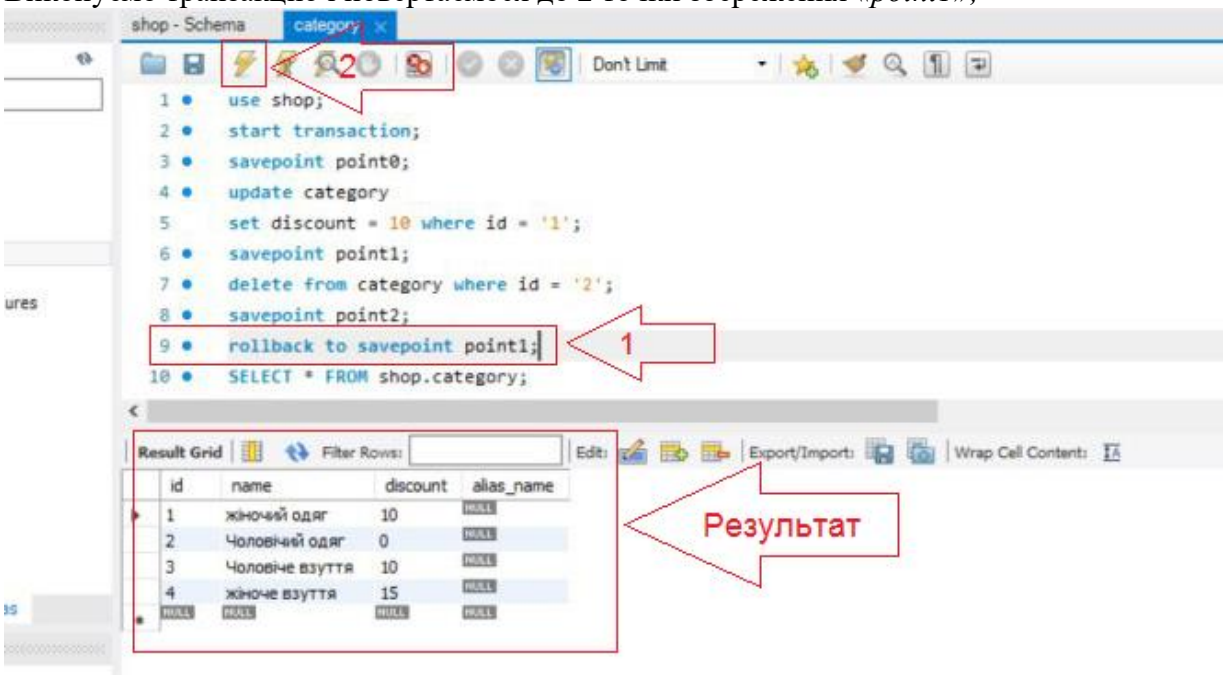
Команда SAVEPOINT

SAVEPOINT (точка збереження) - це спеціальна позначка в транзакції, яка дозволяє відкотити всі команди, виконані після неї, і відновити таким чином стан на момент установки цієї точки.

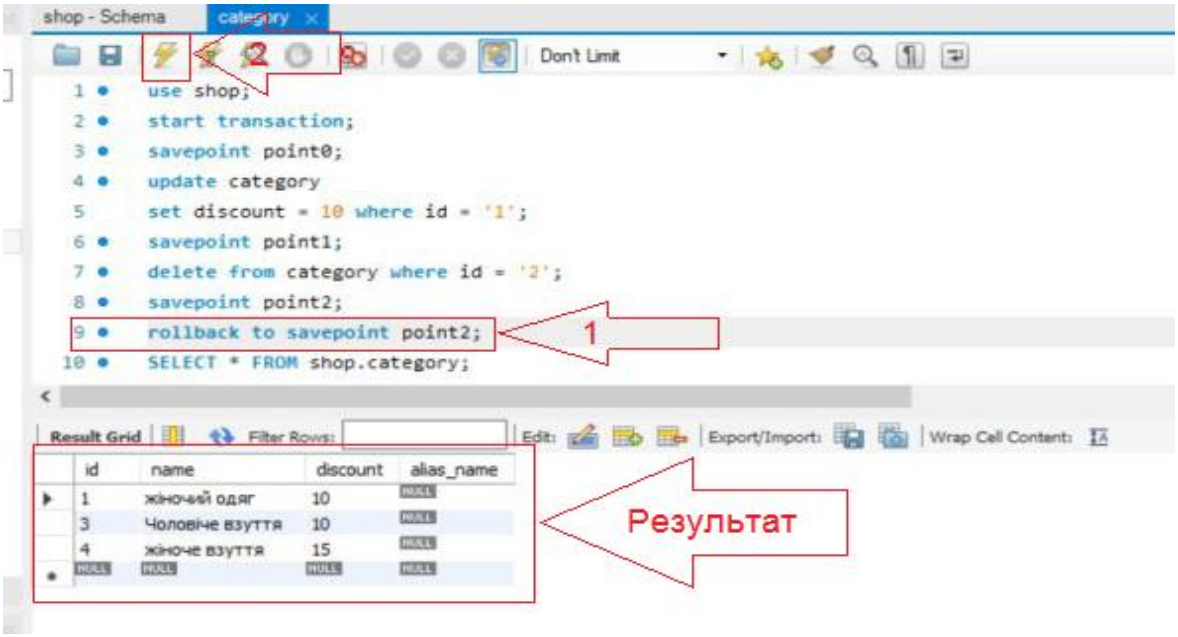
Створюємо 1 точку збереження *«point0»* не змінивши таблицю*;*
Змінюємо значення стовпця *«discount»* на значення 10 в таблиці «*category*»;
Створюємо 2 точку збереження *«point1»;*
Видаляємо рядок з id=2 в таблиці *«category»;*
Створюємо 2 точку збереження *«point1»;*
Виконуємо транзакцію і повертаємося до 1 точки збереження *«point0»;*



Виконуємо транзакцію і повертаємося до 2 точки збереження *«point1»;*



Виконуємо транзакцію і повертаємося до 3 точки збереження *«point2».*



4. Індекси

Індекси допомагають SQL сервер знаходити дані. Вони прискорюють вибірку даних, вказуючи SQL Серверу положення табличних даних на диску. Одна таблиця може мати кілька індексів.

Індекси безпосередньо недоступні користувачеві, оскільки в мові SQL не можна явно вказати на індекс у запиті. Користувач може лише створити або видалити табличний індекс, а SQL Сервер сам вирішує як використовувати його при запиті до таблиці. У міру того як змінюються дані в таблиці, SQL Сервер може змінити індекс, який буде відображати зміни, що відбулися. Знову ж таки це відбувається автоматично без втручання з боку користувача.

Якщо пошук в таблиці здійснюються за значенням у неіндексованому стовпці, то система просто переглядає один рядок за одним для знаходження потрібного значення. У цьому випадку час пошуку прямо пропорційний числу рядків у таблиці.

SQL Сервер підтримує наступні види індексів:

· Складові (складні) індекси - включають більше одного табличного стовпця. Ці індекси використовуються, коли дані в декількох стовпцях логічно взаємопов'язані.

· Унікальні індекси - забороняють використання повторюваних значень у зазначених стовпцях. SQL Сервер перевіряє, чи немає повторюваних значень, коли створюється такий індекс (якщо в таблиці вже є дані) і повторює перевірку при кожній модифікації даних.

· Кластеризовані та некластеризовані індекси - дозволяють пов'язувати фізичне і логічне розташування даних. За наявності кластеризованого індексу фізичне розташування рядків таблиці на накопичувачі відповідає їх логічному (індексованому) розташуванню. У таблиці може бути лише один кластеризований індекс. Некластеризований індекс не забезпечує такої відповідності і дані можуть розташовуватися в будь-якому порядку.

Порівняння двох способів створення індексів

Табличний індекс можна створити або за допомогою оператора create index (створення індексу), або вказавши обмеження цілісності у вигляді унікального (unique) або головного (primary) ключа в операторі створення таблиці create table.

Якщо індекс задається як унікальне поле, то недоліки:

· Не дозволяє створювати індекси за неунікальними полями.

· Не дозволяє скористатися опціями, передбаченими в команді create index, які уточнюють спосіб використання індексу.

· Ці індекси можна видалити разом з ключами оператором alter table.

Індекси слід створювати оператором create index. Також можна використовувати унікальний або головний ключ, оскільки це найпростіший спосіб зв'язування індексу з таблицею.

Рекомендації з використання індексів

Індекси прискорюють вибірку даних. Наявність індексу у табличного стовпця часто призводить до значної зміни часу відповіді на запит, прискорюючи його виконання.

Індекс не можна зв'язувати з кожним стовпцем. Створення індексу пов'язано з витратами часу і пам'яті.

Наприклад, при перевизначенні кластеризованого індексу автоматично створюється некластеризованний індекс.

Друга причина полягає в тому, що модифікація даних у індексованих стовпцях займає більше часу в порівнянні з неіндексованими.

З табличним стовпцем, за яким найбільш ймовірно проводиться сортування даних і який зазвичай вказується в умові order by, необхідно пов'язати індекс, щоб SQL Сервер міг проводити впорядкування значень за цим індексом.

Стовпці, за якими часто проводиться з'єднання таблиць, завжди повинні індексуватися, оскільки в цьому випадку дані розташовуються в порядку зростання індексу і з'єднання відбувається значно швидше.

Зі стовпцем таблиці, який оголошений головним ключем (primary), зазвичай зв'язується кластеризованний індекс, особливо тоді, коли він часто використовується при з'єднанні з іншими таблицями. (Слід пам’ятати, що у таблиці може бути тільки один кластеризованний індекс).

Зі стовпцем, в якому дані вибираються з деякого діапазону, доцільно пов'язати кластеризованний індекс. У цьому випадку як тільки буде знайдено перший рядок з потрібним значенням, всі наступні значення будуть розташовані поруч з ним. Кластеризованний індекс не такий ефективний при пошуку рядків з конкретними значеннями даних.

Випадки недоцільного використання індексів:

· Стовпці, які рідко використовуються в запитах, не варто індексувати, оскільки виграш у часі пошуку буде дуже маленьким.

· Стовпці, які містять всього два або три значення. Наприклад, чоловіча, жіноча стать або значення "так", "ні", також не варто індексувати.

Створення індексів для прискорення вибірки даних

Індекси зв'язуються зі стовпцем таблиці з метою прискорення вибірки даних. Найпростіша форма команди create index (створення індексу) має наступний вигляд:

```
create index teacher_id_ind on teacher (teacher_id)
```

Тобто команда для створення індексу в стовпці student_id таблиці students має наступний вигляд:

```
create index teacher_id_ind on students (student_id)
```

Назва індексу має задовольняти загальним правилам, встановленим для ідентифікаторів.

Назва стовпця і назва таблиці, вказують стовпець таблиці, з яким потрібно зв'язати індекс.

Індекси не можна пов'язувати зі стовпцями типу bit, text і image.

Користувач повинен бути власником таблиці, щоб мати право створювати або видаляти індекс. Власник таблиці може в будь-який час створити або видалити індекс незалежно від того, чи містить таблиця які-небудь дані чи ні. Індекси можна пов'язувати і з таблицями в іншій базі даних шляхом відповідного розширення назв таблиць.

Синтаксис команди створення індексів

Повний синтаксис команди створення індексів create index має наступний вигляд:

```
create [unique] [clustered | nonclustered]
 index назва_індекса
on [[база_даних.]власник.]назва_таблиці  назва_стовпця [,назва_стовпця]...)
[with {{fillfactor | max_rows_per_page}= x, ignore_dup_key, sorted_data,
 [ignore_dup_row | allow_dup_row]}]
[on назва_сегмента]
```

Опція назва_сегмента в команді створення індексу дозволяє розташувати індекс на зазначеному сегменті запам'ятовуючого пристрою. Перш ніж використовувати цю опцію, потрібно дізнатися у системного адміністратора або власника бази даних перелік сегментів, які можна використовувати. Деякі сегменти з міркувань підвищення продуктивності можна використовувати тільки для спеціальних таблиць або індексів.

Індексування за кількома стовпцями: складені індекси

Якщо у визначенні індексу вказуються назви декількох стовпців, то створюється складений індекс, з яким пов'язуються комбінації значень у зазначених стовпцях.

Складений індекс слід використовувати в тому випадку, коли потрібно вести пошук по комбінації значень у декількох стовпцях. У цьому випадку комбінацію значень даних можна розглядати як одне складне значення. Наприклад, припустимо, що в таблиці students потрібно зв'язати складений індекс зі стовпцями first_name і last_name. У цьому випадку в дужках треба вказати назви всіх стовпців, що включаються в складений індекс, як у наступному операторі:

```
create index nmind
on students (first_name, last_name)
```

Порядок задавання назв стовпців у цьому списку впливає на порядок сортування. Цей порядок може відрізнятися від порядку проходження цих стовпців в операторі створення таблиці create table.

У складений індекс можна включити до 16 стовпців. Всі ці стовпці повинні належати одній таблиці. Максимальний об’єм даних, з якими зв'язується складений індекс повинен бути не більше 256 байт. Іншими словами, сумарний об’єм пам’яті всіх стовпців, що включаються в складений індекс, повинна бути не більше 256 байт.

У складений індекс, як уже вказувалося, можна включати назви двох і більше стовпців. Ці стовпці разом зі стовпцем sensitivity(цей стовпець додається SQL Сервером автоматично) утворюють складений індекс. Складений індекс слід використовувати в тому випадку, коли комбінацію значень даних можна розглядати як одне складне значення, за яким зручно вести пошук. Наприклад, у таблиці students був визначений складений індекс, пов'язаний зі значеннями у стовпцях first_name і last_name і sensitivity. Як вже було зазначено, оператор визначення індексу в цій таблиці має наступний вигляд:

```
create index nmind
on students (first_name, last_name)
```

Використання опції unique

Унікальний (unique) індекс, пов'язаний з деяким стовпцем, перешкоджає появі в цьому стовпці двох однакових значень, включаючи невизначене значення NULL. У цьому випадку система сама перевіряє відсутність дубльованих значень, під час визначення такого індексу, якщо в таблиці вже є деякі дані, і така перевірка повторюється після кожного оператора модифікації даних insert або update.

Визначення унікального індексу має сенс лише в тому випадку, якщо дані у відповідному стовпці унікальні за своєю природою. Наприклад, було б помилково пов'язувати унікальний індекс зі стовпцем last_name (прізвище), оскільки навіть у невеликій таблиці з декількох сотень рядків можуть зустрітися два рядки.

З іншого боку, доцільно пов'язати унікальний індекс зі стовпцем, в якому міститься номер залікової книжки, оскільки ці номери різні в різних людей і отже в цьому випадку унікальність є властивістю самих даних. Крім того, унікальний індекс використовується для перевірки цілісності даних. Наприклад, поява двох однакових залікових книжок може бути викликано або помилкою введення, або помилкою відповідної державної установи.

Якщо користувач спробує створити унікальний індекс в табличному стовпці, де є повторювані значення, то відповідна команда буде перервана і SQL Сервер згенерує повідомлення про помилку, в якому буде вказано перше повторюване значення.

Якщо дані змінюються у стовпці, з яким вже пов'язаний унікальний індекс, то результат буде залежати від опції ignore_dup_key.

Ключове слово unique (унікальний) можна використовувати і для складених індексів.

Але воно не використовувалось в нашому прикладі при визначенні складеного індексу в таблиці students.

Використання лічильників в неунікальних індексах

Опція identity in nonunique index (лічильник в не унікальному індексі) автоматично включає стовпець лічильника в усі індекси таблиці, тим самим роблячи їх всі унікальними. Ця опція бази даних робить логічно неунікальні ключі фактично унікальними, що дозволяє використовувати їх в оновлюваних курсорах (uptable cursors) і при зчитуванні на нульовому рівні ізоляції (isolation level 0 reads).

Якщо курсор є обновлюваним, то через нього можна оновлювати вміст рядків або видаляти рядки повністю. Якщо курсор призначений тільки для читання, то через нього можна тільки зчитувати дані. За замовчуванням SQL Сервер намагається зробити курсор обновлюваним і якщо це не вдається, то курсор призначається для читання.

Мінімальний рівень ізоляції, гарантує тільки фізичну цілісність при запису даних. Процеси-читачі можуть зчитувати дані незавершеною транзакції процесу-записувача.

Таблиця повинна вже мати стовпець лічильника, який повинен бути визначений або в операторі створення таблиці, або включений в неї автоматично, якщо перед виконанням цього оператора була включена опція auto identity.

Опцію identity in nonunique index слід використовувати в тому випадку, якщо користувач планує працювати з курсором або виконувати зчитування на нульовому рівні в таблиці з неунікальним індексами. Унікальний індекс забезпечує установку курсору на потрібному рядку, перед тим як виконується наступна операція fetch (завантаження) за допомогою цього курсору.

Використання опцій fillfactor і max_rows_per_page

Досить рідко у користувача виникає необхідність зайнятися тонким налаштуванням продуктивності системи, і в цьому випадку він може використовувати опції fillfactor (фактор щільності) і max_rows_per_page (максимальне число рядків на сторінку). Ці опції має сенс використовувати тільки при створенні індексів для таблиць, які вже містять деякі дані.

FILLFACTOR

За допомогою опції fillfactor користувач задає ступінь заповнювання індексної сторінки, яку повинен підтримувати SQL Сервер. Тут береться до уваги кількість вільної пам'яті, яку можна залишити на індексній сторінці, оскільки при повному заповненні сторінки системі потрібен додатковий час, щоб звільнити місце для інформації, що надійшла.

За замовчуванням цей фактор (fillfactor) дорівнює 0 і це ж значення використовується, якщо користувач не вказав опції заповнення. Системний адміністратор може змінити значення цього чинника, що приймається за замовчуванням, за допомогою системної процедури sp_configure (конфігурація).

Правильні значення, які може вказати користувач, змінюються в інтервалі від 1. Далі наведено приклад оператора створення індексу з використанням опції fillfactor:

```
create index postalcode_ind
on students(postalcode)
with fillfactor = 100
```

Якщо значення чинника щільності дорівнює 100, то повністю заповнюється кожна сторінка. Такий фактор доцільно задавати в тому випадку, якщо користувач точно знає, що ніяке індексоване значення в таблиці не змінюватиметься.

MAX_ROWS_PER_PAGE

Опція max_rows_per_page обмежує число рядків, які SQL Сервер може розмістити на одній сторінці пам'яті. Мале значення цього параметра обмежує блокування і має сенс тільки для часто використовуваних таблиць. Малі значення цього параметра призводять до додаткової витрати пам'яті для індексу.

За замовчуванням значення цієї опції дорівнює 0, це ж значення використовується, якщо користувач не вказав ніякого максимуму. Користувач може змінити це значення, прийняте за замовчуванням, за допомогою системної процедури sp_relimit.

Правильне значення цього параметра, що задається користувачем, лежить в інтервалі від 1 до 256.

Наступний оператор створення індексу використовує опцію max_rows_per_page:

```
create index postalcode_ind
on students(postalcode)
with max_rows_per_page = 10
```

Використання кластеризованих і некластеризованих індексів

За наявності кластеризованого індексу SQL Сервер впорядковує рядки таблиці у відповідності зі значеннями цього індексу таким чином, що фізичне розташування рядків відповідає їх логічному розташуванню в таблиці. Самий нижній або листовий (leaf) рівень кластеризованого індексу відповідає фізичним сторінкам табличних даних. Кластеризований індекс потрібно створювати перед визначенням будь-яких некластеризованих індексів, оскільки некластеризовані індекси автоматично перебудовуються при появі кластеризованого індексу.

За визначенням у таблиці може бути тільки один кластеризованний індекс. Часто такий індекс створюється для головного ключа (primary key), тобто стовпця або декількох стовпців, значення в яких однозначно визначають цей рядок.

Логічно головний ключ вибирається на етапі проектування бази даних. Однак користувач може явно визначити головні ключі, зовнішні ключі і загальні ключі (пари ключів часто використовувані при з'єднаннях) за допомогою системних процедур sp_primarykey, sp_foreignkey і sp_commonkey. Інформацію про ключі можна отримати за допомогою системних процедур sp_helpkey, а інформацію про стовпці, за якими доцільно здійснювати з'єднання за допомогою системної процедури sp_helpjoins.

Головний ключ у вигляді обмеження цілісності даних можна також визначити в операторах створення таблиці (create table) або зміни таблиці (alter table), тим самим створивши табличний індекс. Інформацію про обмеження цілісності можна отримати за допомогою системної процедури sp_helpconstraint.

За наявності некластеризованого індексу фізичне розташування рядків таблиці може не відповідати їх індексованому розташуванню. На листовому (нижньому) рівні некластеризованого індексу розташовуються вказівники (адреси) місця розташування рядків таблиці на сторінках даних. Тобто кожна листова сторінка містить значення індексу і вказівник на рядок, відповідний цьому значенню. Іншими словами, некластеризований індекс має додатковий рівень між індексною структурою і власне табличними даними.

З одною таблицею може бути пов'язано до 249 некластеризованих індексів, які дозволяють вибирати дані з таблиці в різному порядку.

Пошук даних за наявності кластеризованого індексу майже завжди здійснюється швидше ніж за наявності некластеризованого індексу. Крім того, кластеризований індекс забезпечує більш швидку вибірку масиву рядків, відповідних послідовним значенням індексу, тобто масиву відповідного діапазону значень в індексованому стовпці. Як тільки буде знайдено перший рядок з потрібним значенням, всі інші рядки можна вибирати відразу без додаткового пошуку.

Якщо при визначенні індексу не вказані ключові слова clustered (кластеризований) або nonclustered (некластеризованний), то створюється некластеризований індекс.

Далі наводиться приклад створення індексу, пов'язаного зі стовпцем student_id, таблиці students (якщо користувач дійсно збирається виконати цю команду, то необхідно потім видалити цей індекс командою drop index):

```
create clustered index studentind
on students (student_id)
```

Якщо необхідно розташувати студентів по їх поштових індексах, то в таблиці students потрібно створити некластеризований індекс, пов'язаний зі стовпцем postalcode, наступною командою:

```
create nonclustered index postalcodeind on students (postalcode)
```

Не має сенсу визначати унікальний індекс у цьому операторі, оскільки дуже ймовірно, що деякі студенти будуть мати однаковий поштовий індекс. Тут також не можна визначати кластеризований індекс, оскільки поштовий індекс не є головним ключем у цій таблиці.

Кластеризований індекс в таблиці students повинен бути складеним і повинен включати в себе поля з ім'ям і прізвищем. Для створення кластеризованого індексу потрібно попередньо видалити, раніше визначений некластеризований індекс nmind, за допомогою наступного оператора:

```
drop index students.nmind
```

Потім вже можна визначити кластеризований індекс:

```
create clustered index nmind
on students (first_name, last_name)
```

Зауваження: Оскільки нижній (листовий) рівень кластеризованого індексу містить за визначенням сторінки табличних даних, то використання розширення on назва_сегменту в операторі створення кластеризованого індексу викликає переміщення таблиці з пристрою, на якому вона була створена, на вказаний сегмент запам'ятовуючого пристрою.

Перш ніж створювати таблиці або індекси на конкретних сегментах запам'ятовуючого пристрою, необхідно дізнатися у системного адміністратора або власника бази даних які сегменти можна використовувати, оскільки деякі сегменти можуть бути зарезервовані для інших цілей.

Використання індексних опцій

Індексні опції ignore_dup_key (ігнорувати дублювання ключа), ignore_dup_row (ігнорувати повторювані рядки) та allow_dup_row (дозволити дублювання рядків) визначають реакцію системи на появу повторюваних значень при виконанні операторів вставки та оновлення (insert, update). У наступній таблиці наведено умови, коли можна використовувати ці опції:

Індексні опції

Тип індексу	Опція
Кластеризований	ignore_dup_row allow_dup_row
Унікальний кластеризований	ignore_dup_key
Некластеризований	
Унікальний некластеризований	ignore_dup_key
Унікальний некластеризований	ignore_dup_row

Використання опції ignore_dup_key

Якщо користувач намагається вставити повторюване значення в табличний стовпець, який має унікальний індекс, то команда модифікації переривається. Після цієї відміни може виконуватися будь-яка активна транзакція наче не було ніяких спроб оновлення даних. Щоб система не перервала при цьому виконання всієї транзакції, користувач може включити опцію ignore_dup_key у визначення унікального індексу (unique).

Користувач не може пов'язувати унікальний індекс зі стовпцем, в якому вже перебувають однакові значення, незалежно від того вказана опція ignore_dup_key чи ні. Якщо користувач все ж зробить спробу створити такий індекс, то SQL Сервер видасть повідомлення про помилку і покаже однакові значення.

Наведемо приклад використання опції ignore_dup_key:

```
create unique clustered index phone_ind
on students(phone)
with ignore_dup_key
```

Використання опцій ignore_dup_row і allow_dup_row

Опції ignore_dup_row і allow_dup_row використовуються при створенні неунікальних кластеризованих індексів. Ці опції не мають значення при створенні неунікальних некластеризованиих індексів, оскільки SQL Сервер заводить для будь-якого некластеризованого індексу свій внутрішній унікальний номер рядка і тому в цьому випадку виникнення однакових значень неважливо.

Опції ignore_dup_row і allow_dup_row є взаємовиключними.

Якщо вказана опція allow_dup_row (дозволити дублювання рядків), то можна визначити новий неунікальний кластеризований індекс в таблиці, яка містить однакові рядки, і в подальшому записувати значення, які з’явилися в результаті виконання операцій модифікації даних insert і update.

Якщо який-небудь з табличних індексів унікальний, то вимога унікальності є пріоритетною по відношенню до інших функцій, тобто вона продовжує діяти (по відношенню до значень у зв'язаних з індексом стовпцях) навіть тоді, коли для іншого індексу вказана опція allow_dup_row. Таким чином, цю опцію є сенс використовувати тільки в таблицях, які не мають унікальних індексів. Цю опцію не можна вказувати, якщо в таблиці вже є унікальний кластеризованний індекс.

Опція ignore_dup_row (ігнорувати повторювані рядки) використовується для усунення повторень в таблиці даних. Коли вставляється рядок, який вже зустрічався, то цей рядок ігнорується, а відповідний оператор вставки переривається і видається повідомлення про помилку. Недубльовані рядки вставляються звичайним чином.

Опцію ignore_dup_row можна застосовувати лише до таблиць з неунікальними індексами.

Таким чином, її не можна використовувати, якщо таблиця має принаймні один унікальний індекс.

Зауваження: оператор оновлення скасовується, якщо при його виконанні записується рядок, який вже зустрічався. Після цієї відміни може виконуватися будь-яка активна транзакція наче б не було ніяких спроб оновлення даних.

Наступна таблиця ілюструє, як впливають опції ignore_dup_row і allow_dup_row на створення неунікального кластеризованого індексу в таблиці, в якій вже є однакові рядки, або на спробу запису таких рядків у таблицю.

Індексні опції, пов'язані з дублюванням рядків

Опція	Існують повторення	Вводяться повторення
Немає опцій	Команда створення індексу **creat index** не виконується.	Відміняється виконання поточної команди.
Вказана опція **allow_dup_row**	Команда створення індексу повністю виконується.	Поточна команда повністю виконується.
вказана опція **ignore_dup_row**	Індекс створюється, але дублікати рядків видаляються. Видається повідомлення про помилку.	Всі рядки вставляються за виключенням дублікатів. Видається повідомлення про помилку. Див. попередні застереження.

Використання опції sorted_data

Опція sorted_data (сортовані дані) прискорює створення індексу, коли табличні дані вже розташовані в потрібному порядку, наприклад, якщо вони були скопійовані в пусту таблицю з уже відсортованої таблиці за допомогою процедури bcp. Програма bcp використовується для масового копіювання даних між примірником Microsoft SQL Server і файлом даних в призначеному для користувача форматі. За допомогою програми bcp можна виконувати імпорт великої кількості нових рядків у таблиці SQL Server або експорт даних з таблиць у файли даних. Економія часу стає дуже помітною на великих таблицях і зростає в кілька разів, коли розмір таблиці більше гігабайта. Цю опцію можна використовувати разом з іншими індексними опціями, оскільки вона від них повністю незалежна.

Якщо вказана опція sorted_data, а розташування табличних даних не відповідає значенням індексу, то видається повідомлення про помилку і команда не виконується.

Ця опція прискорює створення тільки кластеризованих і унікальних некластеризованих індексів. Однак, створення неунікального некластеризованого індексу (з цією опцією) буде успішно завершено, якщо в таблиці немає однакових значень ключа. Якщо такі значення є, то буде видане повідомлення про помилку і команда не буде виконуватися.

Використання опції on segment_name

У виразі on назва_сегмента можна вказати назву сегмента, на якому слід створити індекс. Некластеризований індекс може бути створений на сегменті, відмінному від сегментів, на яких розташовані сторінки даних. Наприклад:

```
create index bookind on books(title)
on seg1
```

Видалення індексів

Команда drop index (видалення індексу) використовується для видалення індексів з бази даних. Ця команда має наступний синтаксис:

```
drop index назва_таблиці.назва_індекса [,назва_таблиці.назва_індекса]...
```

Коли виконується ця команда, SQL Сервер видаляє зазначені індекси з бази даних і звільняє займану ними пам'ять.

Тільки власник індексу може видалити його. Права на видалення індексів не можуть передаватися іншим користувачам. Не можна видаляти індекси системних таблиць, розташованих в базі даних master, або в базі даних користувача.

Індекс корисно видалити в тому випадку, якщо він не використовується в більшості запитів. Щоб видалити індекс phone_ind в таблиці students, потрібно виконати наступну команду:

```
drop index students.phone_ind
```

Інформація про табличні індекси

Щоб отримати інформацію про індекси в таблиці, можна скористатися системною процедурою sp_helpindex. Наприклад, при виконанні команди буде наступний результат:

```
sp_helpindex friends_etc
```

```
index_name     index_description                 index_keys
-------------   --------------------------------  -------------
nmind          clustered located on default      firtst_name, last_name
postalcode_ind nonclustered located on default   postalcode
postalcodeind  nonclustered located on default   postalcode

(3 rows affected, return status = 0)
```

Процедура sp_help також видає інформацію про індекси таблиці.

Оновлення індексної статистики

Команда update statistics (оновлення статистики) допомагає SQL Серверу зробити оптимальний вибір індексів при виконанні запитів шляхом запам'ятовування самого останнього розподілу ключових значень в індексах. Цю команду слід використовувати, коли велика кількість даних додається, змінюється або видаляється з індексного стовпця.

Право на використання цієї команди за замовчуванням належить власникові таблиці і не може передаватися іншим особам. Ця команда має наступний синтаксис:

```
update statistics назва_таблиці [назва_індекса]
```

Якщо назву індексу в цій команді не вказано, то оновлюється статистика розподілу значень для всіх індексів таблиці. Якщо ж назву індексу вказано, то оновлюється лише статистика для цього індексу.

Назви табличних індексів, як було зазначено, можна дізнатися за допомогою системної процедури sp_helpindex. Параметром цієї процедури є назва таблиці.

Наступна команда показує як можна дізнатися індекси таблиці teacher:

```
sp_helpindex teacher
```

Результат виконання:

```
index_name     index_description   index_keys
----------     -----------------   ----------
teacher_id_ind clustered,unique    teacher_id
```

Щоб оновити статистику для всіх індексів цієї таблиці, потрібно виконати наступну команду:

```
update statistics teacher
```

Щоб оновити статистику, що стосується тільки індексу, пов'язаному зі стовпцем teacher_id, потрібно виконати наступну команду:

```
update statistics teacher teacher_id_ind
```

Оскільки назви індексів у різних таблиць можуть збігатися, то в цій команді потрібно вказати назву таблиці, з якою пов'язаний даний індекс. SQL Сервер виконує команду оновлення статистики update statistics автоматично, коли створюється індекс в таблиці з уже записаними даними.

Створення індексів в PostgreSQL

Індекс – спеціалізована структура даних, котра усуває необхідність повного перебору екземплярів даних при виконанні запиту до БД. Індексні структури забезпечують оптимізацію запитів до СБД, зменшуючи кількість запитів до файлу бази даних, розміщеного в зовнішній пам’яті.

Прискорення перебору екземплярів даних БД теж має межі. Без індексів довелося б переглянути всі екземпляри даних для з’ясування необхідності зчитування окремих рядків таблиці. Продемонструємо дане твердження на прикладі.

Таблиця 3.1. Відповідність екземплярів даних таблиці до умови запиту

Чи відповідає «Christmas Day»?	Ні	LARP Club	2	1
Чи відповідає «Christmas Day»?	Ні	April Fools Day		2
Чи відповідає «Christmas Day»?	Так	Christmas Day		3

В відповідь на запит створення таблиці СУБД «CREATE TABLE», PostgreSQL автоматично створює індекс за первинним ключем. Ключем індексу events.index є значення стовпця events.event_id, а «значенням» – посилання на розташування в зовнішній пам’яті.

Таблиця 3.2. Індекс за значенням первинного ключа

events.index	events.title	venues.venue_id	events.event_id
1	LARP Club	2	1
2	April Fools Day		2
3	Christmas Day		3

Для створення індексу за конкретним стовпцем зазначають умову UNIQUE.

Декларативно індекс створюється запитом CREATE INDEX.

```
CREATE INDEX events_index ON events USING hash (event_id);
```

```
booking=# CREATE INDEX events_index ON events
CREATE INDEX
```

Рисунок 3.1. Створено хеш-індекс

За потреби в співставленні значень стовпців за операторами «<», «>», індекс необхідно зберігати в гнучкішій структурі даних, наприклад, B-дереві.

Сформуємо запит зчитування всіх заходів з датою проведення «не раніше 1-го квітня».

```
SELECT * FROM events WHERE starts >= '2012-04-01';
```

```
booking=# SELECT * FROM events WHERE starts
 event_id |      title       |   starts   |
----------+------------------+------------+--
        2 | April Fools Day | 2012-04-01 | 2
```

Рисунок 3.2. Знайдено екземпляри даних за умовою щодо значень

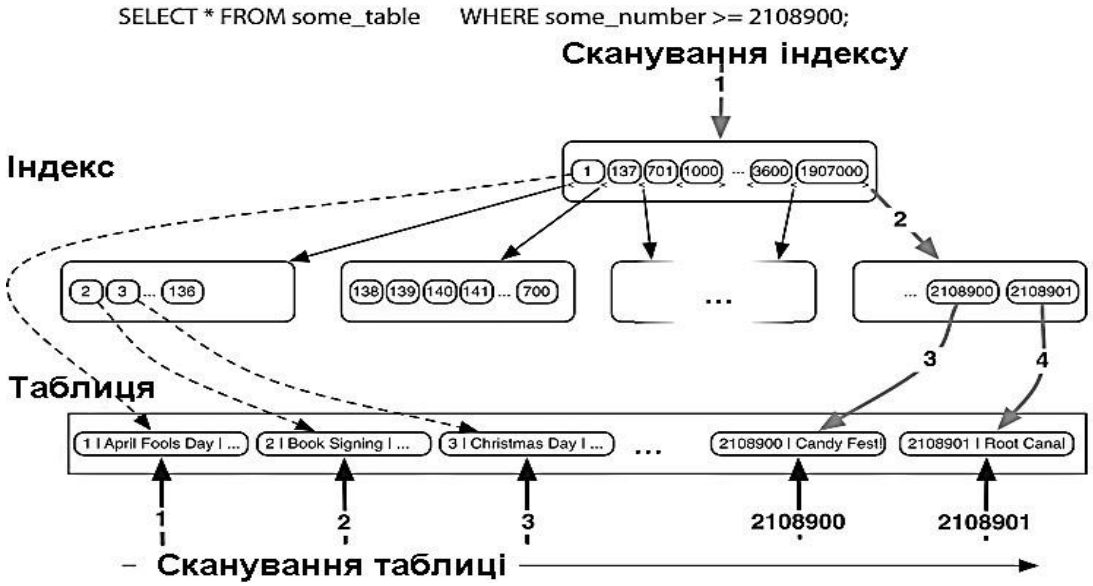


Рисунок 3.3. Зчитано значення за інтервалом з індексу зі структурою B-дерево

Для даного запиту ідеальною структурою є дерево. Для побудови B-дерева (btree) за полем events.starts побудуємо відповідний індекс.

CREATE INDEX events_starts ON events USING btree (starts);

```
booking=# CREATE INDEX events_starts ON events
```

Рисунок 3.4. Створено індекс зі структурою B-дерево

Тепер при виконанні запиту за діапазоном дат немає потреби в перегляді всієї таблиці, так як при перегляді млн. або млрд. екземплярів даних різниця в затримці відповіді СБД буде значною.

СБД автоматично створює індекс за стовпцями з умовою FOREIGN KEY. Зчитаємо список всіх індексів бази даних – « booking=# \di ».

```
booking=# booking=# \di
                    List of relations
 Schema |              Name              | Type
--------+--------------------------------+-------
 public | cities_pkey                    | index
 public | countries_country_name_key     | index
 public | countries_pkey                 | index
 public | events_index                   | index
```

Рисунок 3.5. Зчитано наявні індекси

III. Завдання на лабораторну роботу

1. Реалізувати запити на виконання інструкцій INSERT, UPDATE, DELETE з лабораторної роботи №2 у вигляді збережених процедур (по одній на кожен тип операції).

2. Створити збережену процедуру, яка б створювала нову таблицю Admission. Ця таблиця повинна включати id студента та дату його зарахування.

3. Створити тригер, що додає нові значення у таблицю Admission одночасно з додаванням нового студента в таблицю Student, дата зарахування формується як час створення запису. Продемонструвати його працездатність на прикладі додавання трьох студентів.

4. Створити збережену процедуру, яка б створювала нову таблицю TheBestOfTheBest. Ця таблиця повинна включати П.І.Б 5-х найкращих студентів (за фактичним рейтингом таблиці Table_Subject_Success), Назву групи, Назву предмета по якому в студента найвищий бал і П.І.Б. викладача , що викладає цей предмет.

5. Виконати транзакцію над довільною таблицею. Видаліть рядки з id=2;3;6, створюючи точки збереження перед кожним видаленням. Перевірте точки збереження.

6. Виконати транзакцію над довільною таблицею. Змініть числовий параметр в одному з атрибутів з довільним id на +10 і -5 для іншого id.

7. У своїх базах даних для кожної з таблиць додайте індекси, які можуть використовуватися оптимізатором запитів для виконання цих запитів.

8. Виведіть на екран інформацію про індекси, що містяться у ваших таблицях.

V. Контрольні питання

1. Що таке тригер, процедура?
2. З якими правами користувач може створювати тригер в наявній БД?
3. Які є переваги використання тригерів в базах даних?
4. Що таке транзакція? Для чого її використовують?
5. Що свідчить про закінчення транзакції?
6. Назвіть основні властивості транзакцій і що вони собою являють.
7. Які переваги і недоліки використання транзакцій?
8. Для чого використовують команди COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION.
9. Поясніть як вирішується проблема паралельних транзакцій.

Перелік літератури

1. Глоба Л.С. Створення та обробка баз даних: навч. посібник для студ. / Л. С. Глоба, М. Ю. Терновой, Р. Л. Новогрудська, О. С. Штогрина. – Київ: НТУУ "КПІ", 2013. – 477 с.
2. Грабер М. Введение в SQL / Мартин Грабер. – М.: Лори, 2010. – 227 с.
3. Пасічник В. В. Сховища даних навчальний посібник / В. В. Пасічник, Н. Б. Шаховська. – Львів: Видавництво "Магнолія 2006", 2020. – 490 с.
4. Гороховатський В. О. Методи інтелектуального аналізу та оброблення даних навчальний посібник / В. О. Гороховатський, І. С. Творошенко. – Харків: ХНУРЕ, 2021. – 90 с.
5. Гайна Г. А. Основи проектування баз даних навчальний посібник для студентів вищих навчальних закладів, що навчаються за напрямом підготовки 0804 "Комп'ютерні науки" / Г. А. Гайна. – Київ: Кондор, 2021. – 204 с.
6. Сергеєв-Горчинський О. О. Системи баз даних: Комп’ютерний практикум [Електронний ресурс] : навч. посіб. для студ. спеціальності 122 «Комп’ютерні науки» і спеціалізації «Інтелектуальні сервіс-орієнтовані розподілені обчислювання», спеціальності 172 «Телекомунікації та радіотехніка» і спеціалізації «Інформаційно-комунікаційні технології» / О. О. Сергеєв-Горчинський, Л. С. Глоба ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 3 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 139 с.: Іл.
7. Швець М. Ю. Порівняння SQL та NoSQL баз даних / М. Ю. Швець, Д. С. Заруба, Ю. В. Хохлов // Вчені записки Таврійського національного університету імені В. І. Вернадського. Серія : Технічні науки. - 2018. - Т. 29(68), № 6(2). - С. 21-25. - Режим доступу: http://nbuv.gov.ua/UJRN/sntuts_2018_29(68)_6(2)__7.
8. Нікітіна Т. С. Порівняльний аналіз продуктивності баз даних SQL та NOSQL / Т. С. Нікітіна, О. І. Морозова // Системи управління, навігації та зв'язку. - 2019. - Вип. 1. - С. 125-128. - Режим доступу: http://nbuv.gov.ua/UJRN/suntz_2019_1_26.
9. Мулеса О. Ю. Особливості використання додатку PHPMYADMIN в ході вивчення мови запитів SQL / О. Ю. Мулеса, Ф. Е. Гече, Г. М. Розлуцька // Фізико-математична освіта. - 2017. - Вип. 4. - С. 234-238. - Режим доступу: http://nbuv.gov.ua/UJRN/fmo_2017_4_46.
10. Фісун М. Т. Порівняльний аналіз засобів Data Mining у СКБД SQL SERVER та ORACLE / М. Т. Фісун, Є. О. Давиденко, О. М. Крайник // Проблеми інформаційних технологій. - 2016. - № 1. - С. 231-238. - Режим доступу: http://nbuv.gov.ua/UJRN/Pit_2016_1_28.
11. Брацький В. О. Дослідження особливостей застосування реляційних і нереляційних баз даних на прикладі SQL Server та MongoDB / В. О. Брацький, О. М. М’якшило // Наукові праці Національного університету харчових технологій. - 2016. - Т. 22, № 5. - С. 15-24. - Режим доступу: http://nbuv.gov.ua/UJRN/Npnukht_2016_22_5_4.
12. Буй Д. Б. Композиційна семантика SQL-подібних мов та суміжні питання / Д. Б. Буй, Н. Д. Кахута, О. В. Шишацька // Вісник Київського національного університету імені Тараса Шевченка. Серія : Фізико-математичні науки. - 2013. - Вип. 1. - С. 116-121. - Режим доступу: http://nbuv.gov.ua/UJRN/VKNU_fiz_mat_2013_1_22.
13. Тарасов О. В. Особливості використання мови визначення даних SQL у сучасних СКБД / О. В. Тарасов // Системи обробки інформації. - 2012. - Вип. 8. - С. 50-53. - Режим доступу: http://nbuv.gov.ua/UJRN/soi_2012_8_14.
14. Жученко, А. І. Основи проектування баз даних [Електронний ресурс] : навчальний посібник для студентів спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / А. І. Жученко, Л. Д. Ярощук ; НТУУ «КПІ». – Електронні текстові дані (1 файл: 2, 50 Мбайт). – Київ : НТУУ «КПІ», 2015. – 158 с. – Назва з екрана.

Додаток А

Схема бази даних для предметної області «Навчальний процес»

Додаток Б

Детальний опис таблиць для предметної області «Навчальний процес»

Опис таблиці «СТУДЕНТ»

Ім’я поля	Ключове поле	Тип даних	Пояснення	Умова на значення	Повідомлення про помилку
GroupID		Числовий	Унікальний ідентифікатор групи		
StudentID	Так	Числовий	Унікальний ідентифікатор студента		
Number		Числовий	Номер студента в групі	0…30	Невірно задане значення
LastName		Текстовий	Прізвище студента		
FirstName		Текстовий	Ім’я студента		
BirthDate		Дата	Дата народження	1950…поточний рік	Невірно задане значення
Adress		Текстовий	Адреса студента		
P_ball		Числовий	Прохідний бал студента	3…5	Невірно задане значення

Опис таблиці «ГРУПА»

Ім’я поля	Ключове поле	Тип даних	Пояснення	Умова на значення	Повідомлення про помилку
GroupID	Так	Числовий	Унікальний ідентифікатор групи		
ChairID		Числовий	Унікальний ідентифікатор кафедри		
Name		Числовий	Назва групи	0…25 символів	Невірно задане ім’я
MaxCountStudents		Числовий	Кількість студентів в групі	0…30	Кількість студентів більше допустимої
P_ball		Числовий	Прохідний бал	2…5	Невірно задане значення

Опис таблиці «КАФЕДРА»

Ім’я поля	Ключове поле	Тип даних	Пояснення	Умова на значення	Повідомлення про помилку
ChairID	Так	Числовий	ID номер кафедри		
Name		Текстовий	Назва	10 символів	Назва введена невірно
Phone		Текстовий	Телефон	Виду 111-111-11	Невірний вид введених даних
ManagerID		Числовий	ПІБ зав. кафедри		

Опис таблиці «ВИКЛАДАЧ»

Ім’я поля	Ключове поле	Тип даних	Пояснення	Умова на значення	Повідомлення про помилку
TeacherID	Так	Числовий	Унікальний ідентифікатор викладача		
ChairID		Числовий	Унікальний ідентифікатор кафедри		
LastName		Текстовий	Прізвище викладача		
FirstName		Текстовий	Ім’я викладача		
PatronymicName		Текстовий	По-батькові викладача		
Degree		Текстовий	Вчена степінь		
Title		Текстовий	Вчене звання	“професор” чи “доцент”	Невірно введені дані
Photo		Поле об’єкта OLE	Фотографія		

Опис таблиці «ПРЕДМЕТ»

Ім’я поля	Ключове поле	Тип даних	Пояснення	Умова на значення	Повідомлення про помилку
SubjectID	Так	Числовий	Унікальний ідентифікатор предмета		
Name		Текстовий	Назва предмета		
Program		Поле МЕМО	Програма предмета		

Опис таблиці ВИВЧЕННЯ

Ім’я поля	Ключове поле	Тип даних	Підпис поля	Умова на значення	Повідомлення про помилку
GroupID	Так	Числовий	Унікальний ідентифікатор групи		
SubjectID	Так	Числовий	Унікальний ідентифікатор предмета		
TeacherID	Так	Числовий	Унікальний ідентифікатор викладача		
ExamTypeID		Числовий	Унікальний ідентифікатор типу контрольного заходу		
SubjectTypeID		Числовий	Унікальний ідентифікатор виду занять		
Hours		Числовий	Кількість годин		
Semesters		Числовий	Кількість семестрів		
Access_mark_to_exam		Числовий	Середній бал по предмету	1…5	Невірно задане значення

Опис таблиці «УСПІШНІСТЬ»

Ім’я поля	Ключове поле	Тип даних	Підпис поля	Умова на значення	Повідомлення про помилку
StudentID	Так	Числовий	Унікальний ідентифікатор студента		
SubjectID	Так	Числовий	Унікальний ідентифікатор предмета		
ExamTypeID	Так	Числовий	Унікальний ідентифікатор типу контрольного заходу		
Mark		Числовий	Оцінка	1…5	Невірно задане значення
Date		Дата	Дата сдачі		

Опис таблиці «ТИП КОНТРОЛЬНОГО ЗАХОДУ»

Ім’я поля	Ключове поле	Тип даних	Підпис поля	Умова на значення	Повідомлення про помилку
ExamTypeID	Так	Числовий	Унікальний ідентифікатор типу контрольного заходу		
Name		Текстовий	Назва типу контрольного заходу	«екзамен», «залік», «диф. залік»	Невірно задане значення

Опис таблиці «ТИП ЗАНЯТЬ»

Ім’я поля	Ключове поле	Тип даних	Підпис поля	Умова на значення	Повідомлення про помилку
SubjectTypeID	Так	Числовий	Унікальний ідентифікатор виду занять		
Name		Текстовий	Вид занять	«лекція», «практика», «лабораторна робота»	Невірно задане значення