Лабораторна робота

Тема: Початок роботи з Microsoft SQL Server. Основи T-SQL. DDL.

Мета: Створення та видалення бази даних. Створення та видалення таблиць. Типи даних T-SQL. Атрибути та обмеження стовпців та таблиць. Зовнішні ключі. Зміна таблиці. Пакети. Команда GO.

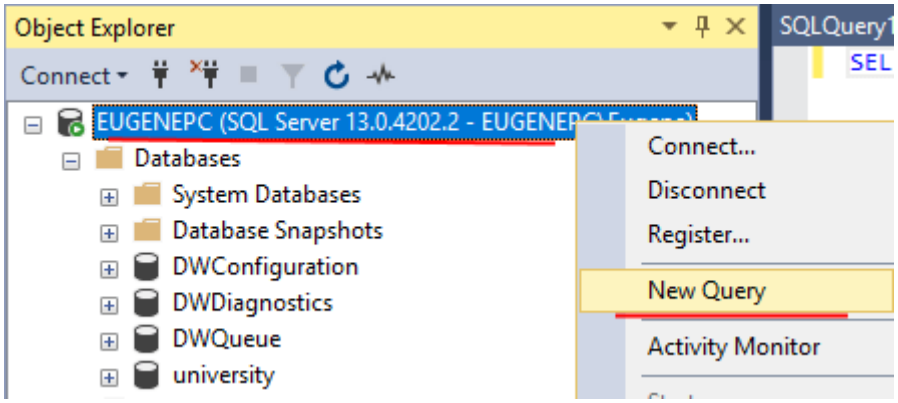
Хід роботи

Створення та видалення бази даних

Створення бази даних

Для створення бази даних використовується команда CREATE DATABASE.

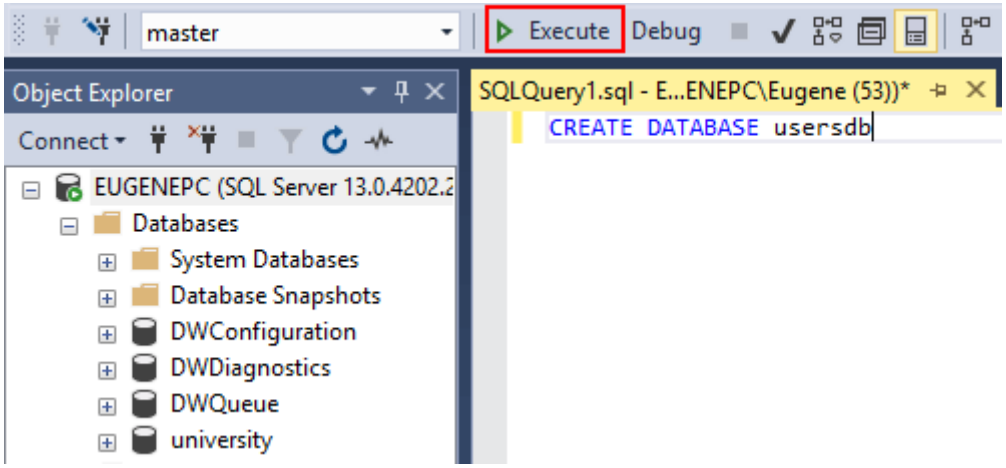
Щоб створити нову базу даних, відкриємо SQL Server Management Studio. Натиснемо на призначення сервера у вікні Object Explorer і в меню виберемо пункт New Query.



У центральне поле для введення виразів SQL введемо наступний код:

```
1   CREATE DATABASE usersdb
```

Тим самим ми створюємо базу даних, яка називатиметься "usersdb":



Для виконання команди натисніть на панелі інструментів кнопку Execute або на клавішу F5. І на сервері з'явиться нова база даних.

Після створення бази даних, ми можемо встановити її як поточну за допомогою команди USE:

```
1 USE usersdb;
```

Прикріплення бази даних

Можлива ситуація, що ми вже маємо файл бази даних, який, наприклад, створено іншому комп'ютері. Файл бази даних є файлом з розширенням mdf, і цей файл в принципі ми можемо переносити. Однак навіть якщо ми скопіюємо його комп'ютер із встановленим MS SQL Server, просто так скопійована база даних на сервері не з'явиться. Для цього необхідно виконати прикріплення бази даних до сервера. У цьому випадку застосовується вираз:

```
1 CREATE DATABASE db_name
2 ON PRIMARY(FILENAME=file_path')
3 FOR ATTACH;
```

Як каталог для бази даних краще використовувати каталог, де зберігаються інші бази даних сервера. Наприклад, нехай у нашому випадку файл із даними називається userstoredb.mdf. І ми хочемо додати цей файл на сервер як базу даних. Спочатку його треба скопіювати у зазначений каталог. Потім для прикріплення бази до сервера слід використовувати таку команду:

```
1   CREATE DATABASE contactsdb
2   ON PRIMARY(FILENAME='your_file_path')
3   FOR ATTACH;
```

Після виконання команди на сервері з'явиться база даних contactsdb.

Видалення бази даних

Для видалення бази даних застосовується команда DROP DATABASE, яка має наступний синтаксис:

```
1   DROP DATABASE database_name1 [, database_name2]...
```

Після команди через кому ми можемо перерахувати всі бази даних, що видаляються. Наприклад, видалення бази даних contactsdb:

```
1 DROP DATABASE contactsdb
```

Варто враховувати, що навіть якщо база даних, що видаляється, була прикріплена, то все одно будуть видалені всі файли бази даних.

Створення та видалення таблиць

Для створення таблиць використовується команда CREATE TABLE. З цією командою можна використовувати низку операторів, які визначають стовпці таблиці та їх атрибути. Крім того, можна використовувати ряд операторів, які визначають властивості таблиці в цілому. Одна база даних може містити до 2 мільярдів таблиць.

Загальний синтаксис створення таблиці виглядає так:

```
1   CREATE TABLE table_name
2   (column_name1 data_type column_attributes1,
3   column_name2 data_type column_attributes2,
4   ..........................................
5   column_nameN data_type column_attributesN,
6   атрибути_таблиці
7   )
```

Після команди CREATE TABLE йде назва створюваної таблиці. Ім'я таблиці виконує роль її ідентифікатора у базі даних, тому має бути унікальним. Ім'я повинно мати не більше 128 символів. Ім'я може складатися з алфавітно-цифрових символів, символів $ і символу підкреслення. Причому першим символом має бути буква чи знак підкреслення.

Ім'я об'єкта не може включати пробіли і не може представляти одне з ключових слів Transact-SQL. Якщо ідентифікатор все ж таки містить пробільні символи, його слід укладати в лапки. Якщо необхідно як ім'я використовувати ключові слова, ці слова поміщаються в квадратні дужки.

Приклади коректних ідентифікаторів:

```
1   Users
2   tags$345
3   users_accounts
4   "users accounts"
5   [Table]
```

Після імені таблиці в дужках вказуються параметри всіх стовпців і наприкінці атрибути, які стосуються всієї таблиці. Атрибути стовпців та атрибути таблиці є необов'язковими компонентами і їх можна не вказувати.

У самому простому вигляді команда CREATE TABLE повинна містити як мінімум ім'я таблиці, імена та типи стовпців.

Таблиця може містити від 1 до 1024 шпальт. Кожен стовпець повинен мати унікальне в рамках поточної таблиці ім'я, і він повинен бути призначений тип даних.

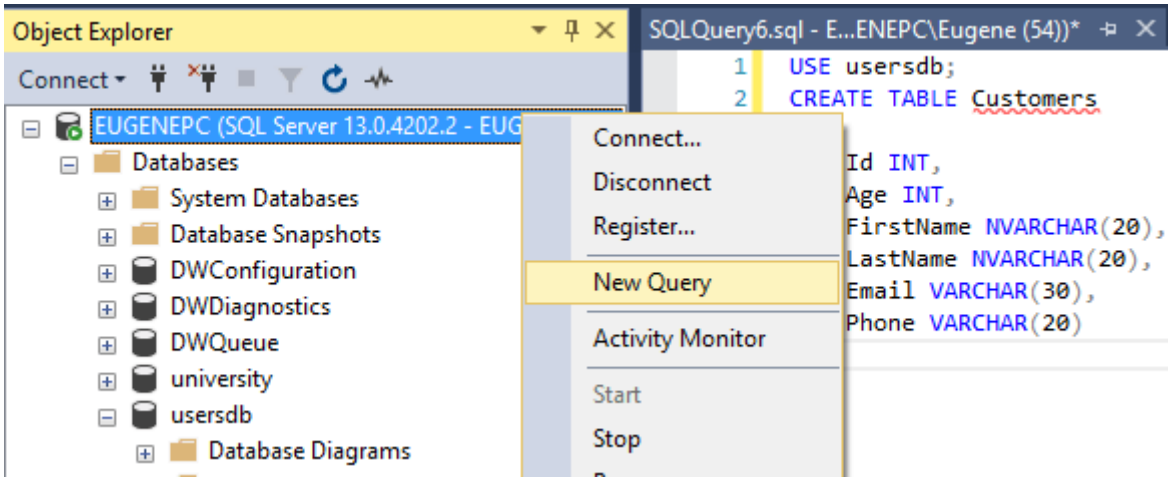
Наприклад, визначення найпростішої таблиці Customers:

```
1 CREATE TABLE Customers
2 (
3 Id INT,
4 Age INT,
5 FirstName NVARCHAR(20),
6 LastName NVARCHAR(20),
7 Email VARCHAR(30),
8 Phone VARCHAR(20)
9 )
```

У разі у таблиці Customers визначаються шість стовпців: Id, FirstName, LastName, Age, Email, Phone. Перші два стовпці представляють ідентифікатор клієнта та його вік і мають тип INT, тобто зберігатимуть числові значення. Наступні два стовпці представляють ім'я та прізвище клієнта і мають тип NVARCHAR(20), тобто представляють рядок UNICODE довжиною трохи більше 20 символів. Останні два стовпці Email і Phone представляють адресу електронної пошти та телефон клієнта і мають тип VARCHAR(30/20) - вони також зберігають рядок, але не кодування UNICODE.

Створення таблиці в SQL Management Studio

Створимо просту таблицю на сервері. Для цього відкриємо SQL Server Management Studio і натиснемо правою кнопкою миші назву сервера. У контекстному меню виберемо пункт New Query.

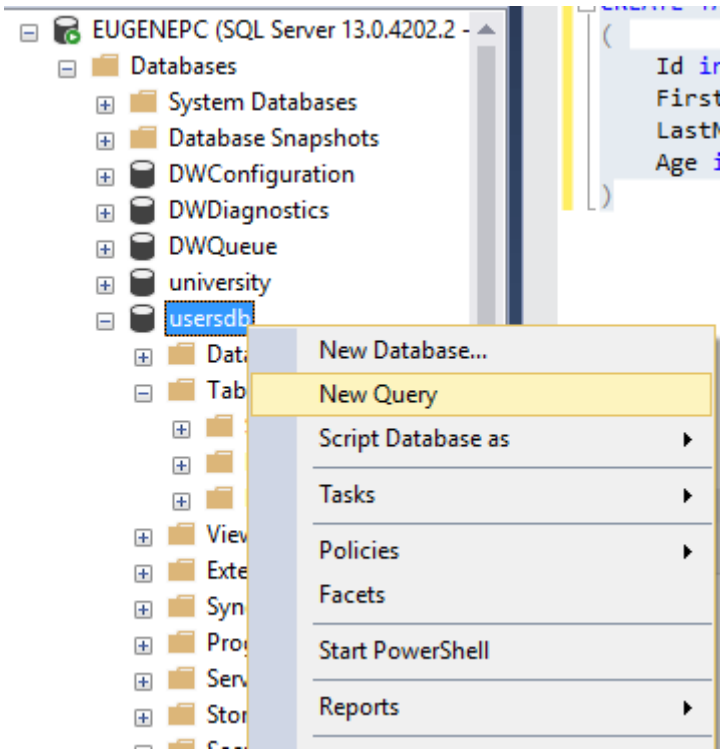


Таблиця створюється у межах поточної бази даних. Якщо ми запускаємо вікно редактора SQL як це зроблено вище - з-під назви сервера, то база даних за умовчанням не встановлена. І для її встановлення необхідно застосувати команду USE, після якої вказується ім'я бази даних. Тому введемо в поле редактора SQL-команд такі вирази:

```
USE usersdb;

CREATE TABLE Customers
(
Id INT,
Age INT,
FirstName NVARCHAR(20),
LastName NVARCHAR(20),
Email VARCHAR(30),
Phone VARCHAR(20)
);
```

Тобто до бази даних додається таблиця Customers, яка була розглянута раніше. Також можна відкрити редактор з-під бази даних, також натиснувши на неї правою кнопкою миші і вибравши New Query:



У цьому випадку як поточна буде розглядатися та база даних, з-під якої був відкритий редактор, і додатково її встановлювати за допомогою команди USE не потрібно.

Видалення таблиць

Для видалення таблиць використовується команда DROP TABLE, яка має наступний синтаксис:

```
1 DROP TABLE table1 [, table2, ...]
```

Наприклад, видалення таблиці Customers:

```
1 DROP TABLE Customers
```

Перейменування таблиці

Для перейменування таблиць застосовується системна процедура "sp_rename". Наприклад, перейменування таблиці Users на UserAccounts у базі даних usersdb:

```
1 USE usersdb;
2 EXEC sp_rename 'Users', 'UserAccounts';
```

Типи даних T-SQL

Під час створення таблиці всім її стовпців необхідно вказати певний тип даних. Тип даних визначає, які значення можуть зберігатися в стовпці, скільки вони займатимуть місця у пам'яті.

Мова T-SQL надає багато різних типів. Залежно від характеру значень їх можна розділити на групи.

Числові типи даних

- **BIT**: зберігає значення від 0 до 16. Може виступати аналогом булевого типу в мовах програмування (у цьому випадку значення true відповідає 1, а значення false - 0). При значеннях до 8 (включно) займає 1 байт, при значеннях від 9 до 16 – 2 байти.
- **TINYINT**: зберігає числа від 0 до 255. Займає 1 байт. Добре підходить для зберігання невеликих чисел.
- **SMALLINT**: зберігає числа від –32 768 до 32 767. Займає 2 байти
- **INT**: зберігає числа від -2147483648 до 2147483647. Займає 4 байти. Найбільш використовуваний тип зберігання чисел.
- **BIGINT**: зберігає дуже великі числа від -9223372036854775808 до 9223372036854775807, які займають у пам'яті 8 байт.
- **DECIMAL**: зберігає числа з фіксованою точністю. Займає від 5 до 17 байт залежно від кількості чисел після коми.

  Цей тип може приймати два параметри precision та scale: DECIMAL(precision, scale).

  Параметр precision представляє максимальну кількість цифр, які можуть зберігати число. Це значення має знаходитися в діапазоні від 1 до 38. За замовчуванням воно дорівнює 18.

  Параметр scale представляє максимальну кількість цифр, які можуть містити число після коми. Це значення має знаходитися в діапазоні від 0 до значення параметра precision. За умовчанням воно дорівнює 0.
- **NUMERIC**: цей тип аналогічний типу DECIMAL.

- **SMALLMONEY**: зберігає дробові значення від -214 748.3648 до 214 748.3647 Призначений для зберігання грошових величин. Займає 4 байти. Еквівалент типу DECIMAL(10,4).
- **MONEY**: зберігає дробові значення від -922337203685477.5808 до 922337203685477.5807. Представляє грошові величини та займає 8 байт. Еквівалент типу DECIMAL(19,4).
- **FLOAT**: зберігає числа від –1.79E+308 до 1.79E+308. Займає від 4 до 8 байт залежно від дрібної частини.

  Може мати форму визначення у вигляді FLOAT(n), де n є число біт, які використовуються для зберігання десяткової частини числа (мантиси). За промовчанням n = 53.
- **REAL**: зберігає числа від –340E+38 to 3.40E+38. Займає 4 байти. Еквівалентний типу FLOAT(24).

  Приклади числових стовпців:

```
1   Salary MONEY,
2   TotalWeight DECIMAL(9,2),
3   Age INT,
4   Surplus FLOAT
```

Типи даних, що представляють дату та час

- **DATE**: зберігає дати від 0001-01-01 (1 січня 0001 року) до 9999-12-31 (31 грудня 9999 року). Займає 3 байти.
- **TIME**: зберігає час у діапазоні від 00:00:00.0000000 до 23:59:59.9999999. Займає від 3 до 5 байт.

  Може мати форму TIME(n), де n становить кількість цифр від 0 до 7 у дробовій частині секунд.
- **DATETIME**: зберігає дати та час від 01/01/1753 до 31/12/9999. Займає 8 байт.
- **DATETIME2**: зберігає дати та час у діапазоні від 01/01/0001 00:00:00.0000000 до 31/12/9999 23:59:59.9999999. Займає від 6 до 8 байт, залежно від точності часу.

  Може мати форму DATETIME2(n), де n представляє кількість цифр від 0 до 7 у дрібній частині секунд.

- **SMALLDATETIME**: зберігає дати та час у діапазоні від 01/01/1900 до 06/06/2079, тобто найближчі дати. Займає від 4 байти.
- **DATETIMEOFFSET**: зберігає дати та час у діапазоні від 0001-01-01 до 9999-12-31. Зберігає детальну інформацію про час з точністю до 100 наносекунд. Займає 10 байт.

  Поширені формати дат:
- yyyy-mm-dd- 2017-07-12
- dd/mm/yyyy- 12/07/2017
- mm-dd-yy- 07-12-17

  У такому форматі двоцифрові числа від 00 до 49 сприймаються як дати в діапазоні 2000-2049. А числа від 50 до 99 як діапазон чисел 1950 – 1999.
- Month dd, yyyy- July 12, 2017

  Розповсюджені формати часу:
- hh:mi- 13:21
- hh:mi am/pm- 1:21 pm
- hh:mi:ss- 1:21:34
- hh:mi:ss:mmm- 1:21:34:12
- hh:mi:ss:nnnnnnn- 1:21:34:1234567

  Рядкові типи даних
- **CHAR**: зберігає рядок довжиною від 1 до 8000 символів. На кожен символ виділяє по 1 байти. Не підходить для багатьох мов, оскільки зберігає символи не в кодуванні Unicode.

  Кількість символів, які можуть зберігати стовпець, передається в дужках. Наприклад, для шпальти з типом CHAR(10) буде виділено 10 байт. І якщо ми збережемо в стовпці рядок менше 10 символів, то він буде доповнений пробілами.
- **VARCHAR**: зберігає рядок На кожен символ виділяється 1 байт. Можна вказати конкретну довжину стовпця - від 1 до 8 000 символів, наприклад, VARCHAR(10). Якщо рядок повинен мати більше 8000 символів, то

визначається розмір MAX, а на зберігання рядка може виділятися до 2 Гб: VARCHAR(MAX).

Не підходить для багатьох мов, оскільки зберігає символи не в кодуванні Unicode.

На відміну від типу CHAR якщо стовпець з типом VARCHAR(10) буде збережено рядок у 5 символів, то столці буде збережено саме п'ять символів.

- **NCHAR**: зберігає рядок у кодуванні Unicode довжиною від 1 до 4000 символів. На кожен символ виділяється 2 байти. Наприклад, NCHAR(15)
- **NVARCHAR**: зберігає рядок у кодуванні Unicode. На кожен символ виділяється 2 байти. Можна задати конкретний розмір від 1 до 4 000 символів: . Якщо рядок повинен мати більше 4000 символів, то визначається розмір MAX, а на зберігання рядка може виділятися до 2 Гб.

  Ще два типи TEXT та NTEXT є застарілими і тому їх не рекомендується використовувати. Замість них застосовуються VARCHAR та NVARCHAR відповідно.

  Приклади визначення рядкових стовпців:

```
1   Email VARCHAR(30),
2   Comment NVARCHAR(MAX)
```

Бінарні типи даних

- **BINARY**: зберігає бінарні дані у вигляді послідовності від 1 до 8000 байт.
- **VARBINARY**: зберігає бінарні дані у вигляді послідовності від 1 до 8 000 байт, або до 2^31-1 байт при використанні значення MAX (VARBINARY(MAX)).

  Ще один бінарний тип – тип IMAGE є застарілим, і замість нього рекомендується застосовувати тип VARBINARY.

Інші типи даних

- **UNIQUEIDENTIFIER**: унікальний ідентифікатор GUID (по суті, рядок з унікальним значенням), який займає 16 байт.
- **TIMESTAMP**: деяке число, яке зберігає номер версії рядка таблиці. Займає 8 байт.
- **CURSOR**: представляє набір рядків

- **HIERARCHYID**: представляє позицію в ієрархії
- **SQL_VARIANT**: може зберігати дані іншого типу даних T-SQL.
- **XML**: зберігає документи XML або фрагменти документів XML. Займає у пам'яті до 2 Гб.
- **TABLE**: представляє визначення таблиці
- **GEOGRAPHY**: зберігає географічні дані, такі як широта та довгота.
- **GEOMETRY**: зберігає координати місцезнаходження на площині

Атрибути та обмеження стовпців та таблиць

При створенні стовпців у T-SQL ми можемо використовувати низку атрибутів, які є обмеженнями. Розглянемо ці атрибути.

PRIMARY KEY

За допомогою виразу PRIMARY KEY стовпець можна зробити первинним ключем.

```
1   CREATE TABLE Customers
2   (
3   Id INT PRIMARY KEY,
4   Age INT,
5   FirstName NVARCHAR(20),
6   LastName NVARCHAR(20),
7   Email VARCHAR(30),
8   Phone VARCHAR(20)
9   )
```

Первинний ключ унікально ідентифікує рядок таблиці. Як первинний ключ необов'язково повинні виступати стовпці з типом int, вони можуть представляти будь-який інший тип.

Установка первинного ключа лише на рівні таблиці:

```
1    CREATE TABLE Customers
2    (
3    Id INT,
4    Age INT,
5    FirstName NVARCHAR(20),
6    LastName NVARCHAR(20),
7    Email VARCHAR(30),
8    Phone VARCHAR(20),
```

```
9    PRIMARY KEY(Id)
10   )
```

Первинний ключ може бути складеним (compound key). Такий ключ може знадобитися, якщо у нас одразу два стовпці мають унікально ідентифікувати рядок у таблиці. Наприклад:

```
1   CREATE TABLE OrderLines
2   (
3   OrderId INT,
4   ProductId INT,
5   Quantity INT,
6   Price MONEY,
7   PRIMARY KEY(OrderId, ProductId)
8   )
```

Тут поля OrderId та ProductId разом виступають як складовий первинний ключ. Тобто в таблиці OrderLines не може бути двох рядків, де для обох з цих полів одночасно були б ті самі значення.

IDENTITY

Атрибут IDENTITY дозволяє зробити стовпець ідентифікатором. Цей атрибут може бути призначений для стовпців числових типів INT, SMALLINT, BIGINT, TYNIINT, DECIMAL і NUMERIC. При додаванні нових даних до таблиці SQL Server інкрементуватиме на одиницю значення цього стовпця в останньому записі. Як правило, у ролі ідентифікатора виступає той самий стовпець, який є первинним ключем, хоча в принципі це необов'язково.

```
1   CREATE TABLE Customers
2   (
3   Id INT PRIMARY KEY IDENTITY,
4   Age INT,
5   FirstName NVARCHAR(20),
6   LastName NVARCHAR(20),
7   Email VARCHAR(30),
8   Phone VARCHAR(20)
9   )
```

Також можна використовувати повну форму атрибуту:

```
1   IDENTITY(seed, increment)
```

Тут параметр seed вказує на початкове значення, з якого починатиметься відлік. А параметр increment визначає, наскільки збільшуватиметься таке значення. За умовчанням атрибут використовує такі значення:

```
1   IDENTITY(1, 1)
```

Тобто, відлік починається з 1. А наступні значення збільшуються на одиницю. Але ми можемо цю поведінку перевизначити. Наприклад:

```
1   Id INT IDENTITY (2, 3)
```

В даному випадку відлік почнеться з 2, а значення кожного наступного запису буде збільшуватися на 3. Тобто перший рядок матиме значення 2, другий - 5, третій - 8 і т.д.

Також слід враховувати, що у таблиці лише один стовпець повинен мати такий атрибут.

UNIQUE

Якщо хочемо, щоб стовпець мав лише унікальні значення, то можна визначити атрибут UNIQUE.

```
1   CREATE TABLE Customers
2   (
3   Id INT PRIMARY KEY IDENTITY,
4   Age INT,
5   FirstName NVARCHAR(20),
6   LastName NVARCHAR(20),
7   Email VARCHAR(30) UNIQUE,
8   Phone VARCHAR(20) UNIQUE
9   )
```

У цьому випадку стовпці, які представляють електронну адресу та телефон, матимуть унікальні значення. І ми не зможемо додати до таблиці два рядки, які мають значення для цих стовпців співпадати.

Також ми можемо визначити цей атрибут на рівні таблиці:

```
1    CREATE TABLE Customers
2    (
3    Id INT PRIMARY KEY IDENTITY,
4    Age INT,
5    FirstName NVARCHAR(20),
6    LastName NVARCHAR(20),
7    Email VARCHAR(30),
```

```
8    Phone VARCHAR(20),
9    UNIQUE(Email, Phone)
10   )
```

NULL та NOT NULL

Щоб вказати, чи може стовпець набувати значення NULL, при визначенні стовпця йому можна задати атрибут NULL або NOT NULL. Якщо цей атрибут явно не буде використаний, то за умовчанням стовпець допускатиме значення NULL. Винятком є той випадок, коли стовпець виступає ролі первинного ключа - у разі за умовчанням стовпець має значення NOT NULL.

```
1   CREATE TABLE Customers
2   (
3   Id INT PRIMARY KEY IDENTITY,
4   Age INT,
5   FirstName NVARCHAR(20) NOT NULL,
6   LastName NVARCHAR(20) NOT NULL,
7   Email VARCHAR(30) UNIQUE,
8   Phone VARCHAR(20) UNIQUE
9   )
```

DEFAULT

Атрибут DEFAULT визначає значення за промовчанням для стовпця. Якщо при додаванні даних для стовпця не буде передбачено значення, то для нього використовуватиметься значення за замовчуванням.

```
1   CREATE TABLE Customers
2   (
3   Id INT PRIMARY KEY IDENTITY,
4   Age INT DEFAULT 18,
5   FirstName NVARCHAR(20) NOT NULL,
6   LastName NVARCHAR(20) NOT NULL,
7   Email VARCHAR(30) UNIQUE,
8   Phone VARCHAR(20) UNIQUE
9   );
```

Тут для стовпця Age передбачено значення за промовчанням 18.

CHECK

Ключове слово CHECK визначає обмеження для діапазону значень, які можуть зберігатися в стовпці. Для цього після слова CHECK вказується в

дужках умова, якій має відповідати стовпець або кілька стовпців. Наприклад, вік клієнтів не може бути меншим за 0 або більше 100:

```
1   CREATE TABLE Customers
2   (
3   Id INT PRIMARY KEY IDENTITY,
4   Age INT DEFAULT 18 CHECK(Age >0 AND Age < 100),
5   FirstName NVARCHAR(20) NOT NULL,
6   LastName NVARCHAR(20) NOT NULL,
7   Email VARCHAR(30) UNIQUE CHECK(Email !=''),
8   Phone VARCHAR(20) UNIQUE CHECK(Phone !='')
9   );
```

Тут також вказується, що стовпці Email і Phone не можуть мати порожній рядок як значення (порожній рядок не еквівалентний NULL).

Для з'єднання умов використовується ключове слово AND. Умови можна задати у вигляді операцій порівняння більше (>), менше (<), не дорівнює (! =).

Також за допомогою CHECK можна створити обмеження загалом для таблиці:

```
1    CREATE TABLE Customers
2    (
3    Id INT PRIMARY KEY IDENTITY,
4    Age INT DEFAULT 18,
5    FirstName NVARCHAR(20) NOT NULL,
6    LastName NVARCHAR(20) NOT NULL,
7    Email VARCHAR(30) UNIQUE,
8    Phone VARCHAR(20) UNIQUE,
9    CHECK((Age >0 AND Age<100) AND (Email !='') AND (Phone! = ''))
10   )
```

Оператор CONSTRAINT. Встановлення імені обмежень.

За допомогою ключового слова CONSTRAINT можна встановити ім'я для обмежень. Як обмеження можуть використовуватися PRIMARY KEY, UNIQUE, DEFAULT, CHECK.

Імена обмежень можна встановити на рівні стовпців. Вони вказуються після CONSTRAINT перед атрибутами:

```
1    CREATE TABLE Customers
2    (
3    Id INT CONSTRAINT PK_Customer_Id PRIMARY KEY IDENTITY,
4    Age INT
5    CONSTRAINT DF_Customer_Age DEFAULT 18
```

```
6    CONSTRAINT CK_Customer_Age CHECK(Age >0 AND Age < 100),
7    FirstName NVARCHAR(20) NOT NULL,
8    LastName NVARCHAR(20) NOT NULL,
9    Email VARCHAR(30) CONSTRAINT UQ_Customer_Email UNIQUE,
10   Phone VARCHAR(20) CONSTRAINT UQ_Customer_Phone UNIQUE
11   )
```

Обмеження можуть носити довільні назви, але, як правило, застосовуються такі префікси:

- "PK_" - для PRIMARY KEY
- "FK_" - для FOREIGN KEY
- "CK_" - для CHECK
- "UQ_" - для UNIQUE
- "DF_" - для DEFAULT

В принципі необов'язково вказувати імена обмежень, при установці відповідних атрибутів SQL Server автоматично визначає їх імена. Але знаючи ім'я обмеження, ми можемо до нього звертатися, наприклад, для його видалення.

Також можна задати всі імена обмежень через атрибути таблиці:

```
1    CREATE TABLE Customers
2    (
3    Id INT IDENTITY,
4    Age INT CONSTRAINT DF_Customer_Age DEFAULT 18,
5    FirstName NVARCHAR(20) NOT NULL,
6    LastName NVARCHAR(20) NOT NULL,
7    Email VARCHAR(30),
8    Phone VARCHAR(20),
9    CONSTRAINT PK_Customer_Id PRIMARY KEY (Id),
10   CONSTRAINT CK_Customer_Age CHECK(Age >0 AND Age < 100),
11   CONSTRAINT UQ_Customer_Email UNIQUE (Email),
12   CONSTRAINT UQ_Customer_Phone UNIQUE (Phone)
13   )
```

Зовнішні ключі

Зовнішні ключі використовуються для встановлення зв'язку між таблицями. Зовнішній ключ встановлюється для стовпців із залежної, підлеглої таблиці, і вказує на один із стовпців із головної таблиці. Хоча, як правило, зовнішній

ключ вказує на первинний ключ із пов'язаної головної таблиці, але це необов'язково має бути неодмінною умовою. Зовнішній ключ може вказувати на якийсь інший стовпець, який має унікальне значення.

Загальний синтаксис встановлення зовнішнього ключа на рівні шпальти:

```
[FOREIGN KEY] REFERENCES головна_таблиця (стовпець_головної_таблиці)
[ON DELETE {CASCADE|NO ACTION}]
[ON UPDATE {CASCADE|NO ACTION}]
```

Для створення обмеження зовнішнього ключа на рівні стовпця після ключового слова REFERENCES вказується ім'я пов'язаної таблиці та у круглих дужках ім'я зв'язаного стовпця, на який вказуватиме зовнішній ключ. Також зазвичай додаються ключові слова FOREIGN KEY, але їх необов'язково вказувати. Після виразу REFERENCES йде вираз ON DELETE та ON UPDATE.

Загальний синтаксис встановлення зовнішнього ключа на рівні таблиці:

```
FOREIGN KEY (Стобець1, стовпець2, ... стовпецьN)
REFERENCES головна_таблиця (стовпець_головної_таблиці1, стовпець_головної_таблиці
[ON DELETE {CASCADE|NO ACTION}]
[ON UPDATE {CASCADE|NO ACTION}]
```

Наприклад, визначимо дві таблиці та зв'яжемо їх за допомогою зовнішнього ключа:

```
CREATE TABLE Customers
(
Id INT PRIMARY KEY IDENTITY,
Age INT DEFAULT 18,
FirstName NVARCHAR(20) NOT NULL,
LastName NVARCHAR(20) NOT NULL,
Email VARCHAR(30) UNIQUE,
Phone VARCHAR(20) UNIQUE
);

CREATE TABLE Orders
(
Id INT PRIMARY KEY IDENTITY,
CustomerId INT REFERENCES Customers (Id),
CreatedAt Date
);
```

Тут визначено таблиці Customers та Orders. Customers є головним і представляє клієнта. Orders є залежною та представляє замовлення, зроблене клієнтом. Ця таблиця через стовпець CustomerId пов'язана з таблицею Customers та її стовпцем Id. Тобто стовпець CustomerId є зовнішнім ключем, який свідчить про стовпець Id з таблиці Customers.

Визначення зовнішнього ключа лише на рівні таблиці виглядало б так:

```
1   CREATE TABLE Orders
2   (
3   Id INT PRIMARY KEY IDENTITY,
4   CustomerId INT,
5   CreatedAt Date,
6   FOREIGN KEY (CustomerId) REFERENCES Customers (Id)
7   );
```

За допомогою оператора CONSTRAINT можна встановити ім'я для обмеження зовнішнього ключа. Зазвичай це ім'я починається з префіксу "FK_":

```
1 CREATE TABLE Orders
2 (
3 Id INT PRIMARY KEY IDENTITY,
4 CustomerId INT,
5 CreatedAt Date,
6 CONSTRAINT FK_Orders_To_Customers FOREIGN KEY (CustomerId) REFERENCES
  Customers (Id)
7 );
```

У цьому випадку обмеження зовнішнього ключа CustomerId називається "FK_Orders_To_Customers".

ON DELETE та ON UPDATE

За допомогою виразів ON DELETE та ON UPDATE можна встановити дії, які виконуватимуться відповідно при видаленні та зміні зв'язаного рядка з головної таблиці. І для визначення дії ми можемо використовувати такі опції:

- **CASCADE**: автоматично видаляє або змінює рядки із залежної таблиці під час видалення або зміни пов'язаних рядків у головній таблиці.
- **NO ACTION**: запобігає будь-яким діям у залежній таблиці при видаленні або зміні зв'язаних рядків у головній таблиці. Тобто фактично якихось дій відсутні.

- **SET NULL**: при видаленні зв'язаного рядка з головної таблиці встановлює значення NULL для стовпчика зовнішнього ключа.
- **SET DEFAULT**: у разі видалення зв'язаного рядка з головної таблиці встановлює для стовпчика зовнішнього ключа значення за замовчуванням, яке задається за допомогою атрибути DEFAULT. Якщо для стовпця не встановлено значення за умовчанням, то як нього застосовується значення NULL.

  Каскадне видалення

  За замовчуванням, якщо на рядок із головної таблиці за зовнішнім ключем посилається якийсь рядок із залежної таблиці, то ми не зможемо видалити цей рядок із головної таблиці. Спочатку нам необхідно буде видалити всі зв'язані рядки із залежної таблиці. І якщо при видаленні рядка з головної таблиці необхідно, щоб було видалено всі зв'язані рядки із залежної таблиці, то застосовується каскадне видалення, тобто опція CASCADE:

```
1   CREATE TABLE Orders
2   (
3   Id INT PRIMARY KEY IDENTITY,
4   CustomerId INT,
5   CreatedAt Date,
6   FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE CASCADE
7   )
```

  Аналогічно працює вираз ON UPDATE CASCADE. При зміні первинного ключа автоматично зміниться значення пов'язаного з ним зовнішнього ключа. Але оскільки первинні ключі, як правило, змінюються дуже рідко, та й з принципу не рекомендується використовувати як первинні ключі стовпці зі змінними значеннями, то на практиці вираз ON UPDATE використовується рідко.

  Встановлення NULL

  У разі встановлення для зовнішнього ключа опції SET NULL необхідно, щоб стовпець зовнішнього ключа допускав значення NULL:

```
1   CREATE TABLE Orders
2   (
3   Id INT PRIMARY KEY IDENTITY,
```

```
4   CustomerId INT,
5   CreatedAt Date,
6   FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE SET NULL
7   );
```

Встановлення значення за промовчанням

```
1 CREATE TABLE Orders
2 (
3 Id INT PRIMARY KEY IDENTITY,
4 CustomerId INT,
5 CreatedAt Date,
6 FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE SET DEFAULT
7 )
```

Зміна таблиці

Можливо, в якийсь момент ми захочемо змінити таблицю, що вже є. Наприклад, додати або видалити стовпці, змінити тип стовпців, додати або видалити обмеження. Тобто потрібно змінити визначення таблиці. Для зміни таблиць використовується вираз ALTER TABLE.

Загальний формальний синтаксис команди виглядає так:

```
1   ALTER TABLE table_name [WITH CHECK | WITH NOCHECK]
2   { ADD column_name column_data_type [column_attributes] |
3   DROP COLUMN column_name |
4   ALTER COLUMN column_name column_data_type [NULL|NOT NULL] |
5   ADD [CONSTRAINT] constraint_definition |
6   DROP [CONSTRAINT] constraint_name}
```

Таким чином, за допомогою ALTER TABLE ми можемо провернути різні сценарії зміни таблиці. Розглянемо деякі з них.

Додавання нового стовпця

Додамо до таблиці Customers новий стовпець Address:

```
1   ALTER TABLE Customers
2   ADD Address NVARCHAR(50) NULL;
```

У цьому випадку стовпець Address має тип NVARCHAR і для нього визначено атрибут NULL. Але якщо нам треба додати стовпець, який не повинен набувати значення NULL? Якщо таблиці є дані, то наступна команда нічого очікувати виконано:

```
1   ALTER TABLE Customers
```

```
2   ADD Address NVARCHAR(50) NOT NULL;
```

Тому в даному випадку рішення полягає в установці значення за промовчанням через атрибут DEFAULT:

```
1   ALTER TABLE Customers
2   ADD Address NVARCHAR(50) NOT NULL DEFAULT 'Невідомо';
```

У цьому випадку, якщо в таблиці вже є дані, для них для стовпця Address буде додано значення "Невідомо".

Видалення стовпця

Видалимо стовпець Address з таблиці Customers:

```
1   ALTER TABLE Customers
2   DROP COLUMN Address;
```

Зміна типу стовпця

Змінимо в таблиці Customers тип даних у стовпця FirstName на NVARCHAR(200):

```
1   ALTER TABLE Customers
2   ALTER COLUMN FirstName NVARCHAR(200);
```

Додавання обмеження CHECK

При додаванні обмежень SQL Server автоматично перевіряє наявні дані на відповідність обмеженням, що додаються. Якщо дані не відповідають обмеженням, такі обмеження не будуть додані. Наприклад, встановимо для стовпця Age у таблиці Customers обмеження Age > 21.

```
1   ALTER TABLE Customers
2   ADD CHECK (Age > 21);
```

Якщо в таблиці є рядки, в яких у стовпці Age є значення, що не відповідають цьому обмеженню, то команда SQL завершиться з помилкою. Щоб уникнути подібної перевірки на відповідність і додати обмеження, незважаючи на наявність невідповідних йому даних, використовується вираз WITH NOCHECK:

```
1   ALTER TABLE Customers WITH NOCHECK
2   ADD CHECK (Age > 21);
```

За замовчуванням використовується значення WITH CHECK, яке перевіряє відповідність обмеженням.

Додавання зовнішнього ключа

Нехай спочатку до бази даних буде додано дві таблиці, ніяк не пов'язані:

```
1    CREATE TABLE Customers
2    (
3    Id INT PRIMARY KEY IDENTITY,
4    Age INT DEFAULT 18,
5    FirstName NVARCHAR(20) NOT NULL,
6    LastName NVARCHAR(20) NOT NULL,
7    Email VARCHAR(30) UNIQUE,
8    Phone VARCHAR(20) UNIQUE
9    );
10   CREATE TABLE Orders
11   (
12   Id INT IDENTITY,
13   CustomerId INT,
14   CreatedAt Date
15   );
```

Додамо обмеження зовнішнього ключа до стовпця CustomerId таблиці Orders:

```
1   ALTER TABLE Orders
2   ADD FOREIGN KEY(CustomerId) REFERENCES Customers(Id);
```

Додавання первинного ключа

Використовуючи певну таблицю Orders, додамо до неї первинний ключ для стовпця Id:

```
1   ALTER TABLE Orders
2   ADD PRIMARY KEY (Id);
```

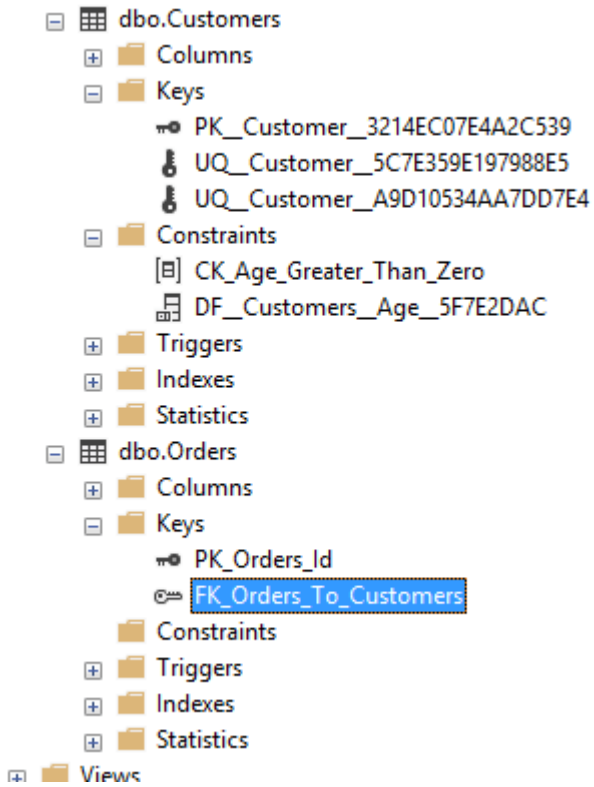
Додавання обмежень з іменами

При додаванні обмежень ми можемо вказати ім'я, використовуючи оператор CONSTRAINT, після якого вказується ім'я обмеження:

```
1 ALTER TABLE Orders
2 ADD CONSTRAINT PK_Orders_Id PRIMARY KEY (Id),
3 CONSTRAINT   FK_Orders_To_Customers   FOREIGN   KEY(CustomerId)   REFERENCES
  Customers(Id);
4
5 ALTER TABLE Customers
6 ADD CONSTRAINT CK_Age_Greater_Than_Zero CHECK (Age > 0);
```

Видалення обмежень

Для видалення обмежень необхідно знати їхнє ім'я. Якщо ми точно не знаємо ім'я обмеження, його можна дізнатися через SQL Server Management Studio:



Розкривши вузол таблиць у підвузлі Keys можна побачити назви обмежень первинного та зовнішніх ключів. Назви обмежень зовнішніх ключів починаються з FK. А в підвузлі Constraints можна знайти всі обмеження CHECK та DEFAULT. Назви обмежень CHECK починаються з CK, а обмежень DEFAULT - з DF.

Наприклад, як видно на скріншоті, в моєму випадку ім'я обмеження зовнішнього ключа в таблиці Orders називається "FK_Orders_To_Customers". Тому для видалення зовнішнього ключа ми можемо використовувати такий вираз:

```
1   ALTER TABLE Orders
2   DROP FK_Orders_To_Customers;
```

Пакети. Команда GO

У попередніх випадках спочатку створювалася база даних, а потім до цієї БД додавалася таблиця за допомогою окремих команд SQL. Але можна одразу поєднати в одному скрипті кілька команд. І тут окремі набори команд називаються пакетами (batch).

Кожен пакет складається з одного або декількох SQL-виразів, які виконуються як ціле. В якості сигналу завершення пакета та виконання його виразів служить команда GO.

Сенс поділу SQL-виразів на пакети полягає в тому, що одні вирази повинні успішно виконатись до запуску інших виразів. Наприклад, при додаванні таблиць ми повинні впевнені, що було створено базу даних, у якій ми збираємося створити таблиці.

Наприклад, визначимо наступний скрипт:

```
1    CREATE DATABASE internetstore;
2    GO
3
4    USE internetstore;
5
6    CREATE TABLE Customers
7    (
8    Id INT PRIMARY KEY IDENTITY,
9    Age INT DEFAULT 18,
10   FirstName NVARCHAR(20) NOT NULL,
11   LastName NVARCHAR(20) NOT NULL,
12   Email VARCHAR(30) UNIQUE,
13   Phone VARCHAR(20) UNIQUE
14   );
15
16   CREATE TABLE Orders
17   (
18   Id INT PRIMARY KEY IDENTITY,
19   CustomerId INT,
20   CreatedAt DATE,
21   FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE CASCADE
22   );
```

Спочатку створюється бд internetstore. Потім йде команда GO, яка сигналізує, що можна виконувати наступний виразний пакет. І далі виконується другий пакет, який додає до неї дві таблиці - Customers та Orders.