Урок 2

План заняття:

1. Пакетні запити та сценарії

Сценарій – це набір операторів, які зберігаються окремим файлом, що може запускатись на виконання і використовуватись повторно.

В T-SQL також виділяють поняття пакету. **Пакетний запит (пакет)** – це послідовність операторів T-SQL, інтерпретуємих сервером разом, тобто як одна логічна одиниця. Оператори, які є складовою частиною пакету, посилаються на сервер як єдине ціле. Щоб розділити сценарій на кілька пакетів, використовується оператор **GO**. Оператор GO викликає компіляцію всіх операторів від початку сценарію або попереднього оператора GO (в залежності від того, що ближче), після чого отриманий план виконання передається на сервер незалежно від всіх інших пакетів. Наприклад:

```
--1 команда. Робимо базу даних master активною за допомогою оператора use
use master
--2 команда. Виводимо на екран всю інформацію з системної таблиці sysobjects
select *
from sysobjects
--посилаємо на сервер пакет з двох команд для обробки
go
```

Пакети підпорядковуються наступним **правилам**:

- Всі оператори пакета компілюються як єдине ціле.
- Якщо в пакеті існує синтаксична помилка, - відміняється виконання всього пакета.
- Якщо під час виконання пакета в одному з операторів відбувається помилка, то цей оператор пропускається і продовжується виконання інших операторів. Наприклад, якщо пакет містить три оператора CREATE TABLE і в другому операторі відбувається помилка, то SQL Server створить лише першу та третю таблицю.
- В межах одного пакету неможна спочатку змінювати поля таблиці, а потім використовувати ці нові поля.
- Оператори SET виконуються одразу, крім випадків, коли встановлені опції QUOTED_IDENTIFIER і ANSI_NULLS.
- В один пакет **не можна сумісно поміщати** наступні оператори:
  - CREATE RULE;
  - CREATE TRIGGER;
  - CREATE PROCEDURE;
  - CREATE DEFAULT;
  - CREATE VIEW.

2. Типи даних в MS SQL Server. Розрахункові поля. Поля з обмеженнями (check)

MS SQL Server, як вже було сказано, являється реляційною базою даних, і тому всі її дані зберігаються в таблицях, які складаються з записів та полів. Кожне поле таблиці має ім'я та містить дані лише одного типу. Тип даних дозволяє вказати на те, які саме дані можна зберігати в кожному окремому полі та обмежувати діапазон їх значень, що дозволить перешкодити введенню невірних даних. В MS SQL Server 2008 виділяють наступні типи даних:

Тип даних	Опис
Рядкові типи даних	
char(к-ть_символів)	Рядок фіксованої довжини, кількість символів якого не може перевищувати встановленого значення. Якщо в полі буде міститись рядок довжиною, наприклад, 5 символів, а тип для зберігання даних був визначений як char(20), то на диску для нього буде все рівно виділено 20 байт, а не використовувані байт заповняються пропусками (space). Якщо рядок, який вводиться буде більше 20 символів, він буде обрізаний до необхідної довжини. Максимальна довжина – 8000 байт (8000 символів).
varchar(к-ть_символів)*	Рядок змінної довжини, максимальна кількість символів якого визначається параметром. Якщо в полі буде міститись рядок довжиною, наприклад, 5 символів, а тип для зберігання даних був визначений як varchar(20), то на диску для нього буде виділено 5 байт. Якщо рядок, який вводиться буде більше 20 символів, він буде обрізаний до необхідної довжини. Максимальна довжина - 8000 байт (8000 символів). При використанні в якості параметра значення «MAX» (varchar(MAX)) максимальна довжина зберігаємих даних сягає 2 Гб (до 1 073 741 824 символи)
nchar(к-ть_символів) **nvarchar(к-ть_символів)***	Аналогічні типам char і varchar, але призначені для збереження даних в форматі Unicode (2 байти на символ). Максимальна кількість необхідних байт – 8000, тобто максимальна кількість символів – 4000. При використанні в якості параметра значення «MAX» (nvarchar(MAX)) максимальна довжина зберігаємих даних сягає 2 Гб (до 536 870 912 символи)
Цілочисельні типи даних	
tinyint	Розмір - 1 байт. Діапазон значень: 0..255
smallint	Розмір - 2 байти. Діапазон значень: -32767..+32676
int	Розмір - 4 байти. Діапазон значень: -2147483647..+2147483647
bigint	Розмір - 8 байт. Діапазон значень: -9223372036854775808..+9223372036854775808
Типи даних для збереження дійсних чисел	
real(к-ть_розрядів)	Розмір - 4 байта. Максимальна розрядність - 7. Наприклад, при типові даних real(5), в ньому можна зберігати значення типу 1.2345, 123.45, але не більше визначеної розрядності, тобто 5. Даний тип підтримується лише для сумісності з стандартом SQL-92 і замінений на float.
float(к-ть_розрядів)	Розмір – 4 або 8 байт. Максимальна розрядність – 38. По замовчуванню встановлюється точність15 розрядів.
numeric(загальна_к-ть_розрядів, скільки_для_дробної_частини)	Зберігає дійсні числа з максимальною кількістю розрідів не більше 38. По замовчуванню розрядність встановлюється 18 і 0. При введенні 0 для десяткових розрядів числові значення приводяться до цілочисельних Наприклад, при типові даних numeric(7,2) можна зберігати значення типу 123,45 і 12345,67, але не 1234567,89. Якщо кількість розрядів після коми більша, ніж вказана, то при збереженні буде здійснено математичне скорочення. У випадку перевищення розрядності мантиси буде виведена помилка при збереженні даних.
decimal(загальна_к-ть_розрядів, скільки_для_дробної_частини)	Аналог numeric.
Типи даних для збереження дати та часу	
date	Розмір - 3 байти. Діапазон дат: 0001/01/01..9999/12/31 Дата представлена в форматі YYYY-MM-DD Новий тип даних в SQL Server 2008
time[(точність_в_секундах)]	Розмір – 5 байт. Діапазон часу: 00:00:00..23:59:59.9(7) Дата представлена в форматі HH:MM:SS[.nnnnnnn] Позначення n* вказує на долі секунди (0..9999999). Новий тип даних в SQL Server 2008

smalldatetime	Розмір - 4 байта. Діапазон дат: 1900/01/01..2079/06/06. Дата представлена в форматі YYYY-MM-DD HH:MM:SS
datetime	Розмір – 8 байт. Діапазон дат: 1753/01/01..9999/12/03 Діапазон часу: 00:00:00..23:59:59.0997 Дата представлена в форматі YYYY-MM-DD HH:MM:SS.MS
datetime2[(точність_в_ секундах)]	Розмір – 6 байт для представлення точності меньше 3 цифр, 7 байт - для точності в 3 і 4 цифри. Для представлення інших значень точності необхідно 8 байт. Діапазон дат: 0001/01/01..9999/12/31. Діапазон часу: 00:00:00..23:59:59.9(7) Дата представлена в форматі YYYY-MM-DD HH:MM:SS.MS Новий тип даних в SQL Server 2008
datetimeoffset [(точність_в _секундах)]	Дата і час з врахуванням часового поясу в 24-годинному форматі. Необов'язковий параметр p вказує на точність в секундах. Розмір – 10 байт. Діапазон дат: 0001/01/01..9999/12/31 Діапазон часу: 00:00:0(7)..23:59:59.9(7) Дата представлена в форматі YYYY-MM-DD HH:MM:SS[.nnnnnnn][{+|-}hh:mm] Позначення n* вказує на долі секунди (0..9999999). Позначення hh може приймати значення від -14 до +14. Позначення mm приймає значення від 00 до 59. Новий тип даних в SQL Server 2008
Типи даних для збереження грошових величин	
smallmoney	Розмір - 4 байти. Діапазон значень: -214 748,3647..+214 748,3647.
money	Розмір - 8 байт. Діапазон значень: -922 337 203 685 477,5808..+922 337 203 685 477,5808.
Бітові типи даних	
binary(к-ть_байт) **varbinary(к-ть_байт)***	Призначені для збереження інформації в бітовому вигляді, тобто дані, що в них зберігаються являють собою послідовність 0 і 1, представлених в шіснадцятковій системі числення і організованих в пари. При введенні даних в бінарному вигляді до них додається префікс **0x**. Наприклад, для створення в полі бітового значення 10, введіть 0x10. Принцип роботи аналогічний типам char та varchar. Максимальний розмір – 8000 байт. При використанні в якості параметра типу varbinary значення «MAX» (varbinary(MAX)) максимальна довжина зберігаємих даних сягає 2 Гб
filestream	Новий тип даних в SQL Server 2008 забезпечує збереження великих обсягів двійкових даних в файловій системі NTFS, при цьому вони залишаються частиною БД з підтримкою цілісності транзакцій. Це дозволяє розміщувати двійкові дані за межами БД і при цьому забезпечувати доступ до них.
Додаткові типи даних	
bit	Розмір: 1 біт Двійковий тип даних, який використовується для збереження булевих значень: 0 (false) або 1 (true). Поля, що мають тип даних bit можуть містити NULL значення і не можуть бути проіндексовані. При створенні в таблиці кількох полів даного типу SQL Server об'єднує їх в групи з 8-ми полів в кожній, тобто сумарно по 1 байту.
uniqueidentifier (RowGUID)	Розмір – 16 байт При реплікації даних кожне поле реплікованої таблиці повинно мати унікальний ідентифікатор, для якого бажано використовувати тип даних uniqueidentifier, що має властивість ROWGUIDCOL. При підключенні даної властивості полю надається GUID (Global Unique Identifier – глобально унікальний ідентифікатор). GUID визначається двома способами: • за допомогою функції NEWID; • шляхом приведення рядкової константи до 16 системи числення в форматі xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.Наприклад, 8FE17A24-B1AA-C790-23DA-2749A3E09AA2 В типі даних uniqueidentifier можна викостовувати символи =, <>, IS NULL та IS NOT NULL

text **image**	Ці типи даних вважаються застарівшими і замість них рекомендується використовувати типи varchar(max) та varbinary(max) відповідно (див. примітку). Використовувався для збереження в полі даних розміром більше 8000 байт. Ці типи даних потрібні при збереженні об'єктів BLOB (Binary Large Object – великий об'єкт двійкових даних), оскільки вони оголосити поля, що містять до 2Гб бітової і текстової інформації. При оголошенні типу даних text і image в рядок додається 16-бітовий вказівник, який визначає сторінку даних розміром 8 Кб, яка містить додаткові дані. Якщо додаткові дані перевищують 8 Кб, тоді створюються вказівними на додаткові сторінки Вашого BLOB. Збереження та використання текстових даних та малюнків дуже знижує продуктивність бази даних, оскільки великі обсяги даних повільно обробляються при виконанні запитів на вставку, обновлення або видалення. З ціллю прискорення виконання операцій з великими типами даних використовується команда WRITETEXT.
ntext	Аналогічно типу даних text, але дані зберігаються в форматі Unicode (2 байта на символ), тому при максимумі даних 2 Гб, можна поміщати символів в два рази меньше. Замість даного типу рекомендується використовувати тип nvarchar(max).
sql_variant	Дозволяє зберегти значення будь-якого типу, крім типів даних ntext та timestamp. Цей тип використовується для збереження значень полів в змінних, в якості значень параметрів, для представлення результатів виконання користувацьких функцій. Цей тип даних НЕБЕЗПЕЧНИЙ (!), оскільки він дозволяє оголошувати поле або змінну без явного вказання типу даних, які будуть в ньому зберігатись. Після цього sql_variant автоматично приводиться до типу даних, додаваємих в поле. В зв'язку з цим, може виникнути помилка у випадку повторного внесення в дане поле даних, які мають відмінний тип.
table	Використовується для збереження набору даних, які будуть використовуватись в майбутньому. Оголошення даного типу аналогічно створенню тимчасових таблиць.
geography	Являє собою просторовий тип даних на основі Microsoft .NET Framework, в якому використовується сферична модель нашої планети. В цьому типі зберігаються точки, лінії, багатокутники і набори координат довготи і широти. Поява даного типу даних в SQL Server 2008 зумовлена використанням в багатьох додатках картографічних функцій.
geometry	Просторовий тип даних SQL Server 2008 на основі Microsoft .NET Framework, основним призначення якого, на відміну від типу GEOGRAPHY, є навігація і картографія на сферичній моделі Землі. Даний тип відповідає специфікації Open Geospatial Consortium (OGC) і оснований на плоскій моделі.
hierarchyid	Тип даних SQL Server 2008 змінної довжини, який призначений для збереження даних, які мають деривовидну структуру, наприклад, організаційні схеми. Він реалізований як користувацький тип CLR, що містить кілька вбудованих методів для створення вузлів ієрархії і гнучкого маніпулювання ними.
cursor	Розмір – 1 байт Містить посилання на курсор, який використовується для виконання операцій. Даний тип не можна використовувати в таблицях.
xml	Збереження XML-документів розміром до 2 Гб. Задаваємі параметри дозволяють зберігати в полях лише документи, відповідної структури.
Автоматично обновляємі типи даних	
rowversion (стара назва - **timestamp**)	Розмір – 8 байт. При кожному додаванні запису в таблицю з полем даного типу, в нього автоматично додається нове значення часу. При обновленні таблиці значення поля rowversion також обновляються. Нове значення являє собою набір автоматично згенерованих чисел в двійковому форматі і тому його дані зберігаються в вигляді binary(8) з властивістю NOT NULL або varbinary(8) з властивістю NULL. В одній таблиці може існувати кілька полів даного типу. Згенеровані значення зручно використовувати для відслідковування порядку додавання елементів в таблицю.

Примітка! * Типи даних text та ntext призначені для зберігання великих масивів текстових даних, а image - двійкових. Але все ж ряд операцій для полів даного типу заборонені, наприклад, до них не можна застосовувати оператор рівності або конкатенації, їх не можна використовувати в багатьох системних функціях. В зв'язку з цими обмеженнями, ще в SQL Server 2005 з'явились типи даних varchar(max), nvarchar(max) та varbinary(max). Типи varchar(max) та nvarchar(max) об'єднують можливості типів text/ntext і varchar/nvarchar, можуть зберігати дані до 2 Гб і не мають обмежень по використанню з різними операціями і функціями. Щодо типу varbinary(max), то він може зберігати дані такого ж обсягу, як і image (до 2 Гб), та може використовуватись у всіх операціях і функціях, де допустимі типи даних binary/varbinary.

Варто також відмітити, що завдяки інтеграції CLR та SQL Server (починаючи з версії SQL Server 2005) можна створювати власні користувацькі типи даних CLR. Для цього слід здійснити наступні кроки:

1. Створити клас на одній з мов програмування Microsoft .NET, який відповідає специфікації користувацьких типів, наприклад, C#.
2. Написаний клас скомпілювати в динамічно підключаєму бібліотеку (DLL).
3. Зареєструвати бібліотеку в екземплярі SQL Server. Це може зробити тільки член серверної ролі sysadmin.
4. В базі даних включити підтримку типів даних CLR за допомогою утиліти Surface Area Configuration. При відключенні CLR всі поля з користувацькими типами даних CLR стануть недоступними.

Нажаль, вивчення створення та використання користувацьких типів даних CLR виходить за рамки нашого курсу. Більш детальніше про типи даних CLR дивіться документацію SQL Server.

3. Домени та їх створення

В SQL Server існує можливість створювати власні типи даних, що базуються на базових типах та включають ряд обмежень. Для чого? Як ви знаєте, існує певний набір типів даних, які визначають, які дані будуть зберігатись в окремих полях таблиці. Але, коли необхідно встановити більш жорсткі умови на введення даних, характеристик базового типу даних може бути недостатньо. Наприклад, необхідно задати умову на введення коректного віку працівника (додатнє значення) або встановити умову на введення значення в певному діапазоні тощо. Для встановлення таких обмежень використовують користувацькі типи даних, які називають **доменами**.

Домен – це тип, визначений користувачем для зручності застосування певних обмежень чи сукупності параметрів базових типів. В стандарті SQL2 вказується, що домен реалізований як частина бази даних. Згідно цього стандарту, **домен** являється іменованою сукупністю значень і використовується в БД як додатковий тип даних. Після створення домена на нього можна ссилатись як на звичайний тип даних.

Як вже було сказано, домени прив'язані до конкретної бази даних, але, якщо існує необхідність використання його у всіх базах даних в рамках поточного сервера, слід включити його в базу даних model.

В SQL Server 2008 було було знято 8-кілобайтне обмеження для користувацьких типів (User Defined Type, UDT), тобто для доменів, яке існувало в ранніх версіях. Це дозволило значно розширити можливості користувачів.

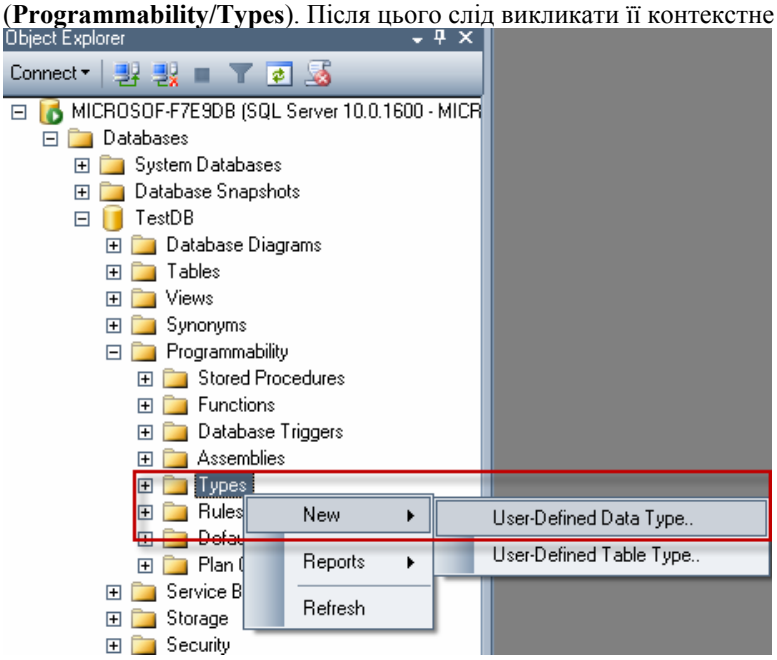
Існує **<u>три способи створення доменів</u>**:

- за допомогою візарда студії;
- за допомогою системних зберігаємих процедур;
- за допомогою команд SQL.

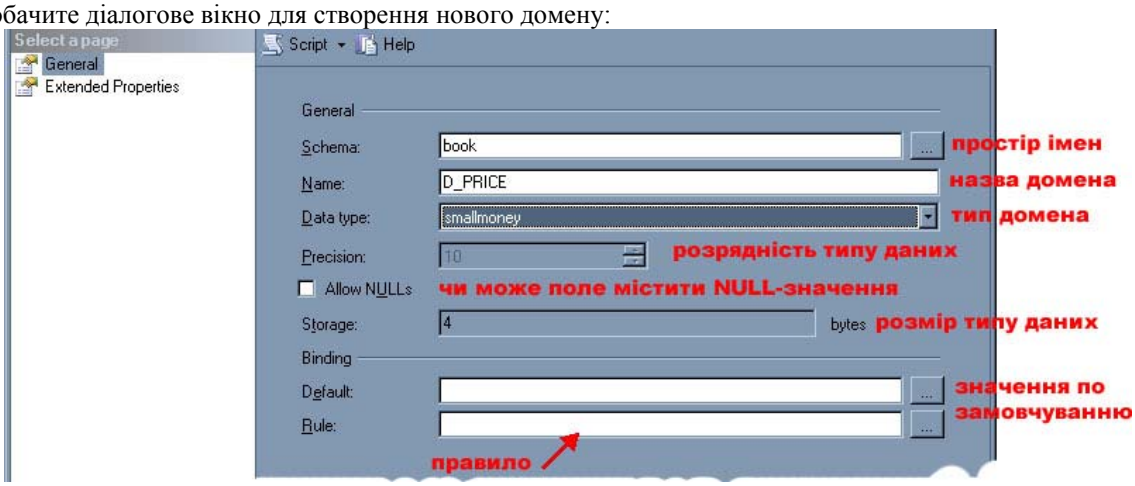
3.1. Засобами MS SQL Server

3.1.1. Створення, модифікація та видалення домену

Для цього потрібно обрати необхідну базу даних, в ній обрати папку, де розміщуються її користувацькі типи даних (**Programmability/Types**). Після цього слід викликати її контекстне меню та обрати пункт **New/User-Defined Data Type**.



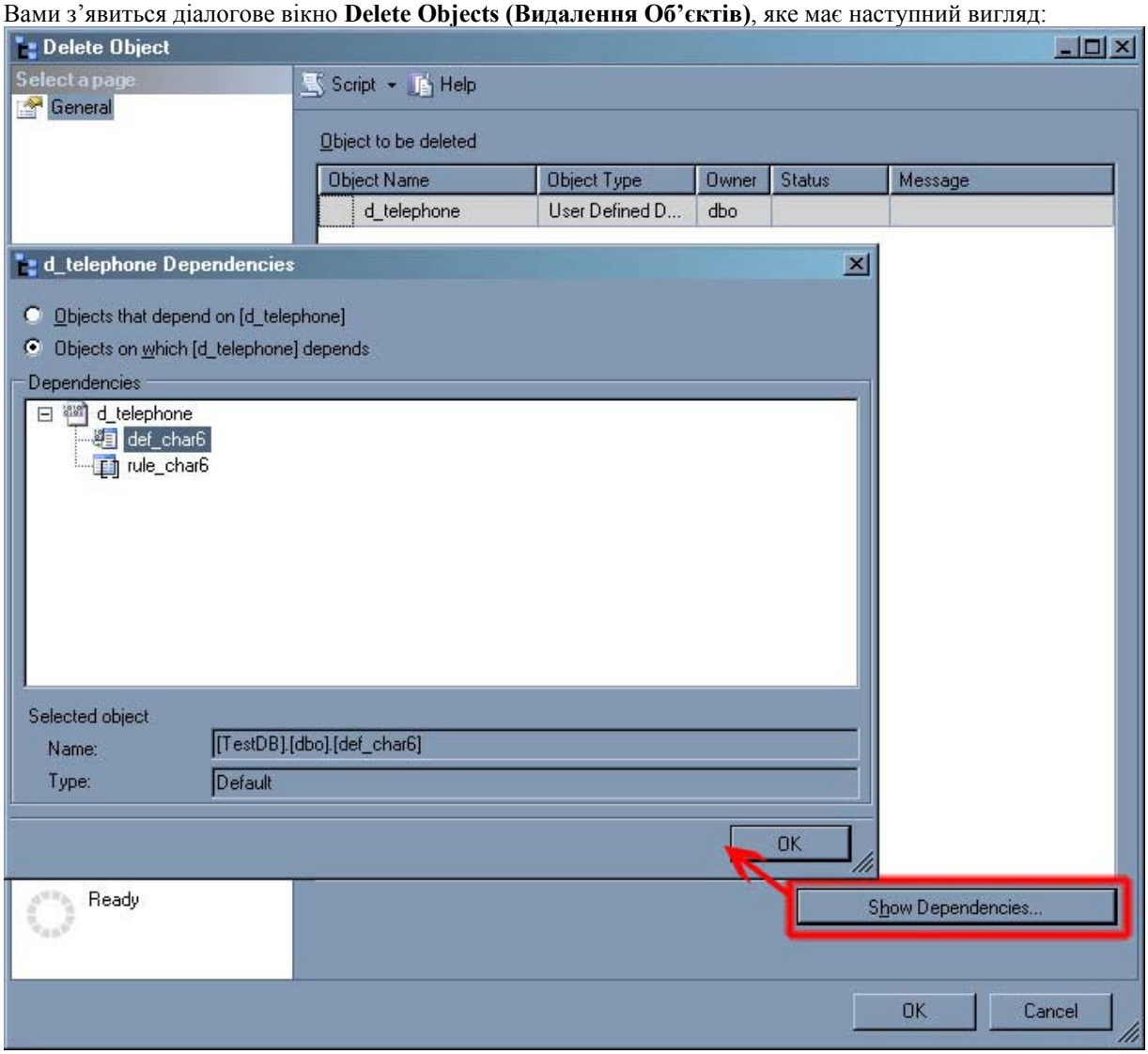
Далі ви побачите діалогове вікно для створення нового домену:



Слід відмітити, що поле **Precision** буде активне у випадку, якщо для типу даних можна вказати розрядність, а поле **Storage**, якщо існує можливість задати розмір. Наприклад, ви можете обрати тип даних varchar та задати йому довжину 10.

Для видалення доменів в Management Studio слід відкрити папку User-defined Data Types і в контекстному меню необхідного типу даних обрати пункт **Delete (Видалити)**.

Перед Вами з'явиться діалогове вікно **Delete Objects (Видалення Об'єктів)**, яке має наступний вигляд:



В ньому ви можете ще раз передивитися тип, який ви видаляєте і, при натисненні кнопки «OK», тип даних буде знищено. Але слід пам'ятати, що видалення типу домена буде неможливим у випадку його використання в базі даних в даний момент. Для того, щоб перевірити чи використовується користувацький тип, натисніть на кнопку «Show Dependencies…» (Показати залежності), як показано вище. При цьому на екрані з'явиться список таблиць та полів, в яких присутній перевіряємий користувацький тип.

3.1.2. Значення по замовчуванню та правила

Значення по замовчуванню – це значення, яке автоматично присвоюється записам поля, якщо їм не було присвоєне значення явно. Значення по замовчуванню задаються в директиві INSERT за допомогою оператора **DEFAULT** або у випадку проску даних в списку введення.

SQL Server підтримує **два види значень по замовчуванню**:

- **Обмеження DEFAULT в форматі ANSI**, до яких відносять: властивість IDENTITY (для генерації унікальних значень подібно типу даних «Лічильник»), обмеження на перевірку даних (check), обмеження первинного, зовнішнього, унікального ключів, інформація про які розміщується в системних таблицях sysreferences, syscomments, sysobjects та іноді в sysforeignkeys.
- **Окремі об'єкти-значення**, для створення яких використовується оператор CREATE DEFAULT:

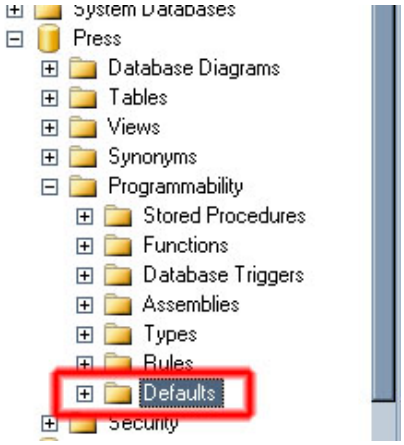
```
CREATE DEFAULT ім'я_значення_по_замовчуванню
AS вираз_константа
```

Вираз-константа відповідає типу даних полів, для яких використовується значення по замовчуванню. Наприклад:

- ✓ для типу char(3) - константа 'x' або 'xx', або 'xxx'
- ✓ для цілочисельного типу - довільне число
- ✓ для бітових типів - константа починається з 0x
- ✓ для грошових типів - константа починається з $
- ✓ для типів даних в форматі Unicode - константа починається з N

У виразі також можна використовувати функції SQL, але за умови, якщо вони повертають дані визначеного типу.

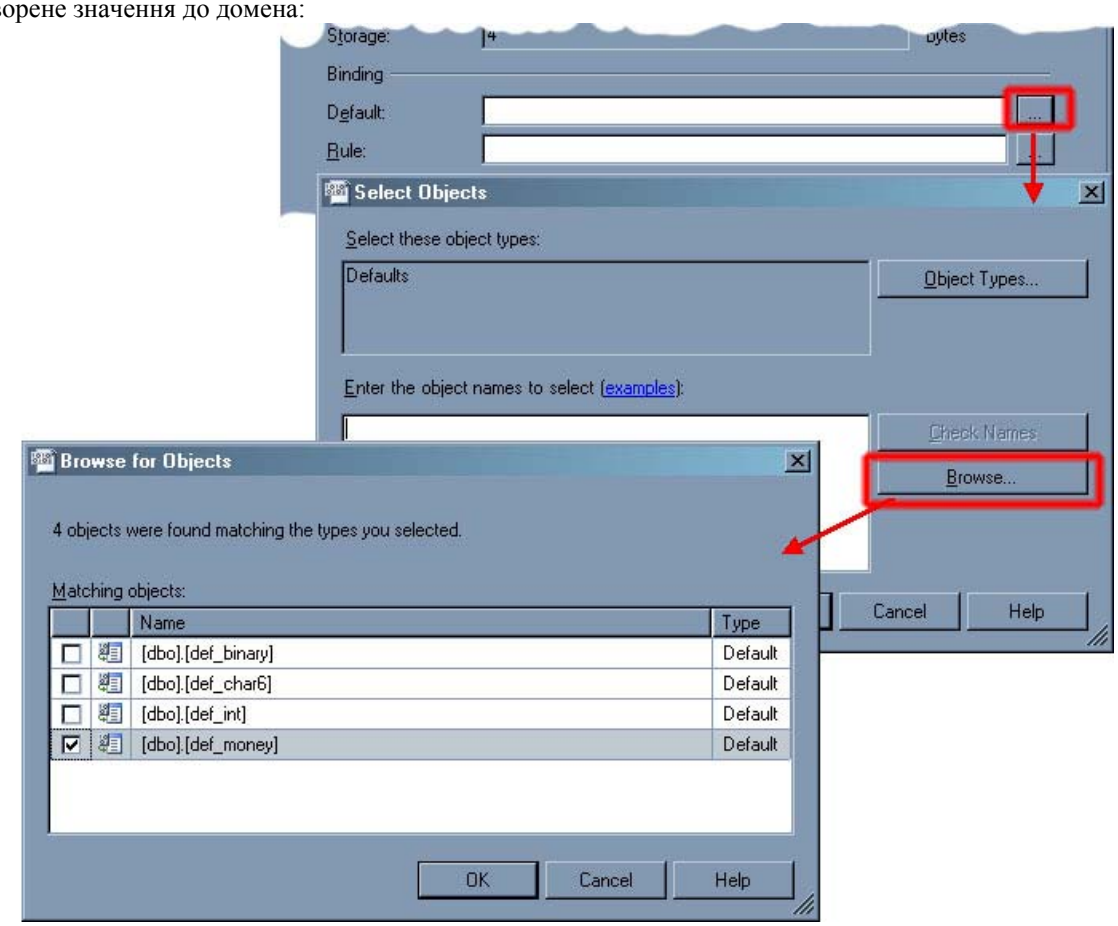
В кожній базі даних значення по замовчуванню (як і більшість об'єктів бази даних) зберігаються в системній таблиці **sysobjects**, текст оператора записується в таблицю **syscomments**, а фізичне розміщення знаходиться у папці по шляху **Programmability/Defaults**.



Отже, створимо кілька значень по замовчуванню:

```
create default def_int as 5            /*для цілочисельних типів*/
go
create default def_char6 as 'Ivanko' /*для рядкових типів, розміром не меньше 6 символів*/
go
create default def_money as $5.0      /*для грошових типів даних*/
go
create default def_binary as 0x00      /*для двійкових типів*/
go
```

Отже, маючи в розпорядженні кілька значень по замовчуванню, оберемо одне з них для нашого домена, тобто прив'яжемо створене значення до домена:



Видалити існуюче значення по замовчуванню можна за допомогою оператора **DROP DEFAULT**.

```
DROP DEFAULT ім'я_домена [,...n]
--наприклад
drop default def_int
```

З значенням по замовчуванню розібрались, залишились правила. **Правила** забезпечують додаткову підтримку доменної цілісності за допомогою перевірки допустимості значень. Задані за допомогою правил **значення перевіряються на**:

a) відповідність шаблону за допомогою оператора LIKE;
b) відповідність значенню з множини за допомогою оператора IN;
c) приналежність діапазону за допомогою оператора BETWEEN.

Для створення правил використовують оператор **CREATE RULE**:

```
CREATE RULE імя_правила
AS умова_вираз

-- умова-вираз повинна мати наступну форму
@змінна директива_where
```

Ім'я правила задається довільно, але у більшості випадків його задають таким чином, щоб воно було співзвучним з іменем поля, на яке накладається правило.

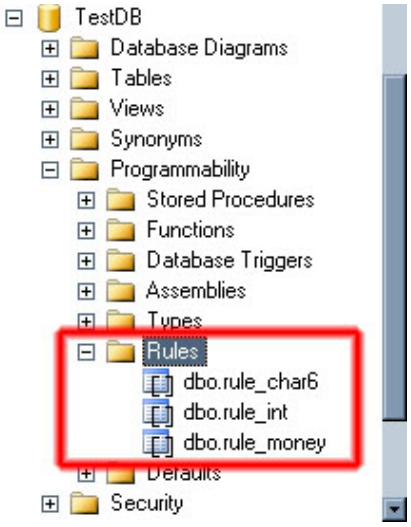
Умова в даному операторі являє собою довільну допустиму директиву WHERE, включаючи арифметичні оператори, оператори BETWEEN, LIKE, IN, AND, OR, NOT тощо. Але правило не може посилатись на змінні значення або інші поля бази даних. Для цього потрібно скористатись обмеженнями, які задаються значеннями по замовчуванню або тригерами.

Наприклад:

```
create rule rule_int as @ivar >= 0
go
```

```
create rule rule_char6 as @name like 'A%' or @name like 'B%'
go
create rule rule_money as @price between $10.0 and $50.0
go
```

Робочі правила, як і значення по замовчуванню, являють собою окремі об'єкти бази даних і зберігаються в тих же системних таблицях: sysobjects і syscomments, а фізично розміщуються по шляху **Programmability/Rules**.

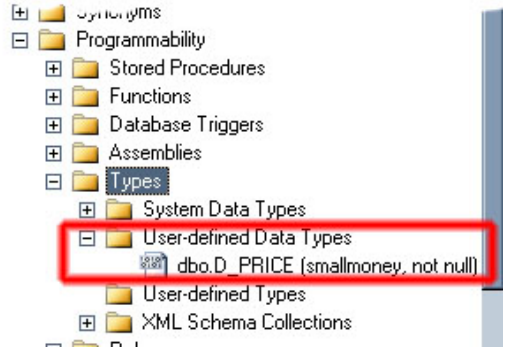


Прив'язка правила до домена здійснюється аналогічно як і привя'язка значення по замовчуванню.

Видалити вже існуюче правило можна за допомогою оператора **DROP RULE**.

```
DROP RULE ім'я_правила [,...n]
--наприклад
drop rule rule_char6
```

Після всіх дій, ми маємо доволі непоганий домен, який містить значення по замовчуванню і правило на перевірку введених значень:



Модифікувати домен, тобто змінити його параметри в SQL Server 2008 неможливо. Допускається лише його переіменування, яке можна зробити за допомогою пункта контекстного меню необхідного домена **«Rename»**.

3.2. За допомогою SQL запитів

В Transact-SQL існує інструкція **CREATE TYPE**, який дозволяє створити користувацький тип даних (домен).

```
CREATE TYPE [схема.] ім'я_домена
{
  FROM базовий_тип_даних
            [ ( загальна_кількість_розрядів [, розрядність_дробової_частини ] )  ]
            [ NULL | NOT NULL ]
| EXTERNAL NAME ім'я_збірки [.ім'я_класу ]
| AS TABLE ( список_полів_типу [ конструкції ] [ ,...n ] )
}
```

При створенні домену потрібно вказати назву схеми, до якої він буде належати (більш детальніше про схеми читайте в розділі «Схеми»), назву домена, базовий тип (який тип домен розширює) з розрядністю і точністю, якщо це необхідно. Поле типу домена можна також визначити як NOT NULL, що вкаже на обов'язковість внесення даних в поле.

Якщо планується використовувати в якості домена тип даних CLR, то необхідно вказати збірку SQL Server поточної бази даних (!), в якій він описаний. Після цього можна вказати назву класу в середині збірки, який реалізує визначений користувачем тип. При цьому ім'я класу може бути вказане в квадратних дужках ([ ]) з вказанням простору імен. Якщо аргумент «ім'я_класу» не вказаний, SQL Server вважає, що його значення рівне значенню аргумента «базовий_тип_даних».

Ви можете також створити домен табличного вигляду, задавши при цьому список полів такої таблиці з переліком необхідних конструкцій. Більш детальніше про оголошення полів таблиці дивіться у розділі «Таблиці. Основи побудови таблиць засобами SQL».

Наведемо кілька прикладів на створення доменів за допомогою оператора SQL. Спочатку приведемо простий приклад створення доменів на основі базових типів.

```
create type d_name from varchar(32) not null;
create type dbo.d_PersonName from nvarchar(32) not null;
```

Створимо домен Utf8String, який ссилається на клас utf8string в збірці utf8string. Відмітимо, що перед тим як створити такий тип, збірку utf8string потрібно зареєструвати в локальній базі даних. Зареєструвати збірку можна за допомогою оператора **CREATE ASSEMBLY** (див. документацію по SQL Server).

```
-- 1. Реєструємо збірку в базі даних
create assembly utf8string
from '\\127.0.0.1\library\utf8string.dll'
go
-- 2. Створюємо домен на основі типу utf8string з збірки utf8string.dll
create type Utf8String
external name utf8string.[Microsoft.Samples.SqlServer.utf8string]
go
```

Наступний приклад продемонструє створення домену вигляду таблиці, яка має два поля.

```
create type tabletype
as table (name varchar(10), digit int)
```

Такий табличний домен буде розміщуватись в папці **User-Defined Table Types**, яка також розміщується в директорії Programmability/Types.

Для видалення домену використовується оператор **DROP TYPE**:

```
DROP TYPE [схема.] імя_домена [,...n]
-- наприклад
drop type d_name, d_PersonName;
```

Нажаль, оператора SQL для модифікації домена в MS SQL Server не існує.

3.3. За допомогою зберігаємих процедур

Працювати з доменами можна також за допомогою системних зберігаємих процедур. Розглянемо їх по черзі.

1. Для створення доменів існує системна процедура **sp_addtype**:

```
sp_addtype [ @typename = ] ім'я_домена,
           [ @phystype = ] базовий_тип_даних
           [, [ @nulltype = ] 'null_type' ]
```

Аргумент nulltype вказує на спосіб обробки значень NULL доменом. Даний аргумент має тип varchar(8), значення по замовчуванню NULL, і його слід вказувати в одинарних лапках ('NULL', 'NOT NULL' або 'NONULL').

Наприклад:

```
exec sp_addtype d_salary, 'money'
exec sp_addtype d_telephone, 'varchar(24)', 'NOT NULL'
exec sp_addtype d_birthday, datetime, 'NULL'
```

2. Видалення домена здійснюється за допомогою системної зберігаємої процедури **sp_droptype**:

```
sp_droptype [ @typename = ] ім'я_домена [,...n]
```

Наприклад:

```
exec sp_droptype d_salary
exec sp_droptype d_birthday
```

3. Модифікувати домен неможна (тобто змінити його параметри) але допускається його переіменування. Переіменувати домен можна за допомогою системної зберігаємої процедури **sp_rename**.

```
sp_rename [ @objname = ] 'ім'я_домена',
          [ @newname = ] 'нове_ім'я_домена'
```

```
          [, [ @objtype = ] 'тип_об'єкта' ]
```

Аргумент objtype вказує на тип об'єкта, який буде переіменований і може приймати одне з наступних значень.

Значення	Опис
COLUMN	Поле, яке буде переіменоване
DATABASE	Користувацька база даних. Даний тип необхідний при переіменуванні бази даних.
INDEX	Користувацький індекс.
OBJECT	Елемент типу, який відслідковується в списку каталогів sys.objects. Наприклад, значення OBJECT може бути використано для переіменування об'єктів з обмеженнями (CHECK, FOREIGN KEY, PRIMARY/UNIQUE KEY), користувацьких таблиць і правил.
USERDATATYPE	Тип даних домену або користувацькі типи CLR

Наприклад:

```
exec sp_rename 'd_salary', 'd_s'
```

4. До вже існуючого домена ви можете прив'язати або від'єднати існуючі правила одним з нижчеописаних способів.

```
-- прив'язати правило до об'єкта БД
sp_bindrule [ @rulename = ] 'назва_правила',
            [ @objname = ] {'домен' | 'таблиця.поле' }
            [, [ @futureonly = ] 'futureonly' ]
-- від'єднати правило від об'єкта БД
sp_unbindrule [ @objname = ] {'домен' | 'таблиця.поле' }
              [, [ @futureonly = ] 'futureonly' ]
```

Якщо флаг futureonly вказується при прив'язуванні правила до домена, передбачається успадкування нового правила лише визначеним і існуючим полем типу домена. Якщо даний аргумент має значення NULL, тобто він не встановлений, то нове правило прив'язується до довільного поля типу домена, яке на даний момент не має правила, або до того поля, яке використовує існуюче правило домена.

Аргумент futureonly може вказуватись і при від'єднанні правила. В такому разі передбачається, що існуючі поля цього типу даних не будуть втрачати заданного правила.

5. Приєднати або від'єднати існуючі значення по замовчуваню можна за допомогою наступних процедур.

```
-- прив'язати значення по замовчуванню до об'єкта БД
sp_bindefault [ @defname = ] 'імя_значення_по_замовчуванню',
              [ @objname = ] {'домен' | 'таблиця.поле'}
              [,[ @futureonly = ] 'futureonly' ]
-- від'єднати значення по замовчуванню від об'єкта БД
sp_unbindefault [ @objname = ] {'домен' | 'таблиця.поле' }
                [, [ @futureonly = ] 'futureonly' ]
```

Наприклад:

```
--створюємо домен та таблицю, які будуть містити значення по замовчуванню
exec sp_addtype d_telephone, 'varchar(24)', 'NOT NULL'
create table Test
(
      id  int not null primary key,
      age int not null
);
go
--створюємо значення по замовчуванню
create default def_char6 as 'Ivanko'
create default def_int as 5
go
-- прив'язуємо до необхідних полів та доменів значення по замовчуванню
--exec sp_bindefault @defname = def_char6, @objname = 'd_telephone'
exec sp_bindefault def_char6, 'd_telephone'
exec sp_bindefault def_int, 'Test.age'
go
--створюємо правила
create rule rule_int as @ivar >= 0
create rule rule_char6 as @name like 'A%' or @name like 'B%'
go
```

```
-- прив'язуємо до необхідних полів та доменів правила
exec sp_bindrule 'rule_char6', 'd_telephone'
exec sp_bindrule 'rule_int', 'Test.age'

-- правило прив'язується до домена d_telephone,
-- але на існуючі поля такого типу це не впливає
-- exec sp_bindrule rule_char6, 'd_telephone', 'futureonly'
go
-- відміняємо прив'язку правил до всіх майбутніх полів типу домена.
-- Для всіх поточних полів типу rule_int правила "залишаються в силі"
exec sp_unbindrule 'rule_int', 'futureonly'
```

4. Таблиці. Основи побудови таблиць засобами SQL

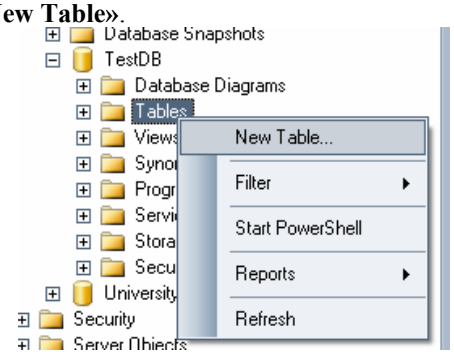
4.1. Засобами MS SQL Server

SQL Server - це реляційна СУБД. Отже, дані в ній представлені у вигляді таблиць. Всі правила щодо створення та нормалізації таблиць, які ви вчили на попередніх курсах баз даних, зберігаються і в SQL Server. Існує також два **способи** роботи з таблицями:

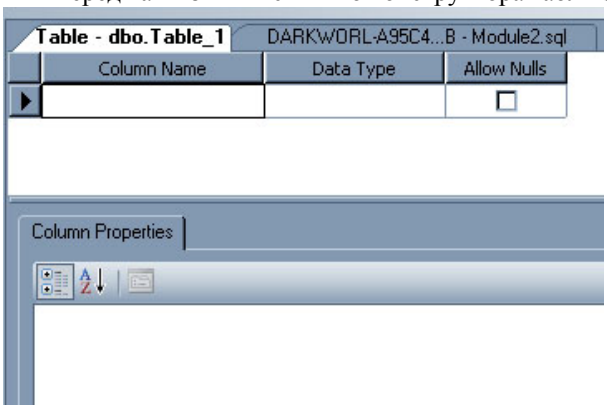
1. Засобами Management Studio;
2. За допомогою операторів мови запитів SQL.

Щоб наша розповідь була повною, розглянемо обидва способи.

Для створення таблиці засобами Management Studio, необхідно в контекстному меню папки **«Tables»** обрати пункт **«New Table»**.



Перед вами з'явиться вікно конструктора таблиць, в якому Вам необхідно заповнити дані про нову таблицю:



- **Column name** – ім'я поля створюваної таблиці.
- **Data type** - тип даних для поточного поля. Тип поля по замовчуванню nchar(10).
- **Allow NULLs** – чи допускаються пусті (NULL) поля, тобто чи воно є обов'язковим. По замовчуванню флажок не встановлений, тобто поле є обов'язковим для заповнення.

Крім того, кожне поле може мати і інші розширені характеристики, які залежать від його типу даних. До основних додаткових **характеристик належать**:

- ✓ **Length** – кількість символів (якщо поле текстове) або кількість необхідних байт (якщо поле числове або бітове);
- ✓ **Default Value or Binding** - значення по замовчуванню;
- ✓ **Description** - опис створюваного поля;
- ✓ **Precision** – загальна кількість розрядів для дійсних типів даних;
- ✓ **Scale** – кількість розрядів після десяткової крапки для дійсних типів даних;
- ✓ **Identity Specification** – використовується для задання лічильника в поля (зазвичай використовується для первинного ключа). Якщо флаг **Is Identity** встановлений в **Yes**, то значення в полі можна прирощувати. При цьому поле **Identity Seed** визначає початкове значення лічильника, а **Identity Increment** величину прирощування;
- ✓ **Not For Replication** – активне за умови використання лічильника в полі. Якщо поле приймає значення Yes, то лічильник в реплікаціях не використовується;

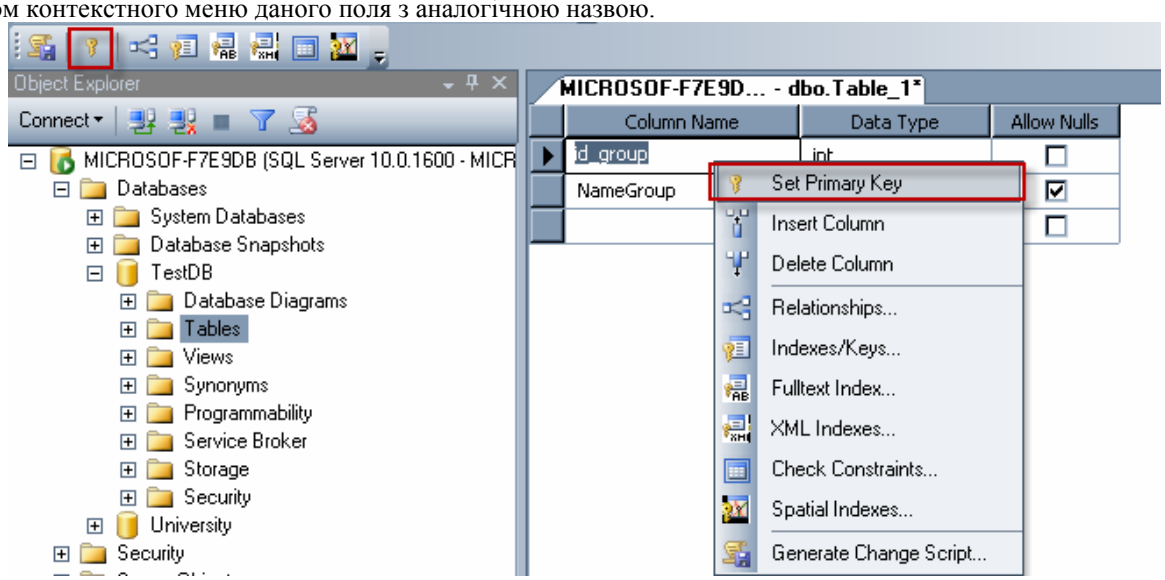
- ✓ **Collation** – чи буде здійснюватись сортування поля при видаленні даних;
- ✓ **Size –** розмір поля в байтах;
- ✓ **RowGUID –** використовується лише з типом даних **uniqueidentifier**.

Для прикладу, створимо маленьку таблицю, яка буде мати наступну структуру:

- id_group – тип int not null
- nameGroup – тип varchar(10) not null

Після того, як таблиця створена її можна зберегти під певним ім'ям, наприклад, **Groups**.

Як бачите, створити виявилось не дуже складно, але це ж не все. Не слід забувати про те, що згідно правил нормалізації, кожна таблиця повинна мати первинний ключ. Для того, щоб задати первинний ключ певному полю нашої новоствореної таблиці, слід обрати необхідне поле і скористатись або кнопкою **«Set Primary Key»** на панелі інструментів, або ж пунктом контекстного меню даного поля з аналогічною назвою.

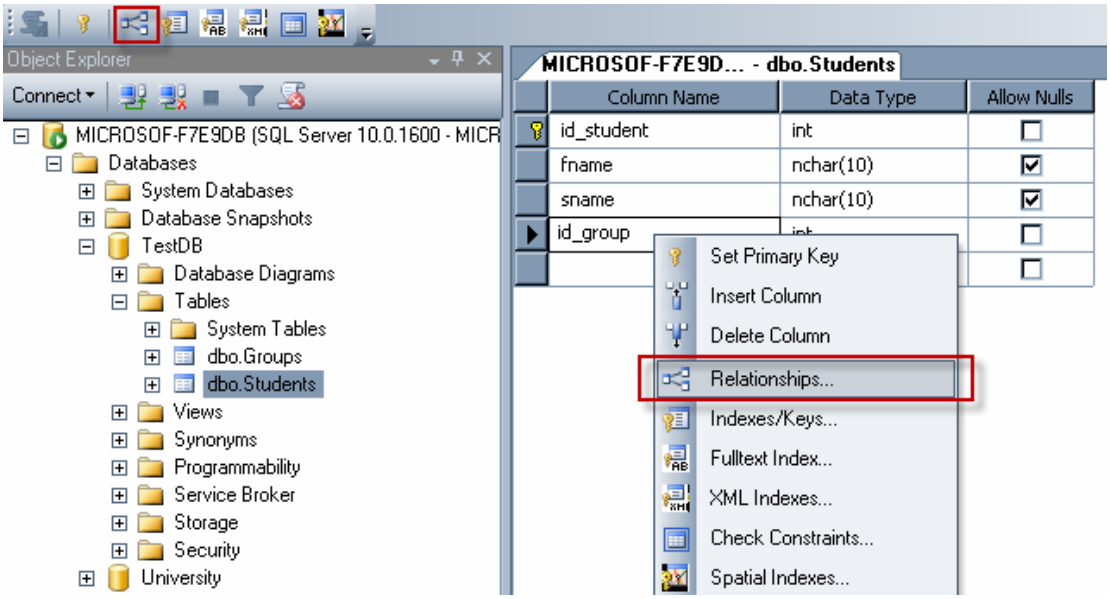


Якщо після створення таблиці необхідно змінити параметри того чи іншого поля, потрібно в контекстному меню таблиці, параметри якої потребують модифікації, обрати пункт меню **«Design»**. Він відкриє вже знайомий конструктор таблиці.

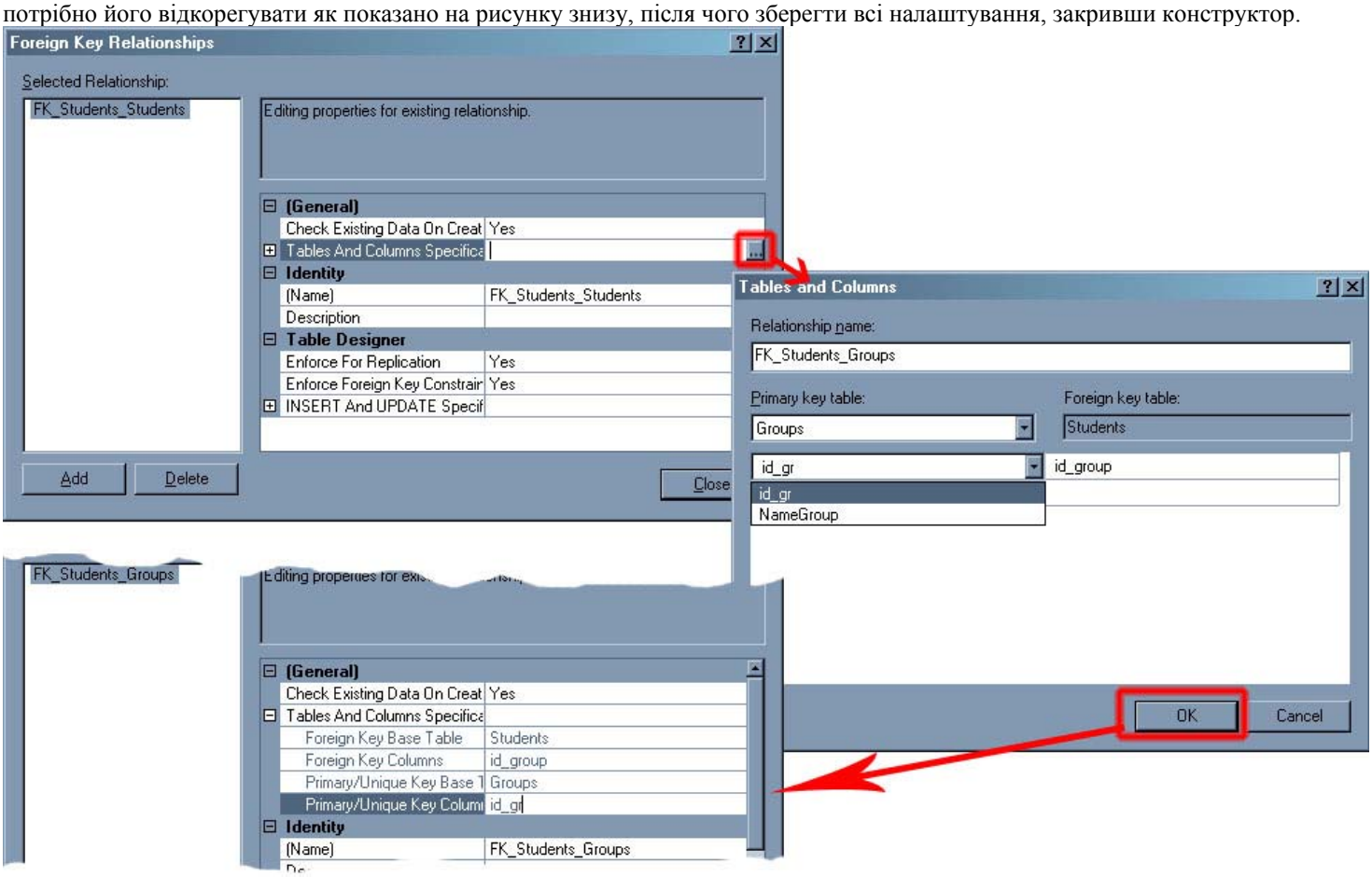
Йдемо далі. Як задати первинний ключ таблиці ми вже знаємо, але крім первинних ключів, існують зовнішні (вторинні) ключі, які дозволяють зв'язати дві таблиці між собою. Щоб краще зрозуміти як це можна зробити, створимо ще одну таблицю **Students**, яка буде містити певну інформацію про студента. При цьому дозволимо кожному студенту навчатись в різних групах, дані про які зберігаються у вже створеній таблиці Groups. Отже, таблиця Students матиме наступну структуру:

- id_student – тип int not null
- fname – тип varchar(10) null
- sname – тип varchar(10) not null
- id_group - тип int null

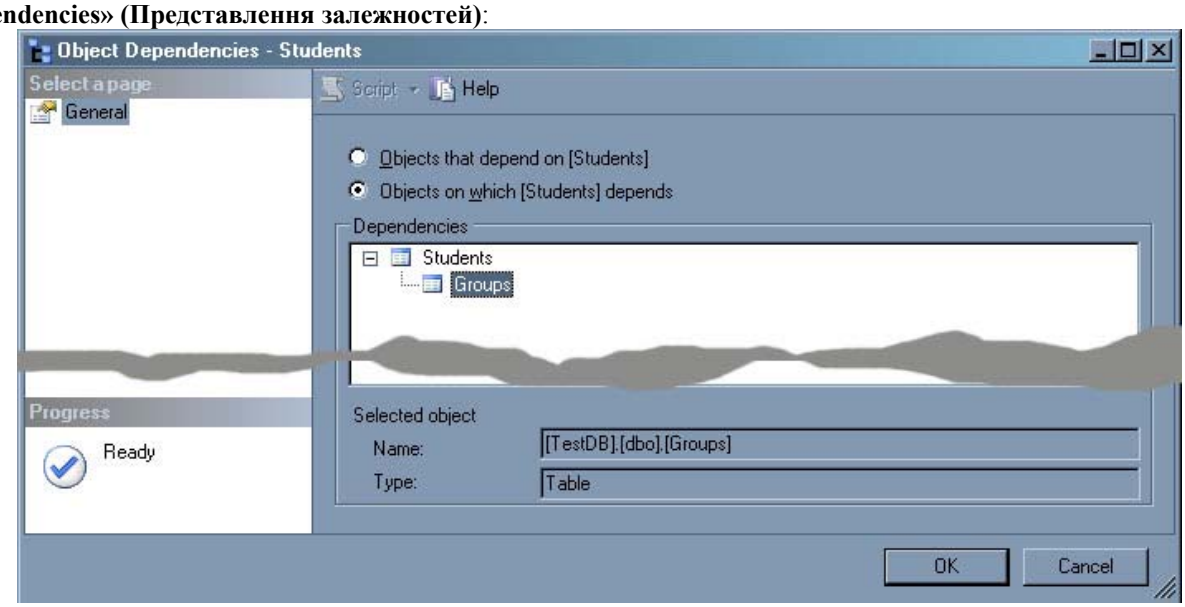
Тепер зв'яжемо таблиці Students та Groups по полю id_group. Для цього робимо дане поле активним і викликаємо конструктор створення зв'язків або за допомогою кнопки **«Relationships»** на панелі інструментів, або за допомогою пункта контекстного меню з аналогічною назвою:



Перед Вами з’явиться діалогове вікно, в якому при натисненні на кнопку **«Add»** буде доданий новий зв’язок. Далі потрібно його відкорегувати як показано на рисунку знизу, після чого зберегти всі налаштування, закривши конструктор.



Щоб переглянути в будь-який момент встановлені зв'язки, слід обрати в контекстному меню таблиці Students пункт **«View Dependencies» (Представлення залежностей)**:



Ну і насамкінець. Для видалення вже існуючої таблиці потрібно в тому ж контекстному меню таблиці обрати пункт **«Delete» (Видалити)**.

4.2. За допомогою SQL запитів

Створення таблиць в SQL Server здійснюється за допомогою інструкції SQL **CREATE TABLE**, узагальнений синтаксис якої наступний:

```
CREATE TABLE [ім'я_БД.][схема.] назва_таблиці
( назва_поля тип_даних
                        [ FILESTREAM ]
                        [ COLLATE напрям_сортування ]
                        [ SPARSE ]                          /* розріжене поле */
                        [ NULL | NOT NULL ]              /* чи допускаються NULL значення */
                        [ CONSTRAINT назва_конструкції ] /* назва конструкції */
                        [ DEFAULT вираз_константа ]      /* значення по замовчуванню */
                        [
                           [ IDENTITY [ ( поч_значення, крок_прирощення ) ]
                           [ NOT FOR REPLICATION ]
                        ]
                        [ ROWGUIDCOL ]
                        [ [CONSTRAINT] конструкції_поля [ ...n ] ]
 [ ,...n ]
)
[ON { ім'я_схеми_секції ( ім'я_поля_секції ) | група_файлів | "default"  } ]
[ TEXTIMAGE_ON { група_файлів | "default" } ]
[ FILESTREAM_ON { ім'я_схеми_секції | група_файлів | "default"  }]
[ WITH ( опції_таблиці [ ,...n ] ) ]
```

Інструкція **ON** вказує, де фізично буде зберігатись таблиця. Якщо файлова група не задана, SQL Server створить таблицю, використовуючи файлову групу по замовчуванню. У випадку встановлення аргумента **«ім'я_схеми_секції»**, таблиця буде розбита на секції, які будуть збережені в одній або кількох вказаних файлових групах.

Інструкція **TEXTIMAGE_ON** вказує на те, що поля типів text, ntext, image, xml, varchar(max), nvarchar(max), varbinary(max), а також користувацьких типів середовища CLR будуть зберігатись в окремій вказаній файловій групі. Зрозуміло, що даний параметр недопустимий, якщо в таблиці не існує полів з великими значеннями.

Параметр **FILESTREAM_ON** задає файлову групу для даних FILESTREAM у випадку, якщо таблиця містить такі дані та являється секціонованою. В такому разі вказується схема секціонування файлових груп файлового потоку. Більш детальнішу інформацію про параметр FILESTREAM дивіться в розділі «Проектування і реалізація даних FILESTREAM» документації.

Інструкція **WITH** використовується для вказання додаткових характеристик таблиці, наприклад, можна вказати підтримку компресію (зжаття) рядків.

Параметр **IDENTITY** використовується для встановлення лічильника для поля (зазвичай використовується для первинного ключа). При цьому можна визначити початкове значення лічильника та величину прирощування, яка може бути як додатнім, так і від'ємним числом. По замовчуванню початкове значення рівне 1, а крок прирощення +1.

Параметр **NOT FOR REPLICATION** може вказуватись для властивостей IDENTITY, а також обмежень FOREIGN KEY і CHECK (див. інформацію про обмеження даного розділу). Якщо ця інструкція вказана для властивості IDENTITY, то значення в полях ідентифікаторів не прирощуються, якщо вставка виконується в наслідок реплікації.

ROWGUIDCOL вказує на те, що в таблиці можна назначити лише одне поле типу uniqueidentifier. Цю властивість можна встановити лише полю з типом uniqueidentifier і використовувати її не допускається, якщо рівень сумісності бази даних рівний 65 або нижче.

SPARSE вказує на розріжене поле. Розріжені поля з'явились в SQL Server 2008 і являють собою звичайні поля, які мають оптимізоване сховище для NULL значень. Для розріжених полів не можна вказувати параметр NOT NULL. Завдяки цьому компоненту значення NULL більше не займають фізичний простір, що робить управління пустими даними ефективним. Крім того, розріжені поля дозволяють створювати об'єктні моделі з великою кількістю NULL значень, не займаючи при цьому багато місця на диску. Але, не дивлячись на переваги таких полів, використовувати їх слід лише у випадку, якщо економиться не менше 20-40% місця.

Опція **FILESTREAM** допустима лише для полів типу varbinary(max) і вказує на сховище FILESTREAM для даних BLOB типу varbinary(max). При цьому таблиця не повинна містити поле типу uniqueidentifier з атрибутом ROWGUIDCOL. Це поле не повинно допускати NULL значень і повинно мати обмеження UNIQUE або PRIMARY KEY. GUID значення для такого поля задається при вставці даних або обмеженням DEFAULT, в якому використовується функція NEWID().

Щодо конструкцій, які вказуються в кінці оголошення поля, то вони можуть починатись з ключового слова CONSTRAINT і вказувати на обмеження PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY або CHECK для поля.

Для прикладу, створимо спочатку простеньку таблицю:

```
use TestDB
go
create table Test1
(
 id_test1   int identity not null,
 fname      varchar(20) default 'Unknown',
 Surname    d_name,                 /*поле типу домена*/
 Salary     money not null,  /*ненулеве поле з можливістю збереження грошових величин*/
 percent_   as (Salary * 0.2),      /*розрахункове поле*/
 age        int check(age > 20),    /*поле з обмеженням*/
 test       nvarchar(20) sparse     /*розріжене поле*/
);
```

В наступному прикладі створимо таблицю, яка підтримує компресію рядків:

```
create table Test2
(
 id_test2   int identity(1,2) not null,
 literal       nvarchar(20)
)
with (DATA_COMPRESSION = ROW);
```

Для модифікації вже створеної таблиці використовується оператор **ALTER TABLE**, узагальнений синтаксис якого наступний

```
ALTER TABLE [ім'я_БД.][схема.] назва_таблиці
{
    /* зміна інформації про поле */
    ALTER COLUMN назва_поля
    {
        [ схема. ] тип_даних
        [ COLLATE напрям_сортування ]
        [ SPARSE | NULL | NOT NULL ]
        | {ADD | DROP } { ROWGUIDCOL | PERSISTED | NOT FOR REPLICATION | SPARSE }
    }
    | [ WITH { CHECK | NOCHECK } ]

    /* додавання та знищення характеристик полів та самих полів */
    | ADD  { опис_нового_поля | [ CONSTRAINT ] конструкція } [ ,...n ]
    | DROP {
        [ CONSTRAINT ] ім'я_конструкції
```

```
        [ WITH (параметри_видалення_кластеризованого_обмеження [ ,...n ] ) ]
        | COLUMN назва_поля
    } [ ,...n ]

    /* включені чи вимкнені конструкції. ALL – всі конструкції. Даний параметр може
       використовуватись лише для конструкцій FOREIGN KEY та CHECK */
    | { CHECK | NOCHECK } CONSTRAINT
        { ALL | ім'я_конструкції [ ,...n ] }

    /* включені чи вимкнені тригери, які встановлені на таблицю. ALL – всі тригери */
    | { ENABLE | DISABLE } TRIGGER
        { ALL | ім'я_тригера [ ,...n ] }

    /* дозволено чи ні відслідковування змін в даній таблиці. Щоб дозволити таке
       відслідковування, в таблиці повинен бути первинний ключ */
    | { ENABLE | DISABLE } CHANGE_TRACKING
        [ WITH ( TRACK_COLUMNS_UPDATED = { ON | OFF } ) ]

    | SWITCH [ PARTITION секції_основної_таблиці ]
        TO [ схема. ] цільова_таблиця
        [ PARTITION секції_цільової_таблиці ]

    /* вказує місцезнаходження сховища даних FILESTREAM.
     "NULL" вказує на видалення всіх посилань на файлові групи FILESTREAM для таблиці */
    | SET ( FILESTREAM_ON = { ім'я_схеми_секції | група_файлів | "default"  | "NULL" } )

    /* для перебудови таблиці */
    | REBUILD
      [ [PARTITION = ALL]
        [ WITH ( опції_перебудови [ ,...n ] ) ]
      | [ PARTITION = номер_секції
          [ WITH ( <single_partition_rebuild_option> [ ,...n ] ) ]
        ]
      ]

    /* опції для блокування таблиці*/
    | (SET ( LOCK_ESCALATION = { AUTO | TABLE | DISABLE } ))
}
```

Опція **«WITH CHECK | WITH NOCHECK»** вказує на те, чи задовольняють дані в таблиці нещодавно доданому або повторно включеному обмежеенню FOREIGN KEY або CHECK. Якщо нічого іншого не вказано, то для нових обмежень рекомендується використовувати WITH CHECK, а для повторно включених обмежень - WITH NOCHECK.

Опція **SWITCH** дозволяє переключити блок даних одним з наступних способів:

- переназначає всі табличні дані як секцію у вже існуючій секціонованій таблиці;
- перемикає секції з однієї секціонованої таблиці в іншу;
- перемикає всі дані однієї секції секціонованої таблиці у вже існуючу несекціоновану таблицю.

Якщо основна таблиця являється секціонована, то необхідно вказати аргумент PARTITION після опції SWITCH, якщо ж цільова таблиця секціонована, то повинен вказуватись аргумент PARTITION після її імені. Якщо відбувається переназначення даних таблиці як секції у вже існуючу секціоновану таблицю або переключення секції з однієї секціонованої таблиці на іншу, то цільова секція повинна існувати і бути пустою.

Аргументи «секції_основної таблиці» та «секції_цільової_таблиці» являються постійними виразами, які можуть ссилатись на змінні, включаючи домени, і функції. При цьому вони не можуть ссилатись на вирази мови Transact-SQL.

Опція **SET** для блокування таблиці може приймати одне з наступних параметрів, які вказують на метод блокування:

- AUTO – дозволяє SQL Server Database Engine обрати гранулярність укрупнення блокування, яка підходить до даної схеми таблиці.
- TABLE (значення по замовчуванню) – укрупнення блокування буде виконуватись на рівні гранулярності таблиці, незалежно від того, секціонована таблиця чи ні.
- DISABLE – забороняє укрупнення блокування, але не повністю. Наприклад, при скануванні таблиці, яка не має кластеризованого індекса на рівні ізоляції SERIALIZABLE, компонент Database Engine повинен встановити блокування таблиці для захисту цілісності даних.

Опція **REBUILD** використовується для перебудови таблиці. Параметр REBUILD WITH використовується для перебудови всієї таблиці, включаючи всі секції у секціоновану таблицю. Для перебудови однієї секції в секціонованій таблиці використовуйте параметр REBUILD PARTITION. Даний параметр може приймати одне з двох значень:

- PARTITION = ALL - перебудовує всі секції при зміні налаштувань компресії секцій;
- REBUILD WITH (опції_перебудови) – всі вказані опції будуть використані для таблиці з кластеризованим індексом. Якщо в таблиці немає кластеризованого індекса, то на структуру кучі впливають лише певні параметри.

Наведемо кілька прикладів на модифікацію структури таблиці.

```
-- додоємо нове поле NullCol типу nvarchar(20)
alter table MyTable alter column NullCol nvarchar(20) not null

-- додати додоткове розріжене поле test
alter table MyTable
add test char(100) sparse null;

-- перетворюємо розріжене поле test в нерозріжене
alter table MyTable
alter column test drop sparse;
```

Для видалення таблиці існує оператор **DROP TABLE**.

```
DROP TABLE [ім'я_БД.][схема.] назва_таблиці [ ,...n ]

-- наприклад
drop table test1;
drop table testdb.dbo.test2;
```

Щоб таблиці бази даних були нормальними формами і підлягали всім правилам нормалізації, вони повинні містити первинні та зовнішні ключі. Первинний ключ для певного поля таблиці можна встановити одним з наступних способів:

```
-- 1. Шляхом встановлення конструкції для поля при первинному створенні таблиці:
create table Table1(id_test1   int identity not null primary key,
                    ..............
                    );

-- 2. Шляхом встановлення конструкції для поля при первинному створенні таблиці після
-- оголошення всіх полів
create table Table1(id_test1   int identity not null,
                    .............. ,
                    constraint pkTable1 primary key(id_test1)
                    );

-- 3. Шляхом модифікації вже існуючої таблиці, тобто коли таблиця створена, а первинний
-- ключ ще не заданий
alter table Test1
add constraint pkTable1 primary key (id_test1);
```

Для видалення вже встановленого первинного ключа потрібно написати наступний запит на зміну таблиці:

```
alter table Test1
drop constraint pkTable1 primary key(id_test1);
```

Для вставлення зовнішнього ключа слід скористатись одним з нижчеописаних сценаріїв:

```
-- 1. Шляхом встановлення конструкції для поля при первинному створенні таблиці:
create table test2 (id_test2 integer not null primary key,
            id_test1 integer not null constraint fkTest2_1 references Test1 (id_test1));
-- або
create table test2 (id_test2 integer not null primary key,
                    id_test1 integer not null references Test1 (id_test1));

-- 2. Шляхом встановлення конструкції для поля при первинному створенні таблиці після
-- оголошення всіх полів
```

```
create table test2 ( id_test2   integer not null primary key,
                     id_test1   integer not null,
                  constraint fkTest2_1 foreign key (id_test2) references Test1 (id_test1))

-- 3. Шляхом модифікації вже існуючої таблиці, тобто коли таблиця створена, а вторинний
-- ключ ще не заданий
alter table test2
add constraint fkTest2_1 foreign key (id_test2) references Test1 (id_test1))
```

SQL Server також дозволяє виконувати ряд автоматичних дій при редагуванні або видаленні зовнішнього ключа, тобто даних, на які він ссилається. Це дозволяє забезпечити цілісність даних в базі даних. Для цього використовується наступний набір опцій зовнішнього ключа:

```
[ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
[ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL }]
```

При цьому можливі наступні **варіанти**:

- **No Action** (по замовчуванню) – неможливо змінити або видалити значення в головній таблиці (первинний ключ), поки існують відповідні значення в підпорядкованій таблиці (зовнішній ключ);
- **Cascade** – при зміні або видаленні значень в зв'язаному полі (первинний ключ) з головної таблиці змінюються або знищуються (відповідно) значення в зв'язаному полі з підпорядкованої таблиці (зовнішній ключ);
- **Set Null** – значення поля, яке являється зовнішнім ключем встановлюється в NULL;
- **Set Default** - значення поля, яке являється зовнішнім ключем встановлюється в значення по замовчуванню, яке повинно бути в списку значень первинного ключа головної таблиці.

Наприклад, забезпечимо цілісність даних, на які ссилається зовнішній ключ таблиці test2.

```
create table test2 ( id_test2   integer not null primary key,
                     id_test1   integer not null,
                  constraint fkTest2_1 foreign key (id_test2) references Test1 (id_test1)
                                                              on delete no action
                                                              on update cascade )
```

Ви можете розглянути ще кілька прикладів використання оператора CREATE TABLE для створення таблиць в файлі University.sql поточної директорії.

5. Схеми

В SQL Server вирізняють таке поняття як схеми, яке співзвучне з просторами імен, в межах яких можуть знаходитись таблиці, представлення, зберігаємі процедури і тригери та інші об'єкти бази даних. Починаючи з версії MS SQL Server 2005 схеми реалізовані у відповідності до стандарту ANSI і являють собою колекції об'єктів баз даних, тоді як в ранніх версіях об'єкти знаходились в межах їх власника. Ім'я користувача тепер не являється частиною імені об'єкта, тому імена користувачів бази даних можна змінювати і навіть видаляти, не вносячи при цьому змін в додаток. У кожної схеми є свій власник – користувач або роль, і якщо їх необхідно видалити, володіння схемою слід передати іншому користувачу або ролі.

Для створення схем використовується оператор **CREATE SCHEMA**, скорочений синтаксис якого виглядає наступним чином:

```
CREATE SCHEMA { назва_схеми | AUTHORIZATION власник
                | назва_схеми AUTHORIZATION власник }
              [ елемент_схеми [ ...n ] ]

-- елемент схеми описується:
{ таблиця | представлення
  оголошення_GRANT | оголошення_REVOKE | оголошення_DANY }
```

Як видно з інструкції, при створенні схеми ви можете їй вказати конкретного власника або ж залишити власником поточного користувача, який являється її створювачем. Інструкції, які містять **CREATE SCHEMA AUTHORIZATION**, тобто не вказують ім'я, дозволені лише для зворотної сумісності.

Після вказаного імені та власника, можна здійснити перелік елементів схеми, тобто перелік таблиць та інших об'єктів, які будуть створені всередині схеми. Цей перелік задається інструкціями CREATE TABLE, CREATE VIEW, GRANT, REVOKE або DENY. Останні три інструкції надають, відміняють або забороняють права доступу на захищені об'єкти, за виключенням новоствореної схеми.

Після створення, інформація про схему заноситься в представлення каталога **sys.schemas**.

Приведемо кілька прикладів:

```
-- створюється схема, яка належить поточному користувачу
create schema mgmt;

-- створюється схема people, яка належить користувачу Pupkin
create schema people authorization Pupkin;

-- створюється схема people, яка належить користувачу Pupkin і містить таблицю
-- testTable. Інструкція також надає право на SELECT для користувача Jeckson
-- і забороняє SELECT для Volydemar.
-- Примітка! І схема people і таблиця TestTable створюються в одній інструкції
create schema people authorization Pupkin
    create table testTable (id int not null, number int)
    grant select to Jeckson
    deny select to Volydemar;
```

Назначити таблицю (або інший об'єкт) певній схемі не обов'язково при її безпосередньому створенні. Набагато зручніше вказати область бачення об'єкта при створенні самого об'єкта, тобто таблиці, вказавши перед її іменем назву схеми. Наприклад:

```
create table people.testTable (id int not null, number int);
go
select *
from people.testTable;
go
```

Щоб змінити схему, використовується інструкція **ALTER SCHEMA**:

```
ALTER SCHEMA назва_схеми
      TRANSFER [ тип_сутності :: ] ім'я_об'єкта_для_переміщення
```

В інструкції alter schema необхідно обов'язково вказати назву нової схеми та ім'я об'єкта для переміщення (як правило, з вказанням імені старої схеми). У випадку необхідності можна вказати тип переміщаємої сутності, який може приймати одне з наступних значень:

- object (по замовчуванню);
- type;
- xml_schema_collection.

Наприклад:

```
-- перемістити таблицю Address з схеми People в схему Resources
alter schema Resources transfer People.Address;

-- перемістити тип TestType з схеми People в схему Resources
alter schema Resources transfer type::People.TestType ;
```

Для видалення існує оператор **DROP SCHEMA**:

```
DROP SCHEMA назва_схеми [, ...n ]
-- наприклад
drop schema people;
```

6. Оператори вставки, модифікації та видалення таблиць

З створенням бази даних, таблиць, схем, в яких можна розміщувати таблиці та інші об'єкти розібрались. Залишилось розібратись з тим, як додавати, видаляти та обновлювати дані в базі даних. Засобами Management Studio, починаючи з версії SQL Server 2008 це зробити неможливо. Здійснити такі дії можна лише засобами мови структурованих запитів SQL, в якій для цього використовується три **операторі**:

1. Вставки записів – **INSERT.**
2. Видалення записів в таблиці – **DELETE.**
3. Зміни значень в існуючих записах таблиці - **UPDATE**.

Отже, оператора INSERT призначений для додавання записів в таблицю або представлення. Різні варіанти оператора INSERT дозволяють додавати в таблицю більше одного запису методом зчитування даних з іншої таблиці або представлення, а також при виконанні зберігаємої процедури або функції. В кожному з перелічених випадків **необхідно враховувати структуру таблиці**:

- кількість полів;
- тип даних кожного поля;
- імена полів, в які заносяться дані;
- обмеження і властивості полів.

Спрощений синтаксис оператора INSERT наступний:

```
INSERT
    [ TOP (кількість) [ PERCENT ] ] /* кількість або відсоток випадкових записів */
    [ INTO ]
    { таблиця | представлення }
{
    [ ( список_полів ) ]
    { VALUES ( { DEFAULT | NULL | значення } [ ,...n ] ) [ ,...n ]
      | інструкція_SELECT
      | інструкція_EXECUTE
      | DEFAULT VALUES      /* заповнює рядок значеннями по замовчуванню */
    }
}
```

Згідно вищеописаного синтаксису у вказану таблицю (представлення) вставляється запис з значеннями полів, вказаними в переліку фрази **VALUES (значення)**, причому і-е значення відповідає і-му полю в списку полів (поля, не вказані у списку, заповнюються NULL-значеннями або відповідними значеннями по замовчуванню). Якщо в списку VALUES вказані всі поля модифікуємої таблиці і порядок їх перелічення відповідає порядку полів в описі таблиці, то список полів при INTO можна проігнорувати.

Крім того, таблицю можна заповнити результуючими значеннями вибірки SELECT або даними, які повертаються оператором EXECUTE.

Наприклад, додамо в таблицю Students нового студента:

```
-- дані додаються в таблицю у відповідності з їх фізичним порядком
INSERT INTO Students
VALUES (1, 'Леся', 'Поваренко', 1);

-- дані додаються лише у визначені поля, а відсутні в списку заповняються
-- NULL значеннями, значеннями по замовчуванню, а для полів з типом IDENTITY або
-- rowversion (timestamp) нові значення будуть згенеровані самостійно
INSERT INTO Students(fname, sname, id_group)
VALUES ('Вова', 'Писаренко', 2),
       ('Олег', DEFAULT, 2);

-- додати новий рядок заповнений значеннями по замовчуванню
INSERT INTO Students
DEFAULT VALUES;
```

Оператор **DELETE** призначений для видалення певної кількості полів з таблиці або представлення та має наступний скорочений синтаксис:

```
DELETE
    [ TOP (кількість) [ PERCENT ] ] /* кількість або відсоток перших записів */
    [ FROM ]
    { таблиця | представлення }
    [ FROM вихідна_таблиця [ ,...n ] ]
    [ WHERE { умова
              | [ CURRENT OF /* видалення з поточної позиції */
                  { [ GLOBAL ] ім'я_курсора } /* з позиції, на яку вказує курсор */
                ]
            }
    ]
```

Опція **«FROM вихідна_таблиця»** - це розширення мови Transact-SQL для інструкції DELETE, яке дозволяє задати дані з додаткової таблиці і видаляти відповідні рядки з таблиці в першому описі оператора FROM. Таке розширення може використовуватись замість вкладеного запиту в параметрі WHERE для вказівки рядків, що видаляються.

Наприклад:

```
-- видалити всі записи таблиці Students
delete
from Students;

-- видалити з бази даних студентів з іменем Вова
delete
from Students
where fname = 'Вова';
```

Слід також сказати, що для видалення даних з таблиці крім оператора DELETE можна використовувати оператор **TRUNCATE TABLE**, який знищує або всі дані з таблиці, або ж ті, які відповідають вказаній умові, але структуру залишає незмінною. Але, на відміну від DELETE, цей оператор не повертає повідомлення про кількість видалених записів і перевстановлює значення поля типу IDENTITY заново.

Наприклад, очистимо таблицю Groups, використовуючи оператор TRUNCATE TABLE:

```
TRUNCATE TABLE [ім'я_БД.][схема.] назва_таблиці

-- наприклад
truncate table Groups;
```

Третій оператор – це оператор обновлення даних **UPDATE**. Він дозволяє змінити значення поля в середині існуючих записів.

```
UPDATE таблиця | представлення
SET   назва_поля = { значення | NULL | вираз | (оператор_SELECT) }
[WHERE умова];

UPDATE
    [ TOP (кількість) [ PERCENT ] ] /* кількість або відсоток перших записів */
    { таблиця | представлення }
    /* вказуємо що необхідно обновити */
    SET
        { назва_поля = { вираз | DEFAULT | NULL }
          | { користувацький_тип.{ { властивість = вираз | назва_поля = вираз }
                                   | нестатичний_метод ( аргумент [ ,...n ] )
                                 }
          }
          | назва_поля { .WRITE ( вираз , @Offset , @Length ) }
          | @змінна = вираз
          | @змінна = поле = вираз /* змінній і полю присвоюється однакове значення */
          | назва_поля { += | -= | *= | /= | %= | &= | ^= | |= } вираз
          | @змінна { += | -= | *= | /= | %= | &= | ^= | |= } вираз
          | @змінна = поле { += | -= | *= | /= | %= | &= | ^= | |= } вираз
        } [ ,...n ]

    [ FROM вихідна_таблиця [ ,...n ] ]
    [ WHERE { умова
              | [ CURRENT OF /* видалення з поточної позиції */
                  { [ GLOBAL ] ім'я_курсора } /* з позиції, на яку вказує курсор */
                ]
            }
    ]
```

Опція **.WRITE (вираз, @Offset , @Length)** вказує на те, що повинен бути змінений розділ значення «назва_поля». Аргумент @Length – це довжина розділу в полі, починаючи з @Offset, який замінюється на вказаний вираз. Аргумент @Offset відраховується з нуля, має тип bigint і не може бути від'ємним числом. Дану опцію можна вказувати лише для полів типу varchar(max), nvarchar(max) або varbinary(max). При вказуванні поля не можна вказувати імені таблиці.

Наприклад:

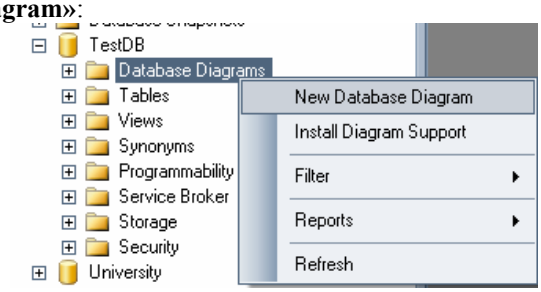
```
-- замінити ім'я студента з іменем Вова на Вован
update Students
set fname = 'Вова'
where fname = 'Вован';

-- знижуємо поточну ціну на вівсяне печиво на 20% і встановлюємо нове значення знижки
update Product
set price = price * 0.2,
    discount = 0.2
where name = 'Вівсяне печиво';
```

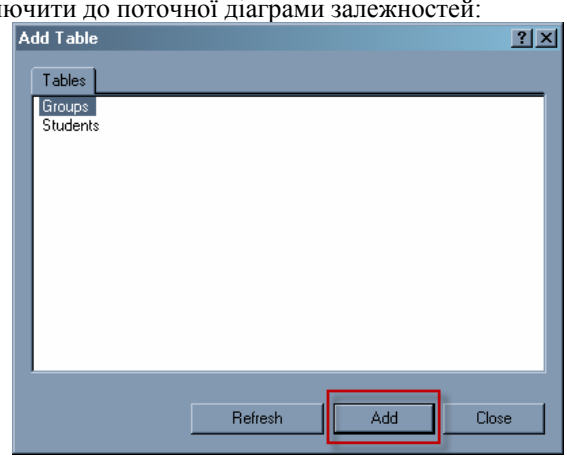
7. Діаграми бази даних

В MS Access в нас існувала можливість зовнішнього представлення бази даних з всіма її таблицями та зв'язками. Здійснювалось це за допомогою **Схеми даних**. В SQL Server є також можливість створення такої схеми даних. Реалізується шляхом побудови діаграм. Причому діаграм може бути кілька в одній базі даних і кожна з них містити своє зовнішнє представлення тих чи інших взаємозв'язків в поточній базі даних. Діаграми є об'єктом бази даних і розміщуються вони в папці «**Database Diagrams**» **(Діаграми бази даних)**.

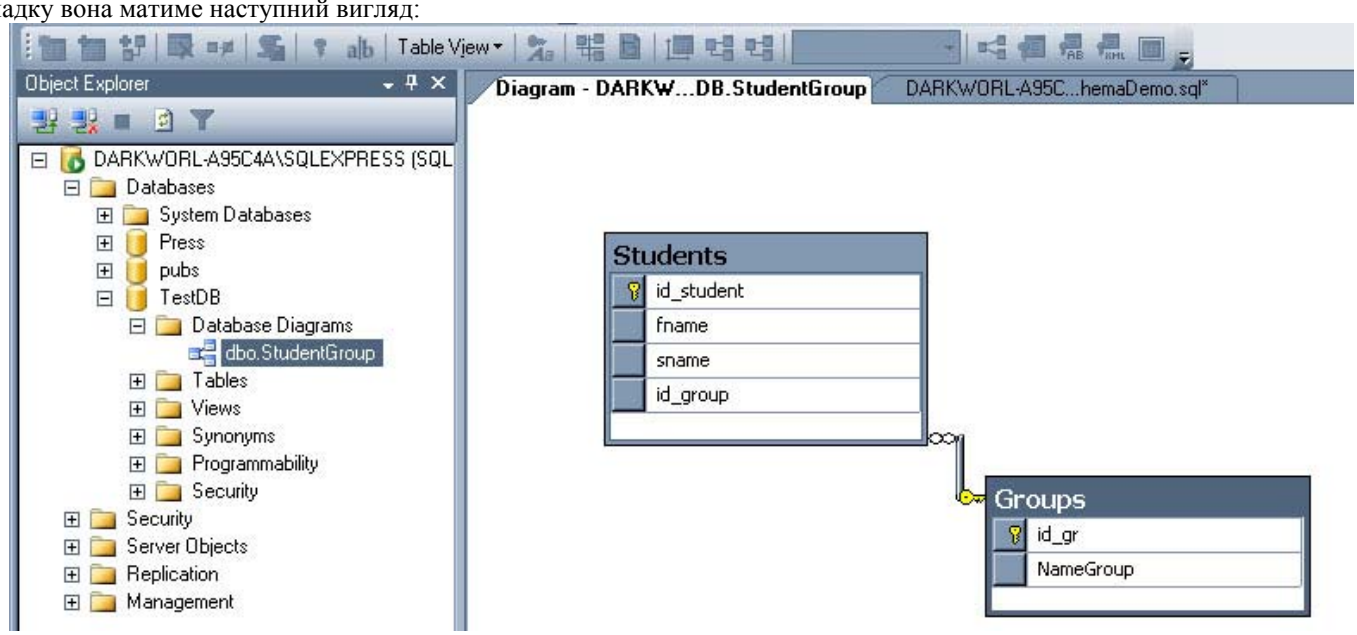
Для створення діаграми потрібно в контекстному меню папки «**Database Diagrams**» обрати пункт «**New Database Diagram**»:



Після цього перед Вами з'явиться діалогове вікно, в якому запитають чи дійсно Ви хочете створити діаграму. Після підтвердження своїх дій, якщо Ви звісно впевнені, з'явиться вікно з списком таблиць на представлень, які необхідно включити до поточної діаграми залежностей:

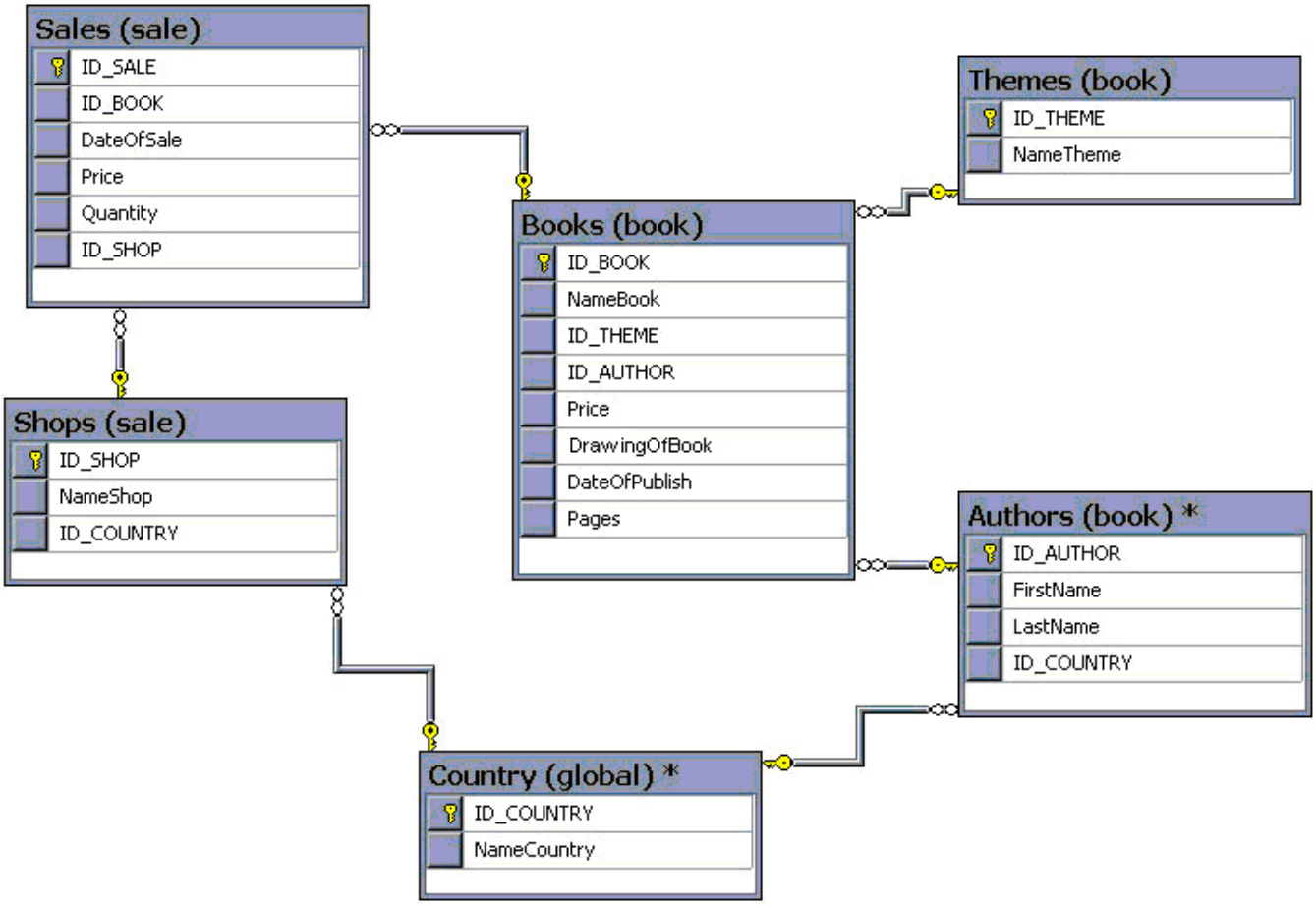


Після натиснення кнопки «**Add**» буде побудована діаграма, яку можна зберегти для майбутніх поколінь. У нашому випадку вона матиме наступний вигляд:



8. Домашнє завдання

Створити базу даних Видавництва, що має наступну структуру:



При створенні обов’язково створити необхідні домени, значення по замовчуванню, правила. Також створити діаграму, яка представляє внутрішню структуру Вашої бази даних.

Copyright © 2010 Iryn Petrova