Урок 6

План заняття:

1. Поняття Transact-SQL та його розширення
2. Транзакції в MS SQL Server
3. Зберігаємі процедури
4. Домашнє завдання

1. Поняття Transact-SQL та його розширення

Transact-SQL (T-SQL) – це розширення мови запитів ANSI SQL, який був розроблений компаніями Microsoft (для Microsoft SQL Server) і Sybase (для Sybase ASE). Одним з основних призначень мови T-SQL – дозволити розробникам баз даних з легкістю створювати запити, повертаючи при цьому дані множиною способів.

Стандарт ANSI SQL був розширений набором елементів та операторів, які часто використовуються в тригерах, транзакціях, зберігаємих процедурах та функціях. Розглянемо ряд **розширень мови T-SQL:**

1. Оператори **BEGIN..END**, які обмежують кілька операторів певного блоку.
2. **Змінні**, які служать для збереження довільних даних. Для того, щоб створити змінну, потрібно її задекларувати:

```
declare {@ім'я_локальної_змінної | @@ім'я_глобальної_змінної} [ AS ] тип [ = value ]
```

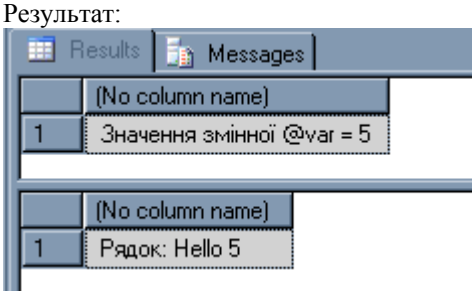
Як видно з синтаксису при створенні змінної її можна проініціалізувати певним значенням. Присвоїти ж значення вже створеній змінній можна двома способами: за допомогою оператора **select** та за допомогою оператора **set**:

```
-- ініціалізація змінної
declare @find varchar(10) = 'Boo%';

-- присвоєння значення за допомогою оператора select
select змінна1 = значення1 [, змінна2 = значення2 ]
-- наприклад
declare @var int, @a char(5)
select @var = 5, @a = 'Hello'

-- присвоєння значення за допомогою оператора set
set змінна = значення
-- наприклад
declare @var int, @a char(5)
set @var = 5
set @a = 'Hello'
```

Самий простий спосіб виведення значення з змінної - це скористатись допомогою вже знайомого оператора **select**, який має здатність виводити літерали та значення переданих полів.

```
select 'Значення змінної @var = ' + convert(char(10), @var)
select 'Рядок: ' + @a + ' ' + convert(char(10), @var)
```

Результат:



За допомогою T-SQL можна створювати також змінні табличного типу. Для прикладу створимо табличну змінну @MyTable та заповнимо її значеннями з таблиці Books.

```
-- створюємо змінну табличного типу
declare @MyTable table( Id int NOT NULL,
                        number int);
```

```
-- заповнюємо змінну даними
insert @MyTable
      select top (5) ID_BOOK, Pages
      from book.Books

-- виводимо на екран дані з змінної типу таблиці
select Id, number
from @MyTable;
```

Результат:

Results | Messages

	Id	number
1	6	640
2	7	440
3	8	720
4	9	200
5	10	288

3. Оператор **PRINT** дозволяє вивести рядок в форматі ASCII, змінну символьного типу або вираз, результатом якого також є рядок. Синтаксис оператора має наступний вигляд:

```
PRINT { 'рядок_ASCII' | @локальна_рядкова_змінна | вираз_який_повертає_рядок }
```

Наприклад:

```
PRINT 'Hello World'

DECLARE @msg nvarchar(50);
SET @msg = N'Сьогодні '+ CAST(GETDATE() AS nvarchar(30)) + N'.';
PRINT @msg;
```

Результат:

Messages

```
Hello World
Сьогодні Mar 23 2010 11:15PM.
```

Слід відмітити, що в даному операторі не можна використовувати оператор конкатенації, а тому він не підходить для виведення форматованого рядка. Щоб вийти з такої ситуації, слід зберегти відформатований рядок в змінній, а тоді вивести її за допомогою оператора PRINT.

4. Функція **Raiserror** виводить повідомлення про помилку. Це повідомлення являється рядком (не більше 2 047 символів), а тому його можна відформатувати довільним чином. Синтаксис функції Raiserror дуже схожий на синтаксис функції printf бібліотеки мови С і виглядає так:

```
Raiserror ( { 'рядок_з_специфікаторами'  | ідентифікатор_помилки | @змінна },
              ступінь_важливості_помилки,
              стан_помилки_на_момент_виклику,
              підставляємі_змінні);
```

В якості першого параметра **може використовуватись**:

- **Рядок**, який містить повідомлення про помилку у відформатованому вигляді;
- **Змінна**, яка повинна мати тип char або varchar, і містити відформатований рядок про помилку;
- **Номер (ідентифікатор)** повідомлення про помилку, який визначений користувачем і збережений в системному представленні sys.messages за допомогою процедури sp_addmessage. Номер користувацького повідомлення повинен бути більше 50 000.

Рядок з повідомленням про помилку для форматування може містити специфікації перетворення наступного формату:

```
% [[flag] [width] [. precision] [{h | l}]] type
```

flag – це код, який дозволяє визначити вирівнювання або проміжок підставляємого значення (-, +, 0, #, ‘’).

width – мінімальна ширина поля, в яке поміщається значення аргумента. Символ (*) означає, що ширина визначається одним з аргументів в функції raiserror.

precision – точність (максимальна кількість символів рядка). Символ (*) означає, що точність визначається одним з аргументів в функції raiserror.

type – специфікатори, які вказують на те, якого типу дані можна використовувати в рядку:

- %d або %i - ціле число з знаком (signed int);
- %s – рядок символів (string);
- %u - беззнакове ціле (unsigned int);
- %o – беззнакове число в вісімковій системі числення (unsigned octal);
- %x або %X – беззнакове число в 16-значній системі числення (unsigned hexdecimal);
- Дійсні числа не підтримуються.

Другий параметр – це **ступінь важливості** помилки, який вказується в межах від 0 до 25.

- Від 0 до 18 – можуть вказуватись користувачами.
- Від 19 до 25 – критичні помилки, які можуть вказувати лише члени серверної ролі sysadmin і користувачами з правами ALTER TRACE.

Вважається, що помилки від 20 до 25 неможливо усунути. У випадку таких помилок з'єднання клієнта з сервером розривається і реєструється повідомлення про помилку в журналах додатку та помилок.

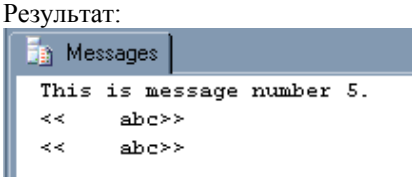
Стан помилки на момент виклику повинно бути цілим числом в діапазоні від 0 до 255. По замовчуванню це значення рівне 1. Якщо одна і та ж користувацька помилка виникає в кількох місцях, то за допомогою цього унікального значення для кожного місцярозташування можна визначити, де була згенерована помилка.

Підставляємі змінні – це змінні, які повинні бути підставлені на місці специфікаторів.

Наприклад:

```
RAISERROR (N'This is message %s %d.', 10, 1, /*аргументи: */'number', 5);
go
RAISERROR (N'<<%*.*s>>', 10, 1,
           7, -- перший аргумент використовується для ширини поля
           3, -- другий аргумент використовується для точності (кількість символів)
           N'abcde'); -- третій аргумент - сам рядок
go
RAISERROR ('<<%7.3s>>', 10, 1, 'abcde');
```

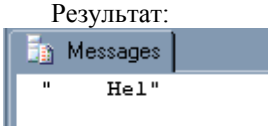
Результат:



І ще маленький приклад коду, в якому ми спробуємо скористатись власним номером помилки. Повідомлення, якому ми хочемо присвоїти номер, слід додати в системне представлення sys.messages за допомогою системної зберігаємої процедури sp_addmessage:

```
sp_addmessage @msgnum = 50005, @severity = 10, @msgtext = '"%7.3s"';
go
RAISERROR (50005, -- ідентифікатор повідомлення
           10, 1, 'Hello');
go
sp_dropmessage @msgnum = 50005;
```

Результат:



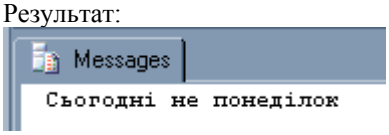
5. Умовний оператор **`if..else`**, який використовується для перевірки умови та має наступний синтаксис:

```
if [(]булевий_вираз[)]
      дія
[else [булевий_вираз]
      дія
]
```

Наприклад, нам необхідно визначити поточний день тиждня. Якщо день тиждня «Понеділок», тоді виводимо на екран значення «Поточний день тижня», інакше вказати на невірний результат.

```
if(datename(dw, GetDate())) = 'Monday'
begin
      PRINT 'Сьогодні понеділок'
end
else
      PRINT 'Сьогодні не понеділок'
```

Результат:



Для отримання назви тижня ми скористались функцією **datename**:

```
DATENAME ( datepart,  date )
```

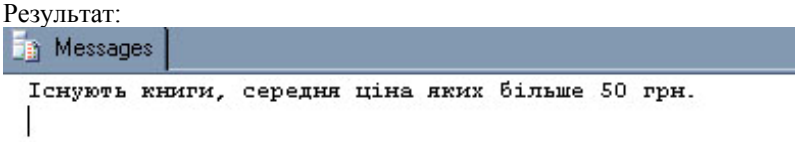
Параметр **datepart** вказує на те, що саме ви хочете отримати з дати:

Datepart	Абревіатура	Datepart	Абревіатура
year	yy, yyyy	**weekday**	dw
quarter	qq, q	**hour**	hh
month	mm, m	**minute**	mi, n
dayofyear	dy, y	**second**	ss, s
day	dd, d	**millisecond**	ms
week	wk, ww		

Булевий вираз біля if може містити оператор **SELECT**, який потрібно ОБОВ'ЯЗКОВО заключити в дужки. Якщо SELECT повертає одне значення, тоді його можна використати для порівняння з іншим значенням і побудовою булевого умовного виразу. Наприклад, визначимо ціну книг і, якщо отримана ціна буде більше 50 грн., тоді виведемо відповідне повідомлення:

```
if (select avg(price)
     from book.Books) > 50
begin
      PRINT 'Існують книги, середня ціна яких більше 50 грн.'
end
```

Результат:



Якщо SELECT повертає більше одного значення, то в умовному виразі if використовується ключове слово **EXISTS**, який повертає true, якщо SELECT повернув хоча б один запис, інакше - false. Синтаксис оператора if разом з exists буде мати наступний вигляд:

```
if exists (оператор SELECT)
      дія
[else [булевий_вираз]
      дія
]
```

Наприклад, виведемо всю інформацію при кожну книгу, дата видавництва якої знаходиться в проміжку від 01.01.2006 до сьогоднішнього дня:

```
if exists   (select *
             from book.Books
             where DateOfPublish between '2006.01.01' and current_timestamp)
begin
      PRINT 'Інформація про книги'
      select *
      from book.Books
      where DateOfPublish between '01.01.2006' and current_timestamp
```

```
      return
end
```

Результат:

Results | Messages

	ID_BOOK	NameBook	ID_THEME	ID_AUTH...	Price	Tiraz	DateOfPublish	Pa
1	18	Путь к LINUX. 2е изд.	15	3	120,00	3000	2006-12-01 00:00:00.000	56
2	19	Как программировать на C	16	15	151,50	3000	2006-03-22 00:00:00.000	10
3	20	C++ за 21 день	16	14	52,00	2000	2006-03-22 00:00:00.000	56
4	21	Язык програмирования C	16	13	78,00	3000	2006-12-01 00:00:00.000	72
5	24	Основы CORBA	20	NULL	58,00	2000	2006-04-20 00:00:00.000	38
6	25	Macromedia Flash 4. Интерактивная веб- анимация	22	1	89,00	3000	2006-04-21 00:00:00.000	67
7	26	Adobe Web-дизайн и публикация. Энциклопедия ...	22	5	76,00	3000	2006-07-09 00:00:00.000	65

6. **Оператор розгалуження CASE**, який дозволить повернути різні значення в залежності від певного контролюючого значення або умови. Оператори, які включають в себе структуру CASE, можуть використовувати одну з двух синтаксичних форм, в залежності від того, чи буде змінюватись оцінюваний вираз:
   1) **Проста**, в якій результуюче значення повертається лише у випадку, якщо вираз після **WHEN** логічно рівний вказаному значенню. Ви можете використовувати довільну кількість інструкцій WHEN. Інструкція **ELSE** необов'язкова і виконується лише, якщо всі інструкції WHEN оцінюються як FALSE.

```
case умовний_вираз
when вираз_константа1 then результуюче_значення1
when вираз_константа2 then результуюче_значення2
[,... n]
[else результуюче_значенняN]
end
```

   2) **З пошуком**. В цій формі CASE можна вказати умовний вираз при кожній інструкції WHEN.

```
case
when умовний_вираз1 then результуюче значення1
when умовний_вираз2 then результуюче значення2
[,... n]
[else результуюче_значенняN]
end
```

Примітка! В SQL Server CASE являється функцією, а не командою. В зв'язку з цим CASE може використовуватись лише як частина оператора SELECT або UPDATE, на відміну від оператора IF, який працює самостійно.

А тепер напишемо **<u>кілька прикладів:</u>**

а) Напишемо запит, який буде виводити на екран назву книги та її тематику в розширеному вигляді:

```
select 'Назва книги' = b.NameBook,
       'Тематика' = case t.NameTheme
                        when 'C & C++' then 'Тематика C & C++'
                        when 'WEB' then 'Все про Web'
                        else 'Невідома тематика'
                        end
from book.Books b, book.Themes t
where b.id_theme = t.id_theme
```

Результат:

Results | Messages

	Назва книги	Тематика
11	Путь к LINUX. 2е изд.	Невідома тематика
12	Как программировать на C	Тематика C & C++
13	C++ за 21 день	Тематика C & C++
14	Язык програмирования C	Тематика C & C++

b) Запит, в якому потрібно перевірити ціну книги. В результуючий запит повертається значення, яке відповідає першій умові true:

```
select 'Назва книги' = b.NameBook,
       'Ціна книги' = case
                        when price < 100 then 'Ціна < 100'
                        when price between 100 and 500 then 'Ціна в діапазоні 100..500'
                        else 'Ціна > 500'
                        end
from book.Books b
```

Результат:

Results | Messages

	Назва книги	Ціна книги
5	Реанимация, проверка, наладка современных ПК	Ціна < 100
6	Руководство для хакеров 2	Ціна < 100
7	Windows NT 5 перспектива	Ціна в діапазоні 100..500
8	Сетевые технологии Windows 2000 для профессионалов	Ціна в діапазоні 100..500

Інколи виникає ситуація, коли необхідно використати оператор CASE для перевірки на **IS NOT NULL**. В результаті структура case набуває вигляду:

```
case
when значення1 IS NOT NULL then вираз1
when значення2 IS NOT NULL then вираз2
[,... n]
[else результуюче_значенняN]
end
```

У такому випадку рекомендується використовувати функцію **COALESCE**, яка служить для отримання значень не рівних NULL.

Наприклад, в таблиці Books замість одного поля ціни в нас існує два поля: оптова та роздрібна ціна. Нам необхідно взнати вартість кожної книги:

```
-- з оператором case
select 'Назва книги' = NameBook,
       'Вартість' = case
                    when TradePrice IS NOT NULL then TradePrice * Quantity
                    when RetailPrice IS NOT NULL then RetailPrice * Quantity
                    end
from book.Books;

-- з використанням функції coalesce
select 'Назва книги' = NameBook,
       'Вартість' = coalesce (TradePrice * Quantity, RetailPrice* Quantity)
from book.Books;
```

Йдемо далі. Оператор case може повернути NULL, якщо порівнювані вирази являються однаковими, інакше він повертає перше значення. У такому випадку структура даного оператора набуває наступного вигляду:

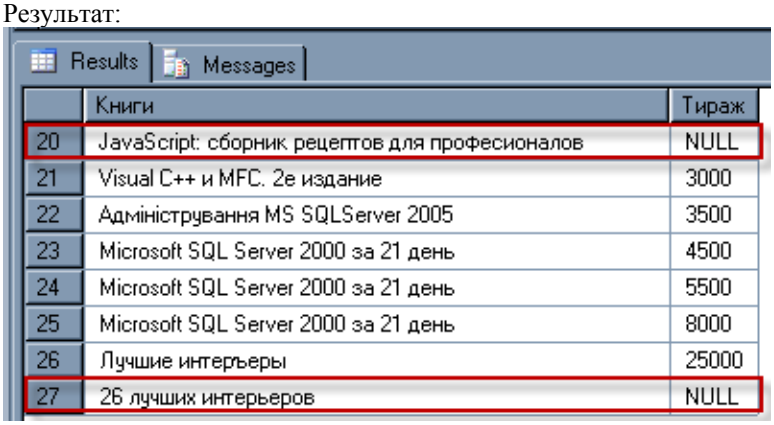
```
case
when значення1=значенняX then NULL
when значення2=значенняY then NULL
[,... n]
[else результуюче_значенняN]
end
```

Щоб спростити роботу слід скористатись функцією **NULLIF**. Припустимо, що нам необхідно вивести на екран назви книг та їх тираж. Якщо значення тиражу відсутнє (тобто рівне нулю), тоді виводимо NULL.

```
-- з оператором case
select 'Книги' = NameBook,
       'Тираж' = case
                    when DrawingOfBook=0 then NULL
```

```
                    else DrawingOfBook
                    end
from book.Books;

-- з використанням функції nullif
select 'Книги' = NameBook, 'Тираж' = NULLIF(DrawingOfBook, 0)
from book.Books;
```

Результат:

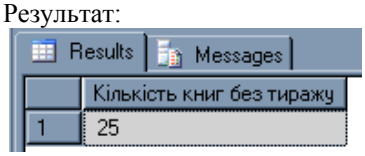


	Книги	Тираж
20	JavaScript: сборник рецептов для професионалов	NULL
21	Visual C++ и MFC. 2е издание	3000
22	Адміністрування MS SQLServer 2005	3500
23	Microsoft SQL Server 2000 за 21 день	4500
24	Microsoft SQL Server 2000 за 21 день	5500
25	Microsoft SQL Server 2000 за 21 день	8000
26	Лучшие интеръеры	25000
27	26 лучших интерьеров	NULL

А тепер підрахуємо скільки книг мають значення тиражу.

```
select 'Кількість книг без тиражу' = count(NULLIF(DrawingOfBook, 0))
from book.Books;
```

Результат:



	Кількість книг без тиражу
1	25

7. Оператор безумовного переходу **GOTO**. Даний оператор передає виконання оператору, який йде після мітки, що на нього вказує. В SQL Server мітки являються невиконуваними операторами та мають наступний синтаксис:

```
ім'я_мітки:
```

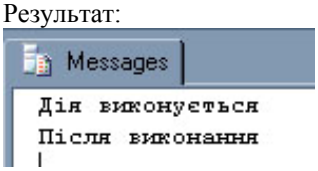
Сама команда GOTO має простий синтаксис:

```
GOTO імя_мітки
```

GOTO як завжди являється небажаною для використання командою, оскільки код з його застосуванням стає важким для сприйняття і аналізу. Напишемо приклад використання оператора GOTO для обробки помилок.

```
PRINT 'Дія виконується'
GOTO label
PRINT 'Дія не виконується'

label:
PRINT 'Після виконання'
```

Результат:



```
Дія виконується
Після виконання
```

Та не слід забувати, що в даному випадку набагато гнучкіше буде застосувати інструкцію try..catch або ж використати механізм обробки транзакцій (про це пізніше).

8. T-SQL також підтримує цикли, які являються послідовністю дій, що можуть виконуватись кілька разів поспіль. Цикли представляються за допомогою єдиного оператора циклу з передумовою **WHILE**:

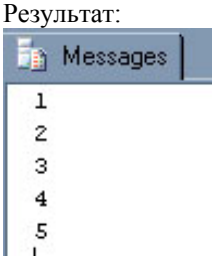
```
while [(] умова [)]
    оператори;
```

В циклах також дозволяэться використання операторів **break** та **continue**. Оператор **BREAK** призводить до виходу з циклу. Після цього виконання продовжується за оператором, який є наступним після оператора **END** (вказує на кінець блоку цикла). Оператор **CONTINUE** використовується, якщо необхідно перейти на початок циклу, і почати всю роботу спочатку. Доречі, в більшості випадків ці два оператори використовуються в межах умовного оператора **if**.

Напишемо кілька прикладів використання цикла:

```
-- створюємо змінну
declare @i int
set @i = 1

-- запускаємо цикл
while @i<10
begin
      print @i
      set @i = @i + 1
      if @i > 5
            break
end
```

Результат:



А тепер обрахуємо середню ціну всіх книг. Якщо вона менше 200 грн., то всі ціни підвищити на 10% до тих пір, поки середня ціна не стане більше 200 грн.

```
while (select avg(price)
      from book.Books) < 200
begin
      update book.Books
      set price = price* 1.1
end
```

9. **Загальні табличні вирази (Common Table Expressions, CTE)** дозволяють задавати тимчасовий іменований набір даних, функціонально схожий на представлення та доступний в межах пакету. В зв'язку з цим їх ще називають **віртуальними представленнями**. Синтаксис їх створення:

```
WITH ім'я_представлення [ ( назва_поля [ ,...n ] ) ]
       AS ( підзапит )
```

При цьому список полів при оголошенні віртуального представлення повинен відповідати кількості полів в підзапиті. В підзапиті CTE **не можуть використовуватись** оператори:

- COMPUTE або COMPUTE BY;
- ORDER BY (за виключенням випадків, коли використовується інструкція TOP);
- INTO;
- FOR XML;
- FOR BROWSE.

Наведемо приклад. Необхідно вибрати всі книги, ціна яких більша середньої ціни на всі книги окремого автора.

```
-- з використанням віртуального представлення
with AvgPrice(ID_AUTHOR, NameAuthor, Price)
as
      (     select a.ID_AUTHOR, a.LastName+' '+a.FirstName, avg(book.Price)
            from book.Books book, book.Authors a
            where book.ID_AUTHOR=a.ID_AUTHOR
            group by a.ID_AUTHOR, a.LastName+' '+a.FirstName
      )
```

```
select book.NameBook, book.Price
from book.Books as book, AvgPrice as p
where book.ID_AUTHOR = p.ID_AUTHOR and book.Price > p.Price

-- з підзапитом
select book.NameBook, book.Price
from book.Books as book,
      (     select a.ID_AUTHOR as ID_AUTHOR,
                  a.LastName+' '+a.FirstName as NameAuthor,
                  avg(book.Price) as Price
            from book.Books book, book.Authors a
            where book.ID_AUTHOR=a.ID_AUTHOR
            group by a.ID_AUTHOR, a.LastName+' '+a.FirstName
      ) as p
where book.ID_AUTHOR = p.ID_AUTHOR and book.Price > p.Price
```

Результат:

Книги

Results | Messages

	NameBook	ID_AUTHOR	Price
1	Самоучитель работы на персональном компьютере: 3...	10	49,6441
2	Основы работы на ПК	9	56,8055
3	Толковый словарь компьютерных технологий	2	82,7405
4	Первые шаги пользователя ПК с дискетой	10	47,6183
5	Реанимация, проверка, наладка современных ПК	NULL	50,4716
6	Руководство для хакеров 2	NULL	32,4402
7	Windows NT 5 перспектива	11	285,3117
8	Сетевые технологии Windows 2000 для профессионалов	2	601,8938
9	Windows 2000 Professional. Руководство Питера Норто...	12	251,0742
...			
18	Macromedia Flash 4. Интерактивная веб- анимация	1	253,9273
19	Adobe Web-дизайн и публикация. Энциклопедия польз...	5	216,8371
20	JavaScript: сборник рецептов для професионалов	6	265,3398
21	Visual C++ и MFC. 2е издание	14	427,9675

Віртуальне представлення

Results | Messages

	ID_AUTHOR	(No column name)	(No column name)
1	1	Веймаер Ричард	189,4636
2	2	White Johnson	243,8275
...			
7	9	Ковязин Андрей	56,8055
8	10	Востриков Михаил	48,6312
9	11	Мочерный Михаил	285,3117
10	12	Нортон Питер	251,0742
11	13	Прата Стивен	232,5291
12	14	Либерти Джес	288,1648

Результуючий набір

Results | Messages

	NameBook	Price
1	Самоучитель работы на персональном компьютере: 3...	49,6441
2	Сетевые технологии Windows 2000 для профессионалов	601,8938
3	Язык програмирования C++	242,515
4	Macromedia Flash 4. Интерактивная веб- анимация	253,9273
5	Visual C++ и MFC. 2е издание	427,9675

Як видно з прикладу, віртуальні представлення та вкладені запити працюють однаково і дають аналогічний результат. Але у випадку повторного використання такого запиту, наприклад, в зберігаємій процедурі, зменшити кількість SQL коду допоможе саме віртуальне представлення.

Розглянемо ще один приклад. Напишемо віртуальне представлення, яке відображає інформацію про кількість книг окремого автора.

```
with CountBooks(NameAuthor, Count_)
as
      (     select a.LastName+' '+a.FirstName, count(ID_BOOK)
            from book.Books b, book.Authors a
            where b.ID_AUTHOR = a.ID_AUTHOR
            group by a.LastName + ' ' + a.FirstName
      ) ;

select *
from CountBooks
order by 2 desc;
```

Одним з основних переваг віртуальних представлень, являється використання **рекурсивних виразів**. Використання таких представлень дуже корисне, якщо необхідно відобразити дані у вигляді ієрархії. Наприклад, можна показати спорідненість дочірних компаній або ж залежність працівників від їх керівників тощо. Загальний принцип побудови рекурсивних віртуальних представлень наступний:

```
WITH ім'я_представлення [ ( назва_поля [ ,...n ] ) ]
AS
(
 SELECT ...        -- Початкова вибірка
 UNION ALL         -- Обєднання результатів
 SELECT ...        -- Вибірка, яка визначає крок рекурсії
 INNER JOIN CTE.ID = Таблиця.ID -- Співставлення таблиць
)
```

В рекурсивних віртуальних представленнях **забороняється** використання наступних операторів:

- SELECT DISTINCT;
- GROUP BY;
- HAVING;
- Функції агрегування;
- TOP;
- LEFT, RIGHT, OUTER JOIN (INNER JOIN допускається);
- Вкладені запити.

Наприклад, виведемо у вигляді ієрархічного списку перелік тематик та книг, які їм належать та видаються видавництвом.

```
WITH Reports(ID_THEME, ID_BOOK, Level_) AS
(
    SELECT ID_THEME, ID_BOOK, 1 AS Level_
    FROM book.Books
    WHERE ID_THEME IS NULL
    UNION ALL
    SELECT b.ID_THEME, b.ID_BOOK, Level_ + 1
    FROM book.Books b INNER JOIN Reports r
        ON b.ID_THEME = r.ID_BOOK
)

select *
from Reports
order by 1
```

Результат нажаль не буде досить наглядний, оскільки в нас не існує принципу підпорядкованості тематик. Тому рівень у всіх книг буде рівний одиниці.

Без використання віртуального представлення, для досягнення аналогічного результату доведеться написати набагато складніший запит. Крім того, рекурсивні віртуальні представлення продуктивніші, ніж тимчасові таблиці.

2. Транзакції в MS SQL Server

Всі дані, які зберігаються в базі даних повинні бути коректними і задача розробників це забезпечити. Основним механізмом, який забезпечує таку узгодженість даних на програмному рівні, являються транзакції. **Транзакція** – це група послідовних операцій, які логічно виконуються як одне ціле. Фактично будь-яка послідовність операторів або оператор, що виконується в базі даних розглядається як транзакція та реєструється в журналі транзакцій. Транзакції можуть бути дуже корисними при тестуванні коду, який вносить зміни в базу даних.

Давайте розглянемо більш детальніше дане визначення. Операції, про які йде мова – це INSERT/UPDATE/DELETE, SELECT тощо. Якщо транзакція об'єднує які-небудь операції в єдиний блок, то говорять що ці дії виконуються в контексті даної транзакції. Слід також відмітити, що після запуску транзакції всі внесені зміни будуть по замовчуванню видимі лише у вашому з'єднанні. Для користувачів, які переглядають дані у інших з'єднаннях, ці зміни будуть невидимі. Далі ви або підтверджуєте транзакцію, зберігаючи всі зміни в базі даних, або робите відкат транзакції, повертаючи цим самим всі дані до їх попереднього стану (до початку транзакції).

Кожна транзакція повинна володіти наступними **4 властивостями**:

1). **Atomicity (атомарності)**. Гарантує, що жодна транзакція не буде зафіксована в системі частково. Тобто оператори, які входять до транзакції можуть бути або виконані всі і повністю або не виконаний жоден з них. Часткове виконання транзацій не допускається.

2). **Consistency (узгодженості або послідовності)** вказує на те, що система знаходиться в узгодженому стані як до початку транзакції, так і після її завершення, а це в свою чергу не порушує бізнес-логіку та відношення між об'єктами бази даних. Ця властивість дуже важлива при розробці клієнт-серверних додатків, оскільки в базі даних відбувається велика кількість транзакцій для різних об'єктів бази від різних клієнтів. І якщо хоча б одна з транзакцій порушує цілісність даних, то всі інші можуть видати невірні результати.

3). **Isolation (ізольованості)**, тобто транзакція не взаємодіє і не конфліктує з іншими транзакціями в базі даних. Це включає і те, що під час виконання транзакції інші процеси не повинні бачити дані в проміжному стані.
4). **Durability (довгочасності або надійності)** вказує на гарантованість виконання всіх дій, незалежно від зовнішніх подій (збій в системі, «падіння» сервера тощо).

В MS SQL Server виділяють наступні **типи транзакцій**:
- явні;
- неявні;
- автоматичні.

Явні транзакції – це транзакції, які оголошуються в програмі напряму. Для визначення початку і кінця транзакції використовуються **такі оператори**:
- **BEGIN TRAN[SACTION] [ім'я_транзакції]** – визначає початок транзакції.
- **COMMIT TRAN[SACTION] [ім'я_транзакції]** – повідомляє сервер, що транзакція закінчилась і потрібно зберегти (зафіксувати) всі зміни. Можна вказувати ім'я транзакції для підтвердження лише її дій.
- **COMMIT WORK** – аналогічно попередньому оператору, але ім'я транзакції не вказується.
- **ROLLBACK TRAN[SACTION] [ім'я_транзакції | ім'я_точки_збереження]** – відміна всіх дій поточної транзакції, або транзакції з певним іменем, або ДО точки збереження. Якщо в програмі існує точка збереження, то всі зміни, що зроблені в базі даних ЛИШЕ до цієї точки, можна відмінити.
- **ROLLBACK WORK** - аналогічно попередньому оператору, але ім'я транзакції не вказується.
- **SAVE TRAN[SACTION] [ім'я_точки_збереження]** – дозволяє встановити точку збереження.

Розглянемо все по порядку. Припустимо в нас існує три дії, які ми хочемо об'єднати в одну транзакцію, причому визначити її потрібно явно. В такому випадку код буде мати наступний вигляд:

```
begin transaction    -- початок транзакції

-- 1
select distinct FirstName
from book.Authors;

-- 2
insert into book.Themes(NameTheme)
values ('MFC')

-- 3
update book.Authors
set id_country = (select id_country
                  for global.Country
                  where NameCountry = 'Україна')

commit transaction -- підтвердити виконання транзакції
-- rollback transaction -- відмінити виконання транзакції
```

Оператори Rollback існують для відміни виконання дій, визначених в межах транзакції. Тобто будь-які зміни, зроблені в базі даних до даного оператора, відміняються. Причому існує можливість створення точки збереження, і тоді можна відмінити лише ті дії, які були здійснені після неї. Точка збереження створюється за допомогою оператора **Save Transaction**. В межах однієї транзакції може існувати кілька точок збереження, а також допускається наявність кількох точок з однаковими іменами. У випадку існування точок з однаковими іменами, відміна операцій здійснюється до останньої точки (від початку тразакції), тобто точки з аналогічними іменами, розміщені вище будуть ігноруватись.

Наприклад:

```
begin transaction    -- початок транзакції
-- 1
select distinct FirstName
from book.Authors;

save transaction pt1  -- перша точка збереження
-- 2
insert into book.Themes(NameTheme)
values ('MFC')

save transaction pt2 -- друга точка збереження
```

```
-- 3
update book.Authors
set id_country = (select id_country
                  for global.Country
                  where NameCountry = 'Україна')

rollback transaction pt2 -- відмінити виконання транзакції до точки pt2 (до update)
--rollback transaction pt1 -- відмінити виконання транзакції до точки pt1 (до insert)
```

Або:

```
begin transaction    -- початок транзакції
-- 1
select distinct FirstName
from book.Authors;

save transaction pt1  -- перша точка збереження
-- 2
insert into book.Themes(NameTheme)
values ('MFC')

save transaction pt1 -- друга точка збереження (з аналогічним іменем)
-- 3
update book.Authors
set id_country = (select id_country
                  for global.Country
                  where NameCountry = 'Україна')

rollback transaction pt1 -- відмінити виконання транзакції до точки pt1.
                         -- Відміна відбудеться до оператора update
```

В SQL Server існує ряд глобальних системних змінниих, серед яких є і корисні в роботі з транзакціями. Найчастіше використовуваними є:

- Глобальна системна змінна **@@error**, яка містить результат виконная транзакції. Якщо транзакція завершилась успішно, то вона містить 0, інакше – код помилки.
- Глобальна системна змінна **@@trancount**, яка є лічильником транзакцій. Працює вона наступним чином:
  - При виклику оператора begin transaction значення цієї змінної збільшується на 1.
  - Оператор save transaction на її значення не впливає
  - Оператор rollback transaction впливає двояко. Якщо ім'я транзакції не вказано, то значення змінної обнуляється, інакше значення зменшується на 1.

Приведемо маленький приклад того, як ми можемо маніпулювати транзакціями, використовуючи значення першої глобальної системної змінної.

```
begin transaction    -- початок транзакції
-- 1
select distinct FirstName
from book.Authors;

save transaction pt1  -- перша точка збереження
-- 2
insert into book.Themes(NameTheme)
values ('MFC')

save transaction pt2 -- друга точка збереження
-- 3
update book.Authors
set id_country = (select id_country
                  for global.Country
                  where NameCountry = 'Україна')

-- вибір дії в залежності від поточного стану помилки
if(@@error >=1 and @@error <= 10)
begin
      print 'Значення помилки 1..10'
```

```
      rollback transaction pt2
end
else if(@@error > 10)
      rollback transaction
else
      commit transaction
```

Для більш гнучкої обробки помилок в T-SQL можна використовувати інструкцію **try…catch**. В блоці **try** (захищений блок) розміщується код, який може генерувати виключення (помилки). У випадку виникнення помилки, обробка негайно призупиняється, всі інструкції try-блоку, які залишились, ігноруються і управління передається catch-блоку, який йде за ним. Отже, в **catch-блок** передається управління, якщо буде згенероване виключення.

Синтаксис інструкції try…catch має наступний вигляд:

```
begin try
      -- блок коду, який перевіряється на помилки
end try
begin catch
      -- обробник виключення
end catch
```

Після попадання в блок catch, ви можете за допомогою системних функцій виявити причину помилки або отримати детальну інформацію про неї. Найпошиніші функції:

- ERROR_NUMBER - номер помилки;
- ERROR_MESSAGE - текст повідомлення про помилку;
- ERROR_LINE - номер рядка, в якому міститься помилка;
- ERROR_SEVERITY - важливість повідомлення про помилку;
- ERROR_STATE - номер стану про помилку.

В явних транзакціях **забороняється одночасно використовувати** наступні оператори:

- ALTER, DROP, RESTORE і CREATE DATABASE;
- BACKUP і RESTORE LOG;
- RECONFIGURE;
- UPDATE STATISTICS.

Неявні транзакції включені в T-SQL для сумісності з стандартом ANSI. Коли вмикається режим неявних транзакцій, автоматично виконується оператор begin transaction. Завершити транзакцію можна лише шляхом явного виклику оператора commit або rollback transaction.

Ввімкнути цей режим можна за допомогою оператора **set implicit_transaction**:

```
set implicit_transaction [on      -- ввімкнути
                         | off]   -- вимкнути
```

При цьому до кінця сеансу наступні операції повинні обов'язково бути зафіксовані або відмінені:

- ALTER, TRUNCATE TABLE;
- всі операції CREATE та DROP;
- SELECT;
- GRANT та REVOKE;
- INSERT, DELETE, UPDATE;
- FETCH;
- OPEN.

Наприклад:

```
set implicit_transaction on

select distinct FirstName
from book.Authors;

commit transaction
```

Слід також пам'ятати, що використання неявних транзакцій не рекомендується, оскільки кожну транзакцію потрібно завершити або відмінити явно. Якщо цього не зробити, то транзакція буде відкрита і дані будуть надовго заблоковані.

Автоматичні транзакції – це транзакції, які виконуються, якщо навіть оператори для роботи з транзакціями не прописані в програмному коді явно. Тобто будь-які зміни даних сервером розцінюються як транзакція. Простіше кажучи,

автоматичні транзакції здійснюються без явних рамок і з використанням оператора **GO**, які посилають пакет даних для обробки на сервер.

3. Зберігаємі процедури

Зберігаємі процедури (stored procedures) – це послідовність компільованих операторів, що зберігаються в базі даних. Слід відмітити, що код зберігаємих процедур компілюється при першому запуску і далі зберігається у відкомпільованому вигляді, тому їх ефективність набагато вища, ніж у звичайних запитів. Зберігаємі процедури являються основним інтерфейсом, який повинен використовуватись прикладними додатками для звернення до довільних даних в базі даних. Крім управління доступом до бази даних, вони також дозволяють ізолювати код бази даних. Тепер не потрібно писати SQL команди для здійснення певних змін, а також це гарантує безпеку між користувачами і таблицями в базі.

В SQL Server існує набір системних зберігаємих процедур, які починаються з префікса **sp_xxx (stored procedure)**, а також можна створювати свої власні користувацькі зберігаємі процедури. Користувацькі зберігаємі процедури створюються за допомогою оператора **CREATE PROCEDURE** і вони можуть містити майже довільні команди.

```
CREATE PROC[EDURE] [схема.] ім'я_процедури [;число]
[@параметр [схема.] тип
                    [VARYING]         /*для управління курсором*/
                    [ = значення_по_замовчуванню | NULL]
                    [OUT | OUTPUT]    /*вказує на те, що даний параметр є повертаємим*/
                    [READONLY]        /*тільки для читання. Для табличних типів*/
]
[,...n]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION
        | EXECUTE AS { CALLER | SELF | OWNER | 'логін_користувача' } } ]
[FOR REPLICATION] /*приймає участь в реплікації*/
AS
{
  [BEGIN]
     тіло_процедури з оператором SELECT
  [END]
  | EXTERNAL NAME збірка.клас.метод
}
```

Зберігаємі процедури можуть як приймати дані (input parameters), так і повертати (output parameters). В процедурі також дозволяється використання локальних змінних. З параметром **ENCRYPTION** ви вже знайомі. Він використовується для створення шифрованих зберігаємих процедур. Якщо вказати параметр **RECOMPILE** SQL Server перекомпілює процедуру при кожному її запуску. Більш детальніше з цим параметром ми познайомимось пізніше. Параметр **EXECUTE AS** визначає контекст безпеки для зберігаємої процедури:

- **CALLER** (по замовчуванню) – вказує, що інструкції, які містяться в процедурі, виконуються в контексті користувача, який її викликав.
- **'логін_користувача' –** вказується який саме користувач може змінювати зберігаєму процедуру.
- **SELF** – інструкції виконуються в контексті користувача, який створив зберігаєму процедуру або може її змінювати.
- **OWNER** – всі інструкції виконуються в контексті поточного власника цієї зберігаємої процедури. Якщо власник процедури не вказаний, тоді мають на увазі власника схеми.

Параметр **EXTERNAL NAME** вказує на збірку .NET Framework, на яку повинна посилатись зберігаєма процедура CLR. При цьому слід вказати назву класу в цій збірці і необхідний метод (повинен бути статичний). Якщо ім'я класу включає в себе назву простору імен, відділену крапками (.), тоді його слід відмежовувати квадратними дужками ([ ]) або подвійними лапками (" ").

Викликати процедуру можна за допомогою оператора **execute**, спрощений синтаксис якого наступний:

```
exec[ute] ім'я_процедури [;число] [список_параметрів] [WITH RECOMPILE]
```

Більш детальніше кожен з параметрів розглянемо на практиці, але для початку ще трішки теорії.

Отже, при створенні зберігаємої процедури слід дотримуватись **наступних правил**:

✓ в зберігаємих процедурах не можна використовувати оператори:

- CREATE RULE,
- CREATE DEFAULT,
- CREATE PROCEDURE,
- CREATE TRIGGER,

- CREATE VIEW,
- USE база_даних,
- SET SHOWPLAN_TEXT,
- SET SHOWPLAN_ALL;

- ✓ під час виконання процедури всі об'єкти, на які вона ссилається, повинні бути присутні в БД. Спеціальна властивість процедури (пізнє зв'язування імені) дозволяє під час компіляції ссилатись на неіснуючий об'єкт. Завдяки цьому зберігаєма процедура при створенні може генерувати тимчасові об'єкти, а потім ссилатись на них при запуску;
- ✓ в процедурі не можна створювати об'єкт, а потім видалити його або створювати заново під одним і тим же іменем;
- ✓ процедура не може мати більше 1024 параметрів;
- ✓ можна створювати вкладені процедури (підтримується до 32 рівнів вкладеності);
- ✓ як і у випадку представлень, якщо в процедурі використовується оператор SELECT * і в базову таблицю додаються поля після створення процедури, то при її виконанні ці нові поля використовувати неможна. Для цього потрібно за допомогою оператора ALTER PROCEDURE змінити зберігаєму процедуру.

Процес виконання зберігаємих процедур проходить **5 етапів**:

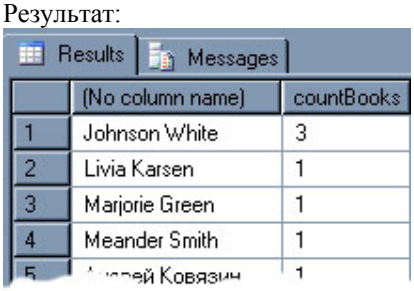
1. Лексикографічний аналізатор виразів розбиває процедуру на окремі компоненти.
2. Компоненти, які ссилаються на об'єкти БД (таблиці, представлення тощо), співставляються з цими об'єктами (вже перевіреними на існування). Цей процес називається **розширенням посилань**.
3. Зберігаєма процедура реєструється, тобто в sysobjects записується її назва, а в syscomments – код створення.
4. Створюється попередній план виконання запиту, тобто дерево запиту, який зберігається в системній таблиці sysprocedures.
5. Зчитується дерево запиту та процедура виконується.

Серед переваг зберігаємих процедур можна виділити те, що план процедури збергається в процедурному кеші після її першого виконання і вже надалі звідти просто зчитується. Тобто процедура компілюється один раз, при першому її виклику. Це призводить до підвищення продуктивності та швидкості виконання зберігаємих процедур. Крім того, існує можливість автоматичного виконання процедур при запуску SQL Server.

Перейдемо до практики. Для початку напишемо просту процедуру, яка дозволяє переглянути список авторів та кількість їх книг:

```
create procedure sp_authors
as
select a.firstname + ' ' + a.lastname, count(b.id_book) as countBooks
from book.Authors a, book.Books b
where b.id_author = a.id_author
group by a.firstname + ' '+ a.lastname;
go
exec sp_authors;
```

Результат:



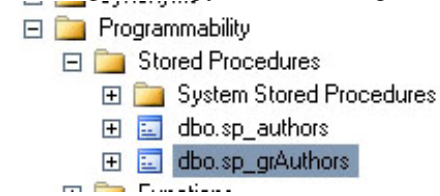
	(No column name)	countBooks
1	Johnson White	3
2	Livia Karsen	1
3	Marjorie Green	1
4	Meander Smith	1
5	Андрей Ковязин	1

А тепер розглянемо як використовується параметр «число» при створенні процедури. Як правило, він служить для створення групи зберігаємих процедур. Це може стати у нагоді, коли необхідно, щоб кілька процедур виконувались як одна, тобто одночасно. Для прикладу, напишемо групу з двох процедур, які отримують різну інформацію про авторів:

```
create proc sp_grAuthors;1
as
select *
from book.Authors;
go

create proc sp_grAuthors;2
as
select a.firstname + ' ' + a.lastname, count(b.id_book) as countBooks
```

```
from book.Authors a, book.Books b
where b.id_author = a.id_author
group by a.firstname + ' '+ a.lastname
```

Навіть фізично група запитів зберігається як одна:



Запускаємо:

```
exec sp_grAuthors;1  -- запускає перший запит з групи (по номеру)
exec sp_grAuthors    -- запускає перший запит з групи
```

Результат:

Results | Messages

	ID_AUTHOR	FirstName	LastName	ID_COUNTRY
1	1	Ричард	Веймаер	4
2	2	Johnson	White	3
3	3	Marjorie	Green	3
4	4	Meander	Smith	5

Для видалення зберігаємої процедури використовується оператор **DROP PROCEDURE**, а у випадку групи процедур – видаляється ціла група. Видалення окремої зберігаємої процедури з групи неможливо. Синтаксис даного оператора типовий, тому розглядати детально його ми не будемо.

Розглянемо приклад передачі переметрів в зберігаєму процедуру. Для цього напишемо зберігаєму процедуру, яка додає два числа, переданих в якості параметрів, а результат записує у вихідний (output) параметр:

```
create procedure sp_summa
@a int,
@b int,
@res int output
as
set @res = @a + @b
```

Передавати при виклику параметри в зберігаєму процедуру можна **двома способами**: явно і по позиції.

```
declare @summ int                       -- оголошуємо змінну, яка буде містити результат
execute sp_summa @a = 5, @b = 25, @res = @summ output -- явна передача параметрів
execute sp_summa 5, 25, @summ output    -- передача параметрів по позиції
select '5 + 25 = ', @summ               -- виводимо результат
```

Результат:

Results | Messages

	(No column name)	(No column name)
1	5 + 25 =	30

Повертати значення з процедури можна використовуючи оператор **return**. Для цього перепишемо нашу процедуру наступним чином:

```
create procedure sp_summa2
@a int,
@b int
as
declare @res int
set @res = @a + @b
return @res
```

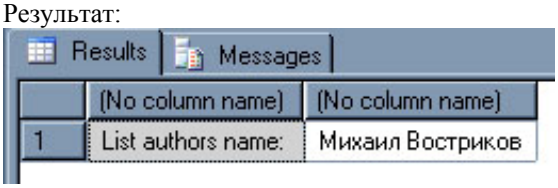
Виклик такої зберігаємої процедури буде наступним:

```
declare @summ int                          -- оголошуємо змінну, яка буде містити результат
execute @summ = sp_summa2 @a = 5, @b = 25     -- явна передача параметрів
execute @summ = sp_summa2 5, 25               -- передача параметрів по позиції
select '5 + 25 = ', @summ                     -- виводимо результат
```

Розглянемо більш складніший приклад. Напишемо процедуру, яка повертає список авторів, які живуть в Україні:

```
create proc sp_ListAuthors
@name varchar(25) output,
@surname varchar(25) output
as
select @name = a.firstname, @surname = a.lastname
from book.Authors a, global.Country c
where a.id_country = c.id_country and c.NameCountry = 'Україна'
go

declare @name varchar(25), @surname varchar(25)
exec sp_ListAuthors @name output, @surname output
select 'List authors name: ', @name + ' ' + @surname
```

Результат:



Залишилось розглянути опцію **RECOMPILE**. При її використанні SQL Server буде ігнорувати існуючий план виконання зберігаємої процедури і при кожному її виконанні створювати новий. На практиці перекомпіляція зберігаємої процедури використовується дуже рідко, але вона стате у нагоді, наприклад, при додаванні нового індекса, який покращує роботу зберігаємої процедури. Опцію RECOMPILE можна **використовувати у двох випадках**:

1) В операторі **CTERATE PROC**, тобто при створенні процедури. В такому випадку план виконання процедури не повинен зберігатись в процедурному кеші і при виконанні вона буде заново перекомпільовуватись. Це корисно для процедур з поточними параметрами. Наприклад:

```
create proc sp_themes
with recompile
as
select *
from book.Themes
```

2) В операторі **EXEC PROC** - при виклику процедури. В такому випадку перекомпіляція здійснюється в поточному сеансі виконання процедури. Новий план зберігається в кеші і потім може ще використовуватись:

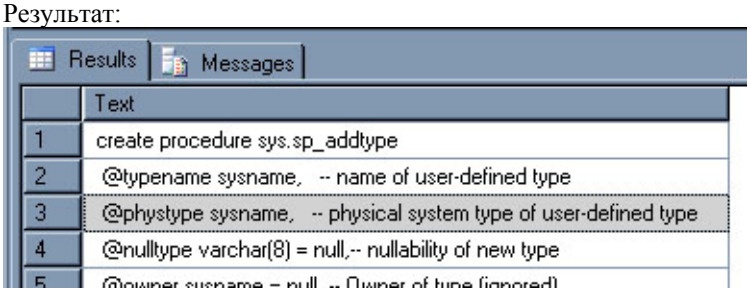
```
exec sp_themes with recompile
```

Щоб перекомпілювати всі зберігаємі процедури і тригери використовується системна процедура **sp_recompile,** а для автоматичного виконання зберігаємої процедури при запуску сервера слід виконати процедуру **sp_procoption**.

Для того, щоб переглянути інформацію про зберігаєму процедуру, тобто код її створення, потрібно викликати системну процеруду **sp_helptext**:

```
exec sp_helptext sp_addtype
```

Результат:



Щоб переглянути список зв'язаних з процедурою об'єктів, слід скористатись системною процедурою **sp_depends**:

```
exec sp_depends ім'я_процедури
```

4. Домашнє завдання

1. Написати представлення, в якому необхідно вивести перелік магазинів з вказанням їх місцярозташування. При цьому назву країни слід вивести на англійській мові і у скороченому вигляді (наприклад, Україна – UA).
2. Написати запит, який змінює дані в таблиці Books наступним чином: якщо книги були видані після 2008 року, тоді їх тираж збільшити на 1000 екзмплярів, інакше тираж збільшити на 100 од. **Примітка!** Скористатись інструкцією CASE.
3. Написати віртуальне представлення, яке виводить загальну кількість продаж і дату останньої реалізації для кожного магазину.
4. Створити зберігаєму процедуру, яка виводить на екран список магазинів, які продали хоча б одну книгу Вашого видавництва. Вказати також місцерозташування (країну) магазину.
5. Написати процедуру, що дозволяє переглянути всі книги певного автора, при цьому його ім'я передається при виклику
6. Створити зберігаєму процедуру, яка повертає максимальне з двух чисел.
7. Написати процедуру, що виводить на екран книги і ціни по вказаній тематиці. При цьому необхідно вказувати напрям сортування: 0 - по ціні, по зростанню, 1 - по спаданню, будь-яке інше - без сортування.
8. Написати процедуру, яка повертає повне ім'я автора, книг якого найбільше всіх було видано.
9. Написати процедуру для обрахунку факторіала числа.
10. Написати зберігаєму процедуру, яка дозволяє збільшити дату видавництва кожної книги, яка відповідає шаблону на 2 роки. Шаблон передається в якості параметра в процедуру.
11. Написати зберігаєму процедуру з параметрами, що визначають діапазон дат випуску книг. Процедура дозволяє обновити дані про тираж випуску книг згідно наступних умов:
    - якщо дата випуску книги знаходиться у визначеному діапазоні, тоді тираж потрібно збільшити в два рази, а ціну за одиницю збільшити на 20%;
    - якщо дата випуску книги не входить в діапазон, тоді тираж залишити без змін.

    Передбачити виведення на екран відповідних повідомлень про помилку, якщо передаваємі дати є однаковими, або кінцева дата проміжку меньше початкої, або ж початкова більше поточної дати.