Урок 5

План заняття:

1. Директиви COMPUTE та COMPUTE BY
2. Надагрегатні оператори ROLLUP, CUBE та GROUPING SETS
3. Оператори PIVOT та UNPIVOT
4. Представлення в MS SQL Server
5. Домашнє завдання

1. Директиви COMPUTE та COMPUTE BY

Оператори **COMPUTE** і **COMPUTE BY** створюють нові рядки на основі даних, які повертаються оператором SELECT. В них використовуються функції агрегування.

Узагальнений синтаксис використання:

```
SELECT запит
[ COMPUTE
   { { AVG | COUNT | MAX | MIN | STDEV | STDEVP | VAR | VARP | SUM } ( вираз ) } [,...n]
   [ BY вираз [ ,...n ] ]
]
```

Оператор **COMPUTE** генерує результуючі значення, які відображаються у вигляді додаткових рядків.

Оператор **COMPUTE BY** повертає нові рядки для групових даних, що схоже з директивою GROUP BY, але тут рядки повертаються як підгрупи з розрахованими значеннями.

Доречі, в одному запиті можна одночасно вказати оператор COMPUTE і COMPUTE BY, але в наступній версії MS SQL Server цю можливість планується усунути.

Наприклад, напишемо запит, який виводить на екран загальну вартість книг кожної тематики. Використаємо при написанні даного запиту звичний нам оператор GROUP BY, адже без нього при побудові даного запиту ніяк не обійтись.

```
select t.NameTheme, sum(b.Price)
from book.Books b, book.Themes t
where b.id_theme = t.id_theme
group by t.NameTheme;
```

Результат:

	NameTheme	(No column name)
1	C & C++	366,50
2	Java, J++, JBuilder, JavaScript	93,00
3	Linux	195,00
4	Visual C++	166,42
5	Windows 2000	298,96
6	Windows NT	100,00
7	Аппаратные средства ПК	17,69
8	Графика для Интернет	165,00

При використанні оператора COMPUTE ми отримаємо додаткове поле з підсумковою сумою цін всіх книг:

```
select t.NameTheme, b.Price
from book.Books b, book.Themes t
where b.id_theme = t.id_theme
-- order by t.NameTheme       -- можна включити сортування по довільному полю
compute sum(b.Price);
```

Результат:

Results | Messages

	NameTheme	Price
1	Учебники	49,6441
2	Учебники	56,8055
3	Учебники	82,7405
4	Учебники	47,6183
5	Аппаратные средства ПК	50,4716
6	Другие книги	32,4402
7	Windows NT	285,3117

	sum
1	4561,4262

→ підсумкова сумма

При використанні оператора COMPUTE BY в запит ОБОВ'ЯЗКОВО включається директива ORDER BY. При цьому, якщо ORDER BY має вигляд:

```
ORDER BY t.NameTheme, b.NameBook
```

то COMPUTE BY повинен бути одним із наступних варіантів

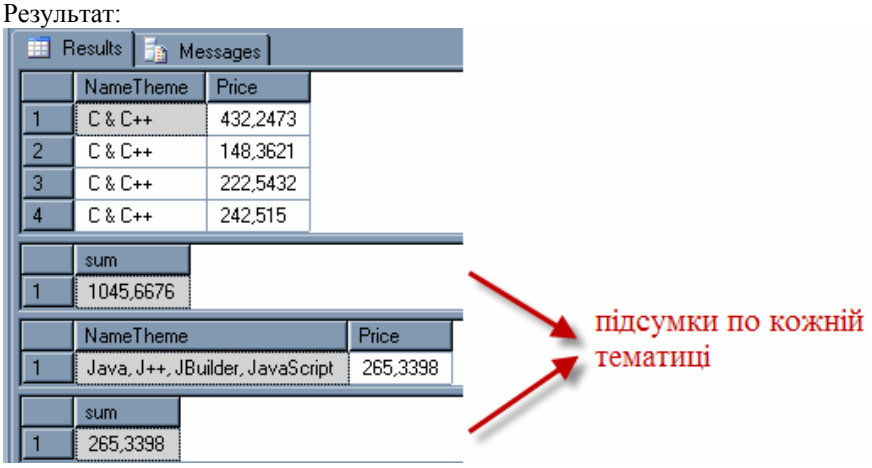
```
-- 1 варіант
COMPUTE функція_агрегування(поле)
BY t.NameTheme, b.NameBook
-- 2 варіант
COMPUTE функція_агрегування(поле)
BY t.NameTheme
```

Тобто поля, які перераховані в операторі COMPUTE BY ті ж, що і в ORDER BY, або складають їх підмножину. Послідовність полів в COMPUTE BY повинна бути такою ж, як і в ORDER BY, пропускати поля не можна.

Отже, перепишемо наш запит з використанням оператора COMPUTE BY:

```
select t.NameTheme, b.Price
from book.Books b, book.Themes t
where b.id_theme = t.id_theme
order by t.NameTheme
compute sum(b.Price)
by t.NameTheme;
```

Результат:



Слід відмітити, що при використанні даного оператора не можна застосовувати оператор SELECT INTO, оскільки COMPUTE і COMPUTE BY створюють нові записи (рядки) нереляційних даних. При використанні вищеописаних операторів також не можна використовувати дані типу text або image, оскільки вони не підлягають впорядкуванню.

2. Надагрегатні оператори ROLLUP, CUBE та GROUPING SETS

Оператори ROLLUP, CUBE GROUPING SETS задекларовані стандартом ANSI/ISO SQL'99 та використовуються для створення додаткових рядків в результаті виконання команди SELECT. Дані оператори ще називають **«надагрегатними»**, оскільки для своєї роботи вони використовують функції агрегування та використовуються разом з оператором GROUP BY.

Узагальнений синтаксис використання виглядає наступним чином:

```
SELECT запит
[GROUP BY
          [ { ALL | CUBE | ROLLUP | GROUPING SETS } ] вираз_групування]
[WITH {CUBE | ROLLUP} ] -- альтернативний варіант з WITH
```

Варто відмітити, що підтримка ANSI стандарту для операторів ROLLUP, CUBE і GROUPING SETS Microsoft реалізував лише у версії SQL Server 2008 (синтаксис з «WITH» являється застарівшим і не рекомендований до застосовування, але підтримується для сумісності з ранніми версіями).

Оператор **ROLLUP** найчастіше використовують для розрахунку середніх значень або сум. Він задає агрегатну функцію для набору полів оператора SELECT з директивою GROUP BY, обробляючи їх зліва направо.

Для кращого розуміння роботи вищезгаданого оператора розглянемо спочатку приклад без використання надагрегатів. Напишемо запит, в результаті якого отримаємо загальну суму книг кожної тематики та кожного автора.

```
select t.NameTheme, a.FirstName + ' ' + a.LastName as 'FullName', sum(b.Price)
from book.Books b, book.Themes t, book.Authors a
where b.id_theme = t.id_theme and b.id_author = a.id_author
group by t.NameTheme, a.FirstName + ' ' + a.LastName
```

Результат:

	NameTheme	FullName	(No column name)
1	Visual C++	Johnson White	16,42
2	Windows 2000	Johnson White	210,96
3	Учебники	Johnson White	29,00
4	Графика для Интернет	Livia Karsen	76,00
5	Linux	Marjorie Green	120,00
6	Linux	Meander Smith	75,00
7	Учебники	Андрей Ковязин	19,91
8	C & C++	Джес Либерти	52,00
9	Visual C++	Джес Либерти	150,00
10	Учебники	Михаил Востриков	34,09
11	Windows NT	Михаил Мочерный	100,00

А тепер перепишемо даний запит з використанням оператора ROLLUP:

```
select t.NameTheme, a.FirstName + ' ' + a.LastName as 'FullName',
       sum(b.Price) as 'Загальна сума'
from book.Books b, book.Themes t, book.Authors a
where b.id_theme = t.id_theme and b.id_author = a.id_author
group by t.NameTheme, a.FirstName + ' ' + a.LastName
with rollup;
-- або
select t.NameTheme, a.FirstName + ' ' + a.LastName as 'FullName',
       sum(b.Price) as 'Загальна сума'
from book.Books b, book.Themes t, book.Authors a
where b.id_theme = t.id_theme and b.id_author = a.id_author
group by rollup (t.NameTheme, a.FirstName + ' ' + a.LastName)
```

Результат:

Results | Messages

	NameTheme	FullName	Загальна сума
1	C & C++	Джес Либерти	148,3621
2	C & C++	Олег Лисовский	432,2473
3	C & C++	Стивен Прата	465,0582
4	C & C++	NULL	1045,6676
5	Java, J++, JBuilder, JavaScript	Сергей Парижский	265,3398
6	Java, J++, JBuilder, JavaScript	NULL	265,3398
7	Linux	Marjorie Green	342,374
8	Linux	Meander Smith	213,9838
9	Linux	NULL	556,3578
10	Visual C++	Johnson White	46,8483

→ підсумок по тематиці "C & C++"

→ підсумок по тематиці "Java, J++, JBuilder, JavaScript"

→ підсумок по тематиці "Linux"

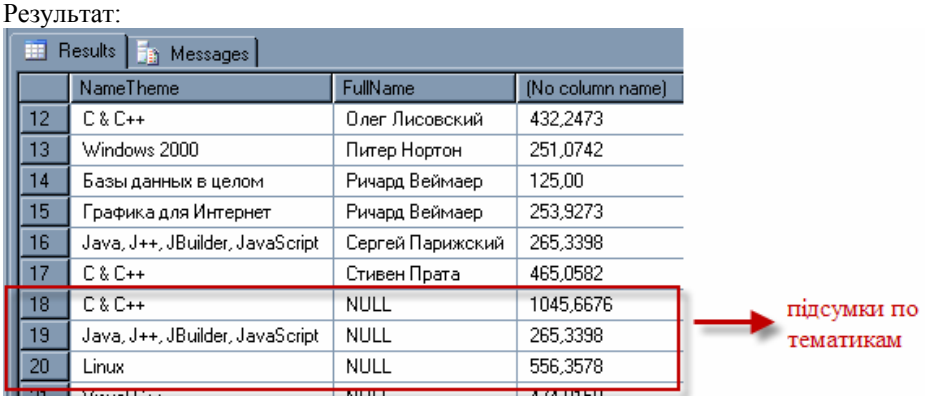
Як видно з результуючого набору, даний оператор створює додаткові рядки для результуючого запиту, в які заносить сумарну інформацію по кільком записам з заданою тематикою (t.NameTheme) та іменем автора (a.FirstName + ' ' + a.LastName). Спочатку створюється новий рядок для першого значення тематики (а саме «C & C++»). Потім для наступного і т.д. Додатковий запис (рядок) відмічається значенням NULL в полі імені автора, а в полі ціни відображається сумма значень (результат дії агрегатної функції), для яких тематика рівна «C & C++». Ці дії повторюються і для інших значень тематики.

В ранніх версіях, щоб отримати той же набір результатів слід скористатись набором запитів та об'єднати їх за допомогою оператора UNION ALL. Виглядатиме такий запит наступним чином:

```
select t.NameTheme, a.FirstName + ' ' + a.LastName as 'FullName', sum(b.Price)
from book.Books b, book.Themes t, book.Authors a
where b.id_theme = t.id_theme and b.id_author = a.id_author
group by t.NameTheme, a.FirstName + ' ' + a.LastName
UNION ALL
select t.NameTheme, NULL, sum(b.Price)
from book.Books b, book.Themes t, book.Authors a
```

```
where b.id_theme = t.id_theme and b.id_author = a.id_author
group by t.NameTheme
UNION ALL
select NULL, NULL, sum(b.Price)
from book.Books b, book.Themes t, book.Authors a
where b.id_theme = t.id_theme and b.id_author = a.id_author
```

Результат:



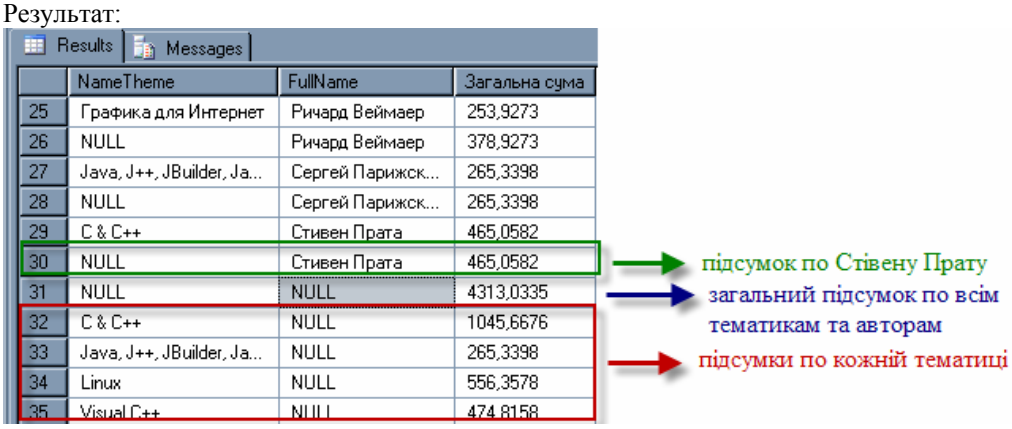
	NameTheme	FullName	(No column name)
12	C & C++	Олег Лисовский	432,2473
13	Windows 2000	Питер Нортон	251,0742
14	Базы данных в целом	Ричард Веймаер	125,00
15	Графика для Интернет	Ричард Веймаер	253,9273
16	Java, J++, JBuilder, JavaScript	Сергей Парижский	265,3398
17	C & C++	Стивен Прата	465,0582
18	C & C++	NULL	1045,6676
19	Java, J++, JBuilder, JavaScript	NULL	265,3398
20	Linux	NULL	556,3578

Оператор **CUBE** створює надагрегатні рядки для всіх можливих комбінацій полів GROUP BY. Як і ROLLUP, він обраховує поточні суми або середні значення, але він створює надагрегати для всіх комбінацій, які не повертаються оператором ROLLUP. В цьому і полягає його відмінність.

Щоб зрозуміти краще вищесказану різницю перепишемо попередній приклад з використанням оператора CUBE.

```
select t.NameTheme, a.FirstName + ' ' + a.LastName as 'FullName', sum(b.Price)
from book.Books b, book.Themes t, book.Authors a
where b.id_theme = t.id_theme and b.id_author = a.id_author
group by t.NameTheme, a.FirstName + ' ' + a.LastName
with cube;
-- або
select t.NameTheme, a.FirstName + ' ' + a.LastName as 'FullName', sum(b.Price)
from book.Books b, book.Themes t, book.Authors a
where b.id_theme = t.id_theme and b.id_author = a.id_author
group by cube (t.NameTheme, a.FirstName + ' ' + a.LastName)
```

Результат:



	NameTheme	FullName	Загальна сума
25	Графика для Интернет	Ричард Веймаер	253,9273
26	NULL	Ричард Веймаер	378,9273
27	Java, J++, JBuilder, Ja...	Сергей Парижск...	265,3398
28	NULL	Сергей Парижск...	265,3398
29	C & C++	Стивен Прата	465,0582
30	NULL	Стивен Прата	465,0582
31	NULL	NULL	4313,0335
32	C & C++	NULL	1045,6676
33	Java, J++, JBuilder, Ja...	NULL	265,3398
34	Linux	NULL	556,3578
35	Visual C++	NULL	474,8158

В результаті роботи оператора CUBE створюються додаткові рядки для результуючого запиту, в які заноситься сумарна інформація по кільком записам з заданою тематикою (t.NameTheme) та іменем автора (a.FirstName + ' ' + a.LastName). Спочатку визначаються сумарні вартості для всіх записів з однаковим значенням тематики. В даному прикладі записи з значеннями NULL в полі з іменем автора містять суму всіх цін по тематикам. Записи з значенням NULL в полі з тематикою (t.NameTheme) містять суму всіх цін для однакових авторів.

Варто відмітити, що при роботі оператора CUBE при N атрибутах, результат складається з 2-х в степені N різноманітних результатів, тому оператор CUBE **являється дуже ресурсоємким** і застосовується лише при малій кількості атрибутів або малій кількості даних.

Оператор **GROUPING SETS** використовують для об'єднання групування, оскільки він дозволяє одночасно групувати як по унікальним значенням одного атрибута, так і по їх комбінаціях. Крім того, він може виступати в якості заміни операторів CUBE і ROLLUP. Для цього необхідно вказати всі допустимі комбінації агрегацій того чи іншого оператора у виразі GROUPING SET.

Роботу даного оператора розглянемо для наглядності на тому ж прикладі, тобто виведемо на екран звіт про загальну суму книг кожної тематики та кожного автора:

```
select t.NameTheme, a.FirstName + ' ' + a.LastName as 'FullName',
       sum(b.Price) as 'Загальна сума'
from book.Books b, book.Themes t, book.Authors a
where b.id_theme = t.id_theme and b.id_author = a.id_author
group by grouping sets (t.NameTheme, a.FirstName + ' ' + a.LastName);
```

Результат:

Results | Messages

	NameTheme	FullName	Загальна сума
9	NULL	Олег Лисовский	432,2473
10	NULL	Питер Нортон	251,0742
11	NULL	Ричард Веймаер	378,9273
12	NULL	Сергей Парижский	265,3398
13	NULL	Стивен Прата	465,0582
14	C & C++	NULL	1045,6676
15	Java, J++, JBuilder, JavaScript	NULL	265,3398
16	Linux	NULL	556,3578
17	Visual C++	NULL	474,8158

Як видно з результату, оператор GROUPING SETS групує по кожному окремому полю в списку. Таким чином, ви можете бачити загальну вартість книг для кожної тематики і автора, що рівносильно роботі звичайного оператора GROUP BY з тією лише різницею, що в полі, яке не враховується, вказується значення NULL.

Отже, якщо наявність всіх групувань не вимагається (як у операторів ROLLUP або CUBE), тоді слід скористатись оператором GROUPING SETS, щоб задати лише унікальні групування.

Але це ще не все. В списку оператора GROUPING SETS можна вказувати кілька наборів групування розділених комами. В такому разі, всі вони вважаються єдиним набором і результат їх дій об'єднується. Фактично результуючий набір може бути перехресним об'єднанням (декартовою множиною значень) групуючих наборів. Наприклад, в додатку **GROUP BY GROUPING SETS ((Colum1, Column2), Column3, Column4)** поля Column1 і Column2 будуть оброблені як одне поле.

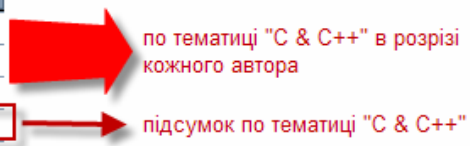
Розглянемо все на прикладі:

```
select t.NameTheme, a.FirstName + ' ' + a.LastName as 'FullName',
       sum(b.Price) as 'Загальна сума'
from book.Books b, book.Themes t, book.Authors a
where b.id_theme = t.id_theme and b.id_author = a.id_author
group by
      grouping sets ((t.NameTheme, a.FirstName + ' ' + a.LastName),
                      t.NameTheme);
```

Результатом буде одночасне групування (тематика, автор) і по тематикам загалом:

Results | Messages

	NameTheme	FullName	Загальна сума
1	C & C++	Джес Либерти	148,3621
2	C & C++	Олег Лисовский	432,2473
3	C & C++	Стивен Прата	465,0582
4	C & C++	NULL	1045,6676
5	Java, J++, JBuilder, JavaScript	Сергей Парижский	265,3398
6	Java, J++, JBuilder, JavaScript	NULL	265,3398
7	Linux	Marjorie Green	342,374



Як вже було вище сказано, оператор GROUPING SETS може давтаи результат аналогічний роботі операторів ROLLUP або CUBE. Розглянемо як це можна зробити, щоб вміти обирати кращий та простіший варіант. Спочатку проаналізуємо схожість операторів CUBE та GROUPING SETS.

Наприклад, напишемо запит, який виведе середню кількість проданих книг за весь період роботи видавництва в розрізі років та магазинів, які реалізовували книги.

```
select convert(char(4), DATEPART(YEAR, s.DateOfSale)) as 'Рік',
       sh.NameShop as 'Магазин',
       avg(s.Quantity) as 'Середня кількість продажу'
from book.Books b, sale.Sales s, sale.Shops sh
where b.ID_BOOK=s.ID_BOOK and s.ID_SHOP=sh.ID_SHOP
group by
       cube ((DATEPART(YEAR, s.DateOfSale)), sh.NameShop);
```

```
-- або
select convert(char(4), DATEPART(YEAR, s.DateOfSale)) as 'Рік',
       sh.NameShop as 'Магазин',
       avg(s.Quantity) as 'Середня кількість продажу'
from book.Books b, sale.Sales s, sale.Shops sh
where b.ID_BOOK=s.ID_BOOK and s.ID_SHOP=sh.ID_SHOP
group by
      grouping sets (  (DATEPART(YEAR, s.DateOfSale), sh.NameShop),
                       (DATEPART(YEAR, s.DateOfSale)),
                       (sh.NameShop),
                       ()
                    );
```

Результат в обох випадках буде наступний:

Results | Messages

	Рік	Магазин	Середня кількість продажу
1	2002	All about PC	6
2	NULL	All about PC	6
3	2007	Book	1
4	NULL	Book	1
5	2001	Booksworld	10
6	2006	Booksworld	5
7	2007	Booksworld	7
8	NULL	Booksworld	7
9	1999	Букинист	4

А тепер порівняємо як будуть виглядати еквівалентні запити з використанням операторів ROLLUP та GROUPING SETS.

```
select convert(char(4), DATEPART(YEAR, s.DateOfSale)) as 'Рік',
       sh.NameShop as 'Магазин',
       avg(s.Quantity) as 'Середня кількість продажу'
from book.Books b, sale.Sales s, sale.Shops sh
where b.ID_BOOK=s.ID_BOOK and s.ID_SHOP=sh.ID_SHOP
group by
       rollup ((DATEPART(YEAR, s.DateOfSale)), sh.NameShop);

-- або
select convert(char(4), DATEPART(YEAR, s.DateOfSale)) as 'Рік',
       sh.NameShop as 'Магазин',
       avg(s.Quantity) as 'Середня кількість продажу'
from book.Books b, sale.Sales s, sale.Shops sh
where b.ID_BOOK=s.ID_BOOK and s.ID_SHOP=sh.ID_SHOP
group by
      grouping sets (   (DATEPART(YEAR, s.DateOfSale), sh.NameShop),
                        (DATEPART(YEAR, s.DateOfSale)),
                        ()
                    );
```

Результат роботи обох запитів:

Results | Messages

	Рік	Магазин	Середня кількість продажу
14	2006	Книга	7
15	2006	Світ книги	8
16	2006	Слово	5
17	2006	NULL	6
18	2007	Book	1
19	2007	Booksworld	7
20	2007	NULL	4
21	NULL	NULL	8

ПРИМІТКА! При використанні операторів ROLLUP, CUBE або GROUPING SETS не можна застосовувати GROUP BY ALL, а в списку GROUP BY не повинно бути більше 10 полів. Також не можна використовувати дані типу text або image, оскільки вони не підлягають впорядкуванню.

3. Оператори PIVOT та UNPIVOT

Різного роду магазини та підприємсва в ході своєї діяльності використовують велику базу даних і час від часу їм необхідно отримувати по цим даним статистику. Наприклад, отримати для порівняння звіт по продажам за різні роки. Для вирішення таких бізнес-задач можна формувати дані у вигляді **звідних** або **перехресних таблиць (cross-tabulation)**. Це спеціальний тип статистичного запиту, в якому згруповані записи для одного з полів перетворюються в окремі поля.

Для створення звідних таблиць використовується оператор **PIVOT**.

```
SELECT  поле_для_заголовка_рядка,
        [перше_поле_для_значень], ...
FROM ( { назва_таблиці | SELECT_запит }) -- звідки отримувати дані
PIVOT
(
      функція_агрегування(поле)
      FOR [поле_для_заголовків_стовпчика]
      IN ( [перше_поле_для_значень],... )
) AS псевдонім_зведеної_таблиці
ORDER BY імя_поля | номер_поля [{ASC | DESC}] --необовязкова інструкція
```

Щоб зрозуміти краще роботу даного оператора і сам принцип побудови зведених таблиць, розглянемо все по-порядку. Для початку напишемо запит, який виводить середню вартість продажу по кожній тематиці:

```
select theme.NameTheme as 'Тематика',
       AVG(sale.Price*sale.Quantity) as 'Середній обсяг продаж'
from sale.Sales sale, book.Books book, book.Themes theme
where book.ID_THEME=theme.ID_THEME AND book.ID_BOOK=sale.ID_BOOK
group by theme.NameTheme;
```

Результат:

Results | Messages

	Тематика	Середній обсяг продаж
1	C & C++	566,75
2	Java, J++, JBuilder, JavaScript	826,6666
3	Windows 2000	1268,00
4	Windows NT	440,00
5	Учебники	413,75

Даний запит повертає два стовпчики даних: в одному – назви тематик, а в іншому - середня вартість продажу по кожній тематиці. Але, іноді користувачу необхідно для наглядності дані у вигляді звідної таблиці, в якій, наприклад, в одному рядку вказуються всі середні обсяги продажу певних тематик.

Щоб створити звідну таблицю слід зробити **наступні дії**:

1. Обрати необхідні дані за допомогою підзапиту, який називають **похідною таблицею (derived table)**.
2. Затосувати оператор PIVOT і вказати функцію агрегування, яку будете використовувати.
3. Визначити, які поля будуть включені у результуючий набір.

На відміну від звичайних звідних таблиць, наприклад, в MS Excel, MS Access (перехресні таблиці) тощо, оператор PIVOT в SQL Server вимагає явно перераховувати поля для результуючого набору. Це являється жорстким обмеженням, оскільки для цього необхідно знати характер даних, з якими ви працюєте.

Отже, перепишемо вищерозглянутий приклад таким чином, щоб утворилась звідна таблиця, яка буде відображати дані для порівняння про середній обсяг продажу підручників та книг таматик «C & C++» і «Windows NT».

```
select 'Середній обсяг продаж' as ' ',    -- поле, значення якого задає заголовок рядка
       [Учебники], [C & C++], [Windows NT]    -- перелік полів, які формують заголовки
                                              -- стовпчиків
from -- підзапит або назва таблиці
(
      select theme.NameTheme, (sale.Price*sale.Quantity) as Cost
      from sale.Sales sale, book.Books book, book.Themes theme
      where book.ID_THEME=theme.ID_THEME AND book.ID_BOOK=sale.ID_BOOK
)as SourceTable
```

```
pivot -- формування зведеної таблиці
(
      AVG(Cost)   -- функція агрегування, яка формує вміст зведеної таблиці
      for NameTheme -- для кого значення, тобто по чому групувати
      in ([Учебники], [C & C++], [Windows NT]) -- для кого формуються значення,
                                               -- для яких КОНКРЕТНО груп
) AS PivotTable
```

Результат:

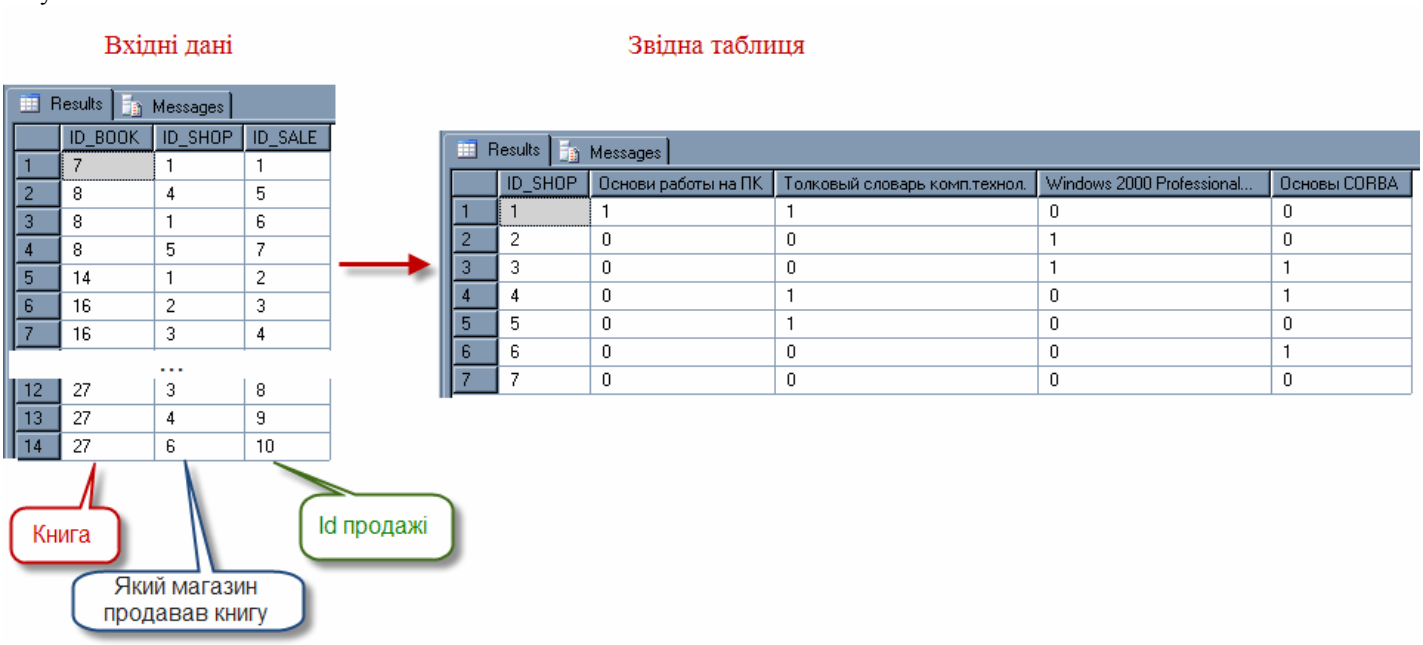
Results | Messages

	(No column name)	Учебники	C & C++	Windows NT
1	Середній обсяг продаж	413,75	566,75	440,00

У нас утворилась однорівнева звідна таблиця. Побудова двохрівневої таблиці дещо складніша. Для прикладу розглянемо побудову звідної таблиці, в якій будуть відображатись дані про кількість продажів магазинами чотирьох визначених книг. Утворені дані слід відсортувати по магазинам.

```
select      pvt.ID_SHOP,      -- дані про ідентифікатори магазинів
            -- перелік книг (їх ідентифікатори), для яких необхідні дані про продаж
            [7]  as 'Основи работы на ПК',
            [8]  as 'Толковый словарь комп.технол.',
            [16] as 'Windows 2000 Professional...',
            [27] as 'Основы CORBA'
from
      ( select ID_BOOK, ID_SHOP, ID_SALE
        from sale.Sales) p
pivot
(
      count (ID_SALE)
      for ID_BOOK in ( [7], [8], [16], [27] )
) as pvt
order by pvt.ID_SHOP;
```

Результат:



Оператор **UNPIVOT** виконує дії, зворотні по відношенню до операції PIVOT, тобто перетворює дані, які мають вигляд звідних таблиць, тобто записані в один рядок, в стовпчик. Цей оператор дуже корисний для нормалізації таблиць, в яких існує кілька полів з однаковим типом даних.

Для демонстрації роботи даного оператора, застосуємо оператор UNPIVOT для перетворення звідної таблиці з даними про середній обсяг продажу 3-х тематик.

```
select NameTheme, Cost -- назви полів: для назв груп і для їх значень відповідно
from
(     -- поміщаємо повністю pivot-запит. Початок
      select 'Середній обсяг продаж' as ' ', [Учебники], [C & C++], [Windows NT]
      from(
            select theme.NameTheme, (sale.Price*sale.Quantity) as Cost
            from sale.Sales sale, book.Books book, book.Themes theme
            where book.ID_THEME=theme.ID_THEME AND book.ID_BOOK=sale.ID_BOOK
      )as SourceTable
      pivot(
            AVG(Cost)
            for NameTheme in ([Учебники], [C & C++], [Windows NT])
      ) AS PivotTable
      -- кінець pivot-запиту
)as Pvt
unpivot
(
      Cost  -- поле, яке буде містити значення
      for NameTheme -- поле, звідки будуть взяті назви груп
      in ([Учебники], [C & C++], [Windows NT]) -- для кого формуються значення,
                                               -- для яких конкретно груп
) AS UnPivotTable
```

Результат:

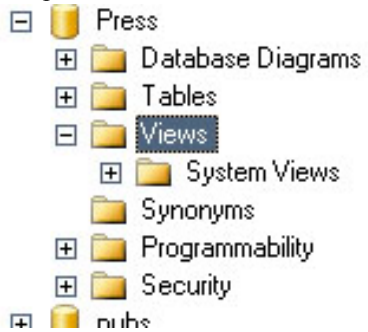
Results | Messages

	NameTheme	Cost
1	Учебники	413,75
2	C & C++	566,75
3	Windows NT	440,00

4. Представлення в MS SQL Server

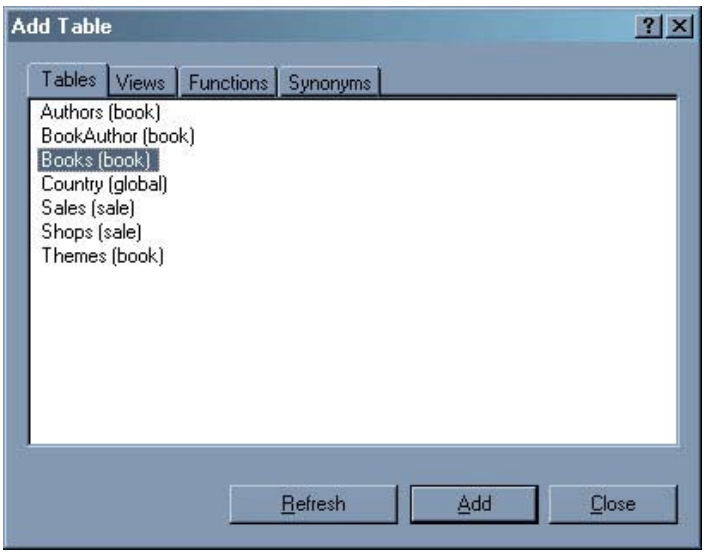
Представлення (view) – це об'єкт БД, який має зовнішній вигляд таблиці, але на відміну від неї немає своїх власних даних. Представлення лише надає доступ до даних однієї або кількох таблиць, на яких воно базується. За допомогою представлення можна здійснювати контроль даних, які вводяться користувачем, а також спростити роботу розробників: запити, які найчастіше виконуються, можна помістити в представлення, щоб не писати один і той же код постійно.

Оскільки, представлення являються об'єктом бази даних, то інформація про них зберігаються в системній таблиці **sysobjects**, текст оператора записується в таблицю **syscomments**, а фізичне розміщення знаходиться в папці **View**. Крім того, автоматично заповнюються наступні представлення: **sys.views**, **sys.columns**, **sys.sql_expression_dependencies** та **sys.sql_modules** (міститься текст інструкції create view).

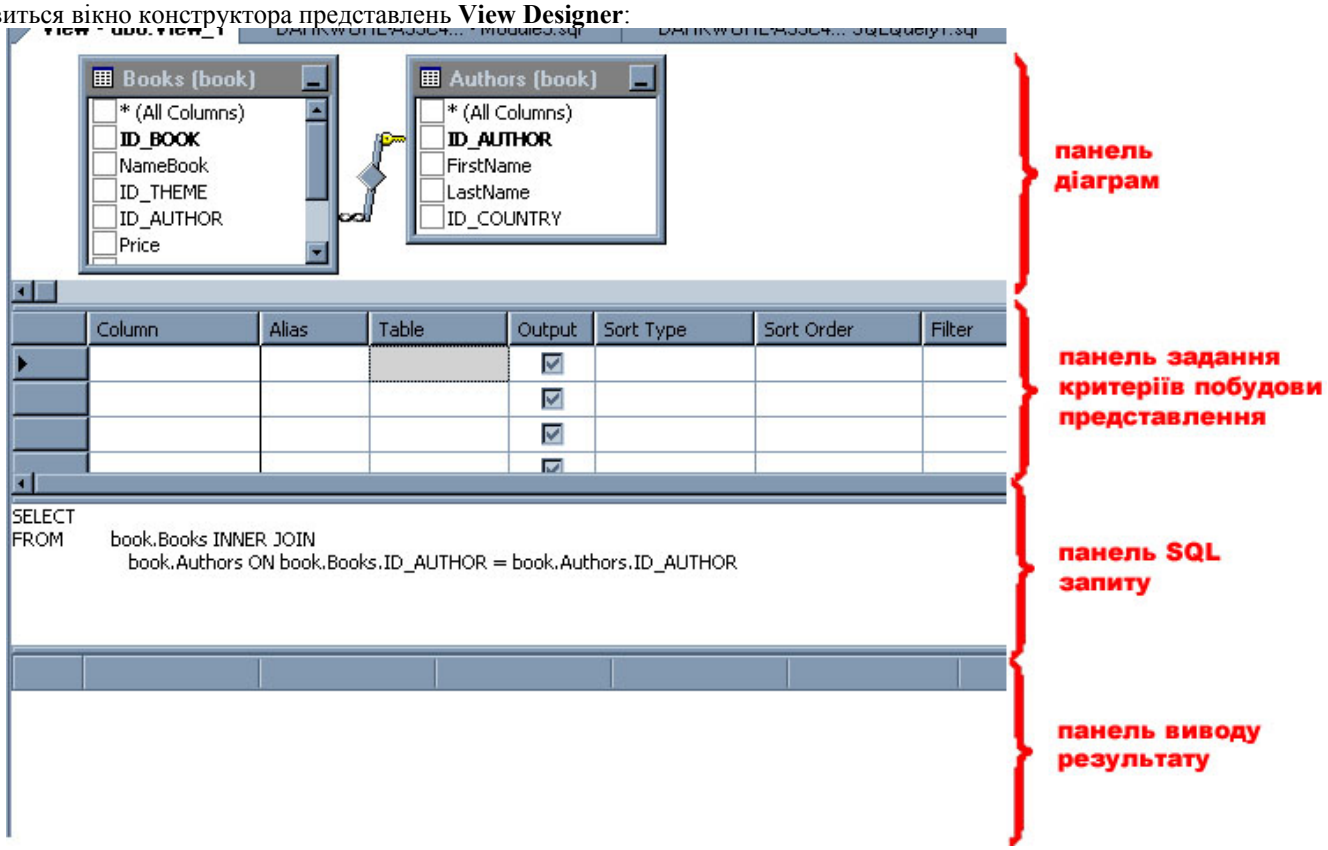


Представлення можна створювати як за допомогою Management Studio, так і за допомогою SQL. Розглянемо обидва способи. Отже, створимо представлення, що дозволяє отримати інформацію про книги та їх авторів, тираж книг яких більше 3000.

Для того, щоб створити такий запит засобами Management Studio, а саме за допомогою **Create View Wizard**, потрібно в контекстному меню папки View обрати пункт **New view**. Після цього перед Вами з'явиться вікно, з пропозицією обрати ті таблиці, з яких буде братись інформація для представлення:

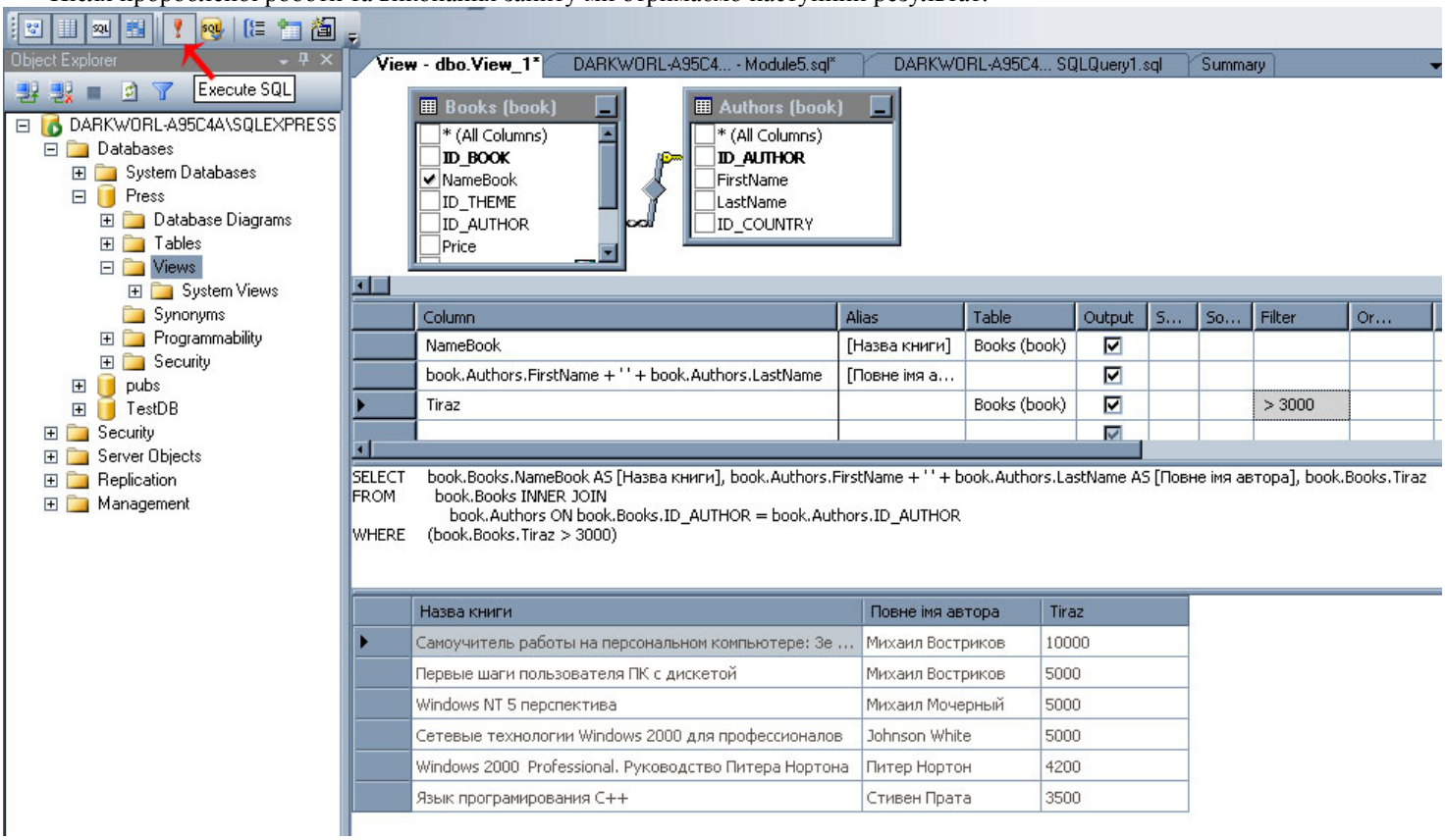


Обравши таблиці, які Вам потрібні (в нашому випадку це Books та Authors) натискаємо Add і Close. Після цього з'явиться вікно конструктора представлень **View Designer**:



Отже, в полі Table обираємо потрібні таблиці, в полі Column – необхідні поля цієї таблиці для виводу. В полі Alias можна задати псевдоніми на поля тощо.

Після проробленої роботи та виконання запиту ми отримаємо наступний результат:



Після цього представлення можна зберегти.

Для створення представлень засобами SQL використовують інструкцію CREATE VIEW, що має наступний синтаксис:

```
CREATE VIEW [схема.] назва_представлення [ (поле [,... n] ) ]
[ WITH ENCRYPTION | SCHEMABINDING | VIEW_METADATA ]
AS
<оператор SELECT>
[ WITH CHECK OPTION];
```

Розшифруємо коротко кожен з **параметрів** даного оператора:

- ✓ **WITH ENCRYPTION** – вказує на те, що представлення буде зберігатись в системній таблиці syscomments в зашифрованому вигляді. Розшифрувати таке представлення неможливо, тому при виборі даного параметра слід впевнитись, що у вас існує його копія.
- ✓ **WITH SCHEMABINDING**. Ця опція встановлює заборону видалення таблиць, представлень або функцій, на які ссилається представлення, раніше, ніж воно буде видалене.
- ✓ **WITH VIEW_METADATA** – вказує на те, що представлення в режимі перегляду буде повертати його метадані, тобто інформацію про його структуру, а не записи таблиць.
- ✓ **WITH CHECK OPTION** – створює модифіковане представлення, в якому існує можливість заборонити виконання операцій INSERT або DELETE, якщо при цьому порушується умова, задана в конструкції WHERE.

Після ключового слова AS повинен розміщуватись запит з використанням оператора **SELECT**, що призначений для вибору даних, що будуть включені в результуюче представлення. Варто відмітити, що для того, щоб змінити довільне представлення, необхідно його перестворювати, тобто видалити і знову створити. А при видаленні потрібно видалити всі залежні від нього об'єкти – тригери, зберігаємі процедури тощо. Це є одним з основних незручностей при роботі з представленнями.

Отже, спробуємо написати представлення, яке ми створювали за допомогою View Designer, використовуючи оператор SQL CREATE VIEW:

```
create view book.BooksEdition_View2 (NameBook, Author, Edition)
as
select b.NameBook, a.FirstName + ' ' + a.LastName as 'FullName', b.Tiraz
from book.Books b, book.Authors a
where b.id_author = a.id_author and b.Tiraz > 3000;
```

Створене представлення можна використовувати як будь-яку таблицю бази даних. І щоб отримати інформацію з нього дані, слід написати простий запит на вибірку за допомогою оператора SELECT:

```
select *
from book.BooksEdition_View2;
```

Результат:

	NameBook	Author	Edition
1	Самоучитель работы на персональном компьютере: 3...	Михаил Вострикov	10000
2	Первые шаги пользователя ПК с дискетой	Михаил Вострикov	5000
3	Windows NT 5 перспектива	Михаил Мочерный	5000
4	Сетевые технологии Windows 2000 для профессионалов	Johnson White	5000
5	Windows 2000 Professional. Руководство Питера Норто...	Питер Нортон	4200
6	Язык програмирования C++	Стивен Прата	3500

Та саме по собі представлення НЕ МІСТИТЬ жодних даних. Воно просто ссилається на вже існуючі дані, тобто являється звичайним SELECT запитом, який просто має ім'я. Фактично, коли за допомогою запиту ви звертаєтесь до представлення, оптимізатор запитів заміняє це посилання на опис представлення (тіло), а потім створює план виконання.

Представлення можуть бути основані на даних з декількох таблиць і навіть на основі інших представлень. Також представлення можуть містити дані, що отримані на основі різноманітних виразів, в тому числі і на основі функцій агрегування.

Розрізняють наступні **типи представлень**:

1) **Звичайні представлення на основі об'єднань** – це представлення на вибірку даних.
2) **Модифіковані (обновлювані) преставлення** – це представлення, які підтримують модифікацію даних.
3) **Індексовані або матеріалізовані представлення** – це представлення з використанням кластеризованого індекса. Такі представлення дозволяють суттєво підвищити продуктивність навіть дуже складних запитів. Індексовані представлення детальніше будуть розглянуті при вивченні індексів.
4) **Секціоновані представлення** – це представлення, яке при створенні використовує оператор UNION ALL для об'єднання кількох запитів, які побудовані на основі таблиць з однаковою структурою та зберігаються або в одному екземплярі SQL Server, або в групі автономних екземплярів SQL Server, які називаються федеративними серверами баз данных. Детальніше інформацію можна прочитати в розділі «Створення секціонованих представлень» MSDN.

Для успішної роботи варто лише зрозуміти, що в основі представлення лежать звичайні запити на вибірку. Отже, все, що можна робити з запитами відноситься і до представлень. Та, якщо необхідно вивести дані у відсортованому порядку нам не пощастить, оскільки сортувати можна лише результуючі дані представлення:

```
select *
from book.BooksEdition_View2
order by 1;
```

Хоча і тут існує нюанс: якщо в операторі SELECT використовується опція TOP, тоді інструкція ORDER BY може використовуватись при створенні представлення. Тобто наступний запит буде вірним:

```
create view TestView
as
select top (3) b.NameBook, t.NameTheme, b.DateOfPublish
from book.Books b, book.Themes t
where b.ID_THEME = t.ID_THEME
order by 1;
```

Не дивлячись на зручність використання представлень, крім обмеження на використання виразу ORDER BY існує ще ряд суттєвих **обмежень**:

a) не можна використовувати в якості джерела даних набір, отриманий в результаті виконання зберігаємих процедур;
b) при створенні представлення оператор SELECT не повинен містити оператори COMPUTE або COMPUTE BY;
c) представлення не може ссилатись на тимчасові таблиці, тому в операторі створення ЗАБОРОНЕНО використовувати оператор SELECT INTO;
d) дані, які використовуються представленням, не зберігаються окремо, тому при зміні даних представлення змінюються дані базових таблиць;
e) представлення не може ссилатись більше, ніж на 1024 поля;
f) UNION і UNION ALL недопустимі при формуванні представлень.

Для модифікації представлення використовується оператор **ALTER VIEW**:

```
ALTER VIEW [схема.] назва_представлення [ (поле [,... n] ) ]
[ WITH ENCRYPTION | SCHEMABINDING | VIEW_METADATA ]
AS
<оператор SELECT>
[ WITH CHECK OPTION];
```

Для того, щоб видалити представлення використовується оператор **DROP VIEW**:

```
DROP VIEW назва_представлення;
```

На початку даного розділу ми говорили, що представлення можуть бути **модифікованими (обновлюваними, updateable view)**. Що ж це за представлення? Як видно з самої назви, такі представлення дають змогу не лише читати дані, але й змінювати їх. Модифікованими (обновлюваними) представлення стають, якщо при їх створенні вказати інструкцію **WITH CHECK OPTIONS**.

До таких представлень можна застосовувати команди INSERT, UPDATE і DELETE. Причому вони дозволяють заборонити виконання операцій INSERT або DELETE, якщо при цьому порушується умова, що задана в конструкції WHERE. Це можна пояснити так: якщо новий запис, який вставляється користувачем або отриманий в результаті обновлення існуючого запису, не задовольняє умову запиту, то вставка цього запису буде відмінена і виникне помилка.

Розглянемо **умови створення модифікованих представлень**:

- Всі модифікації повинні стосуватись лише однієї таблиці, тобто модифіковані представлення будуються лише на однотабличних запитах.
- Всі зміни повинні стосуватись лише полів таблиці, а не похідних полів. Тобто, не можна модифікувати поля, отримані:
  - за допомогою агрегатної функції: AVG, COUNT, SUM, MIN, MAX, GROUPING, STDEV, STDEVP, VAR і VARP;
  - на основі розрахунків за участі інших полів або операцій над полем, наприклад substring. Поля, сформовані за допомогою операторів UNION, UNION ALL, CROSSJOIN, EXCEPT і INTERSECT також вважаються розрахунковимими і також не можуть обновлюватись.
- При визначенні представлення не можна використовувати інструкції GROUP BY, HAVING і DISTINCT.
- Не можна використовувати опцію TOP разом з інструкцією WITH CHECK OPTION, навіть у підзапитах.

Якщо вказані обмеження не дозволяють модифікувати дані, тоді можна скористатись тригерами INSTEAD OF або секціонованими представленнями.

Наприклад, необхідно мати представлення, яке буде відображати інформацію про книги, які мали тираж більше 4000 екземплярів:

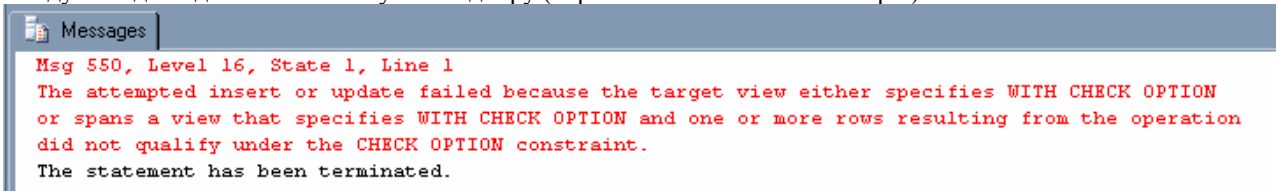
```
create view Books5000 (NameTovar, Price, DrawingOfBook)
as
select NameBook, Price
from book.Books
where DrawingOfBook > 4000
with check option;
```

Саме по собі модифіковане представлення нічим не відрізняється від звичайного, але спробуємо перевірити дане модифіковане представлення на операцію вставки даних.

```
insert into Books5000 (NameTovar, Price, DrawingOfBook)
values ('Microsoft SQL Server 2000 за 21 день', 350.50, 3000);
```

У випадку звичайного представлення, користувач може додати таку книгу і помилка при цьому не виникне. Але даний запит на вставку являється некоректним та призведе до путанини: представлення при наступному зверненні введених даних показувати не буде, оскільки вони не відповідають критерію відбору даних.

При введенні таких даних у модифіковане представлення, сервер одразу видасть повідомлення про помилку, оскільки дані вводу не відповідають основній умові відбору (тираж більше 4000 екземплярів).



Наступний запит на вставку являється коректним і виконається, оскільки тираж випуску вказується більше 4000.

```
Insert into Books5000 (NameTovar, Price, DrawingOfBook)
values ('Microsoft SQL Server 2000 за 21 день', 350.50, 4500);
```

Підсумовуючи, слід сказати, що хоча модифіковані представлення і використовуються для зміни даних, але для цієї цілі вони практично ніколи не використовуються. Найкраще для таких дій підходять зберігаємі процедури. Крім того, вони набагато гнучкіші.

6. Домашнє завдання

1. За допомогою оператора COMPUTE BY написати запит, який виводить назви магазинів і сумарну кількість замовлених магазином книг.
2. За допомогою оператора COMPUTE вивести дані про те, скільки отримало видавництво від продажу книг за останній рік.
3. За допомогою операторів CUBE та ROLLUP написати два запити, які будуть відображати інформацію про середню кількість продаж книг магазинами Англії та України в проміжку 01/01/2008 по 01/09/2008. Показати середню кількість продаж як по кожній країні, так і по окремому магазину.
4. Використовуючи оператор GROUPING SET написати запит, який виведе максимальний тираж книг за весь період роботи видавництва в розрізі авторів та років випуску книг.
5. Використовуючи оператор PIVOT створити звідну таблицю, що відображає мінімальні ціни продажу книг окремими магазинами за останній рік.
6. Використовуючи оператор PIVOT створити двохрівневу звідну таблицю, яка відображає дані про кількість випуску видавництвом книг всіх тематик на протязі трьох обраних років. Відсортувати дані по тематикам.
7. За допомогою оператора UNPIVOT перетворити звідні таблиці у звичайні для прикладів (5) та (6).
8. Знайти авторів, які живуть в тих країнах, де є хоча б один з магазинів по розповсюдженнь книг, знанесених в БД. Результат запиту помістити в окреме представлення.
9. Написати представлення, що містить саму найдорожчу книгу тематики, наприклад, «WEB-програмування».
10. Написати представлення, що дозволяє вивести всю інформацію про роботу магазинів. Відсортувати вибірку по країнам в зростаючому і по назвам магазинів в спадаючому порядку.
11. Написати зашифроване представлення, що показує саму найпропулярнішу книгу.
12. Написати модифіковане представлення, в якому надається інформація про авторів, імена яких починаються з A або B
13. Написати представлення, в якому за допомогою підзапитів вивести назви магазинів, які ще не продають книги вашого видавництва.