## ОСНОВИ ПРОГРАМУВАННЯ У TRANSACT-SQL

Для мови програмування потрібні мінімум чотири речі: змінні, умовні оператори, цикли і процедури.

### ЗМІННІ У TRANSACT-SQL

Змінні позначаються префіксом @; наприклад @MyVariable. Як і тимчасові таблиці, змінні мають дві області дії: локальну та глобальну. Глобальні змінні позначаються подвійним символом @:
@ @ VERSION

Всі глобальні змінні визначаються SQL Server, і не можна визначити їх самостійно. Крім того, область дії локальних змінних ще вужче, ніж у локальних таблиць: вони доступні тільки в межах пакету або процедури, в якій вони оголошені.

*Локальні змінні*
Програміст створює собі потрібні локальні змінні. Оголошення локальної змінної:

```
DECLARE @local_variable type
```

Після створення локальна змінна спочатку має значення NULL. Ви можете привласнити змінній значення наступними способами:

- використовувати команду SET із зазначенням константи або змінної:

  SET @myCharVariable = 'Hello, World'

- використовувати команду SELECT із зазначенням константи або змінної:

  ```
  SELECT @myCharVariable = 'Hello, World'
  ```

- використовувати команду SELECT із зазначенням іншого оператора SELECT:

  ```
  SELECT @myCharVariable = MAX (OilName) FROM Oils
  ```

  Оператор присвоєння (=) заміщає друге ключове слово SELECT; воно вдруге не повторюється.

- використовувати команду INSERT INTO із зазначенням змінної табличного типу:

  ```
  INSERT INTO @myTableVariable
      SELECT * FROM Oils
  ```

  Також можна використовувати синтаксис INSERT INTO ... VALUES.

*Використання змінних*
Змінні можуть використовуватися у всіх виразах мови Transact-SQL. Однак, вони не використовуються замість імені об'єкта або ключового слова. Таким чином, представлені нижче оператори будуть коректними:

```
DECLARE @theOil char(20)
SET @theOil = 'Basil'
```

-- Ця команда буде виконана
```
SELECT OilName, Description
FROM Oils
WHERE OilName = @theOil
```
Але обидва наступних оператора SET не будуть виконані:
```
DECLARE @theCommand char(10), @theField char(10)
SET @theCommand = 'SELECT'
SET @theField = 'OilName'
```

-- Ця команда не буде виконана
```
@theCommand * FROM Oils
```

-- І ця також
```
SELECT @theField from Oils
```

Отже, на місце константи можна підставити значення змінної, а на місце команди, імені відношення або імені атрибуту – не можна.

## БЛОКИ ОПЕРАТОРІВ

При управлінні ходом виконання Transact-SQL зручно інтерпретувати певну групу команд як блок. Transact-SQL дозволяє зробити це з допомогою пари команд BEGIN ... END.

Якщо за будь-якою командою управління ходом виконання (як-от в умовних операторах, циклах) є ключове слово BEGIN, Transact-SQL буде застосовувати цю команду управління для всіх операторів, розташованих між BEGIN і END.

## УМОВНЕ ВИКОНАННЯ

Оператор IF є найпростішим з набору команд управління ходом виконання. Якщо логічний вираз, наступний за командою IF, має значення TRUE, то буде виконаний наступний оператор або блок операторів. Якщо логічний вираз має значення FALSE, то наступний оператор або блок операторів буде пропущений.

Необов'язкова команда ELSE дозволяє задавати оператор або блок операторів, який буде виконуватися, якщо логічний вираз має значення FALSE. Наприклад, команди Transact-SQL, що подаються нижче, повертають 'Істина', якщо @ test має значення "істина", і "Брехня" („Хибність”), якщо ні.

```
IF @test
  SELECT 'Істина'
ELSE
  SELECT 'Хибність'

Оператор IF з блоками команд:
IF @test
BEGIN
  <Команда 1>
  ...
  <Команда m>
END
ELSE
BEGIN
  <Команда 1>
  ...
  <Команда n>
END
```

## CASE

В SQL Server CASE є функцією, а не командою. Вона використовується не сама по собі, як IF, а як частина оператора SELECT або UPDATE.

Синтаксис простої структури CASE:

```
CASE значення
 WHEN вираз_один THEN результуючий_вираз_один
 WHEN вираз_два THEN результуючий_вираз_два
.
.
.
 WHEN вираз_n THEN результуючий_вираз_n
 [ELSE альтернативний_результуючий_вираз]
END
```

У цій формі функції CASE вираз *результуючий_вираз* повертається тільки в тому випадку, якщо вираз, наступний за ключовим словом WHEN, дорівнює зазначеному значенню *значення*. Можна використовувати у виразі будь-яку кількість фраз WHEN. Фраза ELSE необов'язкова - вона виконується, тільки якщо всі фрази WHEN оцінюються як FALSE.

У наступному фрагменті на рис. 1 змінній @Sd треба присвоїти значення змінних @Sd1, або @Sd2, або @Sd3, і т.д., залежно від номеру місяця @j.

```
--Перерахунок сальдо на кінець місяця @j

SET @Sd =       CASE @j
                WHEN 1 THEN @Sd1
                WHEN 2 THEN @Sd2
                WHEN 3 THEN @Sd3
                WHEN 4 THEN @Sd4
                WHEN 5 THEN @Sd5
                WHEN 6 THEN @Sd6
                WHEN 7 THEN @Sd7
                WHEN 8 THEN @Sd8
                WHEN 9 THEN @Sd9
                WHEN 10 THEN @Sd10
                WHEN 11 THEN @Sd11
                WHEN 12 THEN @Sd12
                END
```

Рисунок 1 - Фрагмент збереженої процедури з функцією CASE

## ЦИКЛИ

Цикл дозволяє виконувати оператор або блок операторів, допоки виконується (є істинною) зазначена умова.

*Простий цикл WHILE*

Найпростіша форма циклу WHILE містить булевий вираз і оператор або блок операторів. Оператори будуть повторюватися, поки логічний вираз не стане FALSE. Якщо при першій оцінці булевий вираз має значення FALSE, то оператор або блок операторів не буде виконуватися взагалі.

Query - bunny.Aromatherapy.BUNNY\Rebecca - C:\Documents and Setting

```
USE Aromatherapy
GO

-- Use a local variable as an incremental counter
DECLARE @counter int
SET @counter = 1

WHILE @counter < 11
BEGIN
        PRINT @counter
        SET @counter = @counter + 1
END
```

Рисунок 2 - Приклад простого циклу

*Складні цикли WHILE*
Синтаксис оператора WHILE також дозволяє здійснювати і більш складну логіку. Фраза BREAK викликає вихід з циклу; виконання продовжується з оператора, наступного за фразою END блоку оператора структури WHILE.

```
USE Aromatherapy
GO

-- Use a local variable as an incremental counter
DECLARE @counter int
SET @counter = 1

WHILE @counter < 25
BEGIN
        PRINT @counter
        SET @counter = @counter + 1
        IF @counter > 10 BREAK
END
```

Рисунок 3 - Приклад використання BREAK для виходу з циклу

```
USE Aromatherapy
GO

-- Use a local variable as an incremental counter
DECLARE @counter int
SET @counter = 0

WHILE @counter < 11
BEGIN
        SET @counter = @counter + 1
        IF (@counter % 2) = 0 CONTINUE
        PRINT @counter
END
```

Рисунок 4 - Приклад використання CONTINUE для виходу на початок циклу
(% : @counter по модулю 2)

Фраза CONTINUE повертає виконання на початок циклу, при цьому оператори, наступні за CONTINUE в межах блоку операторів, будуть пропущені.

Якщо потрібно, можна використовувати команди BREAK і CONTINUE в одному і тому ж операторі WHILE.

## КУРСОРИ В TRANSACT SQL

Існують операції, які важко або навіть неможливо виконати на основі принципів роботи з множинами, не обираючи конкретний об’єкт з множини. Щоби впоратися з подібними ситуаціями, в SQL передбачені курсори. Курсор являє собою об'єкт, який вказує на певний рядок у множині. Залежно від суті створеного курсору, можна переміщати курсор всередині множини і модифікувати або видаляти дані.

У збережених процедурах з допомогою курсорів можна виконувати складні обчислення, які важко виразити за допомогою синтаксису інструкції SELECT.

Ніколи не використовуйте курсор, якщо для виконання завдання досить оператора SELECT або UPDATE.

## КУРСОРИ API

Крім курсорів Transact SQL, SQL Server підтримує курсори API, для кожної бібліотеки доступу до БД – свій інтерфейс:

- Курсори ActiveX Data Objects (ADO) – клас Recordset та його методи;
- Курсори ODBC;
- Курсори JDBC - клас ResultSet, аналогічно ODBC.

Класи Recordset і ResultSet створюються програмістом у прикладній мові програмування, як-от C++ чи Java, до якої підключається відповідна бібліотека доступу до БД.

## КУРСОРИ TRANSACT SQL

Курсори Transact-SQL створюються за допомогою команди DECLARE CURSOR. Як об'єкт курсору, так і множина рядків, на яку він вказує, повинні існувати на сервері. Подібні курсори називаються *серверними курсорами*. Якщо ви використовуєте серверний курсор з програми, з’єднаної з SQL Server через мережу, кожна операція з курсором вимагає двосторонньої взаємодії через мережу. Бібліотеки API-курсорів, які підтримують серверні курсори, підтримують також клієнтський курсор, який існує в клієнтській системі і кешує рядки, які він обробляє на клієнті.

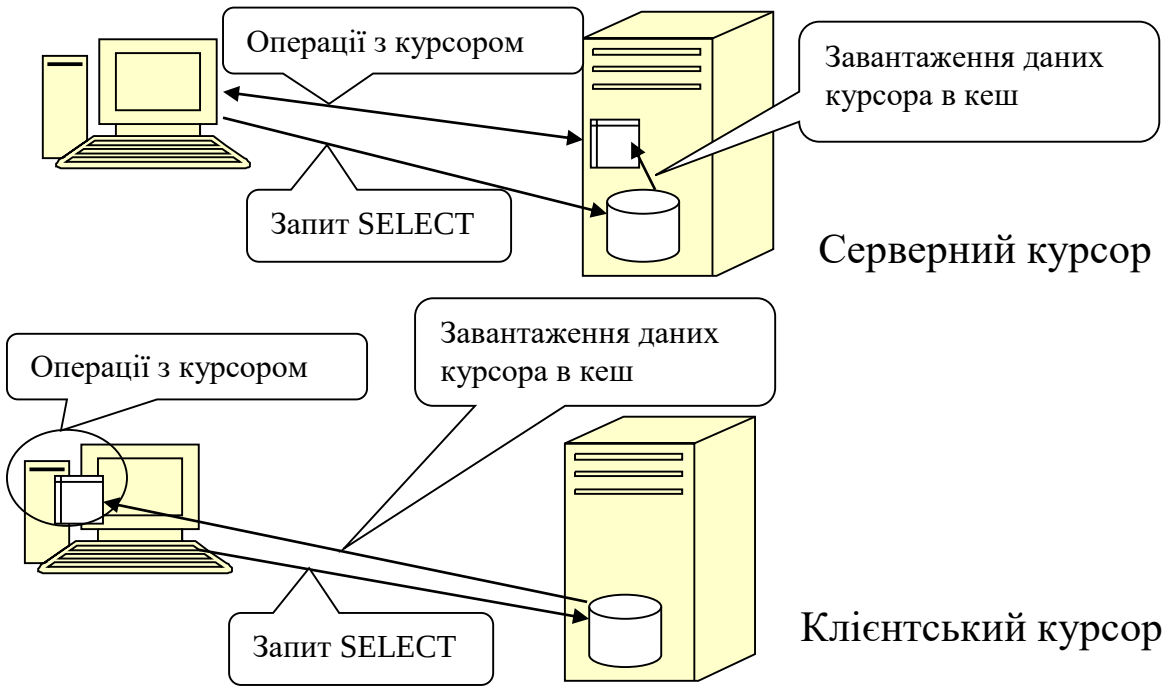


Рисунок 5 - Серверний курсор створюється в кеші в оперативній пам'яті на сервері, а клієнтський курсор в кеші на робочій станції.

Відношення (множина рядків), на яке вказує курсор, визначається за допомогою команди SELECT. При створенні курсору Transact-SQL на команду SELECT накладаються обмеження, зокрема:

- команда SELECT не може містити фразу INTO для створення нової таблиці.

*Характеристики курсорів*

Приймемо до уваги для кожного типу курсору три більш-менш незалежних характеристики:

- здатність відображати зміни у таблицях БД, здійснені, можливо, іншим клієнтом,
- здатність здійснювати прокрутку (переміщення) в відношенні, а також
- здатність модифікувати відношення.

## ОГОЛОШЕННЯ КУРСОРУ В TRANSACT-SQL

(Books Online, стаття Declare Cursor)

DECLARE *cursor_name* CURSOR
[ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
[ TYPE_WARNING ]
FOR *select_statement*
[ FOR UPDATE [ OF *column_name* [ **,**...*n* ] ] ]

Розглянемо параметри оголошення.

## LOCAL|GLOBAL

LOCAL: Ім'я курсору дійсне тільки всередині пакету, збереженої процедури або тригера, в яких курсор був створений. По замовчанню **видимість** LOCAL.

GLOBAL: Ім'я курсору може використовуватися в будь-якій частині збереженої процедури або пакеті, який виконує з'єднання. Курсор неявно звільняється тільки при відключенні з'єднання.

### FORWARD_ONLY | SCROLL

Здатність здійснення **прокрутки** як вперед, так і назад (SCROLL), або тільки вперед (FORWARD_ONLY). Послідовні курсори (forward-only) працюють значно швидше, але мають меншу гнучкість.

### STATIC | KEYSET | DYNAMIC | FAST_FORWARD

Розглянемо типи курсорів з точки зору **оновлення даних**: статичні, ключові, динамічні і курсори швидкого доступу, або "пожежні" (firehose). Кожен тип курсора зберігає різні дані щодо рядків, на які він вказує, крім того, кожному типу курсору властиві різні поєднання характеристик (відображення змін, прокрутка, модифікація).

*Статичні курсори* - STATIC
Статичний курсор робить немовби моментальний знімок даних, що задаються оператором SELECT, і зберігає їх у базі даних tempdb. Він "не відчуває" змін у структурі чи в значеннях даних, які відбулись після його відкриття, а оскільки будь-які модифікації будуть відображені тільки в копії, цей курсор завжди відкривається в режимі "тільки читання". Статичні курсори, однак, можуть бути оголошені як послідовні (FORWARD_ONLY) або як прокручувані (SCROLL).

*Ключові курсори* - KEYSET
Ключовий курсор копіює в базу tempdb тільки ті стовпці, які унікально ідентифікують кожний рядок. Щоб мати можливість оголосити ключовий курсор, кожна таблиця, яка входить до визначення оператора SELECT, повинна мати унікальний індекс.

Ключові курсори можуть бути як "тільки для читання", так і такими, що підлягають модифікації. Вони також можуть бути прокручуваними або послідовними.

Членство в ключовому курсорі фіксується на момент оголошення курсору. Якщо в відкритому стані курсору додається кортеж, що задовольняє умові відбору, він не буде доданий у множину кортежів курсора. Аналогічно, якщо зміна вноситься в кортеж, який після цього не буде задовольняти умові членства в множині, кортеж все ж таки залишиться членом множини. Навіть якщо кортеж видаляється, він як і раніше залишається членом множини, але SQL Server повертає NULL для всіх значень стовпців.

Але зміни значень даних, що вносяться до вихідної таблиці – джерела курсору, знаходять відображення. Проте зміни значень множини ключів відображаються в курсорі лише якщо вони здійснюються усередині курсору.

Для оновлення змісту ключового курсору можна зачинити і повторно відчинити його.

*Динамічні курсори* - DYNAMIC
Динамічний курсор веде себе так, мовби при кожному зверненні до рядка повторно виконується оператор SELECT (насправді все відбувається інакше). Динамічні курсори відображають зміни, пов'язані як з членством, так і зі значеннями вихідних даних, незалежно від того, чи зроблені ці зміни всередині курсору, чи внесені іншим користувачем.

Для динамічних курсорів фразу ORDER BY використовують тільки якщо є індекс, що включає в себе стовпці з ORDER BY.

*Курсори швидкого доступу* - FAST_FORWARD
SQL Server підтримує спеціальну оптимізовану форму не прокручуваного курсору, що допускає тільки читання. Його називають "пожежним" курсором (firehose).
"Пожежні" курсори дуже ефективні, але при їх використанні є два важливих обмеження, додаткових відносно статичного курсора.

- В операторі SELECT курсору не має бути полів типів text, ntext або image (бо насправді частина значень цих полів зберігається в допоміжних таблицях), а також речення TOP.
- Таблиці оператора SELECT курсора не мають містити тригери. Інакше курсор перетвориться в статичний.

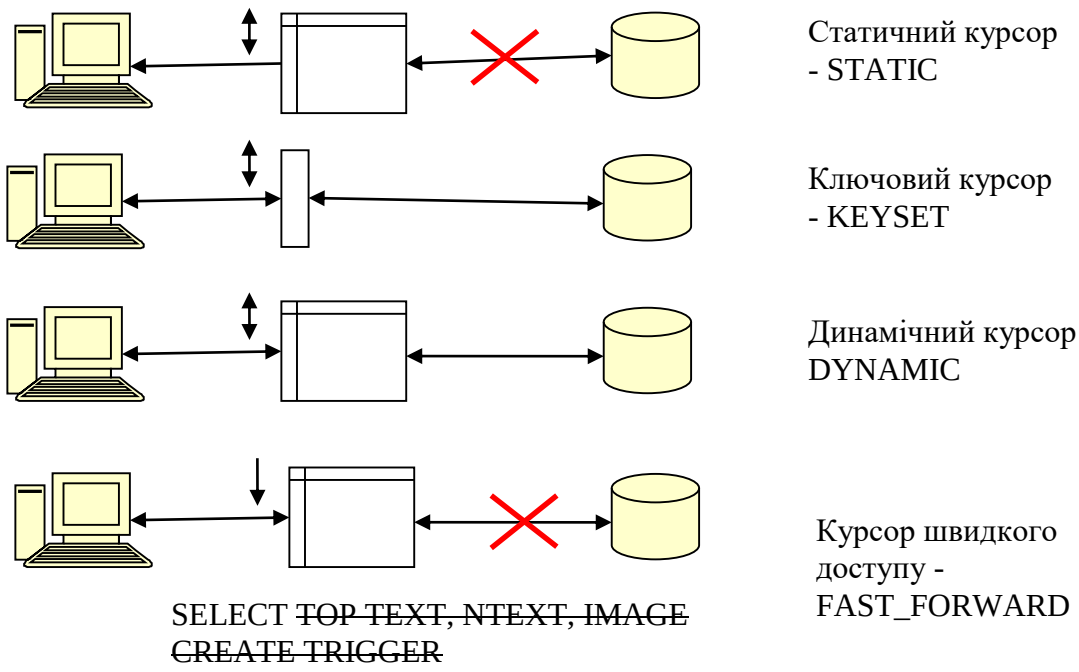


Рисунок 6 - Види курсорів з точки зору оновлення даних

### READ_ONLY | SCROLL_LOCKS | OPTIMISTIC

Задає режим **блокувань** кортежів у базових таблицях курсору.

- READ_ONLY забороняє оновлення через курсор і не блокує дані.
- SCROLL_LOCKS блокує кортежі базових таблиць, які попали в курсор, і гарантує успішне оновлення базових таблиць.
- OPTIMISTIC не блокує кортежі базових таблиць, але якщо певні таблиці оновлені іншим процесом після відчинення курсора, в результаті спроби курсора зберегти своє оновлення буде видане повідомлення про необхідність введення даних ще раз. Для динамічного курсора це режим по замовчанню.

### TYPE_WARNING

Визначає, чи треба надсилати повідомлення про зміну типу курсору, якщо курсор не може набути потрібного типу, вказаного в оголошенні.

### FOR UPDATE [ OF *COLUMN_NAME* [ ,...*N* ] ]

Вказує на можливість оновлення курсору або його вказаних полів.

*Таблиця 5.21. Сумісність типів і параметрів курсора*

| Характеристика\ Тип курсора | STATIC | KEYSET | DYNAMIC | FAST_FORWARD |
|---|---|---|---|---|
| FORWARD_ONLY | + | + | + | + |
| SCROLL | + | + | + | - |
| READ_ONLY | + | + | + | + |
| SCROLL_LOCKS | - | + | + | - |
| OPTIMISTIC | - | + | + | - |
| FOR UPDATE | - | + | + | - |

## КУРСОРНІ ЗМІННІ, ВІДЧИНЕННЯ І ЗАЧИНЕННЯ КУРСОРУ

Transact-SQL дозволяє оголошувати змінні типу CURSOR. У цьому випадку стандартний синтаксис DECLARE не створює курсор, і треба явно встановити змінну для курсору за допомогою ключового слова SET. Приклад створення курсору :

```
DECLARE myCursor CURSOR
        LOCAL
        FAST_FORWARD
        FOR SELECT OilName FROM Oils
DECLARE @myCursorVariable CURSOR
SET @myCursorVariable = myCursor
```

Ви можете оголосити курсорну змінну, а потім використовувати її для безпосереднього створення курсору:

```
DECLARE @myCursorVariable CURSOR
SET @myCursorVariable = CURSOR
        LOCAL FAST_FORWARD FOR SELECT OilName FROM Oils
```

Множина курсора не створюється, поки ви не відчините курсор:

```
OPEN [GLOBAL] курсор_або_змінна
```

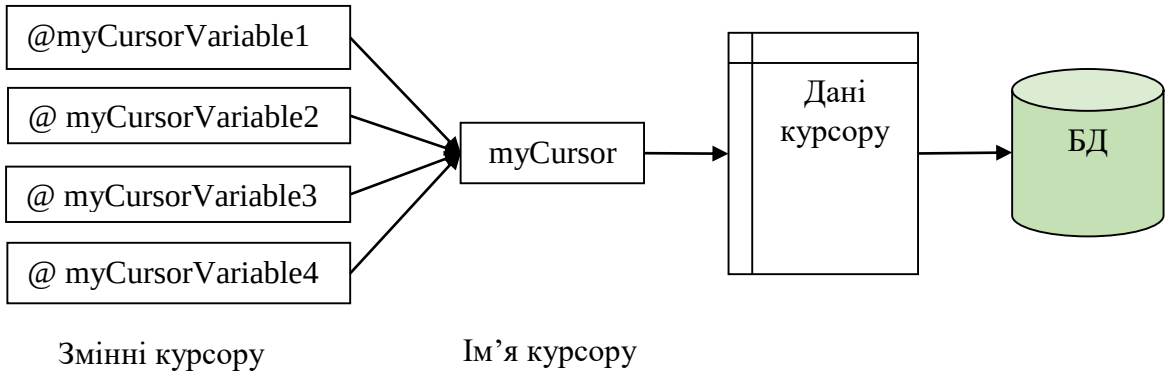


Рисунок 7 – Використання імені курсору і змінних курсору

Закінчивши використання курсору, треба його зачинити. Оператор CLOSE звільняє деякі ресурси (результуючий набір курсору і блокування поточного рядка). Курсор залишється оголошеним і його можна заново відчинити оператором OPEN.

```
CLOSE [GLOBAL] курсор_або_змінна
```

Після завершення роботи з курсором треба завжди його знищувати. Це робить оператор DEALLOCATE - видаляє ідентифікатор або курсорну змінну, але він не обов'язково видаляє сам курсор. Сам курсор не видаляється до тих пір, поки всі ідентифікатори, що посилаються на нього, будуть або звільнені, або перестануть діяти.

Знищення курсору оператором DEALLOCATE тягне його зачинення. Але знищення відчиненого курсору вважається неправильним.

Приклад:

```
-- Створення курсору. myCursor – ім’я курсору.
DECLARE myCursor CURSOR
        KEYSET
        READ_ONLY
        FOR SELECT * FROM Oils
-- Створення курсорної змінної
DECLARE @cursorVariable CURSOR
-- Створення множини записів курсора
OPEN myCursor
-- Призначення змінної курсору
SET @cursorVariable = myCursor
<Використання курсора>
-- Звільнення курсора, доступ до нього призупиняється
CLOSE myCursor
DEALLOCATE myCursor
```

Курсор відчиняється незалежно від того, посилається OPEN на ім’я чи на змінну курсору. Для будь-якої команди, крім DEALLOCATE, ім’я і змінна курсору рівнозначні.

- Команда DEALLOCATE <ім’я курсору> забороняє доступ до курсору по вказаному імені.
- Команда DEALLOCATE <змінна курсору> забороняє доступ до курсору по вказаній змінній.
- Курсор фізично знищується, коли виконується DEALLOCATE для останньої змінної курсору, що посилалась на нього.

## МАНИПУЛЮВАННЯ КОРТЕЖАМИ ЗА ДОПОМОГОЮ КУРСОРУ. КОМАНДА *FETCH*

Команда FETCH витягає вказаний кортеж з множини кортежів курсору. Найпростіший варіант:

```
FETCH курсор_або_змінна
```

Команда FETCH може не тільки повертати рядок безпосередньо, але і дозволяє зберігати значення з повернутого стовпця в змінних.

```
FETCH курсор_або_змінна
        INTO список_змінних
```

В цьому прикладі FETCH повертає поточний рядок.

*Таблиця 5.22. Ключові слова FETCH, які задають рух курсора*

| *Команда* | *Дія* |
|---|---|
| FETCH FIRST | Перейти на перший рядок |
| FETCH LAST | Перейти на останній рядок |

| | |
|---|---|
| FETCH ABSOLUTE n | ABSOLUTE n задає рядок, що відстоїть на n рядків від початку (якщо n додатнє) або від кінця (якщо n від'ємне) множини записів курсору. |
| FETCH NEXT | Перейти на наступний рядок |
| FETCH PRIOR | Перейти на попередній рядок |
| FETCH RELATIVE n | Може задавати рядки, попередні до поточного, якщо n від'ємне, і рядки, наступні за поточним, якщо n додатнє. |

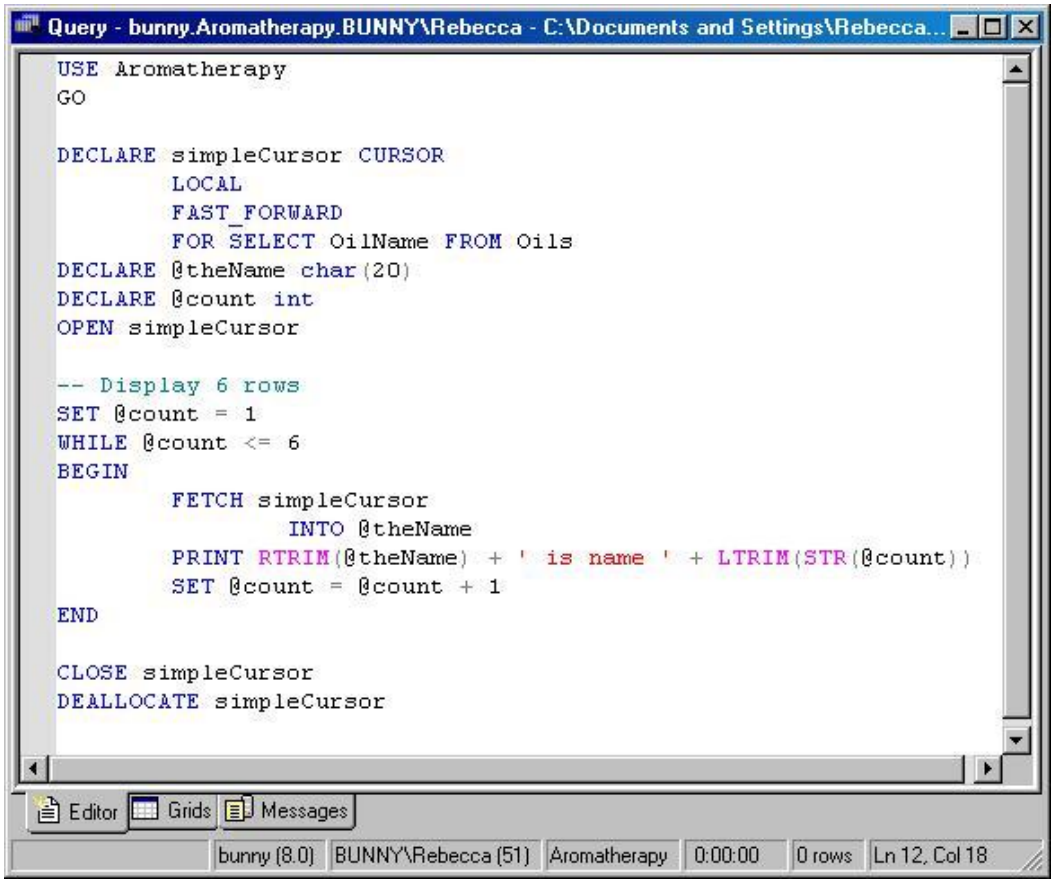


Рисунок 8 - Читання 6 рядків з таблиці за допомогою курсору.

Якщо курсор має тип FORWARD_ONLY або FAST_FORWARD, для вказання позиції може бути використано тільки ключове слово NEXT. SQL Server вважає, що кожен оператор FETCH фактично являє собою оператор FETCH NEXT.

## МОДИФІКАЦІЯ І ВИДАЛЕННЯ РЯДКІВ ЧЕРЕЗ КУРСОРИ

Якщо курсор є таким, що підлягає модифікації, зміна вихідних значень у множині курсору виконується досить просто. Передбачена спеціальна форма речення WHERE, яка підтримує модифікацію через курсор:

```
UPDATE таблиця_або_розріз
SET список_для_модификації
WHERE CURRENT OF курсор_або_змінна
```

Це називається позиційним оновленням.

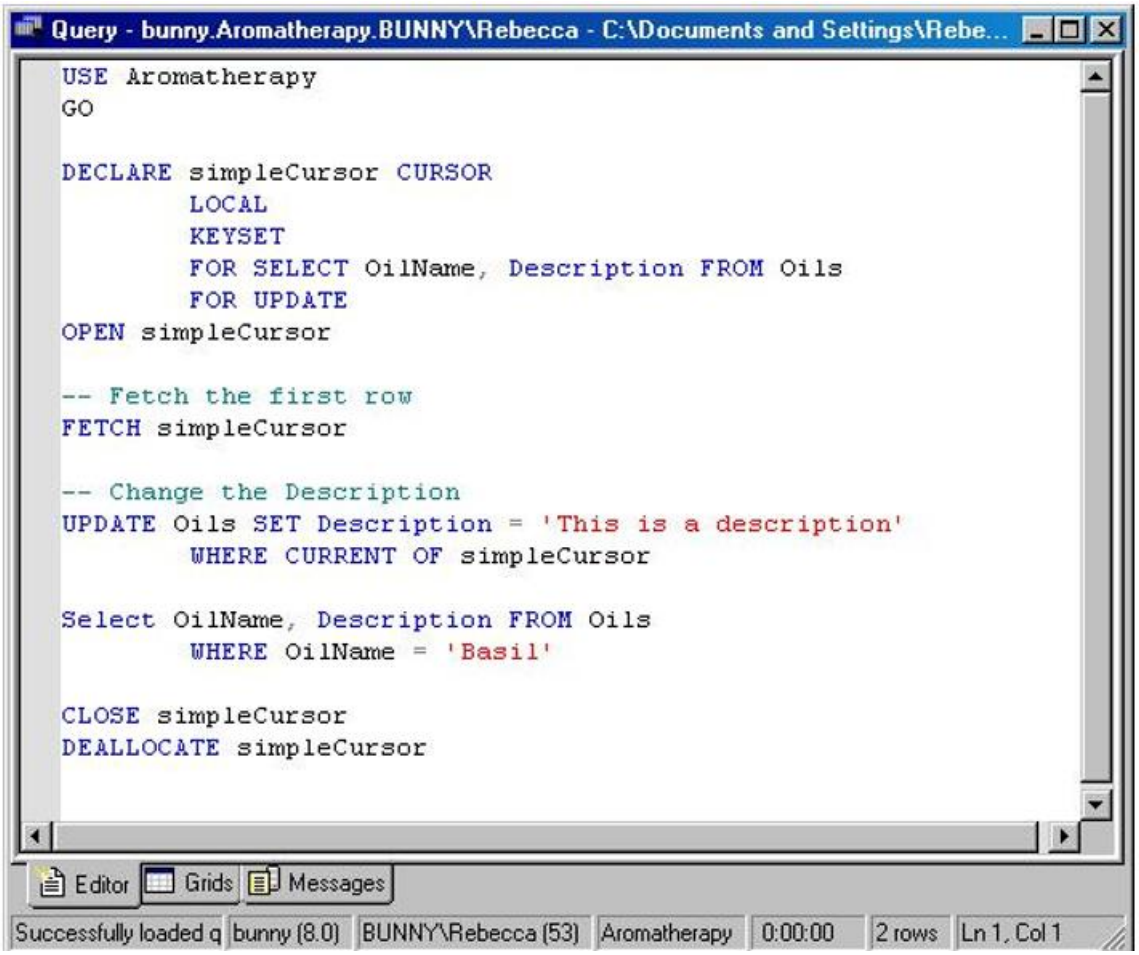


Рисунок 9 - Позиційне оновлення за допомогою курсору

```
1  Зміна контексту бази даних на БД Aromatherapy
2  Оголошення курсору для оновлення
3  Відчинення курсору
4  Читання першого кортежу
5  Позиційне оновлення поля Description
6  Вивід одного кортежу у вікно Grids
7  Закриття курсору та звільнення поточного набору результатів і зняття
   будь-яких виконаних ним блокувань. Структури даних залишаються
   доступні для відновлення, але FETCH і позиційне оновлення не
   допускається, доки курсор не відновлено.
8  Видалення посилання на курсор. Коли останнє посилання на курсор
   видаляється, структура даних видаляється з пам'яті SQL SERVER.
```

Transact-SQL також підтримує позиційне видалення, яке має таку форму запису:

```
DELETE таблиця_або_подання
WHERE CURRENT OF курсор_або_змінна
```

## МОНІТОРИНГ КУРСОРІВ

Дві системні змінні дозволяють контролювати стан курсора.

*Таблиця 5.23. Змінні для моніторингу курсора*

| *Змінна або її значення* | *Значення* | *Зміст* |
|---|---|---|
| @@CURSOR_ROWS | | Кількість рядків у множині останнього курсору, відкритого в з'єднанні (для динамічного курсору -1, оскільки кількість рядків може змінюватись |
| @@FETCH_STATUS | | Повертає інформацію про виконання останньої команди FETCH. |
| | 0 | Оператор FETCH був виконаний успішно. |
| | -1 | Оператор FETCH був виконаний неуспішно. |
| | -2 | Запитаний рядок відсутній |

*Приклад 5.51.*Обробка проводок під час проведення первинного документу (фрагмент)

```
DECLARE cur_Pd_Trans CURSOR FOR
SELECT sTrans.Id,
sTrans.TransTemId,

sTrans.DAccId, sTrans.DOrgId, sTrans.DStatId,
sTrans.DSubdivisionId, sTrans.DSubjectId, sTrans.DProcessId, sTrans.DObjectId,sTrans.DAnalAccId,

sTrans.CAccId, sTrans.COrgId, sTrans.CStatId,
sTrans.CSubdivisionId, sTrans.CSubjectId, sTrans.CProcessId, sTrans.CObjectId, sTrans.CAnalAccId,

kAccPlan_Debit.IsOrg AS DAccIsOrg, kAccPlan_Debit.IsStat AS DAccIsStat,
kAccPlan_Debit.IsSubdivision AS DAccIsSubdivision,
kAccPlan_Debit.ClassSubject AS DAccClassSubject, kAccPlan_Debit.ClassProcess AS DAccClassProcess,
kAccPlan_Debit.ClassObject AS DAccClassObject,
kAccPlan_Debit.RelToBal AS DRelToBal, kAccPlan_Debit.Acts AS DActs,
kAccPlan_Debit.LessZerroRemAction AS DLessZerroRemAction,

kAccPlan_Credit.IsOrg AS CAccIsOrg, kAccPlan_Credit.IsStat AS CAccIsStat,
kAccPlan_Credit.IsSubdivision AS CAccIsSubdivision,
kAccPlan_Credit.ClassSubject AS CAccClassSubject, kAccPlan_Credit.ClassProcess AS CAccClassProcess,
kAccPlan_Credit.ClassObject AS CAccClassObject,
kAccPlan_Credit.RelToBal AS CRelToBal, kAccPlan_Credit.Acts AS CActs,
kAccPlan_Credit.LessZerroRemAction AS CLessZerroRemAction,

kAccClass.IsPlan, kAccClass.RoundingOff,
kAccClass.LessZerroRemAction,
sTrans.Summa

FROM kAccClass INNER JOIN ((sTrans LEFT JOIN kAccPlan AS kAccPlan_Debit
ON sTrans.DAccId = kAccPlan_Debit.Id) LEFT JOIN kAccPlan AS kAccPlan_Credit
ON sTrans.CAccId = kAccPlan_Credit.Id) ON kAccClass.Class = sTrans.AccClass
WHERE PdId = @PdId
FOR READ ONLY

SET NOCOUNT ON

SELECT @ReturnCode = 0
SELECT @NegativAnalAcc = ''

BEGIN TRAN Book_Pd

OPEN cur_Pd_Trans

FETCH cur_Pd_Trans
```

Рисунок 10 - Фрагмент збереженої процедури Transact SQL: організація курсора *cur_Pd_Trans*

```
SET NOCOUNT ON          Stops the message indicating the number of rows
                        affected by a Transact-SQL statement from being
                        returned as part of the results.
BEGIN TRAN Book_Pd      Оголошення початку транзакції Book_Pd
```

```
FETCH cur_Pd_Trans
INTO            @TransId, @TransTemId,
                @DAccId, @DOrgId, @DStatId, @DSubdivisionId, @DSubjectId,
                @DProcessId, @DObjectId, @DAnalAccId,
                @CAccId, @COrgId, @CStatId, @CSubdivisionId, @CSubjectId,
                @CProcessId, @CObjectId, @CAnalAccId,
                @DAccIsOrg, @DAccIsStat, @DAccIsSubdivision, @DAccClassSubject,
                @DAccClassProcess, @DAccClassObject,
                @DRelToBal, @DActs, @DLessZerroRemAction,
                @CAccIsOrg, @CAccIsStat, @CAccIsSubdivision, @CAccClassSubject,
                @CAccClassProcess, @CAccClassObject,
                @CRelToBal, @CActs, @CLessZerroRemAction,
                @IsPlan, @RoundingOff, @LessZerroRemAction,
                @TransSumma

WHILE @@Fetch_Status = 0
BEGIN
        IF (@lBook=0)   SELECT @TransSumma = -@TransSumma
        IF @LessZerroRemAction IS NULL SET @LessZerroRemAction = 0

        --Визначимо призначення шаблону транзакції
```

Рисунок 11 - Продовження фрагменту збереженої процедури Transact SQL: організація циклу по курсору *cur_Pd_Trans*

Далі (опущено) в фрагменті містяться: зміна сальдо аналітичних регістрів по дебету і кредиту, обробка ситуацій від'ємних сальдо. Завершення циклу по курсору:

```
                            GOTO err_Handler
                            END
                    END
            UPDATE sTrans SET CAnalAccId = CASE @lBook
                                    WHEN 1 THEN @AnalAccId
                                    WHEN 0 THEN NULL
                                    END
                                    WHERE Id = @TransId
        END
        -- следующая проводка
        FETCH cur_Pd_Trans
        INTO    @TransId, @TransTemId,
                @DAccId, @DOrgId, @DStatId, @DSubdivisionId, @DSubjectId, @DProcessId, @DObjectId,@DA
                @CAccId, @COrgId, @CStatId, @CSubdivisionId, @CSubjectId, @CProcessId, @CObjectId,@CA
                @DAccIsOrg, @DAccIsStat, @DAccIsSubdivision, @DAccClassSubject,
                @DAccClassProcess, @DAccClassObject, @DRelToBal, @DActs, @DLessZerroRemAction,
                @CAccIsOrg, @CAccIsStat, @CAccIsSubdivision, @CAccClassSubject,
                @CAccClassProcess, @CAccClassObject, @CRelToBal, @CActs, @CLessZerroRemAction,
                @IsPlan, @RoundingOff, @LessZerroRemAction,
                @TransSumma
END


UPDATE sPd SET dOper = CASE @lBook
        WHEN 1 THEN @dateOfBooking
        WHEN 0 THEN NULL
        END,
        dOperOrig = CASE @lBook
        WHEN 0 THEN dOperOrig
        WHEN 1 THEN NULL
        END

WHERE Id = @PdId
```

Рисунок 12 - Фрагмент збереженої процедури Transact SQL: завершення циклу по курсору *cur_Pd_Trans*

| | |
|---|---|
| `GOTO err_Handler` | `Передача управління на обробник помилок err_Handler збереженої процедури` |

```
CLOSE cur_Pd_Trans

EXEC REG_TECHSERVISEDELEMPTYREG
COMMIT TRAN Book_Pd

DEALLOCATE cur_Pd_Trans
--Print 'reg_BookPd: Primary Document was (un)booked. '
RETURN @ReturnCode

err_Handler:
        CLOSE cur_Pd_Trans
        ROLLBACK TRAN Book_Pd
        DEALLOCATE cur_Pd_Trans
        --Print 'reg_BookPd: Error while booking Pd'
        RETURN @ReturnCode

GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO
```

Рисунок 13 - Фрагмент збереженої процедури Transact SQL: завершення процедури і курсору *cur_Pd_Trans*, обробка помилок

| | |
|---|---|
| EXEC REG_TECHSERVISEDELEMPTYREG | Виклик збереженої процедури REG_TECHSERVISEDELEMPTYREG |
| --Print 'reg_BookPd: Primary Document was (un)booked. ' | Відключений вивід відлагоджувального друку у грід |
| RETURN @ReturnCode | Повернення у головну програму зі значенням @ReturnCode |
| err_Handler: | Мітка початку обробника помилок err_Handler збереженої процедури |
| ROLLBACK TRAN Book_Pd | Відкат транзакції Book_Pd |
| GO | Закінчення пакету команд Transact SQL |
| SET QUOTED_IDENTIFIER OFF | Identifiers cannot be quoted and must follow all Transact-SQL rules for identifiers.<br>Literals can be delimited by either single or double quotation marks.<br>OFF:Ідентифікатори не можуть бути в лапках і повинні відповідати всім правилам операторів Transact-SQL для ідентифікаторів. Або ON:<br>Літерали можна обмежувати одинарними або подвійними лапками. |
| SET ANSI_NULLS ON | When SET ANSI_NULLS is ON, all comparisons against a null value evaluate to UNKNOWN. When SET ANSI_NULLS is OFF, the Equals (=) and Not Equal To (<>) comparison operators do not follow the SQL-92 standard. A SELECT statement using WHERE column_name = NULL returns the rows with null values in column_name. A SELECT statement using WHERE column_name <> NULL returns the rows with nonnull values in the column. In addition, a SELECT statement using WHERE column_name <> XYZ_value returns all rows that are not XYZ value and that are not NULL. |