**Лабораторна робота 10.**

**Робота із результуючим набором через курсори.**

**Мета роботи:** познайомиться із типом об’єкту курсор. Навчитись об’являти та використовувати курсори для вибірки та зміни даних.

**Задачі:**

* Поняття курсору, типи курсорів та їх особливості;
* Етап об’явлення курсору
* Переміщення по курсору та маніпуляції із даними
* Використання курсорів для зміни даних
* Виконання індивідуального завдання згідно варіанту

1. Інструкції Microsoft SQL Server створюють повний результуючий набір, але бувають випадки, коли результати зручніше обробляти через підрядник. Відкриття курсору на результуючому наборі уможливлює його порядкову обробку. Нижче наведемо основні частини загального синтаксису створення курсору (рис.1) та коментарі до кожної із них (є 2 синтаксиса команди створення курсору ISO Syntax та Transact-SQL Extended Syntax. Ми розглянемо останній)

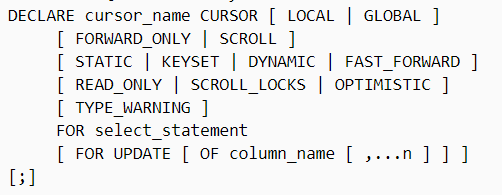


Рис.1. Загальний синтаксис створення курсору

[LOCAL | GLOBAL] – локальний курсор існує у межах поточного пакету команд, глобальний у межах з’єднання

[ FORWARD\_ONLY | SCROLL ] - визначає спосіб переміщення по курсору – або тільки вперед із кроком 1, або як завгодно

[ STATIC | KEYSET | DYNAMIC | FAST\_FORWARD ] – тип курсору STATIC – статична копія, не відслідковує зміни даних та не може бути використаний для зміни; DYNAMIC – вибирає дані із БД динамічно по одному рядку у момент переміщення; FAST\_FORWARD – динамічний forward\_only курсор, є найбільш продуктивним; KEYSET – копіює унікальний ключ, а дані читає на зразок динамічного.

[ READ\_ONLY | SCROLL\_LOCKS | OPTIMISTIC ] - визначає тип блокування даних при переміщенні по курсору

FOR select\_statement - обов’язкова частина, визначає SQL-запит на вибірку даних, з яким буде пов’язано курсор

[ FOR UPDATE [ OF column\_name [ ,...n ] ] ] – набір колонок результуючого набору, що може бути змінено через курсор

Протягом роботи із курсором, можна визначити 5 основних етапів:

* Створення курсору – команда declare (рис.1). На даному етапі ми створюємо курсор, як об’єкт та задаємо всі його властивості.
* Відкриття курсору – команда open. На цьому етапі курсор пов’язується із даними, для статичних та ключових курсорів відбувається копіювання даних
* Переміщення по курсору – fetch, переміщення вказівника, читання та зміна даних
* Закриття курсора – close, розірвання зв’язку із даними
* Вивільнення курсору – deallocate, знищує курсор, як об’єкт

1. Далі наведемо декілька прикладів створення курсору. Перший приклад створить курсор, що вибиратиме номери студентських груп (рис.2)



Рис.2. Створення курсору без додаткових опцій

Ми не вказали жодних додаткових опцій, отже створиться глобальний оптимістичний fast-forward курсор. Створимо курсор із таким же набором даних, але прокручуємий, локальний та динамічний (рис.3)



Рис.3. Створення прокручуємого, локального, динамічного курсору

Дана команда може бути виконана скільки завгодно разів, оскільки локальний курсор існує лише у межах пакету команд та буде автоматично знищений після його завершення. Якщо ж ми спробуємо 2-й раз виконати команду, наведену на рис.2, то отримаємо помилку, оскільки глобальний курсор існуватиме, поки ми не закриємо з’єднання із сервером (рис.4)

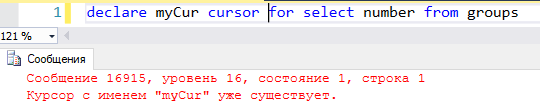


Рис.4. Помилка створення другого глобального курсора

При вказанні параметрів слід бути обережним, оскільки бувають несумісні між собою параметри. Так наприклад, спробуємо створити прокручуємий fast-forward курсор (рис.6)

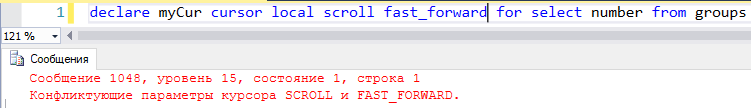


Рис.6. Конфлікт параметрів при створенні курсору

1. Далі виконаємо відкриття курсору командою open, паралельно розглянувши глобальну змінну @@cursor\_rows. Дана змінна містить кількість рядків у курсорі, що був відкритий останнім. Розглянемо приклад створення та відкриття статичного курсору, що містить студенів 305 групи із подальшим визначенням їх кількості (рис.7)

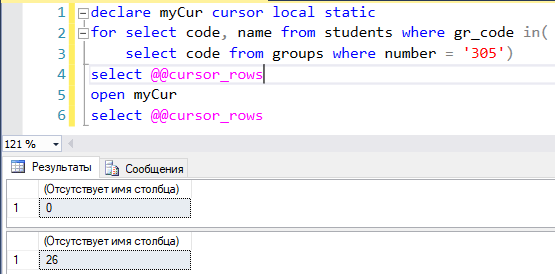


Рис.7. Використання @@cursor\_rows із статичним курсором

Як бачимо, перший раз повертається 0, оскільки немає відкритих курсорів, а 2-й раз фактично маємо кількість студентів групи 305. Зрозуміло, що визначити кількість студентів у групі можна було набагато простіше із використанням досить простого SQL-запиту select.

! УВАГА. Не слід використовувати курсори для задач, що можуть бути вирішені без їх використання, оскільки курсор у переважній більшості випадків працюватиме набагато довше, ніж одиничний SQL-запит до БД.

Далі у прикладі (рис.7) змінимо тип курсора на динамічний та переглянемо результат (рис.8).

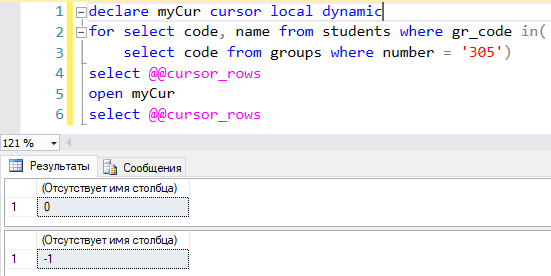


Рис.8. Використання @@cursor\_rows із динамічним курсором

Це є властивістю змінної @@cursor\_rows, що після відкриття динамічного або fast forward курсору вона змінює значення на -1. Якщо пригадати особливості їх поведінки, що полягає у тому, що вони читають дані із БД в момент переміщення по курсору, це стає зрозумілим, оскільки на етапі відкриття курсор ще «не знає», скільки він міститиме рядків.

1. Далі розглянемо команду переміщення по курсору fetch. Наведемо загальний синтаксис команди (рис.9)

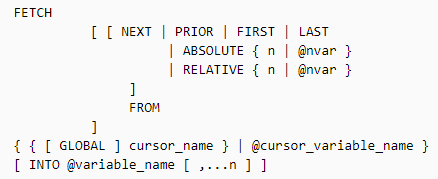


Рис.9. Синтаксис команди fetch

Першим задаємо спосіб переміщення NEXT – на один вперед, PRIOR – на один назад, FIRST – на перший рядок, LAST – на останній, ABSOLUTE – на вказаний рядок від початку (або кінця), RELATIVE – на вказану кількість рядків вперед (або назад). Далі вказуємо по якому саме курсору будемо переміщуватись (оскільки може бути багато одночасно відкритих курсорів) та за допомогою можемо прочитати рядок курсору у змінні.

Розглянемо декілька прикладів. Для початку створимо та відкриємо курсор, що вибирає студенів 308 групи, впорядкованих за прізвищем, а далі виведемо 1-го, 2-го, останнього, передостаннього, 4-го з кінця, 3-го з початку, 5-го з початку та знову 2-го, продемонструвавши при цьому всі типи переміщення (рис.10).

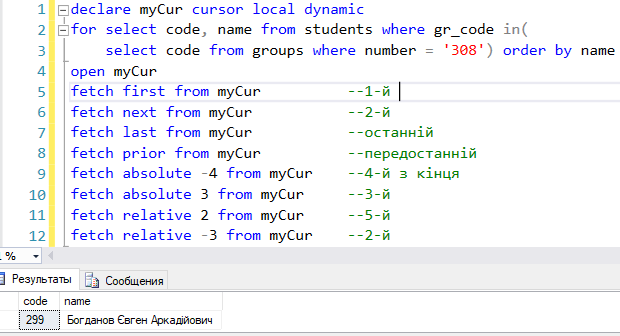


Рис.10. Різні типи переміщення по курсору

Слід нагадати, що для курсору із типом fast\_forward або forward\_only доступна лише опція next. Переконаємося у цьому на прикладі (рис.11)

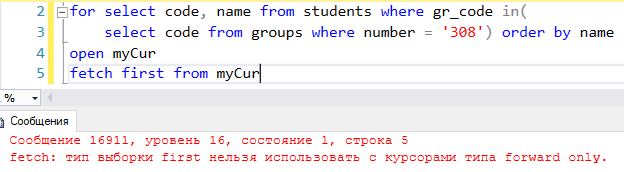


Рис.11. Неможливість використання нічного, окрім next для fast\_forward

Далі реалізуємо приклад, що визначатиме найстаршого студента 308 групи. Для цього створимо курсор статичний студентів 308 групи, за допомогою @@cursor\_rows та while обійдемо його у циклі, та визначимо студента із максимальним віком. При читанні даних із рядку курсора, поміщатимемо їх у змінні, для виконання подальшої операції порівняння (рис.12).

! УВАГА. Курсор у даному прикладі використовується лише у цілях демонстрації такої можливості, але є дуже неоптимальним шляхом вирішення такої задачі.

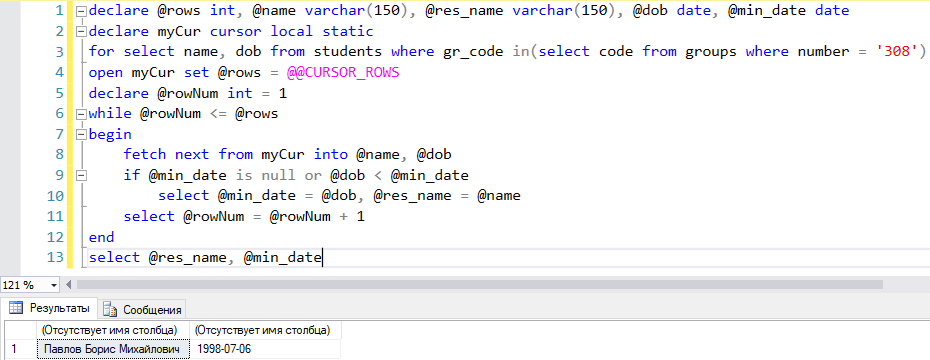


Рис.12. Поміщення результату fetch до змінних

1. Використання змінної @@cursor\_rows не завжди можливе, оскільки для деяких типів курсорів може бути невідомою кількість їх рядків. Тому існує інший шлях – перевірка статусу виконання останньої команди fetch, що зберігається у глобальній змінній @@fetch\_status. Змінна містить 0, якщо переміщення відбулося успішно і -1, якщо вказівник за межами курсору. Реалізуємо попередній приклад отримання найстаршого студента для динамічного або fast\_forward курсору (рис.13)

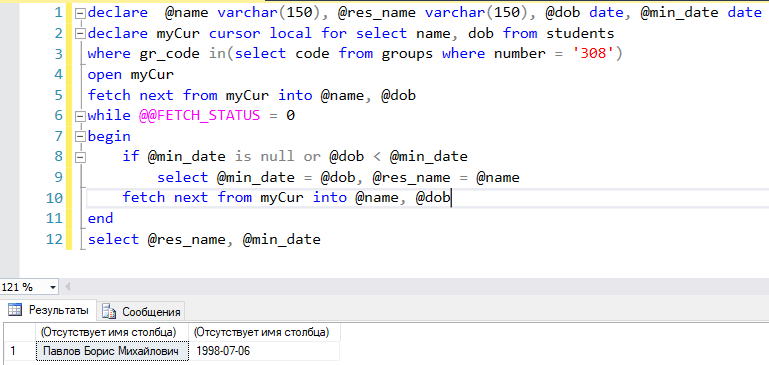


Рис.13. Використання @@fetch\_status для циклічного обходу курсору

1. Після того, як курсор став вже непотрібним, дуже бажано його відразу закрити та вивільнити, щоб звільнити ресурси що були під нього виділені та зняти блокування, якщо такі ще залишаються. Робиться це командами close та deallocate відповідно (рис.14).

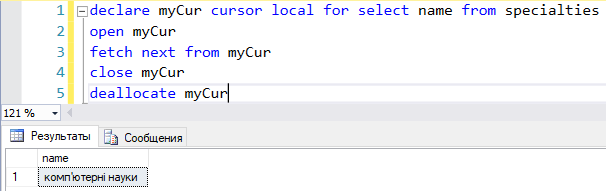


Рис.14. 5 етапів роботи із курсором

1. Оскільки тепер ми володіємо всіма необхідними для цього знаннями, далі розглянемо більш наближений до життєвої ситуації приклад використання курсорів. У табличці groups є поле students\_count, що має містити значення кількості студентів у групі (дане поле є відступленням від 3НФ, але може пришвидшити виконання деяких запитів у майбутньому, тому може мати місце). На даний момент воно не заповнене, але виправимо цю ситуацію – заповнимо це поле відповідно до кількості рядків у табличці students для кожної із студентських груп. Використаємо для цього курсор (рис.15).

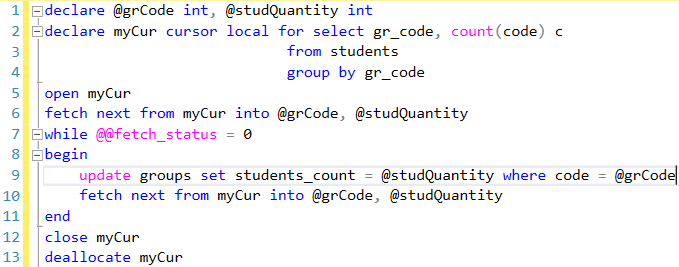


Рис.15. Приклад використання курсорів для підрахунку та збереження кількості студентів по групах

Даний код не повертає жодного результату, але після його виконання можна пересвідчитись, що кількість студентів збережена у табличці groups, виконавши наступну вибірку (рис.16)

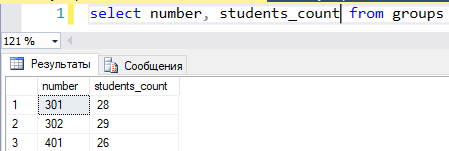


Рис.16. Перевірка зміни поля students\_count таблички groups

1. У попередньому прикладі для зміни таблички студентів була використана звичайна нотація команди update із фільтром where по значенню первинного ключа. Але курсор уже має вказівник на рядок результуючого набору, тому вже сам по собі може бути використаний для зміни даних. Робиться це за допомогою конструкції «current of». Реалізуємо в такий спосіб попередню задачу зміни поля students\_count таблички groups, попередньо скинувши поле students\_count = 0, щоб пересвідчитися, що код працює вірно (рис.17)

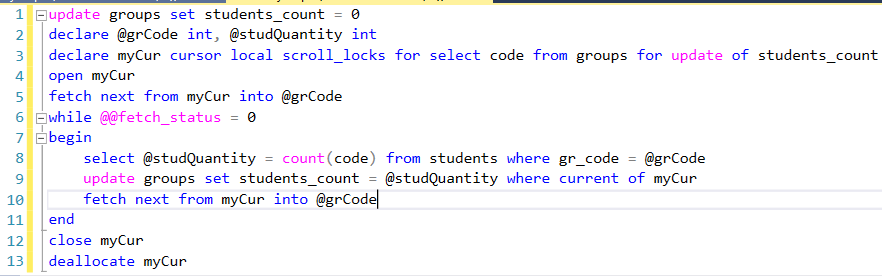


Рис.17. Реалізація підрахунку та збереження кількості студентів по групах (спосіб 2)

1. Незважаючи на те, що розглянута задача є не зовсім примітивною, вона все ще залишається досить простою та засобами T-SQL може (а отже і повинна) бути реалізована без використання курсору. Наведемо реалізацію даної задачі із використанням корельованого підзапиту у команді update (рис.18). Щоб знову ж таки переконатись у вірності роботи нашого коду, на початку скинемо всі students\_count = 0, а наприкінці виведемо групи на екран.

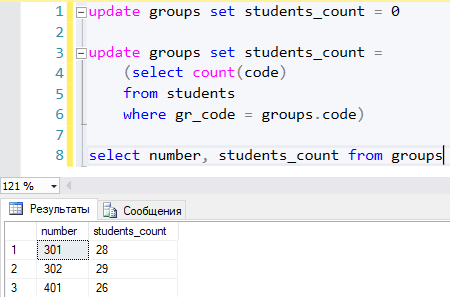


Рис.18. Зміна даних із використанням корельованих запитів

1. Кінець роботи

**Завдання для індивідуального виконання.**

Згідно свого варіанту додати числове поле до таблиці власної БД та заповнити його трьома способами:

* Із використанням курсору та команди update із фільтром по первинному ключу
* Із використанням курсору та конструкції current of
* Без використання курсору за допомогою корельованого підзапиту у команді update

**Варіанти для виконання індивідуальних завдань.**

1. БД Автостоянка, таблиця Автостоянки, поле кількість автомобілів.
2. БД Облік кадрів, таблиця відділи, поле кількість співробітників.
3. БД Облік книжок, таблиця видавництва, поле кількість видань (книжок)
4. БД Лікарські рецепти. Таблиця лікарі, поле кількість виписаних рецептів
5. БД Нерухомість. Таблиця власники, поле кількість квартир
6. БД Облік комунального транспорту. Таблиця типи транспорту (трамвай, тролейбус, … ), поле кількість транспортних засобів
7. БД Кінопрокат. Таблиця Кінофільми поле кількість сеансів
8. БД Ремонт телефонів. Таблиця Моделі поле кількість пристроїв
9. БД Облік тварин ветеринарної клініки. Таблиця Види тварини (кішка, собака, папуга, … ) поле кількість тварин
10. БД Розклад занять. Таблиця Студентські групи, поле кількість занять
11. БД Облік приміщень та обладнання університету. Таблиця Корпуси поле кількість приміщень
12. БД Туристичної компанії. Таблиця Тури поле кількість замовлень
13. БД Регістратура поліклініки. Таблиця Пацієнти поле кількість звернень (відвідувань)
14. БД Оренда транспортних засобів. Таблиця Транспортні засоби поле кількість разів в оренді
15. БД Оренди приміщень. Таблиця Орендар поле кількість приміщень в оренді (без урахування дати та терміну)