**Лабораторна робота 6.**

**Спеціальні оператори T-SQL при вибірці даних.**

**Мета роботи:** ознайомитись із додатковими можливостями вибірки даних в T-SQL із викоистанням віконних функцій та операторів APPLY, PIVOT, UNPIVOT, GROUPING SETS, ROLLUP та CUBE.

**Задачі:**

* Оператор CROSS APPLY та OUTER APPLY
* Робота з віконними функціями. Ранжуємі функції, функції зі зміщенням, агрегатні функції,
* Розгортання та згортання даних (PIVOT та UNPIVOT).
* Групуючі набори із використанням інструкцій ROLLUP та CUBE

1. Підключаємось до сервера баз даних SQL Server відкриваємо БД «StudyBase».
2. Познайомимось із оператором APPLY. Від приймає дві таблиці (праву та ліву), при чому лівою як привило буде табличний вираз або функція, що повертає табличне значення (даний випадок буде розглянуто при роботі із функціями). На перший погляд оператор CROSS APPLY дуже схожий на CROSS JOIN (декартовий добуток) та у випадку «простих» запитів дає еквівалентний результат, як на прикладі використання у вигляді правої та лівої таблиць specialties та departments (рис.1-2)

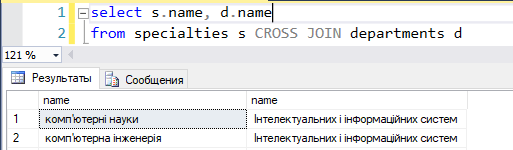


Рис.1. specialties CROSS JOIN departments

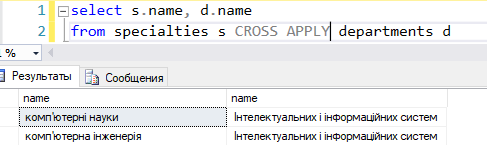


Рис.2. specialties CROSS APPLY departments

CROSS JOIN, як ми пам’ятаємо, повертає множину рядків правої таблиці для кожного рідку лівою таблиці. Як бачимо на прикладі рис.1-2, CROSS APPLY веде себе так само, але на відміну від попереднього, він може бути налаштований таким чином, що біде повертати для кожного рядку лівою таблиці різні набори даних.

Даний ефект може бути отримано за рахунок передачі аргументів лівої таблиці як вхідних параметрів для вкладеного запиту або функції користувача (буде розглянуто пізніше), що виступає у ролі правої таблиці. Переробимо попередній приклад таким чином, щоб права частина повертала назву кафедри для кожної спеціальності (лівої частини), що викладає її студентам найбільшу кількість дисциплін (рис.3)

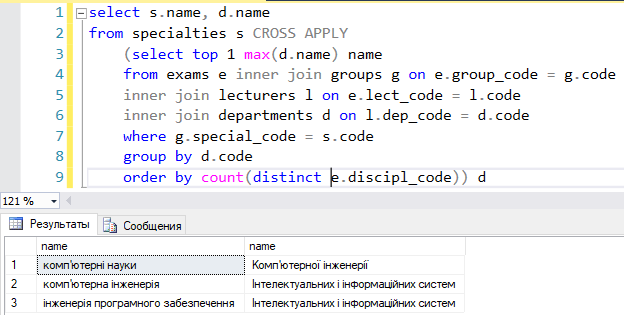


Рис.3. Кафедри, що викладають найбільше дисциплін для спеціальності

Розглянемо іще один приклад. Виберемо для кожної групи перших 2-х студентів за рейтингом (рис.4)

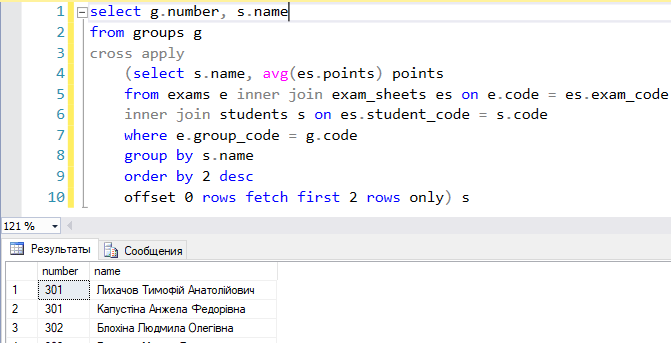


Рис.4. Перші 2 студенти за рейтингом для кожної групи

Оператор CROSS APPLY працює по принципу inner join та не поверне рядків лівої таблиці, для яких немає відповідності у правій. Якщо ми хочемо мати всі рядки лівої таблиці у будь-якому випадку, можна використати його аналог OUTER APPLY. Для рядків, що не матимуть відповідності у правій таблиці, поля правої таблиці будуть заповнені null-ами. Наприклад, виберемо найкращого студента для кожного викладача (рис.5)

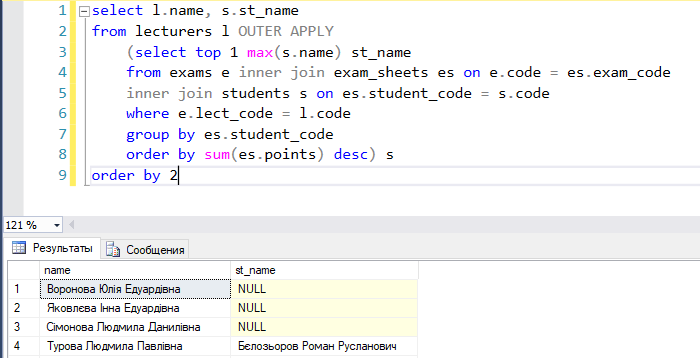


Рис.5. Найкращий студент викладача. OUTER APPLY.

Як бачимо, троє перших викладача ще не приймали іспитів та не мають найкращого студента, але присутні у вибірці завдяки OUTER APPLY.

1. Далі переходимо до розгляду віконних функцій, що дозволяють обчислити скалярний вираз на основі деякої підмножини рядків (вікна). За визначенням може здатися, що віконна функція не що інше, як функція агрегації. Але основна відмінність у тому, що для функції агрегації діапазони підмножини визначаються за допомогою «group by» для всього запиту в цілому, а у випадку віконної функції «вікно» визначається за допомогою інструкції «over» окремо для кожного поля. Розглянемо перший приклад, що реалізує вибірку історії зміни середнього балу студента (рис.6)

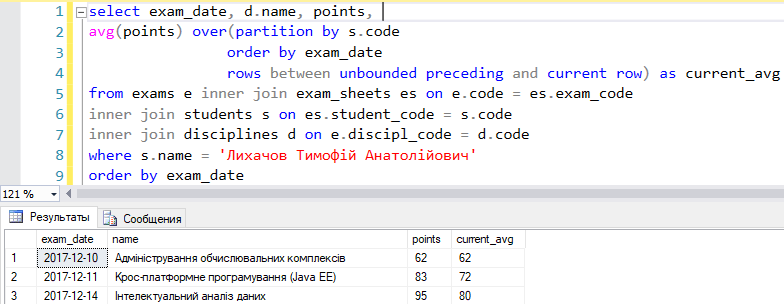


Рис.6. Історія зміни середнього балу студента

Інструкція «over» складається з 3-х основних частин – секціонування, упорядкування та рамок. У випадку порожніх дужок over() робота буде вестись з усім результуючим набором. Інструкцією для секціонування є «partition by», впорядкування «order by» (не впливає на сортування рядків результуючого набору), а рамки ROWS BETWEEN <верхня межа> AND <нижня межа >. В нашому прикладі UNBOUNDED PRECEDING – перший рядок секції, а CURRENT ROW – поточний.

Оскільки віконна функція працює вже з результуючим набором, вона може бути визначена лише у реченнях select та order by.

Далі розглянемо ще декілька прикладів. Почнемо із використання ранжувальних функцій row\_number() та dense\_rank(). Виведемо список студентів, впорядкований за прізвищем, де буде поле місце студента за рейтингом у своїй групі. Місце порахуємо двома способами – за першим студенти, що мають однаковий середній бал займають місця один за одним, а за другим – ділять одне місце (рис.7)

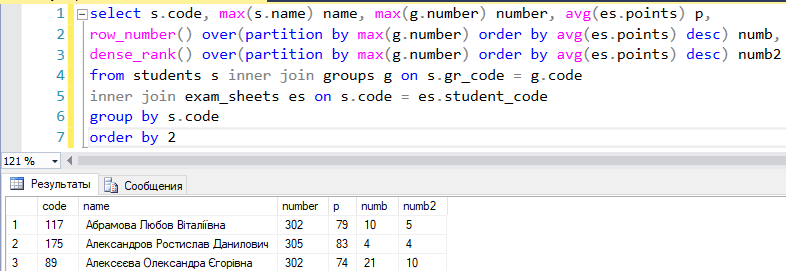


Рис.7. Список студентів із місцем за рейтингом.

Також розглянемо функції зі зміщенням LEAD та LAG, що виводять попереднє та наступне значення вказаного у якості першого параметру поля. В якості прикладу розглянемо вибірку результатів певного студента із зазначенням балів, які студент отримав з попередньої та наступною (за датою складання) дисциплінами у межах календарного року (рис.8)

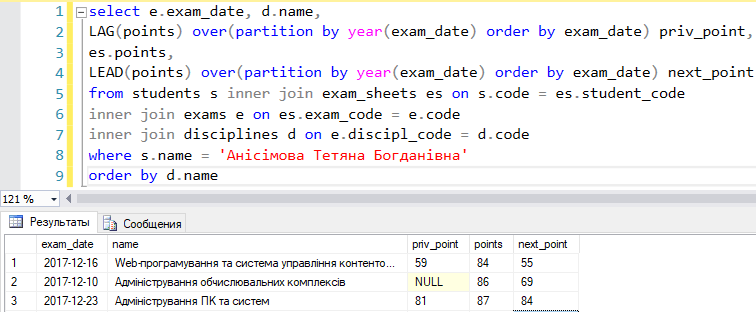


Рис.8. Використання функцій зі зміщенням LEAD та LAG

Один приклад роботи із агрегатними віконними функціями вже був наведений на рис.6. Наведемо іще один приклад, де виведемо студентські групи по спеціальностях із накопичувальною сумою кількості студентів на спеціальності (рис.9)

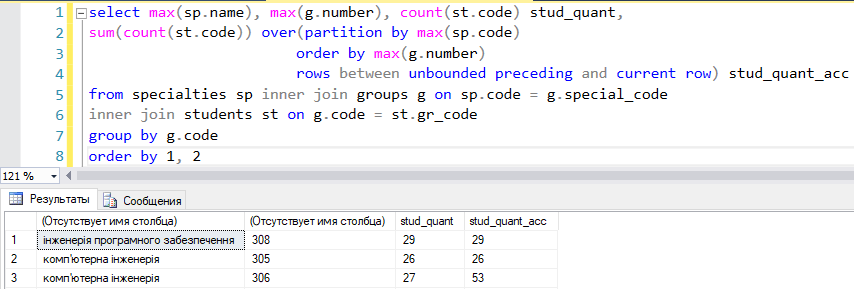


Рис.9. Приклад використання агрегатної віконної функції

1. Наступний розділ – розгортання та згортання даних. Розгортання – це перетворення рядків у стовбці із можливою попутною агрегацією стовбців. Мова T-SQL підтримує нестандартний оператор PIVOT, що виконує групування, розподілення та агрегацію даних. В цілому запит із оператором PIVOT виглядає наступним чином:

SELECT ...

FROM <таблиця\_або\_табличне\_вираження>

PIVOT(<агрегатна\_функція>(<агрегатний\_елемент>)

FOR <разподіляючий >

IN (<цільові стовбці>)) AS <псевдонім\_ітогової\_таблиці>

...;

Припустимо у нас є запит, що повертає бали, які отримали студенти груп 3-го курсу на іспитах (рис. 10)

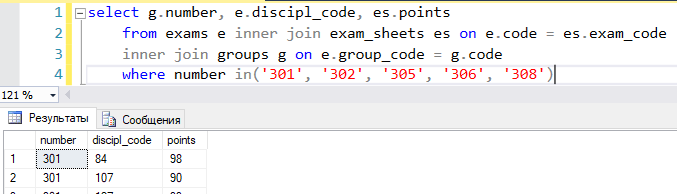


Рис.10. Проміжний запит для звідної таблиці

Далі для зручності порівняння, ми хочемо представити ці дані у вигляді звідної таблиці, де у рядках будуть дисципліни, а у колонках – групи, що їх здавали. Запит на отримання даних такого вигляду наведено на рис. 11.

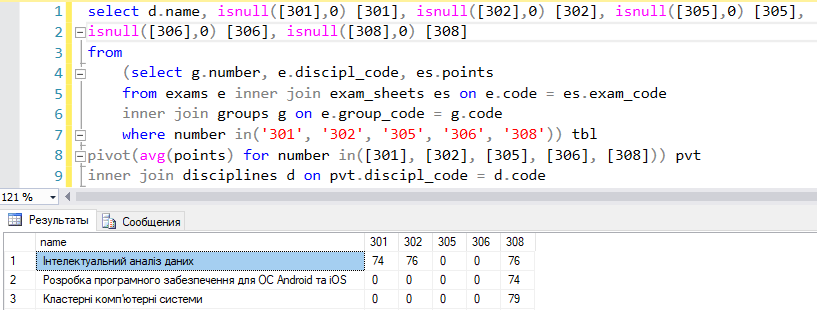


Рис.11. Звідна таблиця середньої успішності студентів по групах

Зворотній оператор UNPIVOT, що виконує згортання даних, має наступний синтаксис:

SELECT ...

FROM < таблиця\_або\_табличне\_вираження >

UNPIVOT(<цільовий\_стовбець\_для\_збереження\_вхідних\_значень>

FOR < цільовий\_стовбець\_для\_збереження \_імен\_ вхідних \_стовбців>

IN (<вхідні\_стовбці>)) AS <псевдонім\_вхідної\_таблиці>

...;

Приклади використання з міркувань економії часу в рамках даного курсу не розглядаються.

1. І, нарешті, розглянемо групуючі набори та інструкції ROLLUP та CUBE.

Групуючий набір – це список атрибутів, за яким відбувається групування. При виконанні агрегатного запиту можна визначити тільки один групуючий набір. Наприклад ми можемо отримати середній бал студентів по групах, або по дисциплінах, або по групах та дисциплінах. Якщо потрібно об’єднати всі три результати в одному наборі, доведеться викорати щось на зразок запиту, наведеного на рис.12.

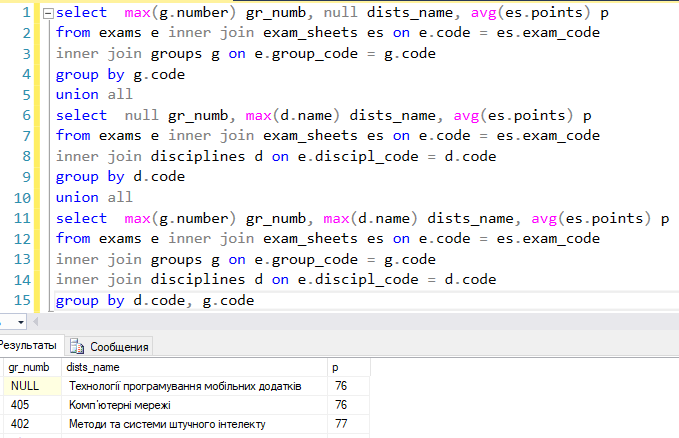


Рис.12. Поєднання різних групуючих наборів в одному запиті

Даний підхід є очевидно громіздким з точки зору кількості зайвого коду, не дуже читабельним та не зовсім оптимальним з точки зору виконання запиту.

Розглянемо вбудовані механізми T-SQL, що дозволяють вирішувати дану задачу більш легко та оптимально. Для початку, реалізуємо наведений вище приклад із використанням інструкції GROPING SETS (рис.13)

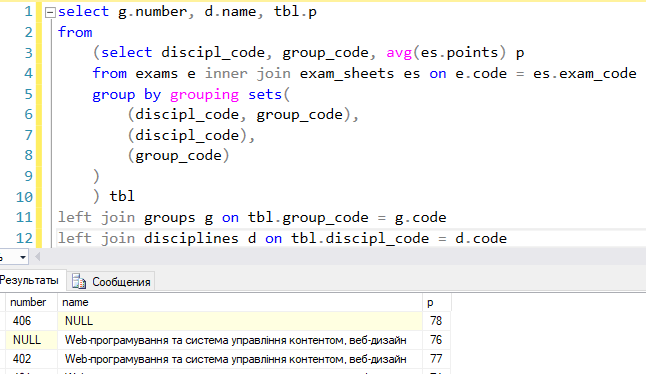


Рис.13. Приклад використання GROPING SETS

Оператор CUBE працює аналогічним чином, але виконує групування по всім можливим комбінаціям групуючих наборів. Тобто, якщо у попередній приклад додати порожній набір (), що виконає підрахунок загального середнього балу студента, то отримаємо результат, аналогічний, як і при використанні оператора CUBE (рис.14)

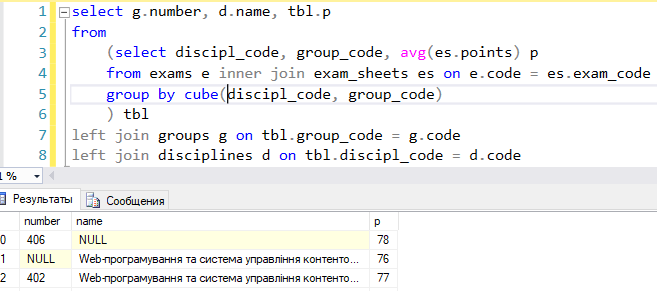


Рис.14. Використання оператора CUBE

Інструкція ROLLUP працює схожим чином із CUBE, але надає не всі можливі варіанти комбінацій наборів, а лише певну їх підмножину. Так, наприклад ROLLUP(a, b, c, d) буде ідентичним GROUPING SETS((a, b, c, d), (a, b, c), (a, b), (a), )()). Використання ROLLUP замість CUBE у попередньому прикладі дасть результат, ідентичний GROUPING SETS((discipl\_code, group\_code), (discipl\_code), ()) (рис.15)

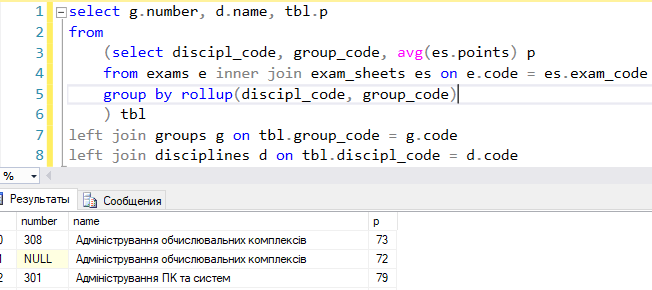


Рис.15. Використання оператора ROLLUP

1. Кінець роботи

**Завдання для індивідуального виконання.**

Виконати запити до власної БД (придумати самостійно), використовуючи наступні засоби (за необхідністю доповнити власну БД таблицями та полями за попереднім погодженням із викладачем) :

* CROSS APPLY або OUTER APPLY
* На вибір по одній ранжуємій, із змішенням та агрегатній віконній функції (всього три)
* Оператор PIVOT
* ROLLUP або CUBE, а аналог запиту із використанням GROUPING SETS

(при здачі має бути наведено текст запитів, результат та словесний опис, які саме дані вибирає запит)

! УВАГА. Запит не обов’язково має повертати дані. Оцінюється правильність тексту запиту, а не наповненість БД даними (хоча мінімальні вимоги щодо наповненості БД із попередньої роботи все ж мають бути виконані)