Лабораторна робота 8. Операції з підзапитами в SQL

Мета:

Навчити студентів використовувати підзапити (subqueries) в SQL для створення складних запитів, застосовуючи різні типи підзапитів у різних частинах SQL-запитів.

Завдання:

1. Реалізувати підзапити в секції FROM.
2. Реалізувати підзапити в секції SELECT.
3. Реалізувати підзапити в секції WHERE, демонструючи різницю використання різних операторів для зв'язку підзапиту з зовнішнім зпитом.
4. Реалізувати корельовані (зв'язані) підзапити.
5. Реалізувати розбиття складного запиту на частини з використанням оператора WITH.
6. Реалізувати запити на модифікацію даних на основі результатів підзапитів.
7. Пояснити результати виконання кожного запиту.

Результат:

Студенти повинні подати SQL-скрипти, що відображають підзапити в різних секціях запиту та з використанням різних операторів згідно завдання та предметної області, їх опис, а також звіт з результатами тестування.

Теоретичні відомості до виконання лабораторної роботи

Підзапит — це SELECT-запит вкладений в інший запит або підзапит.

Підзапити використовуються в SQL для створення вкладених запитів, які дозволяють виконувати операції з даними на основі результатів інших запитів. Це дає змогу розбивати складні запити на частини, що спрощує їх розуміння та підтримку. Також існують завдання, які просто неможливо вирішити за допомогою одного звичайного запиту. Приклад такого завдання — вибірка всіх записів зі значенням більше середнього по всій таблиці. Для одного запиту не можна вибрати значення, і порахувати агрегатну функцію по всій таблиці. Щоб вирішити такі завдання, використовують підзапити.

Типи підзапитів:

1. **Підзапити в секції FROM.** Підзапити у секції FROM дозволяють використовувати результати одного запиту як тимчасову таблицю, що особливо корисно, коли потрібно виконати кілька маніпуляцій над отриманими даними.
2. **Корельовані (зв'язані) підзапити**. Це підзапити, які залежать від зовнішнього запиту і виконуються для кожного рядка зовнішнього запиту. Вони не можуть виконуватися окремо від зовнішнього запиту, оскільки використовують його контекст. Вони ефективні, коли потрібно порівнювати кожен запис із певним набором даних.
3. **Підзапити в секції SELECT.** У цьому випадку підзапит повертає одиночне значення для кожного рядка результатів основного запиту.
4. **Підзапити в секції WHERE.** Найпоширеніший тип підзапиту, який дозволяє фільтрувати результати основного запиту на основі результатів вкладеного запиту.
5. **Узагальнений табличний вираз** (оператор WITH) використовується для розбиття складних запитів на прості частини.
6. **Підзапити в запитах модифікації даних.** Підзапити можуть використовуватися в операціях INSERT, UPDATE, DELETE, дозволяючи автоматично змінювати дані на основі результатів інших запитів.

Ефективніше використовувати підзапити в ситуаціях, коли потрібно отримати результат із окремої таблиці або виконати складні умови фільтрації.

Інколи неефективно: у деяких випадках підзапити можуть бути повільними, особливо корельовані підзапити, які виконуються для кожного рядка. Альтернативою можуть бути операції JOIN або збереження результатів проміжного запиту у тимчасову таблицю.

Підзапити у секції FROM

Такі підзапити іноді називають похідними таблицями або табличними виразами, оскільки зовнішній запит використовує результати підзапиту як джерело даних.

Результату підзапиту слід призначити псевдонім. Якщо не задати псевдонім, то виникне помилка.

Приклад:

```
SELECT SecondName, FirstName, avg_score
FROM (
  SELECT SecondName, FirstName, AVG(mark) AS avg_score
  FROM Rating
  GROUP BY SecondName, FirstName
) avg_scores
WHERE avg_score > 75;
```

Пояснення: Внутрішній підзапит створює тимчасову таблицю avg_scores, що містить середні бали студентів, а зовнішній запит вибирає лише тих студентів, у яких середній бал перевищує 75.

Корельовані (зв'язані) підзапити

Більш складним варіантом підзапитів є так звані корельовані або зв'язані підзапити. Основне застосування корельованих підзапитів – це виконання операцій, пов'язаних із кожним рядком основного запиту. Корельовані підзапити виконуються для кожного рядка зовнішнього запиту та використовують значення з цього рядка для фільтрації або обчислення даних у підзапиті.

Розглянемо задачу: вивести на екран повні номери груп, в яких є студенти, що мають, наприклад, нульовий рейтинг з першої дисципліни.

Варіантом розв'язання може бути таке рішення:

```
SELECT    Spec, GNum   FROM      Student S, Rating R
WHERE R.Kod = S.Kod AND DKod = 1 AND Mark = 0;
```

Однак у результаті з'являться однакові кортежі: скільки студентів з нульовим рейтингом — стільки разів буде повторений шифр та номер групи. Це, по-перше. А по-друге, виконується з'єднання двох таблиць, що є нетривіальною задачею для сервера.

Іншим підходом до розв'язання поставленої задачі може бути реалізація алгоритму:

1. Дістати кортеж таблиці *Student*.
2. Використовуючи значення його поля *Kod*, вибрати з таблиці *Rating* усі кортежі з таким же значенням поля *Kod*.
3. Якщо серед обраних кортежів є кортежі з нульовим значенням поля *Mark* та одиничним значенням поля *DKod*, то поточний кортеж таблиці *Student* включити в результуюче відношення.
4. Повторити алгоритм для всіх інших кортежів таблиці *Student*.

Цей алгоритм легко реалізується за допомогою підзапитів (точніше зв'язаних підзапитів) і в загальному вигляді такий:

1. Вибрати рядок з таблиці, зазначеної в зовнішньому запиті. Це, так званий, поточний рядок-кандидат (у значенні на включення в результат запиту).
2. Виконати підзапит. Причому в умові вибірки кортежів з таблиці, зазначеної в підзапиті, використовуються значення рядка-кандидата.
3. Перевірити істинність умови вибірки кортежів у зовнішньому запиті, використовуючи результат виконання підзапиту.
4. Якщо умова вибірки зовнішнього запиту правдива, то рядок-кандидат включається до результату цього запиту.

Для нашого прикладу такий запит буде виглядати таким чином:

```
SELECT Spec, GNum FROM Student S WHERE 0 IN
(SELECT Mark FROM  Rating R  WHERE  R.Kod = S.Kod AND DKod = 1);
```

Зв'язаність підзапитів ніяк не обмежує глибину їхньої вкладеності. Наприклад, якщо були б потрібні не номера груп, а назви спеціальностей, на яких є студенти, що мають нульовий рейтинг з першої дисципліни, то запит, орієнтований на розв'язання поставленої задачі, виглядав би таким чином:

```
SELECT SpName     FROM  Speciality Sp   WHERE Spec IN
(SELECT Spec FROM  Student S    WHERE 0 IN
(SELECT    Mark FROM Rating R  WHERE R.Kod = S.Kod AND DKod = 1));
```

Префікс і, відповідно, аліас *R* не є обов'язковим, тому що *SQL* звертається спершу до таблиці, зазначеної у підзапиті, і, якщо в неї такого поля немає, то — до таблиці з зовнішнього підзапиту.

Підзапити можуть зв'язувати таблицю і зі своєю копією. Наприклад, **задача**: вивести список студентів, що мають рейтинг з першої дисципліни вище за середній *за своєю спеціальністю*:

```
SELECT SecondName, FirstName, Mark, S1.Spec
FROM Student S1, Rating R1
WHERE S1.Kod = R1.Kod AND DKod = 1 AND Mark >(
                        SELECT AVG(Mark)
                        FROM Rating R2
                        WHERE DKod = 1 AND Kod IN   (
                                SELECT Kod
                                FROM Student S2
                                WHERE S2.Spec = S1.Spec));
```

Тут у підзапиті самого нижнього рівня (корельованому підзапиті) формується множина кодів студентів, які мають той же самий шифр спеціальності, що і студент, описаний поточним рядком-кандидатом. Далі серед цієї множини по таблиці *Rating* підраховується середній бал. У базовому запиті цей результат порівнюється з рейтингом обраного студента знову з таблиці *Rating*. При істинності порівняння з таблиці *Student* дістається його прізвище, ім'я та шифр спеціальності, а з таблиці *Rating* — його бал.

Підзапити у секції SELECT

Підзапити у секції SELECT називають скалярним. Скалярний підзапит — це звичайний запит SELECT у дужках, який повертає рівно один рядок і один стовпець. Після виконання запиту SELECT його єдиний результат використовується в навколишньому виразі. Як скалярний підзапит не можна використовувати запити, що повертають більше одного рядка або стовпця. Але якщо в результаті виконання підзапит не поверне рядків, скалярний результат вважається рівним NULL. У підзапиті можна посилатися на змінні з навколишнього запиту; в процесі одного обчислення підзапиту вони будуть вважатися константами. Приклад:

```
SELECT SecondName, FirstName, (
        SELECT AVG(Mark)
        FROM Rating
        WHERE Rating.Kod = Student.Kod
) AS avg_score
FROM Student;
```

Пояснення: У цьому прикладі підзапит виконується для кожного студента, обчислюючи його середній бал безпосередньо в секції SELECT.

Підзапити у секції WHERE

Підзапити у секції WHERE можуть повертати як скалярні значення, так і табличні значення. Від типу значення, що повертається, залежить, з якими операторами має сенс використовувати підзапит. Скалярне значення – це коли повертається одне значення. Зазвичай це число чи рядок. З скалярними значеннями можна використовувати оператори порівняння (<, >, = тощо), можна передавати як аргумент функції або значення колонки в операторі SELECT. Одне значення можна отримувати в результаті підзапиту, використовуючи агрегатні функції без групових операцій або отримувати вибірку по значенню первинного ключа або іншого унікального атрибуту (потенційного ключа).

Підзапити, зв'язані операторами порівняння. Використовують оператори порівняння, такі як =, >, <, <=, >=, <>, для порівняння результатів підзапиту з даними основного запиту.

Задача: отримати список студентів, які у 2006 році навчалися на третьому курсі спеціальності „Комп'ютерні науки". Шифр невідомий, відомо, що він є в таблиці *Student*, і в таблиці *Speciality*.

Кроки, які необхідно виконати:

1) знайти значення поля *Spec* у таблиці *Speciality* і
2) використати його для вибірки інформації з таблиці *Student*.

SQL дозволяє об'єднати ці два запити в один. Причому, якщо *другий запит повертає* ***єдине*** *значення*, то він включається в *операцію порівняння* вислову WHERE:

```
SELECT SecondName, FirstName
FROM Student
WHERE GNum BETWEEN 041 AND 050 AND Spec = (
                    SELECT Spec
                    FROM Speciality
                     WHERE SpName = 'Комп'ютерні науки');
```

Аналогічно запитам, у підзапитах можна використовувати агрегатні функції.

Наприклад, задача: визначити студентів, що мають рейтинг з першої дисципліни вище за середній. Розв'язання:

```
SELECT Kod, Mark
FROM Rating
WHERE DKod = 1 AND Mark >= (
                    SELECT  AVG(Mark)
                    FROM   Rating
                    WHERE DKod = 1);
```

Спочатку у підзапиті вираховується середній бал по першій дисципліні серед всіх студентів, а далі це значення використовується в умові запиту верхнього рівня для цієї ж таблиці.

Підзапити, зв'язані оператором IN. Як було відзначено, у попередніх прикладах результатом виконання підзапиту було єдине значення. Однак можливі ситуації, коли підзапит повертає безліч значень атрибута. У цьому випадку на допомогу приходить оператор IN (*єдиний* зі *спеціальних*, котрий допустимий в підзапитах). Наприклад, дістати список студентів, як у першому прикладі, але таких, що навчаються на всіх спеціальностях напрямку „Економіка":

```
… AND   Kod IN(
        SELECT Kod
```

```
FROM Student
WHERE  Spec IN(
        SELECT Spec
        FROM  Speciality
        WHERE ScDirect = 'Економіка'));
```

Вкладеність такого роду підзапитів може бути як завгодно великою. Наприклад, **задача**: модифікувати другий приклад так, щоб замість кодів виводилися прізвище, ім'я студента та назва дисципліни. **Розв'язання**:

```
SELECT SecondName, FirstName, DName, Mark
FROM Student S, Rating R, Discipline D
WHERE S.Kod = R.Kod AND R.DKod=D.DKod AND R. DKod = 1 AND R.Kod IN(
        SELECT   Kod
        FROM   Rating
        WHERE DKod = 1 AND Mark >=(
                        SELECT   AVG(Mark)
                        FROM Rating
                        WHERE DKod = 1));
```

Виконання таких запитів здійснюється знизу вгору: спочатку підзапит нижнього рівня, його результат використовується у підзапиті наступного рівня, а далі результат цього підзапиту — в базовому запиті.

```
SELECT SecondName, FirstName
FROM Student
WHERE Kod IN (
        SELECT Kod
        FROM Rating
        WHERE Mark = (
                SELECT MAX(Mark)
                FROM Rating));
```

Пояснення: У цьому прикладі підзапит використовується для пошуку студента з максимальним балом.

Підзапити, зв'язані спеціальними операторами умови: EXISTS, ANY(SOME) та ALL. Вони можуть використовуватися тільки з підзапитами.

Оператор EXISTS призначений для того, щоб фіксувати **наявність** вихідних даних у результаті підзапиту, і залежно від цього повертає значення „істина" чи „неправда". Наприклад:

```
SELECT Kod      FROM   Rating
WHERE NOT (Mark < 60 OR Mark >= 75)  AND EXISTS (
                SELECT      Kod          FROM        Rating
                HAVING  MIN(Mark) >= 95
                GROUP BY Kod);
```

буде виведений список „трієчників" у тому випадку, якщо існують „круглі відмінники".

Якщо видалити всі умови, крім EXISTS, то буде виведена вся таблиця. Очевидно, що запит у такій формі здебільшого не має сенсу. Набагато доцільнішою є перевірка умови існування для кожного кортежу відношення. Для цього необхідно скористатися зв'язаними підзапитами. Наприклад, для вибірки викладачів, що мають однакову сумарну надбавку до заробітної плати, як в одній з попередніх задач, можна замість з'єднання відношень скористатися підзапитом:

```
SELECT LKod, SI1.LongevityInc+ SI1.DegreeInc + SI1.TitleInc + SI1.SpecialInc Summary_Increment
FROM SalaryIncrements SI1
WHERE EXISTS
    (SELECT * FROM SalaryIncrements SI2
    WHERE        SI2.LongevityInc + SI2.DegreeInc + SI2.TitleInc + SI2.SpecialInc =
                 SI1.LongevityInc + SI1.DegreeInc + SI1.TitleInc + SI1.SpecialInc
                 AND SI2.LKod <> SI1.LKod);
```

Втім, якщо у вихідні дані додати прізвище та ім'я викладача з таблиці *Lecturer*, то вийде спільне використання з'єднання і підзапиту з EXISTS в одному запиті:

```
SELECT  SecondName, FirstName, LKod,
SI1.LongevityInc + SI1.DegreeInc + SI1.TitleInc + SI1.SpecialInc Summary_Increment
FROM    Lecturer L, SalaryIncrements SI1
WHERE  EXISTS
      (SELECT * FROM      SalaryIncrements SI2
      WHERE     SI2.LongevityInc + SI2.DegreeInc + SI2.TitleInc + SI2.SpecialInc =
      SI1.LongevityInc + SI1.DegreeInc + SI1.TitleInc + SI1.SpecialInc
      AND SI2.LKod <> SI1.LKod)
AND L.LKod = SI1.LKod;
```

Завдання. Модифікувати приклад виводу назв спеціальностей, на яких є студенти з нульовим рейтингом з першої дисципліни:

```
SELECT      SpName      FROM    Speciality Sp
WHERE  EXISTS
            (SELECT   * FROM Student S
            WHERE S.Spec = Sp.Spec AND EXISTS
                  (SELECT      *      FROM Rating
                  WHERE Kod = S.Kod AND DKod = 1 AND Mark = 0));
```

Фактично використання оператора EXISTS еквівалентно підрахунку числа рядків у результаті підзапиту і порівнянням з 0 чи 1: '> 0' для EXISTS і '<1' — для NOT EXISTS.

Ще більш простим буде розв'язання цієї задачі, якщо скористатися оператором ANY чи SOME. Це той же самий оператор, який просто має дві мнемоніки. Наприклад:

```
SELECT  SpName FROM Speciality Sp WHERE   Spec = ANY
        (SELECT Spec FROM Student WHERE Kod = ANY
              (SELECT Kod FROM Rating WHERE DKod = 1 AND Mark = 0));
```

Цей приклад наочно демонструє, що на відміну від EXISTS, оператор ANY *не вимагає зв'язування підзапитів*. Але, крім цього, приклад показовий і цікавий двома наступними моментами. Оператор ANY бере всі кортежі, отримані в підзапиті, і ***для кожного*** з них порівнює значення поля, зазначеного в підзапиті, *з єдиним значенням* поля із зовнішнього запиту. Єдиним тому, що використовується значення оператора **поточного** кортежу зовнішнього запиту. Якщо ***хоча б одне*** значення з підзапиту задовольняє умову перевірки, то ANY повертає значення „істина", а рядок таблиці з зовнішнього запиту включається в його результат.

У **даному** прикладі умовою є рівність, тому саме в **цьому** випадку дія ANY цілком збігається з дією IN. У загальному випадку ANY відрізняється від IN тим, що може використовуватися з будь-якими операторами порівняння, а IN відповідає тільки рівності (чи нерівності).

Однак тут необхідно бути уважним, тому що ANY позначає „*будь-яке* значення з обраних у підзапиті". Тому умова „<ANY" фактично буде відповідати умові *менше максимального* з

обраних, і навпаки, „>ANY“ відповідає умові *більше мінімального* з обраних.

Друга особливість наведеного прикладу полягає в тому, що заміна в ньому EXISTS на ANY цілком коректна. Зв’язано це з тим, що у використаних таблицях немає NULL-значень. Якби це було не так і треба було б вивести *назви* спеціальностей, у студентів яких немає нульового рейтингу, то ANY і EXISTS реагували б на це по-різному: оператор NOT EXISTS поверне назву спеціальності *незалежно* від того, чи виконується вимога задачі або ж просто в полі Spec якої-небудь з таблиць стоїть NULL-значення:

```
SELECT SpName FROM Speciality Sp WHERE NOT EXISTS
    (SELECT * FROM Student S WHERE S.Spec = Sp.Spec …
```

Справа в тім, що результатом EXISTS може бути тільки значення *„істина“* і *„неправда“*, а для операторів порівняння, в яких бере участь ANY, для NULL-значень генерується значення UNKNOWN, що діє також як FALSE.

Застосування ще одного оператора — ALL — означає, що умову зовнішнього запиту має задовольняти *кожен* кортеж з підзапиту. Відповідно, „>ALL“ чи „<ALL“ буде означати *більше максимального* чи *менше мінімального* зі значень, обраних у підзапиті. „<>ALL“ — відповідає *відсутності значення в множині, сформованій підзапитом*. А „=ALL“ відстежує випадок, коли значення поля у *всіх* кортежах підзапиту *рівні*. **Наприклад**: вивести список групи за умови, що всі студенти групи мають однаковий рейтинг по першій дисципліні. **Розв’язання:**

```
SELECT SecondName, FirstName, Spec, GNum, Mark FROM Student S, Rating R
WHERE S.Kod = R.Kod AND DKod = 1 AND Mark = ALL
      (SELECT  Mark  FROM Rating WHERE DKod = 1 AND Kod IN
            (SELECT Kod FROM Student WHERE Spec = S.Spec AND GNum = S.GNum));
```

Узагальнений табличний вираз (оператор WITH)

Узагальнений табличний вираз або CTE (Common Table Expressions) - це тимчасовий результуючий набір даних, до якого можна звертатися в наступних запитах. Для написання узагальненого табличного виразу використовується оператор WITH.

Вираз з WITH вважається «тимчасовим», тому що результат не зберігається на постійній основі у схемі бази даних, а діє як тимчасове представлення, яке існує тільки на час виконання запиту, тобто воно доступне тільки під час виконання операторів SELECT , INSERT, UPDATE, DELETE. Воно дійсне лише в тому запиті, якому він належить, що дозволяє покращити структуру запиту. Тобто основне призначення SELECT в WITH полягає в розбитті складних запитів на прості частини.

Синтаксис оператора WITH:

```
WITH назва_cte [(стовпець_1 [,стовпець_2 ] …)] AS (підзапит)
    [, назва_cte [(стовпець_1 [,стовпець_2 ] …)] AS (підзапит)] …
```

Наприклад: Виберіть з бази даних напрямок рейсу, клас та кількість квитків, що залишилися упродажу. Результат відсортуйте за напрямком рейсу.

```
WITH count_ticket AS
        (SELECT class, COUNT(id_ticket) AS kolvo FROM ticket GROUP BY class),
    count_place AS
        (SELECT c.id_class, c.name, v.destination, quantity_place FROM class c, voyage v
        WHERE c.voyage=v.id_voyage)
SELECT c.destination, c.name, quantity_place-kolvo AS tail
FROM count_place c, count_ticket K
WHERE c.id_class=k.class ORDER BY c.destination;
```

Підзапити в запитах модифікації даних

Запити і підзапити, засновані на команді SELECT, можна використовувати й в інших командах *SQL*. Наприклад, для того, щоб витягти дані з однієї таблиці і розмістити їх в іншій можна скористатися інструкцією:

```
INSERT INTO      AI
SELECT * FROM StudentWHERE Spec = 'AI';
```

Запит вибере всіх студентів цієї спеціальності і внесе їх у нову таблицю. *Якщо схеми відношень збігаються не цілком*, то можна скористатися *замість* '*' *проекцією*.

Аналогічно можна сформувати таблицю, що зберігає значення середнього рейтингу кожного студента. Приклад:

```
INSERT INTO      AveRat1
SELECT Kod, AVG(Mark) FROM Rating GROUP BY Kod;
```

Якщо замість середнього рейтингу студента потрібен середній рейтинг спеціальності вже необхідно скористатися підзапитом. При цьому підзапит не повинен посилатись (у випадку зв'язаних підзапитів) *на* зазначену у команді INSERT таблицю, що змінюється. Наприклад:

```
INSERT INTO      AveRat2
SELECT SpName, AVG(Mark) FROM Rating R, Speciality        WHERE        Spec IN
                (SELECT Spec        FROM Student        WHERE Kod= R.Kod)
GROUP BY SpName;
```

Вставляє дані студентів, які мають бал вище 90, у таблицю high_scorers.

```
INSERT INTO high_scorers (Kod, SecondName, FirstName)
SELECT Kod, SecondName, FirstName FROM Student WHERE Kod IN (
            SELECT Kod FROM Rating WHERE Mark > 90);
```

Аналогічно формуються підзапити для команди відновлення. **Наприклад**, розділити кожну групу, в якій більше 31 людини на 2. Причому в одну з нових груп додати студентів з рейтингом з другої дисципліни, вищим за середній.

```
UPDATE     Student     SET     GNum = GNum + 1WHERE    Kod IN
(SELECT Kod FROM  Student S1 WHERE  31 <
            (SELECT COUNT(*) FROM Student S2
            WHERE S1.Spec = S2.Spec AND S1.GNum =S2.GNum)
AND Kod IN (SELECT  Kod FROM Rating WHERE DKod = 1 AND Mark >
                  (SELECT AVG(Mark) FROM RatingWHERE DKod = 1 AND Kod IN
                        (SELECT Kod  FROM Student S3
                        WHERE        S3.Spec = S1.Spec
                  AND S3.GNum =S1.GNum))));
```

Зауваження. Цей запит буде коректний, якщо на *кожному* курсі є по *одній* групі.

На відміну від INSERT у командах DELETE і UPDATE можна посилатися на таблицю, зазначену в самому зовнішньому запиті, тобто в самих цих командах. Наприклад:

```
DELETE FROM       Student SWHERE    0 =
                        (SELECT SUM(Mark) FROM  RatingWHERE        Kod = S.Kod);
```

Запит видаляє суцільних двієчників. Для цього у підзапиті для кожного рядка-кандидата таблиці *Student* підсумовуються відповідні кортежі таблиці *Rating*. Якщо студент має нулеві бали по всіх дисциплінах, то в базовому запиті інформація про нього видаляється, **але** тільки з таблиці *Student* і тільки в таких СУБД, як *Firebird,* через те, що такі СУБД не підтримують обмежень ON UPDATE та ON DELETE при створенні таблиць.СУБД *PostgreSQL, Oracle* та інші, які такі обмеження підтримують, взагалі відкинуть цей запит через те, що на *кожний* кортеж таблиці *Student* є посилання з таблиці *Rating*. Як зберегти посилальну цілісність даних і цим же запитом видалити відповідні дані з таблиці *Rating* буде розглянуто у розділі „МВД (DDL) SQL. Тригери".

Більшість СУБД підтримують і таку конструкцію:

```
DELETE FROM RATING WHERE DKod = 1 AND Mark <
    (SELECT AVG(Mark) FROM Rating WHERE DKod = 1);
```

а деякі і таку:

```
DELETE FROM RATING WHERE DKod = 1 AND Mark <
    (SELECT AVG(Mark) FROM Rating WHERE DKod = 1)
  AND MDate =
    (SELECT MAX(MDate) FROM RatingWHERE DKod = 1
    HAVING MAX(MDate) <> MIN(MDate));
```

чи, навіть, таку:

```
  DELETE FROM RATING WHERE DKod = 1 AND Mark <
    (SELECT AVG(Mark) FROM Rating WHERE DKod = 1)
  AND MDate =
    (SELECT MAX(MDate) FROM Rating WHERE DKod = 1 AND NOT MDate = ALL
    (SELECT MDate FROM Rating WHERE DKod = 1));
```

Два останні запити, використовуючи різні засоби, розв'язують задачу видалення кортежів, де в першій умові оцінка з першої дисципліни нижча за середню, а в другій — максимальна дата отримання оцінки з першої дисципліни, але ця дата не єдина (з цієї ж дисципліни). Тобто якщо всі студенти отримали оцінку в один день, то видаляти їх не треба.

Структура звіту до лабораторної роботи

Для кожного з запитів представити:

1) словесна постановка задачі, що вирішується;
2) SQL-код рішення;
3) скриншот отриманого результату.

1. Виконати запити на вибірку даних, які включають:
1.1) підзапит в секції FROM;
1.2) підзапит в секції SELECT, який повертає скалярне значення;
1.3) підзапит в секції WHERE, зв'язаний з зовнішнім запитом оператором порівняння.
1.4) підзапит в секції WHERE, зв'язаний з зовнішнім запитом оператором IN.
1.5) підзапит в секції WHERE, зв'язаний з зовнішнім запитом спеціальним оператором умови EXISTS;
1.6) підзапит в секції WHERE, зв'язаний з зовнішнім запитом спеціальним оператором умови ANY(SOME);
1.7) підзапит в секції WHERE, зв'язаний з зовнішнім запитом спеціальним оператором умови ALL;
1.8) корельовані (зв'язані) підзапити.

1.9) узагальнений табличний вираз (оператор WITH).
2. Виконати запити на модифікацію даних, які включають:
2.1) підзапит при операції додавання даних;
2.2) оновлення даних на основі результатів підзапиту;
2.3) видалення записів на основі результатів підзапиту.
3. Пояснити результати виконання кожного запиту.

Можна об’єднувати декілька пунктів в одному запиті, але загалом **не менше 8 запитів.**

Звіт до лабораторної роботи 8 можна здати онлайн на сайті ДО edu.op.edu.ua до початку вашого заняття.