Лабораторна робота 5. Маніпулювання даними. Підзапити.

Задача: отримати список студентів, які у 1996 році навчалися на третьому курсі спеціальності „Економічна кібернетика“. Шифр невідомий, відомо, що він є в таблиці *Student*, і в таблиці *Speciality*.

Питання: Які кроки необхідно виконати?

Відповідь:

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

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 941 AND 950
            AND   Spec = (SELECT  Spec   FROM   Speciality
                          WHERE   SpName = ‘Економічна кібернетика’);
```

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 серед всіх студентів, а далі це значення використовується в умові запиту верхнього рівня для цієї ж таблиці.

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 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.

Більш складним варіантом підзапитів є так звані **зв’язані підзапити**. 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:

```
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 повторена назва спеціальності. Це, по-перше. А по-друге, виконується з'єднання трьох 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 аж 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*. **Розв'язання**:

```
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 (зв'язаному 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.

Для подальшого поширення запитів і наданих ними можливостей розглянемо ще низку *спеціальних операторів 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.

Розв'язання:

```
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.

Завдання: замінити в попередньому прикладі EXISTS на '0<' і '*' — на COUNT(*) та порівняти результат з наведеним прикладом.

Ще більш простим буде розв'язання цієї задачі, якщо скористатися оператором ANY чи SOME. До речі, 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 (**завдання**: *замінити „=ANY" на „IN" та порівняти результат з наведеним прикладом*). 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));
```

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   OI
        SELECT    *   FROM          Student
            WHERE   Spec = ‘OI’;
```

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 = ANY
                        (SELECT   Spec  FROM   Student
                                WHERE     Kod = R.Kod)
            GROUP BY SpName;
```

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 S
    WHERE     0 =
        (SELECT   SUM(Mark)   FROM    Rating
            WHERE     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      Rating
                  WHERE    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.

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 + 1
   WHERE    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    Rating
                                          WHERE    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.

Приклад рішення завдання до лабораторної роботи 5.

Запишіть *SQL*-запити **з використанням підзапитів** для маніпулювання даними з таблиць, що створені у лабораторній роботі 1.

П.1. Виберіть з бази даних напрямки рейсів та класи, на які ще не продано жодного квитка.

Розв'язання.

```
SELECT DISTINCT V.destination, C.name FROM voyage V, class C
    WHERE C.voyage=V.id_voyage AND C.id_class NOT IN
                                        (SELECT class FROM ticket);
```

Результуюча таблиця.

Destination	**name**
Одеса – Бухарест	бізнес
Одеса – Бухарест	економ
Одеса – Бухарест	економ покращений
Одеса – Нью-Йорк	бізнес
Одеса – Нью-Йорк	економ
Одеса – Пекін	бізнес
Одеса – Пекін	економ

П.2. Виберіть з бази даних пункт призначення рейсу, дату вильоту та назву класу салону, на якій продано всі квитки.

Розв'язання.

```
SELECT   V.destination, V.date_departure, C.name     FROM   voyage V, class C
    WHERE V.id_voyage=C.voyage AND C.quantity_place =
        (SELECT COUNT(class) FROM ticket T
            WHERE C.id_class=T.class GROUP BY class);
```

Результуюча таблиця.

destination	**date_departure**	**name**
Одеса – Нью-Йорк	2012-08-09	економ покращений

П.3. Виберіть з бази даних напрямок рейсу, клас та кількість квитків, що залишилися у продажу. Результат відсортуйте за напрямком рейсу.

Розв'язання.

```
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;
```

Результуюча таблиця.

destination	name	tail
Одеса – Лондон	бізнес	49
Одеса – Лондон	економ покращений	49
Одеса – Лондон	економ	99
Одеса – Нью-Йорк	економ покращений	46

Завдання до лабораторної роботи 5

Запишіть *SQL*-запити (мінімум 8) **з використанням підзапитів** для маніпулювання даними з таблиць, що створені у лабораторних роботах 1 та 2. В роботі обов'язково відобразити використання наступних операторів та конструкцій:

1) спеціальних операторів умови (ANY, ALL, EXISTS);
2) операторів порівняння;
3) оператора IN;
4) зв'язані підзапити;
5) з секцією WITH;
6) підзапит для реалізації будь-якої операції модифікації даних (insert, update або delete).

Структура звіту до лабораторної роботи

Для кожного з запитів:
1) постановка задачі, що вирішується;
2) SQL-код рішення;
3) скриншот отриманого результату.