Реалізація операцій з’єднання

Задача: потрібно вивести поточний рейтинг всіх студентів по всіх дисциплінах з таблиці *Rating*. Однак користувача не цікавить їхній *код* — йому необхідні *прізвища й імена*, що описані в таблиці *Student*:

```
SELECT    S.SecondName, S.FirstName, R.DKod, R.Mark
          FROM       Student S, Rating R
     WHERE    S.Kod = R.Kod    ORDER BY S.SecondName, R.DKod;
```

Правда, необхідно відзначити, що в питанні застосування префіксів цей приклад не абсолютно наочний. І от чому: якщо серед атрибутів, на які з’єднання виконує проекцію, немає однакових імен, то префікси можна опустити.

```
SELECT    SecondName, FirstName, DKod, Mark
          FROM     Student S    INNER JOIN    Rating R     ON    S.Kod = R.Kod
          ORDER BY SecondName, DKod;
```

У всіх СУБД замість „чистої“ команди JOIN використовуються команди:

INNER JOIN — ***внутрішнє*** *з’єднання*

та

LEFT | RIGHT | FULL OUTER JOIN — *ліве*, *праве* чи *повне* ***зовнішнє*** *з’єднання*.

Для розуміння різниці між цими типами з’єднань розглянемо приклад, в якому дещо порушимо правила формалізації біумовного зв’язку 1:1: до таблиці *DWExec* додамо кортежі, що містять ідентифікатори студентів, які *не мають* тем дипломних проектів. Завдяки цьому в полі *DpKod* з’являться значення *NULL*, а таблиця *DWExec* прийме наступний, не зовсім коректний вигляд:

DWExec

SKod	DpKod
1	9
2	1
3	3
4	2
5	5
6	10
7	NULL
8	NULL

Вирішимо задачу формування переліку керівників дипломних проектів з відображенням студентів, якими вони керують, та ідентифікаторів тем дипломних проектів, що виконують ці студенти. Для вирішення цієї задачі виконаємо внутрішнє з'єднання двох наведених таблиць:

```
SELECT    LKod, SKod, L.DpKod
      FROM      DWList L     INNER JOIN    DWExec E    ON    L.DpKod = E.DpKod;
```

LKod	SKod	DpKod
1	1	9
4	2	1
4	3	3
4	4	2
5	5	5
1	6	10

Отримаємо такий результат:

Як можна бачити, у результуючій таблиці відсутні викладачі, що не керують дипломними проектами, та студенти, які ще не отримали тем дипломних проектів. Для того, щоб додати всю таку інформацію треба виконати зовнішнє з'єднання (точніше, повне зовнішнє з'єднання):

```
SELECT    LKod, L.DpKod, SKod, E.DpKod
      FROM      DWList L     FULL OUTER JOIN      DWExec E  ON      L.DpKod = E.DpKod;
```

Результуюча таблиця буде така:

LKod	DpKod	SKod	DpKod
4	1	2	1
4	2	4	2
4	3	3	3
5	5	5	5
1	9	1	9
1	10	6	10
5	4	NULL	NULL
2	6	NULL	NULL
2	7	NULL	NULL
3	8	NULL	NULL
NULL	NULL	7	NULL
NULL	NULL	8	NULL

Такий результат може бути корисним на етапі призначення студентам тем дипломних проектів.

Після цього завідувачу кафедри потрібно знати тільки, хто з викладачів має дипломників, та які теми ще не видані. Для вирішення цієї задачі треба виконати ліве зовнішнє з'єднання, яке додасть до результату внутрішнього з'єднання кортежі першої (що розташована до оператора JOIN) таблиці, які „не мають пари":

```
SELECT  LKod, L.DpKod, SKod, E.DpKod  FROM  DWList L  LEFT OUTER JOIN  DWExec E  ON  L.DpKod = E.DpKod;
```

Результат запиту наступний:

LKod	DpKod	SKod	DpKod
4	1	2	1
4	2	4	2
4	3	3	3
5	5	5	5
1	9	1	9
1	10	6	10
5	4	NULL	NULL
2	6	NULL	NULL
2	7	NULL	NULL
3	8	NULL	NULL

Деканату ж навпаки потрібно додатково знати тільки, хто з дипломників ще не навантажений. Отут допоможе праве зовнішнє з'єднання, яке додасть до результату внутрішнього з'єднання „зайві" кортежі другої таблиці, що знаходиться після оператора JOIN:

```
SELECT  LKod, L.DpKod, SKod, E.DpKod FROM  DWList L  RIGHT OUTER JOIN  DWExec E  ON  L.DpKod = E.DpKod;
```

Результуюча таблиця буде така:

LKod	DpKod	SKod	DpKod
4	1	2	1
4	2	4	2
4	3	3	3
5	5	5	5
1	9	1	9
1	10	6	10
NULL	NULL	7	NULL
NULL	NULL	8	NULL

В одному запиті можна виконувати з’єднання і більшого, ніж два, числа таблиць. Наприклад, додамо до запиту, що вирішує задачу відображення прізвищ студентів та їхнього рейтингу, назву спеціальності:

```
SELECT    SPName, SecondName, FirstName, Mark FROMStudent S, Rating R, Speciality SP
      WHERE    S.Kod = R.Kod AND S.Spec = SP.Spec  ORDER BY        S.Spec, SecondName;
```

З’єднання можна виконати над однією таблицею (*з’єднання таблиці самої із собою*). Префікси та аліаси таблиці для цього просто необхідні. Наприклад, **задача**: з кожної підмножини викладачів, що мають однакову сумарну надбавку, необхідно вибрати тих, у кого надбавка за стаж перевищує 100 грн.

Спочатку виконаємо такий запит:

```
SELECT    LKod, LongevityInc + DegreeInc + TitleInc + SpecialInc          Summary_Increment, LongevityInc
      FROM       SalaryIncrements     ORDER BY                  2 DESC, LKod;
```

який дасть наступний результат:

LKOD	SUMMARY_INCREMENT	LONGEVITYINC
1	1845	300
5	1534	260
2	990	220
4	990	200
3	876	250

Вибірку викладачів з однаковою сумарною надбавкою можна виконати таким чином:

```
SELECT    SI2.LKod, SI2.LongevityInc + SI2.DegreeInc + SI2.TitleInc + SI2.SpecialInc  Summary_Increment, SI2.LongevityInc
      FROM       SalaryIncrements SI1, SalaryIncrements SI2
   WHERESI2.LongevityInc + SI2.DegreeInc + SI2.TitleInc + SI2.SpecialInc =SI1.LongevityInc + SI1.DegreeInc + SI1.TitleInc + SI1.SpecialInc
   ORDER BY 2 DESC, SI2.LKod;
```

Дана інструкція фактично виконує **з’єднання** *двох копій таблиці SalaryIncrements*. Залишається додати тільки до вислову WHERE *другу частину умови*:

```
…     AND SI2. LongevityInc > 100       …
```

Однак у цьому прикладі результуюча таблиця буде включати й одиничні підгрупи, тому що сумарна надбавка кожного викладача дорівнює сумарній надбавці цього ж викладача з другої копії таблиці

Розпізнати такі кортежі можна *за збігом первинних ключів* і, відповідно, виключити їх. Для цього до вислову WHERE додамо *ще одну умову*:

```
…    AND SI2.LKod <> SI1.LKod    …
```

Ця умова забезпечить видалення з результату не тільки одиничних підгруп, але і зайвих кортежів з підгруп, що містять по два рядка, тому що один з них також є з'єднанням кортежу самого із собою. Але якщо в підгрупі було три і більш рядків, то їхнє число зменшується на одиницю. Відповідно залишається по *два і більше однакових кортежів*. Позбутися від них можна за допомогою оператора DISTINCT:

```
SELECT DISTINCT  …
```

І, нарешті, *над результатом з'єднання копій однієї таблиці* можна виконати *з'єднання з іншим відношенням*. Наприклад, замінивши *LKod* на прізвище й ім'я з таблиці *Lecturer*. Кінцевий варіант запиту буде виглядати таким чином:

```
SELECT DISTINCT  SecondName, FirstName,
                 SI2.LongevityInc + SI2.DegreeInc + SI2.TitleInc + SI2.SpecialInc
                 Summary_Increment, SI2.LongevityInc
    FROM     SalaryIncrements SI1, SalaryIncrements SI2, Lecturer L
    WHERE    SI2.LongevityInc + SI2.DegreeInc + SI2.TitleInc + SI2.SpecialInc =
             SI1.LongevityInc + SI1.DegreeInc + SI1.TitleInc + SI1.SpecialInc
             AND SI2. LongevityInc > 100
             AND SI2.LKod <> SI1.LKod
             AND L.Kod = SI2.LKod
    ORDER BY 3 DESC, SI2.LKod;
```

В результаті отримаємо:

SECOND_NAME	FIRST_NAME	SUMMARY_INCREMENT	LONGEVITYINC
Петрушкіна	Валентина	990	200
Сідоров	Георгій	990	220

Підзапити

Задача: отримати список студентів, які у 1996 році навчалися на третьому курсі спеціалізації „Комп'ютерні науки". 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. SQL* дозволяє об'єднати ці два запити в один:

```
SELECT    SecondName, FirstName          FROM       Student
      WHERE     GNum   BETWEEN 941 AND 950
                AND     Spec = (SELECT     Spec      FROM     Speciality          WHERE        SpName = 'Компьютерні 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);
```

У попередніх прикладах результатом виконання підзапиту було єдине значення. Коли підзапит повертає безліч значень атрибута використовуються оператор IN (*єдиний* зі *спеціальних*, котрий допустимий в підзапитах). Наприклад, дістати список студентів, як у першому прикладі, але таких, що навчаються на всіх спеціалізаціях спеціальності 122:

```
… AND   Kod     IN(SELECT      Kod     FROM     Student
                        WHERE  Spec    IN(SELECT    Spec  FROM   Speciality          WHERE        ScDirect = 122));
```

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.

Зв’язані підзапити

Задача: 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. Ми їх не розглянули разом з іншими умовними операторами, тому що 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.

Ще більш простим буде розв'язання цієї задачі, якщо скористатися оператором ANY чи SOME. 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, що не підтримують обмежень ON UPDATE та ON DELETE при створенні таблиць.

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.

Вкладені запити можна створювати з використанням операторів EXISTS та IN, як було показано раніше, або у вигляді посилання на тимчасову таблицю. Посилання на тимчасову таблицю реалізуються двома способами: в FROM-фразі та з використанням оператора WITH.

Використання оператора WITH дозволяє розбити складний запит на безліч підзапитів у зручній для сприйняття людиною формі.

Оператор WITH використовує наступну конструкцію:

```
WITH ім'я_запиту [(список_атрибутів_з_запиту)] AS (опис_запиту)
    Опис_основного_запиту
```

Прикладом використання може бути розв'язання раніш представленої задачі про виведення списку студентів, які мають рейтинг вище за середній за своєю спеціальністю:

```
WITH    Avg_mark AS
        (SELECT   S1.Spec, AVG(Mark) AS Avg_mark
            FROM      Student S1, Rating R1
            WHERE     S1.Kod = R1.Kod
            GROUP BY      S1.Spec)
SELECT    SecondName, FirstName, Mark, S.Spec
    FROM      Student S, Rating R, Avg_mark A
    WHERE     S.Kod = R.Kod AND S.Spec = A.Spec AND R.Mark > A.Avg_mark;
```

Об'єднання

Наприклад, вивести на екран коди студентів та дисциплін, з яких вони мають максимальний і мінімальний рейтинг:

```
SELECT    Kod, DKod, Mark, 'Максимальний␣ рейтинг'   FROM   Rating
    WHERE     Mark = (SELECT    MAX(Mark)   FROM   Rating)

UNION

SELECT    Kod, DKod, Mark, 'Мінімальний␣ ␣ рейтинг'  FROM   Rating
    WHERE     Mark = (SELECT    MIN(Mark)    FROM   Rating);
```

Два пробіли (символ '␣ ') додані для вирівнювання довжини константи, інакше результати запитів будуть несумісні.

Ієрархічні (рекурсивні) запити PostgreSQL

Додавання до оператора WITH необов'язкового оператора RECURSIVE дозволяє запиту звертатися до власних результуючих даних, забезпечуючи їхню рекурсивну обробку.

Алгоритм рекурсивного запиту повинен включати дві частини: перша частина — це *основа*, яка звичайно повертає один рядок з вихідною точкою ієрархії або частини ієрархії, а друга частина — *рекурсивна*, котра буде зв'язуватися з тимчасовою таблицею, оголошеною в операторі WITH. Обидві частини поєднуються оператором UNION або UNION ALL. Строго кажучи, процес формування відповіді на запит скоріше ітераційний, ніж рекурсивний, але поняття RECURSIVE обране комітетом зі стандартизації *SQL*.

Розглянемо приклад запиту, який демонструє підрахунок суми чисел від 1 до 100:

```
WITH RECURSIVE    t(n)    AS    (SELECT      1
                                      UNION ALL
                                  SELECT      n+1    FROM    t    WHERE     n < 100)
      SELECT sum(n) FROM t;
```

У цьому прикладі робоча таблиця містить лише один рядок на кожному кроці.

Щоб отримати перші 10 чисел Фібоначі, можна виконати рекурсивний запит:

```
WITH RECURSIVE    t(n, fn, "fn-1", "fn-2") AS
            (SELECT   1, 1, 1, 0            UNION
             SELECT   n+1, "fn-1"+"fn-2", "fn-1"+"fn-2", "fn-1"              FROM t          WHERE         n <= 10)
      SELECT    n, fn  FROM   t;
```

Запит отримання назв підрозділів ОНПУ (інститутів, кафедр) з урахуванням їхньої ієрархії представлений у наступному вигляді:

```
WITH RECURSIVE    Rec(DeptKod, PKod, DeptName)   AS
      (SELECT   DeptKod, PKod, DeptName     FROM     Department  WHERE     Pkod    IS NULL
         UNION
      SELECT    D.DeptKod, D.PKod, D.DeptName
            FROM     Department D   INNER JOIN    Rec R
                                    ON    (D.PKod = R.DeptKod))
SELECT DeptKod, PKod, DeptName FROM Rec;
```

У цьому прикладі перегляд ієрархії підрозділів починається з підрозділу, в якому відсутній вищий підрозділ, тобто атрибут *PKod* не визначено.

Для виводу на екран назв підрозділів у зручній формі з ієрархічною табуляцією попередній запит модифікований наступним чином:

```
WITH RECURSIVE    Rec(DeptKod, PKod, DeptName, Path, Level) AS
   (SELECT DeptKod, PKod, cast(DeptName AS VARCHAR) AS   DeptName, cast(DeptKod AS VARCHAR) AS Path,1
         FROM Department WHERE PKod IS NULL
       UNION
    SELECT D.DeptKod, D.PKod, lpad(' ', 3*level) || D.DeptName,  Path || cast(D.DeptKod AS VARCHAR), Level+1
         FROM Department D INNER JOIN Rec R ON (D.PKod = R.DeptKod))

   SELECT DeptKod, DeptName FROM Rec ORDER BY Path;
```

Підтримка механізму об’єктно-реляційних зв’язків в PostgreSQL

Об’єктно-реляційні зв’язки між таблицями, створення яких було розглянуто у відповідному параграфі (див. стор. **Ошибка! Закладка не определена.**), накладають певні обмеження на маніпулювання даними в таких таблицях.

Зокрема, внесення даних в таблиці-нащадки не виконує автоматичне внесення в таблицю-предка, але вони логічно видимі в ній. Так у результаті виконання запиту

```
INSERT INTO    Master VALUES(2, 'Петрова', 'Ганна', 'Василівна', 1, 'ОІ', 1, NULL, NULL, 'Тема12');
```

рядок по студенту буде видно і в таблиці *Student*.

Для управлення видимістю рядків у таблиці-предку, які були створені в таблиці-нащадку, використовується оператор ONLY, що передує імені таблиці.

Наприклад, для отримання вмісту тільки таблиці-предка *Student* без зачіпання таблиці-нащадка *Master* використовується запит:

```
SELECT * FROM   ONLY    Student;
```

Наприклад, для оновлення рядків тільки таблиці *Student* без зачіпання рядків таблиці *Master* використовується запит:

```
UPDATE ONLY    Student    SET SecondName = 'Петрушкіна'          WHERE      kod = 1;
```

А для поновлення рядків таблиці *Student* і рядків таблиці *Master* використовується стандартний запит:

```
UPDATE    Student    SET SecondName = 'Перушкіна'          WHERE        Kod = 1;
```