top
КОМПЬЮТЕРНАЯ
АКАДЕМИЯ

Теория Баз Данных

Урок № 6

Объединения

Содержание

1. Операторы для использования в подзапросах

В прошлом уроке вами были изучены подзапросы и возможность их использования при написании SQL-запросов с применением оператора IN, однако совместно с подзапросами можно использовать еще несколько логических операторов, именно их мы и рассмотрим в текущем разделе.

Для того чтобы продемонстрировать вам примеры выполнения SQL-запросов мы будем использовать, знакомую вам по прошлым занятиям, базу данных University, частичная диаграмма, которой представлена на рисунке 1.1.

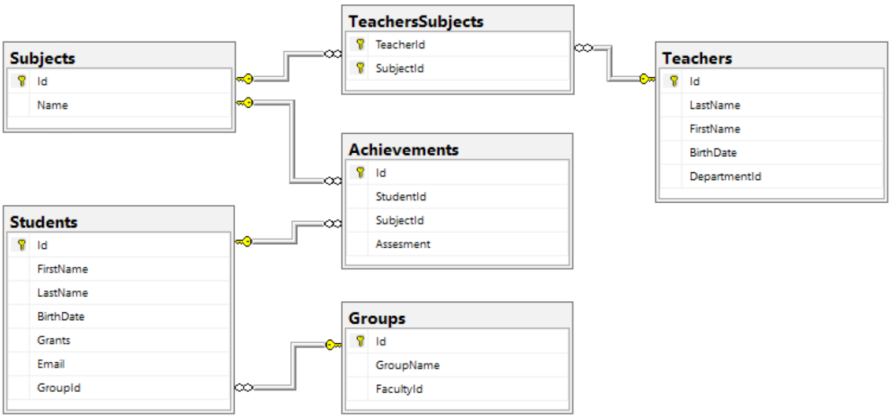


Рисунок 1.1. Диаграмма базы данных University (частично)

Оператор EXISTS

Оператор EXISTS является в своем роде уникальным, потому что проверяет строки, а не сравнивает значения столбцов и предназначен для определения наличия строк, возвращаемых подзапросом. Если подзапрос содержит хотя бы одну строку, то оператор EXISTS возвращает истину и текущая запись помещается в результирующую таблицу, иначе оператор возвращает ложь и, соответственно, данные не записываются.

В качестве примера продемонстрируем SQL-запрос, возвращающий нам информацию о студентах, которые получали оценки по предметам:

```
SELECT FirstName, LastName, BirthDate, Email
FROM Students
WHERE EXISTS (SELECT *
              FROM Achievements
              WHERE Achievements.StudentId =
                                  Students.Id);
```

Следует обратить ваше внимание на ряд особенностей текущего запроса. Во-первых, вы, наверное, заметили, что в данном SQL-запросе подзапрос является связанным с основным запросом через уникальный идентификатор студента (Achievements.StudentId = Students.Id). Во-вторых, в подзапросе после оператора SELECT используется символ *, такая запись характерна для любого подзапроса, с использованием оператора EXISTS, потому что он применяется к строкам и возвращаемые столбцы не имеют никакого значения.

При выполнении текущего SQL-запроса оператор EXISTS вернет истину, если в таблице Achievements существует

хотя бы одна строка, содержащая уникальный идентификатор студента из основного запроса и именно информация по этому студенту будет записана в результирующую таблицу (Рисунок 1.2).

FirstName	LastName	BirthDate	Email
Jack	Jones	1997-11-05	jj@net.eu
Harry	Miller	1998-02-11	hm@net.eu
Grace	Evans	1997-06-24	eg@net.eu

Рисунок 1.2. Студенты, получившие оценки по предметам

Оператор EXISTS можно использовать совместно с оператором NOT для определения данных, которые не удовлетворяют условию. Например, при написании дуального SQL-запроса, который отобразит информацию о студентах, не получивших ни одной оценки по предметам:

```
SELECT FirstName, LastName, BirthDate, Email
FROM Students
WHERE NOT EXISTS (SELECT *
                  FROM Achievements
                  WHERE Achievements.StudentId =
                  Students.Id);
```

В приведенном примере подзапрос также связан с основным запросом по идентификатору студента, однако в данном случае использование оператора NOT приведет к тому, что в результирующую таблицу будет записана информация только о тех студентах, для которых оператор EXISTS вернет ложь. То есть в том случае если в таблице Achievements не существует ни одной строки, содержащей

уникальный идентификатор студента из основного запроса. Результат выполнения текущего запроса представлен на рисунке 1.3.

FirstName	LastName	BirthDate	Email
Lily	Wilson	1998-09-12	lw@net.eu
Joshua	Johnson	1997-05-23	jo@net.eu
Emily	Taylor	1997-12-27	et@net.eu
Charlie	Thomas	1998-01-31	ct@net.eu
Oliver	Moore	1997-07-05	om@net.eu
Jessica	Brown	1997-07-17	jb@net.eu

Рисунок 1.3. Студенты, которые не получили оценки по предметам

Оператор ANY/SOME

Операторы ANY и SOME являются синонимами и выполняют, как это ни странно, одно и то же действие – осуществляют проверку выполнения заданного условия сравнения хотя бы для одного значения из тех, которые возвращает подзапрос.

Следующий SQL-запрос позволит нам выяснить существуют ли в нашей базе данных University записи о студентах, которые получали оценку 10:

```
SELECT FirstName, LastName, BirthDate, Email
FROM Students
WHERE Id = ANY (SELECT StudentId
                FROM Achievements
                WHERE Assesment = 10);
```

Следует обратить ваше внимание на особенность использования оператора ANY — оператор сравнения

необходимо указывать между сравниваемым значением и оператором (WHERE Id = ANY ...).

В текущем запросе подзапрос возвращает уникальные идентификаторы тех студентов, которые получили оценку 10 и если идентификатор студента в записи из основного запроса совпадет с любым из них, то эта запись будет добавлена в результирующую таблицу. Результат выполнения данного запроса представлен на рисунке 1.4.

FirstName	LastName	BirthDate	Email
Jack	Jones	1997-11-05	jj@net.eu
Harry	Miller	1998-02-11	hm@net.eu

Рисунок 1.4. Студенты, которые получали оценку 10

В том случае если вы замените в данном запросе оператор ANY на SOME, то вы получите такой же результат, мы предлагаем вам проверить это самостоятельно.

Внимательный студент может возразить, что для получения такого результата будет достаточно сравнить результаты выполнения подзапроса при помощи оператора IN и будет абсолютно прав. Следующий SQL-запрос вернет такой же результат, как и на рисунке 1.4:

```
SELECT FirstName, LastName, BirthDate, Email
FROM Students
WHERE Id IN (SELECT StudentId
             FROM Achievements
             WHERE Assesment = 10);
```

Однако преимущество операторов ANY/SOME заключается в том, что их можно использовать не только

с оператором сравнения на равенство (=), как оператор IN, но и с операторами сравнения на неравенство (>, <, >=, <=, <>).

В качестве примера приведем SQL-запрос, который позволит нам определить количество студентов, которые старше любого из преподавателей:

```
SELECT COUNT(*) AS [Count]
FROM Students
WHERE BirthDate < ANY (SELECT BirthDate
                       FROM Teachers);
```

В этом запросе дата рождения каждого студента сравнивается с каждой датой рождения преподавателей из списка дат, который возвращает нам подзапрос. Если дата рождения студента меньше чем любая из дат рождения преподавателей, то такой студент старше какого-то преподавателя и функция агрегирования COUNT(*) позволяет определить их количество. Результат выполнения текущего запроса представлен на рисунке 1.5.

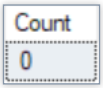


Рисунок 1.5. Количество студентов, которые старше любого из преподавателей

Оператор ALL

Оператор ALL используется при сравнении результатов подзапроса, таким образом, чтобы указанному условию удовлетворяли все результаты подзапроса без исключения.

Данный оператор так же, как и операторы ANY/SOME может применяться совместно с различными операторами сравнения. Например, выполнив следующий SQL-запрос, мы сможем получить список студентов, оценки которых больше, чем средние оценки каждого студента, результат этого запроса представлен на рисунке 1.6.

```
SELECT FirstName, LastName, Assesment
FROM Students AS S, Achievements AS A
WHERE StudentId = S.Id AND
Assesment > ALL (SELECT AVG(Assesment)
                 FROM Achievements
                 GROUP BY StudentId);
```

FirstName	LastName	Assesment
Jack	Jones	11

Рисунок 1.6. Студенты, оценки которых больше, чем средние оценки каждого студента

В данном случае подзапрос возвращает список средних оценок всех студентов, а в основном запросе осуществляется поиск студента, оценка которого больше, чем все оценки из этого списка.

Операторы ANY/SOME и ALL могут применяться не только совместно с оператором WHERE, что было показано во всех предыдущих примерах, но и с оператором HAVING. Для того чтобы продемонстрировать вам это мы, исключительно в учебных целях, напишем следующий SQL-запрос, который возвращает список студентов оценки, которых не равны любой из оценок студентов группы 30PR11:

```
SELECT FirstName, LastName, Assesment
FROM Students AS S,
(SELECT StudentId, Assesment
FROM Achievements
GROUP BY StudentId, Assesment
HAVING Assesment <> ALL (SELECT Assesment
       FROM Groups AS G, Students AS S,
            Achievements AS A
       WHERE S.GroupId = G.Id
            AND A.StudentId = S.Id
            AND GroupName = '30PR11')) AS SA
WHERE SA.StudentId = S.Id;
```

В текущем SQL-запросе применяются два подзапроса. Первый подзапрос, с наибольшей вложенностью, возвращает список оценок, полученных студентами группы 30PR11. Результаты этого подзапроса используются вторым подзапросом для получения уникального идентификатора студента и его оценки при условии, что эта оценка не равна ни одной из оценок, возвращенных первым подзапросом. Результаты второго подзапроса формируются в виде таблицы с псевдонимом SA, которая соединяется с таблицей основного запроса по идентификатору студента (SA.StudentId = S.Id) (Рисунок 1.7).

FirstName	LastName	Assesment
Harry	Miller	9
Jack	Jones	10
Harry	Miller	10
Jack	Jones	11

Рисунок 1.7. Студенты оценки, которых не равны оценкам студентов группы 30PR11

2. Объединение результатов запроса

Принципы объединения

Объединение результатов запросов применяется в том случае если необходимо получить совокупность строк двух и более запросов, которые выполняются независимо друг от друга.

Для того чтобы результаты запросов можно было объединить они должны соответствовать определенным условиям совместимости:

- количество столбцов в каждом запросе должно быть одинаковым;
- типы данных соответствующих столбцов во всех запросах должны быть совместимы.

Также при объединении запросов существует ряд особенностей:

- в результирующем наборе, полученном при объединении, будут использоваться имена столбцов, которые были указаны в первом запросе;
- нельзя сортировать каждый запрос по отдельности, осуществить сортировку можно только всего составного запроса, указав по его окончанию оператор ORDER BY.

Объединение результатов запросов обеспечивается благодаря использованию ключевых слов UNION

и UNION ALL. Общая форма записи SQL-запросов с использованием этих операторов выглядит следующим образом:

```
SELECT columnName1, columnName2, ...
FROM tableName
UNION [ALL]
SELECT columnName1, columnName2, ...
FROM tableName;
```

Ключевое слово UNION

Ключевое слово UNION позволяет объединить результаты запросов и применяется в том случае, когда в результирующем наборе необходимо исключить повторяющиеся строки.

Продемонстрируем использование ключевого слова UNION на примере объединения двух запросов, в результате которого нам необходимо получить студентов и преподавателей, родившихся летом, при этом результирующий набор будет отсортирован по их дате рождения:

```
SELECT FirstName + ' ' + LastName
       AS FullName, BirthDate
FROM Students
WHERE MONTH(BirthDate) > 5
   AND MONTH(BirthDate) < 9
UNION
SELECT FirstName + ' ' + LastName, BirthDate
FROM Teachers
WHERE  MONTH(BirthDate) > 5
   AND MONTH(BirthDate) < 9
ORDER BY BirthDate;
```

Данный SQL-запрос будет выполняться следующим образом: вначале выполнятся два запроса, которые вернут полное имя и дату рождения студентов и преподавателей, родившихся летом, соответственно. После этого результаты запросов объединяются вместе без повторяющихся записей при помощи ключевого слова UNION, полученный вследствие этого результирующий набор будет отсортирован по возрастанию даты рождения с использованием оператора ORDER BY (Рисунок 2.1).

FullName	BirthDate
Daniel Williams	1979-07-30
Grace Evans	1997-06-24
Oliver Moore	1997-07-05
Jessica Brown	1997-07-17

Рисунок 2.1. Преподаватели и студенты, родившиеся летом

Ключевое слово UNION ALL

Ключевое слово UNION ALL также позволяет объединять результаты различных запросов, однако выполняется быстрее, чем UNION, потому что не тратит время на удаление дублирующих строк в объединяемых запросах.

Объединение результатов запросов можно также применять, в том случае если необходимо получить сводную информацию из различных SQL-запросов, которые связаны между собой определенным критерием. В следующем примере мы продемонстрируем возможность получения информации о количестве студентов, которые родились в разные времена года:

```
SELECT 'Spring' AS Seasons, COUNT(*)
                AS [Number of students]
FROM Students
WHERE MONTH(BirthDate) BETWEEN 3 AND 5
UNION ALL
SELECT 'Summer', COUNT(*)
FROM Students
WHERE MONTH(BirthDate) BETWEEN 6 AND 8
UNION ALL
SELECT 'Autumn', COUNT(*)
FROM Students
WHERE MONTH(BirthDate) BETWEEN 9 AND 11
UNION ALL
SELECT 'Winter', COUNT(*)
FROM Students
WHERE MONTH(BirthDate) IN (1, 2, 12);
```

В данном случае каждый из SQL-запросов возвращает количество студентов, родившихся весной, летом, осенью и зимой соответственно. Первым значением каждого запроса указано название времени года в виде строкового литерала, такой подход срабатывает, потому что используется, по сути, одинаковый тип данных. Полученные результаты объединяются при помощи ключевого слова UNION ALL, которое еще обеспечивает требуемый порядок объединения, что не позволяет сделать оператор UNION (Рисунок 2.2).

Seasons	Number of students
Spring	1
Summer	3
Autumn	2
Winter	3

Рисунок 2.2. Количество студентов в соответствии со временами года

Еще один пример продемонстрирует возможность использования объединения при составлении статистических отчетов. Например, нам необходимо получить отдельно количество студентов и преподавателей, которые родились во втором квартале, а также их общее количество:

```
SELECT 'Students' AS [Second quarter], COUNT(*)
AS [Count]
FROM Students
WHERE MONTH(BirthDate) BETWEEN 5 AND 8
UNION ALL
SELECT 'Teachers', COUNT(*)
FROM Teachers
WHERE MONTH(BirthDate) BETWEEN 5 AND 8
UNION ALL
SELECT 'All', SUM(AllSum.AllCount)
FROM
(
      SELECT COUNT(*) AS AllCount
      FROM Students
      WHERE MONTH(BirthDate) BETWEEN 5 AND 8
      UNION ALL
      SELECT COUNT(*)
      FROM Teachers
      WHERE MONTH(BirthDate) BETWEEN 5 AND 8
) AS AllSum
```

Данный SQL-запрос аналогичен предыдущему, только в этом случае два первых объединяемых запроса возвращают количество студентов и преподавателей, родившихся во втором квартале соответственно, а в третьем запросе для подсчета общего количества записей в таблице с прототипом AllSum, которую мы получили

как результат объединения двух запросов, используется функция SUM(AllSum.AllCount). Результат выполнения этого SQL-запроса приведен на рисунке 2.3.

Second quarter	Count
Students	4
Teachers	2
All	6

Рисунок 2.3. Количество студентов и преподавателей, родившихся во втором квартале

3. Объединения JOIN

В прошлом разделе мы изучили возможность объединения результатов запросов при помощи ключевого слова UNION, такой вид объединения можно назвать объединением по вертикали. В свою очередь оператор JOIN позволяет объединять таблицы в пределах одного SQL-запроса и может считаться объединением по горизонтали. Существуют различные виды объединений, которые можно выполнить при помощи оператора JOIN: внутреннее (INNER) и внешние (LEFT, RIGHT, FULL).

В предыдущих уроках при написании SQL-запросов для объединения таблиц мы использовали синтаксис с применением оператора WHERE, который соответствует стандарту ANSI SQL-89. Синтаксис с использованием оператора JOIN соответствует стандарту ANSI SQL-92, который помимо других улучшений языка SQL обеспечивает, в отличие от SQL-89, поддержку выполнения внешних запросов и улучшает читабельность запроса, явно разделяя объединение таблиц и фактическую фильтрацию данных.

В данный момент времени вам может показаться, что предыдущие ваши усилия, потраченные на написание SQL-запросов по стандарту SQL-89, были напрасны, но нет. Дело в том, что оба синтаксиса и SQL-89, и SQL-92 повсеместно применяются различными программистами и для того, чтобы понимать запросы, написанные другими специалистами, вам необходимо научиться использовать оба этих синтаксиса.

В дальнейшем мы рекомендуем вам использовать синтаксис с использованием оператора JOIN, который представляет собой более современный подход при написании SQL-запросов и предоставляет большие возможности, описанные выше.

Понятие INNER JOIN

Внутреннее объединение двух таблиц возможно при помощи оператора INNER JOIN с указанием условия их объединения (предиката) после ключевого слова ON. Общая форма записи SQL-запросов такого вида будет выглядеть следующим образом:

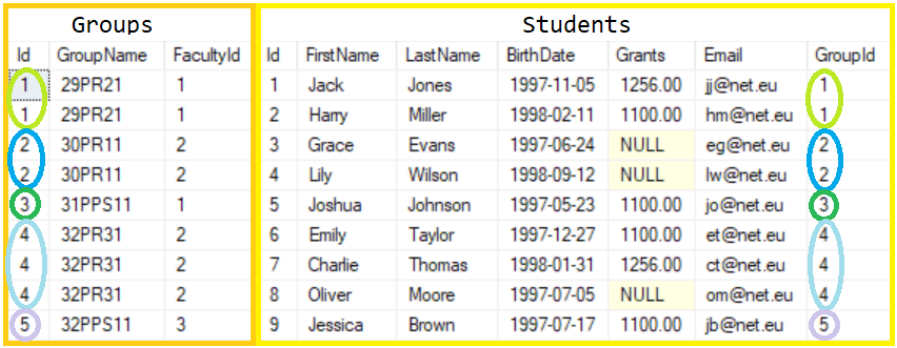
```
SELECT columnName1, columnName2, ...
FROM tableName1 [INNER] JOIN tableName2
ON tableName1.columnName =
                tableName2.columnName;
```

При выполнении оператора INNER JOIN каждая запись из первой таблицы сопоставляется с каждой записью из другой таблицы по условию, указанному после оператора ON, если условие выполняется, тогда строки записываются в результирующую таблицу, которая формируется путем конкатенации строк первой и второй таблиц.

В качестве первого примера внутреннего объединения приведем запрос, при помощи которого мы получим всю информацию о группах и студентах в этих группах:

```
SELECT *
FROM Groups INNER JOIN Students
ON Groups.Id = Students.GroupId;
```

В данном SQL-запросе после оператора SELECT специально прописан символ *, для того чтобы мы смогли увидеть все полученные столбцы (Рисунок 3.1).



Groups

Id	GroupName	FacultyId
1	29PR21	1
1	29PR21	1
2	30PR11	2
2	30PR11	2
3	31PPS11	1
4	32PR31	2
4	32PR31	2
4	32PR31	2
5	32PPS11	3

Students

Id	FirstName	LastName	BirthDate	Grants	Email	GroupId
1	Jack	Jones	1997-11-05	1256.00	jj@net.eu	1
2	Harry	Miller	1998-02-11	1100.00	hm@net.eu	1
3	Grace	Evans	1997-06-24	NULL	eg@net.eu	2
4	Lily	Wilson	1998-09-12	NULL	lw@net.eu	2
5	Joshua	Johnson	1997-05-23	1100.00	jo@net.eu	3
6	Emily	Taylor	1997-12-27	1100.00	et@net.eu	4
7	Charlie	Thomas	1998-01-31	1256.00	ct@net.eu	4
8	Oliver	Moore	1997-07-05	NULL	om@net.eu	4
9	Jessica	Brown	1997-07-17	1100.00	jb@net.eu	5

Рисунок 3.1. Результат использования внутреннего объединения в SQL-запросе

Как вы можете заметить, мы получили результирующую таблицу, которая фактически состоит из значений двух таблиц Groups и Students, связанных между собой уникальным идентификатором группы (Groups.Id).

Естественно, что использовать результаты запроса в таком виде не имеет никакого смысла, однако мы можем указать в операторе SELECT только необходимые нам столбцы, и в результирующей таблице будут именно их значения в указанной нами последовательности:

```
SELECT LastName, FirstName, Email, GroupName
FROM Groups INNER JOIN Students
ON Groups.Id = Students.GroupId;
```

Результат выполнения текущего SQL-запроса представлен на рисунке 3.2.

LastName	FirstName	Email	GroupName
Jones	Jack	jj@net.eu	29PR21
Miller	Harry	hm@net.eu	29PR21
Evans	Grace	eg@net.eu	30PR11
Wilson	Lily	lw@net.eu	30PR11
Johnson	Joshua	jo@net.eu	31PPS11
Taylor	Emily	et@net.eu	32PR31
Thomas	Charlie	ct@net.eu	32PR31
Moore	Oliver	om@net.eu	32PR31
Brown	Jessica	jb@net.eu	32PPS11

Рисунок 3.2. Информация о студентах

При использовании оператора INNER JOIN допускается не писать служебное слово INNER, в этом случае JOIN будет расцениваться как внутреннее объединение.

Как вы, наверное, догадались при помощи оператора INNER JOIN можно осуществлять объединение нескольких таблиц, продемонстрируем это на следующем примере. Допустим нам необходимо получить информацию об успеваемости всех студентов:

```
SELECT FirstName, LastName, Name AS Subject,
       Assesment, GroupName
FROM Groups AS G JOIN Students AS S
     ON G.Id = S.GroupId
JOIN Achievements AS A ON S.Id = A.StudentId
JOIN Subjects AS Sb ON Sb.Id = A.SubjectId;
```

При выполнении данного SQL-запроса нам потребовалось объединить таблицы Groups и Students по идентификатору группы (G.Id = S.GroupId), Achievements и Students связав их по идентификатору студента (S.Id = A.StudentId) и таблицы Subjects и Achievements – по идентификатору предмета (Sb.Id = A.SubjectId) (Рисунок 3.3).

FirstName	LastName	Subject	Assesment	GroupName
Jack	Jones	ADO.NET	11	29PR21
Harry	Miller	C#	10	29PR21
Jack	Jones	SQL Server	10	29PR21
Grace	Evans	C#	8	30PR11
Harry	Miller	ADO.NET	9	29PR21
Grace	Evans	SQL Server	7	30PR11

Рисунок 3.3. Информация об успеваемости студентов

При осуществлении внутреннего объединения вы, как и раньше, можете использовать все операторы языка SQL. Например, если потребуется получить оценки по предметам только у студентов 29 потока и отсортировать полученную информацию по фамилиям студентов, тогда нам достаточно применить операторы WHERE и ORDER BY к написанному ранее запросу (Рисунок 3.4):

```
SELECT FirstName, LastName, Name AS Subject,
       Assesment, GroupName
FROM Groups AS G JOIN Students AS S
     ON G.Id = S.GroupId
JOIN Achievements AS A ON S.Id = A.StudentId
JOIN Subjects AS Sb ON Sb.Id = A.SubjectId
WHERE GroupName LIKE '29%'
ORDER BY LastName;
```

FirstName	LastName	Subject	Assesment	GroupName
Jack	Jones	ADO.NET	11	29PR21
Jack	Jones	SQL Server	10	29PR21
Harry	Miller	C#	10	29PR21
Harry	Miller	ADO.NET	9	29PR21

Рисунок 3.4. Успеваемость студентов 29 потока

Необходимость использования внешнего объединения

При выполнении определенного вида SQL-запросов, которые связаны с получением полной информации из таблиц, использование внутреннего объединения не даст желаемого результата. Например, необходимо вывести информацию обо всех студентах и оценках, которые ими были получены. Запрос к базе данных University с применением оператора INNER JOIN будет выглядеть следующим образом:

```
SELECT FirstName + ' ' + LastName
       AS FullName, Assesment
FROM Students AS S INNER JOIN Achievements AS A
ON S.Id = A.StudentId;
```

В качестве результата выполнения данного запроса мы получим следующий набор данных (Рисунок 3.5).

FullName	Assesment
Jack Jones	11
Harry Miller	10
Jack Jones	10
Grace Evans	8
Harry Miller	9
Grace Evans	7

Рисунок 3.5. Результат выполнения внутреннего объединения

Однако полученный нами результат не удовлетворяет требуемому условию, ведь нам необходимо было получить данные обо всех студентах, а не только о тех из них кто получил оценки. Такое поведение характерно

при использовании оператора INNER JOIN, потому что в результирующую таблицу будут помещены только те записи, которые удовлетворяют условию, указанному после ключевого слова ON, в нашем случае соответствие уникального идентификатора студента (S.Id = A.StudentId).

Для того, чтобы получать правильные результаты при написании подобных SQL-запросов необходимо применять внешние объединения (OUTER JOIN), которые позволяют получить все данные из одной таблицы независимо от того существует ли для них совпадения строк в другой таблице. Существует три типа внешних объединений: левое (LEFT), правое (RIGHT) и полное (FULL). Общая форма записи внешних объединений выглядит следующим образом:

```
SELECT columnName1, columnName2, ...
FROM leftTable LEFT | RIGHT | FULL [OUTER]
                JOIN rightTable
ON tableName1.columnName =
               tableName2.columnName;
```

При написании запросов с использованием внешних объединений ключевое слово OUTER можно опустить, потому что ключевые слова LEFT, RIGHT и FULL однозначным образом определят его тип.

Понятие LEFT JOIN

Оператор LEFT JOIN (LEFT OUTER JOIN) позволяет создать, так называемое левое внешнее объединение, при выполнении которого в результирующий набор помимо строк, удовлетворяющих условию после ключевого слова ON, будут записаны даже те строки из таблицы,

расположенной слева от оператора, которые не удовлетворяют указанному условию.

Попробуем решить задачу обо всех студентах и их оценках поставленную в предыдущем разделе при помощи оператора LEFT JOIN и напишем для этого соответствующий SQL-запрос:

```
SELECT FirstName + ' ' + LastName
       AS FullName, Assesment
FROM Students AS S LEFT JOIN Achievements AS A
ON S.Id = A.StudentId;
```

В данном случае в результирующий набор попадут требуемые данные обо всех студентах из таблицы Students, независимо от того получали они оценки или нет. В том случае если в таблице Achievements отсутствует оценка для соответствующего студента, результат будет равен NULL (Рисунок 3.6).

FullName	Assesment
Jack Jones	11
Jack Jones	10
Harry Miller	10
Harry Miller	9
Grace Evans	8
Grace Evans	7
Lily Wilson	NULL
Joshua Johnson	NULL
Emily Taylor	NULL
Charlie Thomas	NULL
Oliver Moore	NULL
Jessica Brown	NULL

Рисунок 3.6. Результат выполнения левого внешнего объединения

Для того чтобы окончательно убедить вас в полезности внешних объединений, продемонстрируем вам два варианта решения еще одной задачи. Допустим, нам необходимо получить информацию о студентах, которые пока не получали оценок.

Одним из вариантов получения требуемых данных является SQL-запрос с обязательным использованием подзапроса:

```
SELECT FirstName + ' ' + LastName AS FullName
FROM Students
WHERE Id NOT IN (SELECT StudentId
      FROM Achievements);
```

В данном случае подзапрос возвращает уникальные идентификаторы всех студентов, которые получили оценки, и информация о которых находится в таблице Achievements. В основном запросе выводятся данные только о тех студентах, уникальный идентификатор которых отсутствует во множестве, сформированном подзапросом, то есть запись о них отсутствует в таблице Achievements, а значит они еще не получили ни одной оценки (Рисунок 3.7).

FullName
Lily Wilson
Joshua Johnson
Emily Taylor
Charlie Thomas
Oliver Moore
Jessica Brown

Рисунок 3.7. Студенты, которые не получили оценки

При написании подзапросов многие студенты сталкиваются с проблемой понимания их выполнения. Поэтому более изящным решением поставленной задачи без использования подзапроса является возможность применения в данном SQL-запросе оператора LEFT JOIN, который образует более понятную синтаксическую конструкцию. К тому же такой запрос выполнится быстрее, потому что для каждого LEFT JOIN создается отдельной поток и его выполнение происходит параллельно в отличие от подзапроса, выполнение которого осуществляется последовательно с основным запросом в одном потоке.

Для того чтобы понять, как решить поставленную задачу при помощи внешнего объединения, вам необходимо внимательно присмотреться к результату выполнения SQL-запроса на рисунке 3.6. Как вы заметили, у некоторых студентов оценка имеет неопределенное значение (NULL), то есть если отфильтровать записи по этому признаку, мы получим требуемый результат, который полностью совпадает с результатом, который был получен при выполнении предыдущего запроса (Рисунок 3.7):

```
SELECT FirstName + ' ' + LastName AS FullName
FROM Students AS S LEFT JOIN Achievements AS A
ON S.Id = A.StudentId
WHERE Assesment IS NULL;
```

Понятие RIGHT JOIN

При помощи оператора RIGHT JOIN (RIGHT OUTER JOIN) возможно создать, так называемое правое внешнее объединение, при выполнении которого в результирующий набор помимо строк, удовлетворяющих условию

после ключевого слова ON, будут записаны все строки из таблицы, расположенной справа от оператора, независимо от того удовлетворяют они указанному условию или нет.

В качестве первого примера использования оператора RIGHT JOIN продемонстрируем вам решение уже известной вам задачи — вывод информации обо всех студентах и полученных ими оценках:

```
SELECT FirstName + ' ' + LastName
       AS FullName, Assesment
FROM Achievements AS A RIGHT JOIN Students AS S
ON S.Id = A.StudentId;
```

В данном случае мы поменяли местами таблицы Achievements и Students и в результате выполнения данного запроса мы получили результат идентичный результату на рисунке 3.6. Таким образом, при помощи правого внешнего объединения можно получить результаты аналогичные результатам левого внешнего объединения, если изменить порядок следования таблиц в SQL-запросе.

В следующем примере мы продемонстрируем возможность совместного использования правого и левого объединений. Предположим, что нам необходимо получить список всех предметов и информацию о преподавателях, которые их читают, ниже представлен возможный вариант запроса:

```
SELECT [Name] AS [Subject], LastName, FirstName
FROM TeachersSubjects AS TS RIGHT JOIN Subjects
     AS S
ON S.Id = TS.SubjectId LEFT JOIN Teachers AS T
ON T.Id = TS.TeacherId
ORDER BY [Name];
```

Таблица TeachersSubjects содержит информацию о том, кто из преподавателей какой предмет читает. В таблице Subjects находится список всех предметов, то есть именно строки из этой таблицы нам необходимо получить, независимого от того, имеются ли соответствия в других связанных с ней таблицах. Поэтому мы объединили таблицы TeachersSubjects и Subjects при помощи правого внешнего соединения. Для того чтобы сохранились результаты первого объединения мы, при помощи левого внешнего объединения, связываем их с таблицей Teachers, в которой содержится информация о преподавателях. Из полученного результата видно, что предмет Unity на данный момент не читает ни один из преподавателей (Рисунок 3.8).

Subject	LastName	FirstName
ADO.NET	Cooper	Michael
C#	Nelson	Sophia
Discrete Math	Kirk	Emma
ITE1	Williams	Daniel
JavaScript	Nelson	Sophia
SQL Server	MacAlister	Henry
Unity	NULL	NULL
WIN10	Williams	Daniel

Рисунок 3.8. Список всех предметов и данные о преподавателях, которые их читают

Понятие FULL JOIN

Оператор FULL JOIN (FULL OUTER JOIN) позволяет создать полное внешнее объединение, при котором в результирующий набор включаются записи из левой

таблицы даже в том случае, если для них отсутствуют соответствующие записи в правой таблице, а записи из правой таблицы будут помещены в результирующую таблицу даже тогда, когда для них нет записей соответствия в левой таблице. Таким образом, полное внешнее соединение, по сути, является комбинацией левого и правого внешних объединений.

Приведем пример выполнения полного внешнего объединения для получения информации обо всех студентах и группах:

```
SELECT FirstName, LastName, GroupName
FROM Students AS S FULL JOIN Groups AS G
ON G.Id = S.GroupId
ORDER BY FirstName;
```

FirstName	LastName	GroupName
NULL	NULL	33GR12
Charlie	Thomas	32PR31
Charlotte	Becker	NULL
Emily	Taylor	32PR31
Grace	Evans	30PR11
Harry	Miller	29PR21
Jack	Jones	29PR21
Jessica	Brown	32PPS11
Joshua	Johnson	31PPS11
Lily	Wilson	30PR11
Oliver	Moore	32PR31

Рисунок 3.9. Информация обо всех студентах и группах

В полученной результирующей таблице находятся все записи из таблицы Students и все записи из таблицы

Groups там, где отсутствует соответствующая запись из другой таблицы, возвращается неопределенное значение (NULL). В данном случае в группе 33GR12 нет ни одного студента и студентка Charlotte Becker пока не зачислена ни в одну из групп (Рисунок 3.9).

4. Домашнее задание

1. Вам необходимо создать многотабличную базу данных Airport, в которой должна содержаться информация, связанная с работой аэропорта. В этой базе данных обязательно должны быть следующие таблицы: рейсы самолетов, билеты на рейсы (бизнес и эконом класс), пассажиры.
2. Используя полученные в этом уроке знания, вам необходимо написать к созданной базе Airport следующие SQL-запросы:
   - вывести все рейсы в определенный город на произвольную дату, упорядочив их по времени вылета;
   - вывести информацию о рейсе с наибольшей длительностью полета по времени;
   - показать все рейсы, длительность полета которых превышает два часа;
   - получить количество рейсов в каждый город;
   - показать город, в который наиболее часто осуществляются полеты;
   - получить информацию о количестве рейсов в каждый город и общее количество рейсов за определенный месяц;
   - вывести список рейсов, вылетающих сегодня, на которые есть свободные места в бизнес классе;
   - получить информацию о количестве проданных билетов на все рейсы за указанный день и их общую сумму;

- вывести информацию о предварительной продаже билетов на определенную дату с указанием всех рейсов и количестве проданных на них билетов;
- вывести номера всех рейсов и названия всех городов, в которые совершаются полеты из данного аэропорта.