

Database Theory



Урок № 5

Функции агрегирования

Содержание

1. Функции агрегирования	4
Функция COUNT.....	5
Функция AVG.....	7
Функция SUM.....	8
Функция MIN.....	8
Функция MAX	8
2. Понятие группировки.	
Ключевое слово GROUP BY	12
3. Ключевое слово HAVING.....	16
Принципы использования HAVING	16
Сравнительный анализ HAVING и WHERE.....	19
4. Подзапросы.....	21
Необходимость создания	
и использования подзапросов	21

Сравнение подзапросов и многотабличных запросов.....	25
Принцип работы подзапросов.....	28
5. Домашнее задание.....	32

1. Функции агрегирования

Как вы знаете, основным предназначением баз данных является упорядоченное хранение различной информации, которая может быть извлечена удобным способом в случае необходимости. Однако существует еще одно немаловажное применение баз данных — статистический анализ хранимой информации. Для того, чтобы получить статистические данные в цифровом виде используются функции агрегирования, которые предназначены для получения обобщающего значения из определенного количества столбцов, то есть результатом выполнения любой функции будет единственное значение.

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 1.1. Таблица Students

В языке T-SQL существует пять функций агрегирования: **COUNT()**, **AVG()**, **SUM()**, **MIN()** и **MAX()**. Прежде чем приступить к их подробному рассмотрению, следует отметить немаловажную особенность — функции

AVG() и **SUM()** могут применяться только к столбцам цифровых типов данных, а функции **COUNT()**, **MIN()** и **MAX()** как к полям цифровых, так и символьного типа данных.

Для того, чтобы продемонстрировать практическое применение функций агрегирования, мы воспользуемся таблицей *Students*, известной вам из предыдущего урока (Рисунок 1.1).

Функция **COUNT**

Функция **COUNT()** позволяет определить количество записей в определенном столбце либо во всей таблице.

Если вызвать функцию **COUNT()** и передать в качестве параметра символ * (звездочка):

```
SELECT COUNT(*) AS [Number of records]
FROM Students;
```

То в результате вы получите общее количество записей в таблице, включая как повторяющиеся, так и неопределенные значения (NULL-значения) (Рисунок 1.2).

Number of records
9

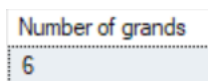
Рисунок 1.2. Определение общего количества записей в таблице

Для того чтобы получить количество записей в конкретном столбце таблицы, при вызове функции **COUNT()** необходимо передать в качестве параметра название требуемого столбца. При использовании функции в таком виде не учитываются записи, имеющие неопределенное

значение в данном столбце. Например, SQL-запрос возвращающий количество записей в столбце **Grants** будет выглядеть следующим образом:

```
SELECT COUNT(Grants) AS [Number of grants]  
FROM Students;
```

Результат, полученный при выполнении этого SQL-запроса, отличается от результата, представленного на рисунке 1.2, так как в столбце **Grants** имеются NULL-значения (Рисунок 1.3).



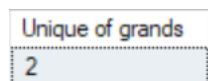
Number of grants
6

Рисунок 1.3. Количество записей
в столбце Grants

В том случае если вам требуется определить количество уникальных записей в определенном столбце, тогда при вызове функции **COUNT()** необходимо указать ключевое слово **DISTINCT** перед именем столбца. Внесем изменения в предыдущий пример:

```
SELECT COUNT(DISTINCT Grants) AS [Unique of grants]  
FROM Students;
```

В результате выполнения данного SQL-запроса мы получим количество записей в столбце **Grants** без повторений (Рисунок 1.4).



Unique of grants
2

Рисунок 1.4. Количество записей
в столбце Grants без повторений


Функция AVG

Функция **AVG()** позволяет получить среднее арифметическое значений определенного столбца.

Использование функции **AVG()** рассмотрим на простом примере нахождения размера средней стипендии студентов, соответствующий SQL-запрос будет выглядеть следующим образом:

```
SELECT AVG(Grants) AS [Average grant]
FROM Students;
```

Результат выполнения данного запроса представлен на рисунке 1.5.




Average grant
1152.000000

Рисунок 1.5. Средняя стипендия студентов

Следующий пример будет немного сложнее, допустим нам необходимо определить средний возраст студентов, тогда мы можем написать следующий SQL-запрос:

```
SELECT AVG(DATEDIFF(dd, BirthDate,
GETDATE())/365.25) AS [Average age]
FROM Students;
```

Для того чтобы определить возраст студентов мы используем функцию **DATEDIFF()**, получая разницу в днях между текущей датой и датой рождения студента, производя деление полученного результата на 365.25 мы получаем более точные значения (Рисунок 1.6).



Average age
20.789717

Рисунок 1.6. Средний возраст студентов

Функция SUM

Функция **SUM()** позволяет вычислить сумму значений заданного столбца.

Например, выполнив следующий SQL-запрос, мы сможем узнать, сколько денег выплачивается студентам ежемесячно в качестве стипендии, результат представлен на рисунке 1.7.

```
SELECT SUM(Grants) AS [Sum grants]  
FROM Students;
```



Sum grants
6912.00

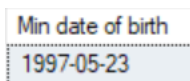
Рисунок 1.7. Сумма стипендий студентов

Функция MIN

Использование функции **MIN()** обеспечивает получение самого малого значения в определенном столбце таблицы.

Для того чтобы определить наименьшую дату рождения студентов необходимо выполнить следующий запрос, результат которого представлен на рисунке 1.8:

```
SELECT MIN(BirthDate) AS [Min date of birth]  
FROM Students;
```



Min date of birth
1997-05-23

Рисунок 1.8. Минимальная дата рождения студентов

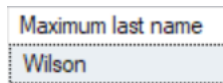
Функция MAX

Функция **MAX()** позволяет определить наибольшее значение в указанном столбце.

Как говорилось ранее, функции **MIN()** и **MAX()** могут использоваться при работе со столбцами символьного типа, в этом случае будут сравниваться цифровые коды соответствующих символов. Например, нам необходимо определить максимальное значение из фамилий студентов, то есть найти фамилию, у которой значение символов наибольшее. Ведь на самом деле будут последовательно сравниваться не сами символы, а их числовые значения в кодировке Unicode, так как в нашем случае столбец **Last-Name** имеет тип данных **nvarchar**. Для того чтобы решить эту поставленную задачу можно написать следующий SQL-запрос:

```
SELECT MAX(LastName) AS [Maximum last name]
FROM Students;
```

Результат этого запроса представлен ниже (Рисунок 1.9).



Maximum last name
Wilson

Рисунок 1.9. Максимальная фамилия
из списка студентов

При написании SQL-запросов с использованием функций агрегирования, при необходимости, можно осуществлять ограничение результата по условию, например, нам нужно получить количество студентов, имя которых начинается на букву **J**. Выполнив следующий запрос, вы получите требуемый результат (Рисунок 1.10):

```
SELECT COUNT(*) AS [Number of students]
FROM Students
WHERE FirstName LIKE 'J%';
```

Number of students
3

Рисунок 1.10. Количество студентов, имя которых начинается на букву J

Для того чтобы у вас не сложилось впечатление, что функции агрегирования применяются только в простых запросах мы, в заключение текущего раздела, продемонстрируем пример многотабличного SQL-запроса. Для того чтобы освежить вашу память, мы приведем часть диаграммы базы данных из прошлого урока (Рисунок 1.11).

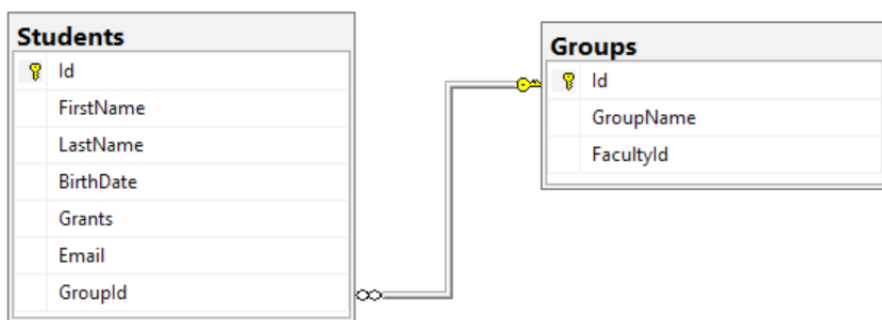


Рисунок 1.11. Связь между таблицами Students и Groups

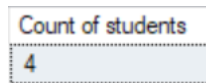
Допустим нам необходимо узнать количество студентов, которые учатся на 32 потоке:

```

SELECT COUNT(*) AS [Count of students]
FROM Students AS S, Groups AS G
WHERE G.Id = S.GroupId
AND G.GroupName LIKE '32%';
  
```

В текущем запросе для того чтобы получить из базы данных нужную информацию необходимо установить

связь между таблицами `Students` и `Groups` по первичному/внешнему ключу (`G.Id = S.GroupId`) и оставить только те записи, которые начинаются на 32 (`G.GroupName LIKE '32%'`), после чего получить их количество при помощи функции `COUNT()`. Результат данного запроса представлен на рисунке 1.12.



Count of students
4

Рисунок 1.12. Количество студентов 32-го потока

2. Понятие группировки. Ключевое слово GROUP BY

Все примеры предыдущего раздела возвращали в качестве результатов SQL-запросов только значения функций агрегирования. А что произойдет, если нам понадобится дополнительная информация? Например, необходимо узнать какое количество студентов учится в каждой группе.

«Что тут сложного? — скажете вы. — Нужно добавить в оператор **SELECT** название группы». Давайте попробуем написать соответствующий запрос, выполнение которого завершится ошибкой (Рисунок 2.1).

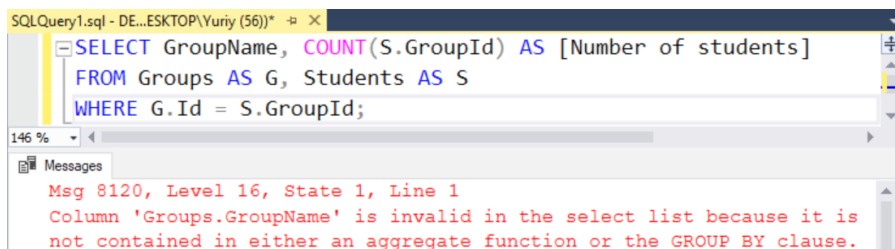


Рисунок 2.1. Ошибка: недопустимо использовать столбец GroupName в списке выбора

Ошибка произошла потому что любая функция агрегирования (в данном случае **COUNT()**) всегда возвращает одно единственное значение, а обращение к столбцу GroupName в операторе **SELECT** возвращает множество значений строк и такая ситуация не может быть корректно обработана оператором **SELECT**. Если представить

невозможный результат выполнения этого SQL-запроса, то выглядел бы он следующим образом (Рисунок 2.2).

GroupName	Number of students
29PR21	2
29PR21	
30PR11	2
30PR11	
31PPS11	1
32PR31	3
32PR31	
32PR31	
32PPS11	1

Рисунок 2.2. Невозможный результат использования функций агрегирования

Как вы можете заметить, нескольким названиям группы соответствует одно значение функции агрегирования, а это в принципе невозможно.

Для того чтобы данный SQL-запрос выполнялся правильно, необходимо сделать так чтобы название группы тоже было в единственном числе, то есть каким-то образом сгруппировать значения в столбце **GroupName** (Рисунок 2.3).

GroupName	Number of students
29PR21	2
30PR11	2
31PPS11	1
32PPS11	1
32PR31	3

Рисунок 2.3. Требуемый результат выполнения запроса

Намек на решение данной проблемы сделан в тексте, который описывает ошибку — необходимо использовать

оператор **GROUP BY**, который позволяет группировать результирующие строки по указанному столбцу:

```
SELECT GroupName, COUNT(S.GroupId)
      AS [Number of students]
FROM Groups AS G, Students AS S
WHERE G.Id = S.GroupId
GROUP BY GroupName;
```

Использование оператора **GROUP BY** в предыдущем запросе позволило создать группы строк, по одной для каждого уникального значения в столбце **GroupName**. После чего при помощи функции **COUNT()** было получено количество записей в каждой группе строк и уже итоговые строки были включены в результирующую таблицу. Результат выполнения этого SQL-запроса совпадает с ожидаемым (Рисунок 2.3).

При необходимости вы можете группировать строки на основе нескольких столбцов, однако в этом случае группирование будет осуществляться, основываясь на уникальном сочетании данных во всех столбцах, которые участвуют в группировке.

Например, если нам понадобится получить количество студентов с одинаковой стипендией по группам с отображением этих стипендий, то в группировке данных помимо названий групп будут участвовать еще и значения стипендий:

```
SELECT GroupName, Grants, COUNT(S.GroupId)
      AS [Number of students]
FROM Groups AS G, Students AS S
WHERE G.Id = S.GroupId
GROUP BY GroupName, Grants;
```

В результате выполнения данного SQL-запроса мы получим количество студентов в группе больше одного только тогда, когда размер стипендии совпадет у нескольких студентов в этой группе. В нашем случае в группе 30PR11 два студента не получают стипендию, тем самым формируя для запроса одинаковое значение — NULL, это происходит потому что функция **GROUP BY** интерпретирует все NULL-значения как равные. Во всех остальных группах значения стипендий у студентов не совпадает (Рисунок 2.4).

GroupName	Grants	Number of students
30PR11	NULL	2
32PR31	NULL	1
29PR21	1100.00	1
31PPS11	1100.00	1
32PPS11	1100.00	1
32PR31	1100.00	1
29PR21	1256.00	1
32PR31	1256.00	1

Рисунок 2.4. Количество студентов в каждой из групп с одинаковой стипендией

3. Ключевое слово HAVING

При использовании функций агрегирования существует необходимость накладывать ограничения на возвращаемые ими результаты, именно это является одним из предназначений оператора **HAVING**. Еще одна возможность применения этого оператора — фильтрация результатов группировки строк на основании значений столбцов, указанных после оператора **GROUP BY**.

Принципы использования HAVING

Оператор **HAVING** синтаксически прописывается после оператора **GROUP BY**, но до оператора **ORDER BY**:

```
SELECT columnName1, columnName2, ...  
FROM tableName  
[WHERE condition]  
[GROUP BY columnName1, columnName2, ...]  
HAVING condition  
[ORDER BY columnName1 ASC | DESC, ...];
```

В действительности наличие операторов **WHERE**, **GROUP BY** и **ORDER BY** является необязательным.

Приведем несколько примеров использования оператора **HAVING**. В первом из них запрос вернет только тех студентов, средняя стипендия которых не превышает 1200, при этом результат будет отсортирован по их фамилиям:

```
SELECT LastName, Grants  
FROM Students  
GROUP BY LastName, Grants
```



```
HAVING AVG(Grants) <= 1200
ORDER BY LastName;
```

Данный SQL-запрос будет выполняться следующим образом: все записи из таблицы **Students** будут сгруппированы по фамилиям и стипендиям студентов (оператор **GROUP BY**), потом полученные группы фильтруются по условию в операторе **HAVING** и после этого полученные результаты сортируются по фамилиям (Рисунок 3.1).

LastName	Grants
Brown	1100.00
Johnson	1100.00
Miller	1100.00
Taylor	1100.00

Рисунок 3.1. Студенты со средней стипендией меньше 1200

При помощи следующего SQL-запроса мы сможем определить название групп, количество студентов в которых превышает 2 человека:

```
SELECT GroupName, COUNT(*)
AS [Number of students]
FROM Groups AS G, Students AS S
WHERE G.Id = S.GroupId
GROUP BY GroupName
HAVING COUNT(S.GroupId)>2;
```

В этом запросе соединяем таблицы **Students** и **Groups** по первичному/внешнему ключу (**G.Id = S.GroupId**), группируем полученные строки по названию группы, после чего ограничиваем количество записей по условию в операторе **HAVING**, сравнивая количество идентификаторов групп

у студентов (**COUNT**(S.GroupId)>2). Результат данного запроса представлен на рисунке 3.2.

GroupName	Number of students
32PR31	3

Рисунок 3.2. Группы, в которых учатся больше 2 человек

Еще один SQL-запрос демонстрирует возможность проверки в операторе **HAVING** значений одного из столбцов, по которым группируются данные. Допустим необходимо вывести имена и фамилии студентов из определенного списка:

```
SELECT FirstName, LastName
FROM Students
GROUP BY LastName, FirstName
HAVING LastName IN ('Moore', 'Thomas', 'Doe');
```

В текущем запросе записи группируются по фамилии и имени, потом остаются только те строки, у которых фамилии совпадают с заданным в операторе **HAVING** списком и только эта информация попадает в результирующую таблицу (Рисунок 3.3).

FirstName	LastName
Oliver	Moore
Charlie	Thomas

Рисунок 3.3. Информация по студентам с определенными фамилиями

Последний запрос демонстрирует возможность использования оператора **HAVING** при отсутствии оператора

GROUP BY. Вывести минимальные значения фамилий студентов, если средняя стипендия превышает 1100:

```
SELECT MIN(LastName) AS [Minimum last name]
FROM Students
HAVING AVG(Grants)>1100;
```

В операторе **SELECT** предыдущего SQL-запроса мы можем вызывать только функции агрегирования, если попытаться указать название любого столбца, то будет сгенерирована ошибка, с требованием использовать оператор **GROUP BY**. Результат выполнения этого запроса представлен на рисунке 3.4.

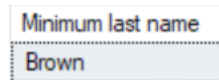


Рисунок 3.4. Использование оператора HAVING без GROUP BY

Если в операторе **HAVING** изменить условие — увеличить сравниваемое значение, например, до 1400, то мы получим пустой результирующий набор, потому что таких средних значений в таблице **Students** нет.

Сравнительный анализ HAVING и WHERE

Операторы **WHERE** и **HAVING** похожи по способу их использования и тот, и другой содержит условия фильтрации данных, однако эти операторы отличаются своим назначением. Условия после оператора **WHERE** определяют, каким образом соединять используемые таблицы и фильтруют отдельные строки данных, тем самым ограничивая количество выводимых записей в результирующей

таблице. В свою очередь оператор **HAVING** предназначен для фильтрации сгруппированных данных или выполнения условий для функций агрегирования, именно поэтому использование оператора **WHERE** в данных ситуациях невозможно. Например, если мы напишем SQL-запрос с использованием оператора **WHERE** для того чтобы получить информацию о студентах, средняя стипендия которых не превышает 1200, то его выполнение приведет к ошибке (Рисунок 3.5).

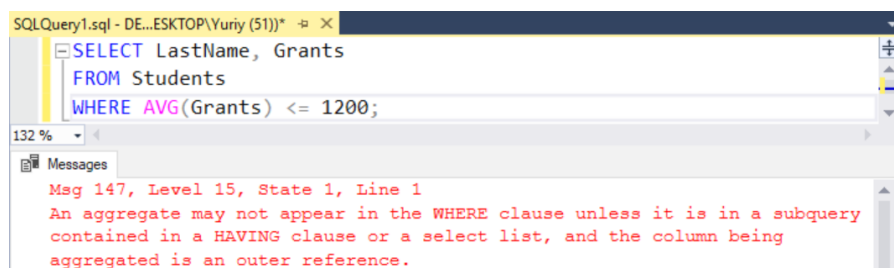


Рисунок 3.5. Ошибка: невозможно использовать агрегатную функцию

4. Подзапросы

В языке SQL реализована возможность создания подзапросов или другими словами вложенных запросов, то есть возможность создания такого запроса, который будет помещен внутрь другого запроса. Существуют ситуации, когда без такого подхода получить требуемую информацию будет очень сложно или вообще невозможно.

Необходимость создания и использования подзапросов

Обсудим необходимость использования подзапросов на простом примере, допустим нам необходимо вывести фамилию, имя и номер группы студентов, которые получают максимальную стипендию. Казалось бы, что написать такой SQL-запрос не составит особого труда, например, так:

```
SELECT LastName, FirstName, GroupName, Grants
FROM Students AS S, Groups AS G
WHERE G.Id = S.GroupId
GROUP BY LastName, FirstName, GroupName, Grants
HAVING Grants = MAX(Grants);
```

Однако полученный результат будет далек от ожидаемого, в данном случае мы получили максимальное значение стипендии для каждой группы данных (Рисунок 4.1).

Такой результат выполнения SQL-запроса легко объясним. В данном случае данные сгруппированы таким образом, что формируют уникальные группы на основании

фамилии и имени студента, названии группы и стипендии, а в операторе **HAVING** сравнивается значение стипендии для каждой группы данных с максимальным значением стипендии той же группы, то есть само с собой, тем самым возвращая истину. Группы, у которых максимальная стипендия имеет неопределенное значение (**Grants = NULL**) отбрасываются, так как сравнение на **NULL**-значение возможно только с использованием ключевого слова **IS NULL**.

LastName	FirstName	GroupName	Grants
Brown	Jessica	32PPS11	1100.00
Johnson	Joshua	31PPS11	1100.00
Jones	Jack	29PR21	1256.00
Miller	Harry	29PR21	1100.00
Taylor	Emily	32PR31	1100.00
Thomas	Charlie	32PR31	1256.00

Рисунок 4.1. Результат неправильно написанного SQL-запроса

Для того чтобы наш SQL-запрос вернул требуемые результаты, необходимо сравнивать стипендии студентов с максимальным значением стипендии (в нашем случае оно равно 1256). Однако, как вы знаете из предыдущего раздела, использование функций агрегирования в операторе **WHERE** запрещено, поэтому запрос можно написать следующим образом:

```
SELECT LastName, FirstName, GroupName, Grants
FROM Students AS S, Groups AS G
WHERE G.Id = S.GroupId AND Grants = 1256;
```

Результат данного запроса представлен на рисунке 4.2.

LastName	FirstName	GroupName	Grants
Jones	Jack	29PR21	1256.00
Thomas	Charlie	32PR31	1256.00

Рисунок 4.2. Студенты, которые получают максимальную стипендию

Несмотря на то, что в результате выполнения этого SQL-запроса мы получили правильный результирующий набор, такой подход не выдерживает никакой критики, ведь максимальное значение стипендии может меняться, что в свою очередь приведет к необходимости изменения «магического числа».

Поэтому для получения максимального значения стипендии целесообразно использовать соответствующий подзапрос (`SELECT MAX(Grants) FROM Students`). Полученное в результате его выполнения значение будет сравниваться со стипендией в каждой записи студентов и в случае совпадения значений, именно эта запись будет добавляться в результирующую таблицу. Перепишем предыдущий SQL-запрос с использованием подзапроса, результат его выполнения будет таким же, как на рисунке 4.2:

```
SELECT LastName, FirstName, GroupName, Grants
FROM Students AS S, Groups AS G
WHERE G.Id = S.GroupId
      AND Grants = (SELECT MAX(Grants)
FROM Students);
```

Дополняя комментарии к данному запросу, следует добавить, что в соответствии с синтаксисом языка SQL подзапросы необходимо помещать в круглые скобки.

В предыдущем примере мы использовали подзапрос, который возвращал скалярное значение, как результат выполнения функции агрегирования **MAX()**. Однако SQL-запросы в основном возвращают множество строк и в этом случае при использовании их в качестве подзапросов существует своя особенность. Например, нам необходимо получить информацию обо всех студентах, которые учатся в группах с номером 11 при этом нам неважно на каком потоке. В этом случае подзапрос выполнится правильно и вернет неопределенное количество значений одного столбца (**Id**), но неправильное сравнение возвращаемых им результатов приведет к ошибке на рисунке 4.3.

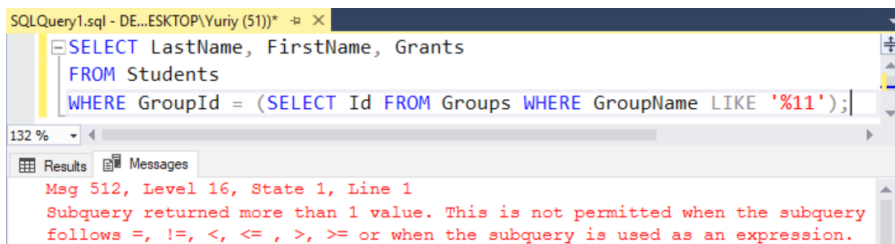


Рисунок 4.3. Ошибка: невозможно применять логические операторы в том случае если запрос возвращает больше одного значения

Данная ошибка связана с тем, что наш подзапрос возвращает уникальные идентификаторы нескольких групп, которые соответствуют заданному условию (**GroupName LIKE '%11'**), а в операторе **WHERE** возвращаемые результаты сравниваются с единственным значением столбца **GroupId**, что в принципе невозможно (подобная ситуация уже описывалась в текущем уроке при использовании функций агрегирования). Поэтому во всех случаях, когда

запрос может вернуть несколько значений, для их сравнения вместо логических операторов следует использовать ключевое слово **IN**:

```
SELECT LastName, FirstName, GroupId
FROM Students
WHERE GroupId IN (SELECT Id
FROM Groups WHERE GroupName LIKE '%11');
```

Результат выполнения текущего запроса представлен на рисунке 4.4.

LastName	FirstName	GroupId
Evans	Grace	2
Wilson	Lily	2
Johnson	Joshua	3
Brown	Jessica	5

Рисунок 4.4. Студенты, которые учатся в группах с номером 11

Сравнение подзапросов и многотабличных запросов

Как многотабличные запросы, так и подзапросы можно отнести к категории «сложных» запросов. Однако если многотабличные запросы возвращают данные, полученные в результате соединения нескольких реально существующих таблиц, то подзапросы формируют виртуальные значения, которые используются для сравнения с реально существующими данными. Подзапросы позволяют обеспечить значительную гибкость при выполнении запросов, потому что их можно использовать не только в операторе **WHERE** (что было уже продемонстрировано), но и в операторах **SELECT**, **FROM** и **HAVING**.

В качестве примера мы напишем подзапрос, который вернет данные (в виде виртуальной таблицы), на основании которых мы выведем информацию основного запроса. Такое использование подзапросов в операторе **FROM** может быть возможным, только если для результатов выполнения подзапроса будет указан псевдоним с использованием ключевого слова **AS**.

Для того чтобы вам было легче понять следующий SQL-запрос мы приведем часть диаграммы базы данных University из прошлого урока (Рисунок 4.5).

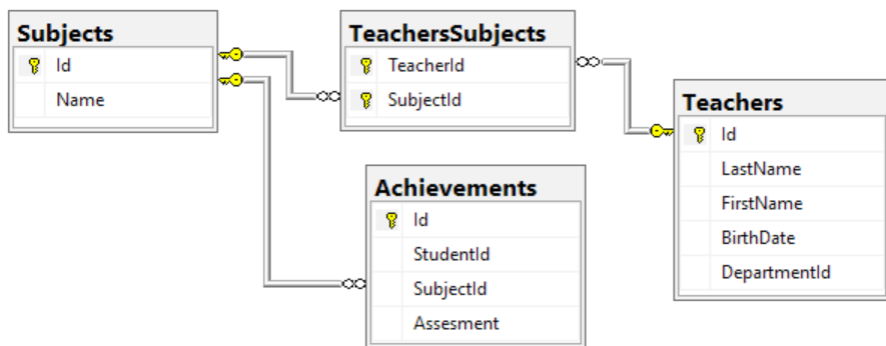


Рисунок 4.5. Диаграмма базы данных University (частично)

Предположим нам необходимо вывести фамилии преподавателей, читающих предметы, по которым студенты получают отличные оценки, для этого выполним следующий запрос:

```

SELECT T.LastName, M.SubjectName
FROM Teachers AS T, TeachersSubjects AS TS,
(SELECT S.Id AS SubId, S.Name AS SubjectName
FROM Subjects AS S, Achievements AS A
WHERE S.Id = A.SubjectId
  
```

```
GROUP BY S.Name, S.Id
HAVING MAX(A.Assesment) >= 10) AS M
WHERE T.Id = TS.TeacherId
      AND TS.SubjectId = M.SubId;
```

В данном случае наш подзапрос формирует виртуальную таблицу, которая возвращает уникальный идентификатор и названия предметов, по которым студенты получают только отличные оценки (**MAX(A.Assesment) >= 10**). Для этой таблицы указан псевдоним (**AS M**), благодаря чему основной запрос может получать доступ к значениям ее столбцов: осуществлять соединение (**TS.SubjectId = M.SubId**) и выводить данные (**M.SubjectName**). Результат выполнения данного запроса представлен на рисунке 4.6.

LastName	SubjectName
Nelson	C#
MacAlister	SQL Server
Cooper	ADO.NET

Рисунок 4.6. Преподаватели и предметы, по которым студенты получают отличные оценки

В следующем SQL-запросе мы продемонстрируем возможность использования подзапросов в операторе **HAVING**. При помощи этого запроса мы получим список преподавателей, среднее значение месяца рождения которых больше среднего значения месяца рождения студентов:

```
SELECT LastName, FirstName
FROM Teachers
GROUP BY LastName, FirstName
HAVING AVG(MONTH(BirthDate)) >
      (SELECT AVG(MONTH(BirthDate)) FROM Students);
```

В этом случае подзапрос возвращает среднее значение месяцев рождения студентов, которое в операторе **HAVING** сравнивается со средним значением месяца рождения каждого преподавателя. Результат выполнения данного запроса представлен на рисунке 4.7.

LastName	FirstName
Williams	Daniel
Cooper	Michael
Nelson	Sophia

Рисунок 4.7. Использование подзапроса в операторе HAVING

Принцип работы подзапросов

В том случае если в SQL-запросе применяются подзапросы, то выполнение SQL-операторов в первую очередь происходит в подзапросе, а уже потом полученные результаты используются в основном запросе. В одном SQL-запросе может быть несколько подзапросов при этом они могут быть вложены друг в друга, в этом случае выполнение начинается с подзапроса, который имеет наиболее глубокое вложение.

Продemonстрируем это при помощи следующего SQL-запроса, который выведет название групп, студенты которых получают максимальную стипендию:

```
SELECT GroupName
FROM Groups
WHERE Id IN (SELECT GroupId
FROM Students
WHERE Grants = (SELECT MAX(Grants) FROM Students));
```

В данном случае первым выполнится подзапрос (`SELECT MAX(Grants) FROM Students`), который вернет размер максимальной стипендии. Потом его результаты будут использованы в подзапросе (`SELECT GroupId FROM Students WHERE Grants = (...)`) для определения уникальных идентификаторов групп, полученные результаты в свою очередь будут использоваться при выполнении основного SQL-запроса (Рисунок 4.8).



GroupName
29PR21
32PR31

Рисунок 4.8. Группы, в которых студенты получают максимальную стипендию

Следует заметить, что существует еще один вид подзапросов, которые называются связанными или коррелированными подзапросами. Особенностью выполнения таких подзапросов является их зависимость от значений в основном запросе и поэтому они не могут быть обработаны раньше, чем основной запрос, для лучшего понимания принципа их использования приведем пример. Допустим нам необходимо вывести максимальную оценку, полученную студентами по каждому предмету. Такую информацию можно получить, написав SQL-запрос и по-другому, но мы в данном случае решили продемонстрировать использование связанного подзапроса в операторе `SELECT`. Необходимо заметить, что обязательным условием такого использования является возврат подзапросом одного значения, для этих целей идеально подходят функции агрегирования:

```
SELECT Subjects.Name, (SELECT MAX(A.Assessment)
FROM Achievements AS A
WHERE Subjects.Id = A.SubjectId) AS Maximum
FROM Subjects;
```

Выполнение текущего связанного подзапроса зависит от сравнения уникального идентификатора предмета (`Subjects.Id = A.SubjectId`), значение которого изменяется по мере построчного прочтения записей в таблице `Subjects`, поэтому количество строк в полученной виртуальной таблице соответствует количеству записей в таблице `Subjects` (Рисунок 4.9).

Name	Maximum
C#	10
Discrete Math	NULL
SQL Server	10
ADO.NET	11
ITE1	NULL
JavaScript	NULL
WIN10	NULL

Рисунок 4.9. Максимальные оценки студентов по каждому предмету

5. Домашнее задание

1. Вам необходимо создать многотабличную базу данных, содержащую информацию о вымышленном фитнес клубе, которая должна содержать следующие таблицы: инструкторы, секции, посетители и т.д.
2. Используя полученные в этом уроке знания, написать к созданной вами базе данных следующие SQL-запросы:
 - вывести количество инструкторов по каждой секции;
 - показать количество людей, которые должны заниматься в определенный момент времени в каждой секции;
 - вывести количество посетителей фитнес клуба, которые пользуются услугами определенного мобильного оператора;
 - получить количество посетителей, у которых фамилия совпадает с фамилиями из определенного списка;
 - показать количество людей с одинаковыми именами, которых занимают у определенного инструктора;
 - получить информацию о людях, которые посетили фитнес зал минимальное количество раз;
 - вывести количество посетителей, которые занимались в определенной секции за первую половину текущего года;
 - определить общее количество людей, посетивших фитнес зал за прошлый год.

Урок № 5

Функции агрегирования

© Юрий Задерей.

© Компьютерная Академия «Шаг».

www.itstep.org

Все права на охраняемые авторским правом фото-, аудио- и видеопроизведения, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объёме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объём и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.