Оглавление

Группировка и агрегирование

Логический порядок обработки инструкции SELECT

Следующие список демонстрирует логический порядок обработки или порядок привязки инструкции SELECT. Этот порядок определяет, когда объекты, определенные в одном шаге, становятся доступными для предложений в последующих шагах. Например, если обработчик запросов можно привязать (для доступа) к таблицам или представлениям, определенным в предложении FROM, эти объекты и их столбцы становятся доступными для всех последующих шагов. И наоборот, поскольку предложение SELECT является шагом 8, любые псевдонимы столбцов или производных столбцов, определенные в этом предложении, не могут быть объектом для ссылки предыдущих предложений. Вместе с тем к ним могут обращаться последующие предложения, например предложение ORDER BY. Фактическое физическое выполнение оператора определяется обработчиком запросов, и порядок может отличаться от этого списка.

1. FROM
2. ON
3. JOIN
4. WHERE
5. GROUP BY
6. WITH CUBE или WITH ROLLUP
7. HAVING
8. SELECT
9. DISTINCT
10. ORDER BY
11. В начало

```
select
       surname
       ,datediff(yy, birthday, getdate()) as 'old'
       ,stipend
from STUDENTS where stipend > 700
```

MySQL

```
select
       surname
       ,timestampdiff(YEAR, birthday, current_date()) as 'old'
       ,stipend
from STUDENTS where stipend > 700
```

id	surname	name	gender	stipend	course	city	birthday	univ_id	old
1	Кабанов	Виталий	m	550.00	4	Харьков	01.12.1990	2	
2	Павленко	Игорь	m	600.00	1	Киев	21.06.1993	5	
3	Цилюрик	Тимофей	m	600.00	4	Херсон	05.11.1990	3	
4	Козьменко	Игнат	m	500.00	1	Киев	26.04.1994	6	
5	Ориненко	Анатолий	m	450.00	4	Львов	08.09.1990	4	
6	Березовский	Роман	m	750.00	2	Киев	09.03.1992	1	25
7	Пименчук	Дмитрий	m	800.00	1	Харьков	20.11.1992	2	25
8	Шуст	Марина	f	750.00	1	Херсон	21.04.1993	3	24
9	Печальнова	Алина	f	500.00	2	Киев	25.02.1993	6	
10	Денисенко	Марк	m	500.00	3	Киев	07.06.1993	1	
11	Корсунская	Вера	f	650.00	2	Ровно	11.07.1992	NULL	
12	Чайка	Ольга	f	550.00	5	Киев	30.12.1990	5	
13	Кожушко	Кристина	f	650.00	3	Львов	18.03.1991	4	
14	Федосеева	Нина	f	750.00	2	Днепр	22.05.1992	7	25
15	Тарара	Леонид	m	900.00	4	Киев	04.05.1991	6	26
16	Земляний	Данил	m	550.00	5	Львов	06.03.1988	4	
17	Осипуков	Виталий	m	500.00	1	Днепр	10.08.1993	7	
18	Зевцов	Андрей	m	650.00	3	Львов	24.10.1991	4	
19	Бородюк	Никита	m	650.00	2	Киев	30.08.1990	6	
20	Миланивская	Ольга	f	750.00	5	Киев	22.10.1989	5	28
21	Запорожец	Владимир	m	750.00	5	Львов	NULL	5	NULL
22	Саенко	Сергей	m	950.00	1	Полтава	17.04.1993	9	24
23	Коваленко	Дмитрий	m	650.00	1	Хмельницкий	28.07.1993	7	
24	Маруга	Ольга	f	850.00	4	Севастополь	03.08.1990	4	27
25	Михайловская	Светлана	f	600.00	2	Винница	18.11.1992	13	25
26	Кораблев	Станислав	m	750.00	2	Горловка	04.05.1992	12	25
27	Бородавко	Михаил	m	750.00	5	Кременчуг	03.01.1989	2	28
28	Литвин	Олег	m	450.00	3	Ровно	10.06.1991	14	
29	Завакевич	Василий	m	800.00	4	Мариуполь	13.11.1990	1	27
30	Забила	Алексей	m	450.00	3	Киев	21.06.1991	13	
31	Залыева	Фатима	f	650.00	4	Горловка	18.05.1990	3	
32	Мироненко	Евгений	m	450.00	2	Белая Церковь	01.09.1992	14	
33	Шелест	Юрий	m	450.00	2	Киев	08.11.1992	4	
34	Сабатовская	Ирина	f	650.00	4	Одесса	14.02.1990	11	
35	Ремезов	Трофим	m	600.00	3	Киев	24.07.1991	8	
36	Винник	Артур	m	500.00	4	Ивано-Франковск	04.03.1990	8	
37	Запорожец	Виталий	m	350.00	5	Луцк	28.02.1989	13	
38	Наталич	Вадим	m	600.00	3	Макеевка	09.05.1991	6	
39	Богацкий	Дмитрий	m	350.00	5	Черновцы	12.10.1989	12	
40	Кот	Екатерина	f	700.00	3	Киев	05.04.1991	14	
41	Денисюк	Данил	m	350.00	4	Херсон	26.01.1990	2	
42	Гладкий	Денис	m	450.00	4	Ровно	20.12.1990	13	
43	Рубцова	Евгения	f	600.00	3	Донецк	10.10.1991	11	
44	Гузенко	Александра	f	500.00	1	Ровно	09.11.1993	8	
45	Рудавский	Игорь	m	500.00	4	Винница	23.10.1990	2	

* – выделены строки, которые не попадут в результирующую выборку
* - выделены столбцы, которые будут включены в результирующую выборку

В результате получаем выборку

surname	old	stipend
Березовский	25	750.00
Пименчук	25	800.00
Шуст	24	750.00
Федосеева	25	750.00
Тарара	26	900.00
Миланивская	28	750.00
Запорожец	NULL	750.00
Саенко	24	950.00
Маруга	27	850.00
Кораблев	25	750.00
Бородавко	28	750.00
Завакевич	27	800.00

Агрегатные функции

Агрегатные функции выполняют вычисление на наборе значений и возвращают одиночное значение. Агрегатные функции не учитывают значения NULL. Агрегатные функции часто используются в выражении GROUP BY инструкции SELECT.

Агрегатные функции позволяют получать из таблицы сводную (*агрегированную*) информацию, выполняя операции над группой строк таблицы.

Функция COUNT

Возвращает число строк в запросе. Имеет две формы:

- **COUNT(*)** - возвращает количество элементов в группе. Сюда входят значения NULL и повторяющиеся значения.
- **COUNT (DISTINCT выражение)** вычисляет выражение для каждой строки в группе и возвращает количество уникальных ненулевых значений.

Указывает, что все строки должны быть подсчитаны для возврата общего числа строк в таблице. COUNT(*) не принимает никаких параметров и не может использоваться с ключевым словом DISTINCT. Функция COUNT(*) возвращает количество строк в указанной таблице, не отбрасывая дублированные значения.* Она подсчитывает каждую строку отдельно. При этом учитываются и строки, содержащие значения NULL.

Функция COUNT(expression) будет учитывать только те записи в счетчике, где значение выражения в COUNT (expression) не равно NULL. Когда выражение содержит значение NULL, оно не включается в вычисления COUNT.

```
select count(*) from STUDENTS;
```
результат 45

```
select count(birthday) from STUDENTS;
```
результат 44

Для пустого множества (например таблицы в которой нет данных) функция count возвращает ноль.

```
select count(*) from STUDENTS;
select count(*) from STUDENTS where stipend > 800;
select count(distinct stipend) from STUDENTS;
```

Функция SUM

Возвращает сумму всех, либо только уникальных, значений в выражении. Функция SUM может быть использована только для числовых столбцов. Значения **NULL** пропускаются. Если выборка (таблица) не содержит данных, то возвращает **NULL.**

```
select sum(stipend) from STUDENTS;
select sum(distinct stipend) from STUDENTS;
```

Функция AVG

Возвращает среднее арифметическое группы значений. Значения **NULL** пропускаются. Если выборка (таблица) не содержит данных, то возвращает **NULL.**

Вызов функции AVG с параметром DISTINCT указывает на то, что функция AVG будет выполнена только для одного экземпляра каждого уникального значения, независимо от того, сколько раз встречается это значение.

```
select avg(stipend) from STUDENTS;
select avg(distinct stipend) from STUDENTS;
```

Функция MAX

Возвращает максимальное значение выражения. При выполнении функции MAX все значения NULL пропускаются. Параметр DISTINCT не имеет смысла при использовании функцией MAX и доступен только для совместимости со стандартом ISO. Если выборка (таблица) не содержит данных, то возвращает **NULL.**

```
select max(birthday) from STUDENTS;
select max(distinct birthday) from STUDENTS;
```

Функция MIN

Возвращает минимальное значение выражения. Параметр DISTINCT не имеет смысла при использовании функцией MAX и доступен только для совместимости со стандартом ISO.

```
select min(birthday) from STUDENTS;
```

Примеры:

Сколько студентов учится на первом курсе и какая средняя стипендия на первом курсе?

```
select count(*), avg(stipend) from STUDENTS where course = 1
```

Сколько разных стипендий выплачивают на третьем курсе?

```
select count(distinct stipend) from STUDENTS where course = 3
```

Сколько студентов родилось летом?

```
select count(*) from STUDENTS where month(birthday) between 6 and 8
```

```
Самая ранняя и самая поздняя даты рождения студентов на 1 и 2 курсах?
select
       min(birthday) as min_bday,
       max(birthday) as max_bday
from students where course in (1, 2)
```

Группировка (GROUP BY)

Для группировки значений в результирующей выборке применяется предложение GROUP BY. При этом выражение select записывается в следующей форме:

```
select
        <list_columns | list_expressions>
from <list_tables>
where <criterias>
group by <list_columns >
having <group_criterias>
```

Предложение GROUP BY принимает результат работы предложений FROM и WHERE, а затем группируют его таким образом, чтобы в указанных в предложении GROUP BY столбцах для каждой группы были одинаковые значения. В результирующей таблице каждая группа уменьшается до размеров одной строки. Полученная таблица называется сгруппированной, и все операции теперь применяются к группам, а не к отдельным строкам.

В соответствии с принятым соглашением NULL-значения считаются одной группой. Порядок столбцов в предложении GROUP BY не имеет значения, но, поскольку они должны затем появиться в списке оператора SELECT, то лучше использовать один и тот же порядок (для облегчения чтения и понимания).

Какая суммарная стипендия по курсам?

```
select
       course
       , sum(stipend) as 'Суммарная стипендия'
from STUDENTS
group by
       course;
```

course	Суммарная стипендия
1	5250.00
2	5550.00
3	5200.00
4	7250.00
5	4050.00

Самая ранняя и самая поздняя даты рождения студентов по курсах?

```
select
        min(birthday) as min_bday
       ,max(birthday) as max_bday
       ,course
from students
group by course;
```

В MySQL реализовано нестандартное поведение операции группировки. Для включения режима соответствия стандарту следует использовать команду

```
set @@sql_mode = 'only_full_group_by';
```

Какая суммарная стипендия по курсам с учетом пола студента?

```
select
       course
       , gender
       , sum(stipend) as 'Суммарная стипендия'
from STUDENTS
group by
       course, gender
order by course;
```

course	gender	Суммарная стипендия
1	f	1250.00
1	m	4000.00
2	f	2500.00
2	m	3050.00

3	f	1950.00
3	m	3250.00
4	f	2150.00
4	m	5100.00
5	f	1300.00
5	m	2750.00

Каковы минимальная, максимальная, общая и средняя стипендии на каждом курсе?

```
select
course
       , min(stipend) as min_stipend
       , max(stipend) as max_stipend
       , sum(stipend) as sum_stipend
       , avg(stipend) as avg_stipend
from students
group by course;
```

course	min_stipend	max_stipend	sum_stipend	avg_stipend
1	500.00	950.00	5250.00	656.250000
2	450.00	750.00	5550.00	616.666666
3	450.00	700.00	5200.00	577.777777
4	350.00	900.00	7250.00	604.166666
5	350.00	750.00	4050.00	578.571428

Оператор HAVING

HAVING — необязательный (опциональный) параметр оператора SELECT для указания условия на результат агрегатных функций (MAX, SUM, AVG, …).

Если предложение WHERE определяет предикат для фильтрации строк, то предложение HAVING применяется после группировки для определения аналогичного предиката, фильтрующего группы по значениям агрегатных функций. Это предложение необходимо для проверки значений, которые получены с помощью агрегатной функции не из отдельных строк источника записей, определенного в предложении FROM, а из групп таких строк. Поэтому такая проверка не может содержаться в предложении WHERE.

HAVING <условия> аналогичен WHERE <условия> за исключением того, что строки отбираются не по значениям столбцов, а строятся из значений столбцов, указанных в GROUP BY, и значений агрегатных функций, вычисленных для каждой группы, образованной GROUP BY.

Пример.
Выбрать все курсы на которых средняя стипендия превышает 600 грн.

```
select
       course
       , avg(stipend) as avgst
from STUDENTS
group by course
having avg(stipend) > 600
```

Выбрать все курсы на которых средняя стипендия студенток превышает 650 грн.

```
select
       course
       , gender
       , avg(stipend) as avgst
from STUDENTS
group by course, gender
having avg(stipend) > 650 and gender = 'f'
```

Убедиться, что тройки городов уникальны.
Первая попытка (неправильная).

Если тройки уникальны, то сгруппировав эти тройки, мы не должны получить группы, которые содержат более одной записи в группе:

```
select distinct
       s.CITY
       , l.CITY
       , u.CITY
from STUDENTS as s
       cross join  UNIVERSITIES as u
       cross join  LECTURERS as l
group by s.CITY, l.CITY, u.CITY
having count(*) > 1;
```

Однако, в результате выполнения этого запроса мы получаем 1812 таких групп. Ошибка в том, что группировка выполняется до выполнения оператора distinct и, в результате группируются не уникальные тройки.

Правильное решение:

```
select sc, lc, uc from
(
       select distinct
              s.CITY as sc
              , l.CITY as lc
              , u.CITY as uc
       from STUDENTS as s
              cross join  UNIVERSITIES as u
              cross join  LECTURERS as l
) as t
group by sc, lc, uc
having count(*) > 1
```

Следует отметить, что предложение HAVING может использоваться и без предложения GROUP BY. При отсутствии предложения GROUP BY агрегатные функции применяются ко всему выходному набору строк запроса, т.е. в результате мы получим всего одну строку, если выходной набор не пуст.

Таким образом, если условие на агрегатные значения в предложение HAVING будет истинным, то эта строка будет выводиться, в противном случае мы не получим ни одной строки.

Пример:

```
select
       MIN(STIPEND) as min_st
       ,MAX(STIPEND) as max_st
       ,AVG(STIPEND) as avg_st
from STUDENTS;
```

Результат:

min_st	max_st	avg_st
350.00	950.00	606.666666

Но если выполнить запрос

```
select
       MIN(STIPEND) as min_st
       ,MAX(STIPEND) as max_st
       ,AVG(STIPEND) as avg_st
from STUDENTS
having AVG(STIPEND) < 500;
```

то получим пустой результат, т.к. средняя стипендия больше 500.

Пример использования суммирования для нечисловых данных

Подсчитать сколько студентов родилось в разные времена года

```
select
    SUM(case
              when MONTH(birthday) in(12,1,2) then 1
              else null
       end) as [born in winter]
   ,SUM(case
              when MONTH(birthday) in(3,4,5) then 1
              else null
       end) as [born in spring]
   ,SUM(case
              when MONTH(birthday) in(6,7,8) then 1
              else null
       end) as [born in summer]
   ,SUM(case
              when MONTH(birthday) in(9,10,11) then 1
              else null
       end) as [born in autumn]
   ,SUM(case
              when birthday is null then 1
       end) as unknown
from students;
```

Напоминание о важности форматирования)

```
sElecT SUM(cAsE WhEn MONTH(birthday) In(12,1,2) THeN 1 eLsE null eND) as
[born in winter],SUM(CasE WheN MONTH(birthday) in(3,4,5) Then 1 eLSE Null end)
AS [born in spring],SUM(cAse WheN MONTH(birthday) in(6,7,8) TheN 1 else Null
ENd) as [born in summer],SUM(CAsE WhEn MONTH(birthday) IN(9,10,11) THEn 1 elsE
nULl ENd) As [born in autumn],SUM(CasE wHen birthday iS NulL THen 1 END) As unknown
FrOM students;
```

Подзапросы (Вложенные запросы)

Простые подзапросы

SQL позволяет использовать одни запросы внутри других запросов, то есть вкладывать запросы друг в друга. Предложение **SELECT** может быть использовано повторно внутри предложений **FROM**, **WHERE** и **HAVING**.

Получить все оценки студента по его фамилии:

```
select
	*
from exam_marks
where student_id = (
	select id from students where surname='Кабанов')
```

Для того, что бы такой запрос работал нужно, что бы внутренний подзапрос возвращал ровно одно значение.

НЕ РАБОТАЕТ!
```
select
	*
from exam_marks
where student_id = (
	select id, name from students where surname='Кабанов')
```

НЕ РАБОТАЕТ!
```
select
	*
from exam_marks
where student_id = (
	select id from students where surname like 'К%')
```

ОПАСНО!
```
select
	*
from exam_marks
where student_id = (
	select univ_id from students where surname = 'Кабанов')
```

Здесь сравниваются значения двух числовых значения полей, которые несут разную смысловую нагрузку. Случайное совпадение этих значений приводит к неправильному результату.

Оператор IN и подзапросы

В случае, когда подзапрос возвращает множество значений, и это ожидаемый результат, следует использовать оператор IN

Пример. Получить список студентов, которые обучаются в университетах Киева (университетов несколько).

```
select
	*
from STUDENTS
where UNIV_ID in (
	select id from UNIVERSITIES where city = 'Киев')
```

Коррелированные подзапросы

Коррелированный подзапрос - это оператор SELECT, вложенный в другой оператор SQL, и ссылающийся на один или несколько столбцов внешнего запроса. Поэтому можно сказать, что коррелированный подзапрос зависит от внешнего запроса. Это - главное различие между коррелированным и простым подзапросом. Простой подзапрос не ссылается на внешний запрос, он может быть выполнен независимо от него.

Коррелированный подзапрос выполняется несколько раз в процессе обработки оператора SQL, содержащего такой подзапрос. Он будет исполняться для каждой строки, отобранной во внешнем запросе. На каждом из этих шагов поля внешнего запроса, на которые ссылается коррелированный подзапрос, будут сравниваться с результатами выборки коррелированного подзапроса. Результат выполнения коррелированного подзапроса определит, попадет ли строка внешнего запроса в результирующую выборку.

Пример. Выбрать преподавателей, которые преподают в городе своего постоянного проживания. Другими словами, в городе, где постоянно проживает преподаватель, есть как минимум один университет и преподаватель преподает в этом (или одном из) этих университетов.

Начнем рассмотрение с выборки:

```
select
	*
from LECTURERS;
```

Преподаватель с ID = 1 (Кравчук) живет в городе Днепр. Проверяем, есть ли в городе проживания этого преподавателя университеты:

```
select
	*
from UNIVERSITIES where city = 'Днепр';
```

Далее проверяем, преподает ли преподаватель Кравчук (проживающий в Днепре) в одном из этих университетов:

```
select
	*
from LECTURERS
where
	id = 1 -- это преподаватель Кравчук
	and univ_id in ( -- преподает ли он в университете (университетах) города Днепр
		select
			id
		from UNIVERSITIES
		where
			city = 'Днепр'
	);
```

В городе Днепр есть университеты, преподаватель Кравчук преподает в одном из них и, следовательно, попадает в выборку. Переходим к следующему преподавателю. Преподаватель ID = 2 (Совва). Город Днепр, повторяются те же рассуждения. Процесс повторяется с теми же рассуждениями до тех пор, пока мы не начинаем рассматривать преподавателя с ID = 20 (Вдовиченко). Он проживает в городе Луцк, выборка:

```
select
	*
from UNIVERSITIES where city = 'Луцк'
```

дает пустое множество, значит, в городе проживания преподавателя Вдовиченко университетов нет, и он не попадает в наше результирующее множество.

```
select
	*
from LECTURERS
where
	id = 20
	and univ_id in (
		select
			id
		from UNIVERSITIES
		where
			city = 'Луцк'
	);
```

Таким образом, выбрать всех преподавателей, которые проживают, например, в городе Днепр, где есть университеты, можно выборкой

```
select
	*
from LECTURERS
where
	city = 'Днепр'		-- это город проживания преподавателя
	and univ_id in
		(
			select
				id
			from UNIVERSITIES
			where city = 'Днепр' -- это город, где расположен университет
		);
```

Для преподавателей из Харькова

```
select
	*
from LECTURERS
where
	city = 'Харьков'
	and univ_id in
		(
			select
				id
			from UNIVERSITIES
			where city = ' Харьков '
		);
```

А теперь передадим информацию о том, в каком городе проживает преподаватель во внутренний запрос, который возвращает идентификаторы университетов:

```
select
	*	-- ШАГ 0. Выбираем преподавателя из таблицы преподавателей
from LECTURERS
-- ШАГ 2. Внешний запрос проверяет, принадлежит ли идентификатор университета
-- в котором преподает преподаватель (текущий в выборке) множеству идентификаторов
-- университетов, которые расположены в городе постоянного проживания этого преподавателя.
-- Если условие принадлежности выполняется, то запись о преподавателе (текущем)
-- передается в результирующую выборку и внешний запрос переходит к обработке
-- следующего преподавателя. В противном случае запись о преподавателе игнорируется
-- и внешний запрос переходит к обработке следующего преподавателя
where univ_id in
	(
		-- ШАГ 1. Во внутренний подзапрос передается информация о городе
		-- постоянного проживания преподавателя (текущего в выборке).
		-- Внутренний подзапрос возвращает множество идентификаторов университетов,
		-- которые расположены в городе проживания преподавателя
		select
			id
		from UNIVERSITIES
		where city = LECTURERS.city -- выбираем те университеты, которые расположены
					-- в городе проживания преподавателя
	);
```

Пояснение. В последней строке приведенного запроса неуточненная ссылка на **city** уточняется неявным образом именем таблицы **UNIVERSITIES**. Другая ссылка явно уточняется именем таблицы **LECTURERS**. Этот пример отличается от предыдущих тем, что внутренний подзапрос не может быть обработан раз и навсегда прежде, чем будет обрабатываться внешний запрос, поскольку этот внутренний подзапрос зависит от переменной, а именно от **LECTURERS.city**, значение которой изменяется по мере того, как система проверяет различные строки таблицы **LECTURERS**. Следовательно, с концептуальной точки зрения обработка осуществляется следующим образом.

- Система проверяет первую строку таблицы **LECTURERS**. Предположим, что это строка лектора, который проживает в городе Харьков. Следовательно, переменная **LECTURERS.city** в данный момент имеет значение Харьков, и система обрабатывает внутренний запрос:

  ```
  select
  	id
  from UNIVERSITIES
  where city = 'Харьков';
  ```

  получая в результате множество (6, 9). Теперь система может завершить обработку для первого лектора (во внешнем запросе). Выборка лектора будет осуществлена тогда и только тогда, когда значение **univ_id** для этого лектора принадлежит найденному множеству.

- Далее система будет повторять обработку такого рода для следующего лектора и т. д. до тех пор, пока не будут рассмотрены все строки таблицы **LECTURERS**.

Такой подзапрос, как в приведенном выше примере, называется коррелированным. Коррелированный подзапрос - это такой подзапрос, результат которого зависит от некоторой переменной. Эта переменная принимает свое значение в некотором внешнем запросе. Следовательно, обработка такого подзапроса должна повторяться для каждого значения переменной в запросе, а не выполняться раз и навсегда.

Пример. Выбрать всех студентов, у которых стипендия меньше чем в среднем по курсу:

```
select
	*
from STUDENTS as s
where stipend < (select avg(stipend) from STUDENTS where course = s.course)
```

Выбрать студентов, у которых стипендия находится в интервале «средняя стипендия на курсе» +/- 50

```
select
	*
from STUDENTS as s1
where
	STIPEND >= (select avg(stipend) - 50 from STUDENTS as s2 where s1.COURSE = s2.COURSE)
	and
	STIPEND <= (select avg(stipend) + 50 from STUDENTS as s3 where s1.COURSE = s3.COURSE)
```