Оглавление

Оператор EXISTS

Оператор **EXISTS** используется для проверки наличия хотя бы одной записи в результате подзапроса. Оператор **EXISTS** возвращает значение TRUE (**истина)**, если подзапрос возвращает одну или несколько записей и значение FALSE (**ложь**) в противном случае.

Обычно оператор EXISTS используется в зависимых (коррелирующих) подзапросах. Этот вид подзапроса имеет внешнюю ссылку, связанную со значением в основном запросе. Результат подзапроса может зависеть от этого значения и должен оцениваться отдельно для каждой строки запроса, в котором содержится данный подзапрос. Поэтому оператор EXISTS может иметь разные значения для разных строк основного запроса.

Работа подзапроса, помещенного в оператор **EXISTS, *прекращается сразу после того, как внутренний подзапрос вернет хотя бы одну запись.***

```
SELECT < *  | list_of_columns | const>
FROM table_name
WHERE EXISTS
(SELECT < * | list_of_columns | const> FROM another_table_name WHERE condition);
```

Пример. Кто из студентов получил положительные оценки?

```
select
	id
	, name
	, surname
from students as s
where exists (
		-- внутренний подзапрос возвращает что-то только в том случае
		-- если студент s.id (значение s.id выбирается из внешнего запроса)
		-- имеет положительные оценки. При этом этот же студент может
		-- иметь и двойки
		select 1
		from exam_marks
		where mark >= 3 and student_id=s.id)
order by id;
```

Пример. Кто из студентов не сдал ни один экзамен?

```
select
	id
	,name
	, surname
from students s
where not exists (
	select 1
	from exam_marks
	where student_id=s.id);
```

id	name	surname
6	Роман	Березовский
17	Виталий	Осипуков
26	Станислав	Кораблев

Проверим.

```
select * from EXAM_MARKS where STUDENT_ID in (6,17,26);
```

Для работы оператора **EXISTS** не важно, что возвращает вложенный оператор **select**. Важны не конкретное значение какого-либо поля, а факт существования записи в результирующем множестве записей. Поэтому можно использовать следующую форму вложенного оператора **select:**

```
select 1 from …
```

Иногда (для MS SQL Server) можно увидеть форму

```
select top 1 1 from …
```

Запрос для получения списка студентов получивших только положительные оценки можно переписать в форме запроса с использованием оператора IN.

```
select
       id
       ,name
       , surname
from STUDENTS as s
where id in (
              -- получаем множество идентификаторов студентов
              -- получивших положительные оценки. По-прежнему в этом
              -- множестве присутствуют студенты, которые могли
              -- получить и двойки
              select student_id
              from EXAM_MARKS
              where mark >= 3)
order by id;
```

Результат будет тот же. Но при использовании IN сначала будет получены ВСЕ данные из внутреннего подзапроса, а потом будет выполнен внешний запрос. В случае с EXISTS работа внутреннего подзапроса останавливается, как только найдена хотя бы одна запись.

Операторы сравнения с множеством значений IN, ANY (SOME), ALL

Рассмотрим задачу: выбрать всех студентов, у которых дата рождения больше чем минимальная дата рождения студентов второго курса:

```
select
       *
from STUDENTS as s1
where s1.birthday > (select min(s2.birthday) from STUDENTS as s2 where s2.course = 2)
```

Использование псевдонимов в этом запросе необязательно и применяются они лишь для большей наглядности оператора select.

Для решения этой задачи мы используем подзапрос, который возвращает одно единственное значение **min(birthday).** Однако мы при помощи подзапроса и оператора > не можем решить следующую задачу: выбрать всех студентов, у которых дата рождения больше любой из дат рождения студентов второго курса. Это невозможно уже хотя бы потому, что даты рождения всех студентов второго курса образуют множество:

```
select s2.birthday from STUDENTS as s2 where s2.course = 2
```

и мы не можем сравнивать одно `s1.birthday` с этим множеством простым оператором `>`. Для того, что бы решить эту задачу необходимо уточнить оператор `>` модификатором сравнения с множеством.

```
select
       *
from STUDENTS as s1
where s1.birthday >ALL (select s2.birthday from STUDENTS as s2 where s2.course = 2)
```

Операторы сравнения с модификаторами ANY (SOME) или ALL

Операторы сравнения с вложенными запросами могут быть уточнены с помощью ключевых слов ALL или ANY. SOME является эквивалентом ANY в стандарте ISO.

Вложенные запросы операторов сравнения с модификаторами возвращают список из нуля или более значений и могут включать предложения GROUP BY или HAVING.

Рассмотрим, например оператор сравнения >: >ALL будет означать «больше любого значения». Другими словами, это сравнение с максимальным значением. Например, >ALL (1, 2, 3) означает «больше 3». >ANY означает «больше, по крайней мере, одного значения», т. е. «больше минимума». Поэтому >ANY (1, 2, 3) означает «больше 1».

Чтобы строка результата вложенного запроса с >ALL удовлетворяла условию, заданному внешним запросом, значение в столбце, для которого вводится вложенный запрос, должно быть больше каждого значения из списка, возвращаемого вложенным запросом.

Аналогичным образом, чтобы строка результата вложенного запроса с >ANY, удовлетворяла условию, заданному внешним запросом, значение в столбце, для которого вводится вложенный запрос, должно быть больше хотя бы одного значения из списка, возвращаемого вложенным запросом.

Оператор =ANY эквивалентен оператору IN. Однако оператор <>ANY отличается от NOT IN: ***< >ANY означает «не равно a, или не равно b, или не равно c». NOT IN означает «не равно a, и не равно b, и не равно c».***

ALL

Сравнивает скалярное выражение с одномерным набором значений. Используется в совокупности с операторами сравнения и иногда также классифицируется как оператор сравнения.
Возвращает значение TRUE, если заданное условие справедливо для всех пар, в противном случае возвращается FALSE.

Оператор ALL, как правило, эффективно используется с неравенствами, а не с равенствами, поскольку значение равно всем, которое должно получиться в этом случае в результате выполнения подзапроса, может иметь место, только если все результаты идентичны. Такая ситуация практически не может быть реализована, так как если подзапрос генерирует множество различных значений, то никакое одно значение не может быть равно сразу всем значениям в обычном смысле. В SQL выражение < > ALL реально означает не равно ни одному из результатов подзапроса.

Таким образом, оператор ALL используется вместе с операторами **=, <>, <, >, <=, >=** и возвращает булево значение ИСТИНА (TRUE) или ЛОЖЬ (FALSE). Значение TRUE получает в том случае если для оператора

X op ALL ($y_1, y_2, \ldots y_N$)

значение всех попарных сравнений **X op y_k** является истиной. Т.е.

(X op y_1) and (X op y_2) and … and (X op y_N) = true

100 > ALL(1, 77, 5, 22) - истина
-1 < ALL(1, 77, 5, 22) – истина

70 > ALL(1, 77, 5, 22) – ложь (70 < 77)

Пример: выбрать все учебные заведения с рейтингом более высоким, чем рейтинг всех учебных заведений Харькова:

```
select * from universities
where rating > all (
select rating from universities where city='Харьков')
```

ANY (SOME)

Сравнивает скалярное выражение с одномерным набором значений. Ключевые слова SOME и ANY полностью взаимозаменяемы. Возвращается TRUE, если заданное условие справедливо для какой-либо пары.

Таким образом, оператор ANY используется вместе с операторами **=, <>, <, >, <=, >=** и возвращает булево значение ИСТИНА (TRUE) или ЛОЖЬ (FALSE). Значение TRUE получает в том случае если для оператора

X op ANY (y_1, y_2, …y_N)

значение всех попарных сравнений **X op y_k** является истиной. Т.е.

(X op y_1) or (X op y_2) or … or (X op y_N) = true

= ANY – полностью соответствует IN

5 = ANY(1, 77, 5, 22) - истина
8 = ANY(1, 77, 5, 22) - ложь
60 > ANY(1, 77, 5, 22) - истина
77 >= ANY(1, 77, 5, 22) – истина
30 < ANY(1, 77, 5, 22) – истина
30 > ANY(1, 77, 5, 22) – истина

Пример: выбрать все учебные заведения с рейтингом более высоким, чем рейтинг ***какого-либо*** учебного заведения Харькова:

```
select * from universities
where rating > any (
select rating from universities where city='Харьков')
```

IN

Операторы **IN** (равен любому из списка) проверяет, принадлежит ли скалярное выражение слева множеству значений в списке, указанном в скобках слева от оператора **IN**. Значения в списке могут быть представлены как литералами, так и быть возвращены подзапросом.

Построенный с использованием **IN** предикат (условие) считается истинным, если значение выражения указано слева от **IN**, совпадает (подразумевается точное совпадение) с одним из значений, перечисленных в списке, указанном в скобках справа от **IN**. Таким образом, оператор **IN** эквивалентен оператору **= ANY**.

Предикат, построенный с использованием **NOT IN**, считается истинным, если значение поля, имя которого указано слева от **NOT IN**, не совпадает ни с одним из значений, перечисленных в списке, указанном в скобках справа от **NOT IN**. Поэтому оператор **NOT IN** не совпадает с оператором **<>ANY**.

Примеры:

5 in (1, 77, 5, 22) – истина
8 in (1, 77, 5, 22) – ложь
9 not in (1, 77, 5, 22) – истина
5 not in (1, 77, 5, 22) – ложь

Таким образом, можно сказать, операторы сравнения с множеством значений имеют следующий смысл.

IN	Значение слева принадлежит множеству значений определенных оператором IN, или - равно любому из значений, полученных во внутреннем запросе.
NOT IN	Не равно ни одному из значений, полученных во внутреннем запросе.
= ANY	То же, что и IN, Соответствует логическому оператору OR.
> ANY, > = ANY	Больше, чем (либо больше или равно) любое полученное число. Эквивалентно > или > = для самого меньшего полученного значения.
< ANY, < = ANY	Меньше, чем (либо меньше или равно) любое полученное число. Эквивалент < или < = для самого большего полученного значения.
= ALL	Равно всем полученным значениям. Эквивалентно логическому оператору AND
> ALL, > = ALL	Больше, чем (либо больше или равно) все полученные числа. Эквивалент > или > =для самого большего полученного значения.
< ALL, < = ALL	Меньше, чем (либо меньше или равно) все полученные числа. Эквивалентно < или < = самого меньшего полученного значения.

Тонкости операторов = ALL и <> ANY

Оператор **= ALL** всегда возвращает false, если только все значения в множестве (подзапросе) не равны друг другу и сравниваемому значению (слева). Например:

X = ALL(1, 2, 3, 4) – false.

По аналогии можно было бы подумать, что выражение **<> ANY** всегда равно false, если значение слева всегда равно какому-либо значению в списке. Например, если X = 3, то выражение

X <> ANY(1, 2, 3, 4) - false

Однако, согласно стандарту SQL выражение **<> ANY** равно true, если значение слева ***не равно хотя бы одному значению*** в сравниваемом множестве. Т.е. выражение, для X = 3, есть true

X <> ANY(1, 2, 3, 4) – true

Работа операторов ANY, ALL, EXISTS с пустыми множествами в подзапросах

Когда правильный подзапрос не генерирует никаких выходных данных, оператор **ALL** автоматически принимает значение истина, а оператор **ANY** - значение ложь.

Так запрос:

```
-- выбрать все университеты с рейтингом выше чем
-- рейтинг любого университета из города New York
select
	*
from UNIVERSITIES
where rating > ANY (
	-- внутренний запрос не вернет ни одной записи поскольку
	-- в таблице UNIVERSITIES нет ни одного университета
	-- расположенного в городе New York
	select
		rating
	from UNIVERSITIES
	where city = 'New York'
);
```

не вернет ни одного значения. В то время как запрос

```
-- выбрать все университеты с рейтингом выше чем
-- рейтинги всех университетов из города New York
select
	*
from UNIVERSITIES
where rating > ALL (
	-- внутренний запрос не вернет ни одной записи поскольку
	-- в таблице UNIVERSITIES нет ни одного университета
	-- расположенного в городе New York
	select
		rating
	from UNIVERSITIES
	where city = 'New York'
);
```

вернет все записи.

Использование <> any тоже может быть неочевидным. Например, решив задачу «выбрать студентов не обучающихся на 1 или 2 курсах» приведенным ниже способом мы получим ошибочный результат.

```
/* Этот запрос вернет всех студентов*/
select *
from STUDENTS
where course <> any (select course from STUDENTS where course in(1, 2));
```

Фактически этот запрос эквивалентен следующему:

```
select *
from STUDENTS
where course <> 1 or course <> 2;
```

Оператор IN на пустом множестве возвращает значение false.

```
select * from STUDENTS where univ_id in (
		select id from UNIVERSITIES where city = 'New York'
	)
```

Оператор EXISTS на пустом множестве возвращает значение false.

Особенности применения операторов ANY, ALL, EXISTS при обработке пустых значений (NULL)

Рассмотрим следующую задачу: найти всех студентов, дата рождения которых, меньше даты рождения всех студентов 4-го курса.

```
select
	*
from STUDENTS
where
	birthday <= all (
		select
			distinct birthday
		from STUDENTS
		where course = 4
	);
```

В результате выполнения запроса мы получим не пустое множество студентов, отвечающих условию задачи. В тоже время запрос, выполненный для студентов пятого курса, вернет пустое множество.

```
select
	*
from STUDENTS
where
	birthday <= all (
		select
			distinct birthday
		from STUDENTS
		where course = 5
	);
```

Это произойдет потому, что среди студентов пятого курса есть студент, для которого неизвестна дата его рождения. Проверим:

```
select birthday from STUDENTS where course = 5;
```

Наш первоначальный запрос полностью эквивалентен следующему запросу:

```
select
	*
from STUDENTS
where
	    birthday <= '19901230'
	and birthday <= '19880306'
	and birthday <= '19891022'
	and birthday <=  NULL       -- <= ПРОБЛЕМА
	and birthday <= '19890103'
	and birthday <= '19890228'
	and birthday <= '19891012';
```

Для получения правильного ответа необходимо должным образом обработать значение null:

```
select
	*
from STUDENTS
where
	birthday <= all (
		select
			distinct birthday
		from STUDENTS
		where
			course = 5 and birthday is not null
	);
```

Вернемся к первоначальному запросу, выбирающему студентов, у которых дата рождения меньше любой даты студентов четвертого курса:

```
select
	*
from STUDENTS
where
	birthday <= all (
		select
			distinct birthday
		from STUDENTS
		where course = 4
	);
```

Результат выполнения запроса:

ID	SURNAME	NAME	GENDER	STIPEND	COURSE	CITY	BIRTHDAY	UNIV_ID
16	Земляний	Данил	m	550.00	5	Львов	1988-03-06	4
20	Миланивская	Ольга	f	750.00	5	Киев	1989-10-22	5
27	Бородавко	Михаил	m	750.00	5	Кременчуг	1989-01-03	2
37	Запорожец	Виталий	m	350.00	5	Луцк	1989-02-28	13
39	Богацкий	Дмитрий	m	350.00	5	Черновцы	1989-10-12	12
41	Денисюк	Данил	m	350.00	4	Херсон	1990-01-26	2

Перепишем запрос с использованием оператора EXISTS:

```
select
	*
from STUDENTS as s1
where
not exists(
	select
		*
	from STUDENTS as s4
	where
		s4.course = 4
		and s4.birthday < s1.birthday
)
```

В результате выполнения запроса мы получим немного другие данные:

ID	SURNAME	NAME	GENDER	STIPEND	COURSE	CITY	BIRTHDAY	UNIV_ID
16	Земляний	Данил	m	550.00	5	Львов	1988-03-06	4
20	Миланивская	Ольга	f	750.00	5	Киев	1989-10-22	5
21	Запорожец	Владимир	m	750.00	5	Львов	NULL	5
27	Бородавко	Михаил	m	750.00	5	Кременчуг	1989-01-03	2
37	Запорожец	Виталий	m	350.00	5	Луцк	1989-02-28	13
39	Богацкий	Дмитрий	m	350.00	5	Черновцы	1989-10-12	12
41	Денисюк	Данил	m	350.00	4	Херсон	1990-01-26	2

Это произойдет потому, что для студента Запорожец будет выполнен подзапрос:

```
	select
		*
	from STUDENTS as s4
	where
		s4.course = 4
		and s4.birthday < NULL
```

который вернет пустое множество. На пустом множестве оператор NOT EXISTS вернет значение true и студент Запорожец попадет в результирующую выборку.

Другой пример:

```
-- 7. Напишите запрос EXISTS, выводящий кол-во студентов каждого курса, которые успешно
--    сдали экзамены, и при этом не получивших ни одной двойки.

select
       course
       ,count(id) [количество студентов]
from STUDENTS s
where not exists (select student_id
                    from EXAM_MARKS
                    where s.id = student_id and mark = 2)
group by course;
```

Решение не правильно. Проверяем: выбираем количество различных студентов обучающихся на первом курсе и не получивших двоек.

```
select count(
                    -- студент мог получить несколько оценок и поэтому конкретное
                    -- значение student_id может повторяться несколько раз в выборке.
                    -- Нас же интересует не количестро оценок, а количество студентов
                    -- (различных), которые не получили двойки.
                    -- Для получения количества различных студентов
                    -- используем distinct
                    distinct student_id
             )
from EXAM_MARKS
where
       student_id in (select id from STUDENTS where course = 1)
       and student_id not in (select student_id from EXAM_MARKS where mark = 2);
```

Запрос возвращает 4. Причина: на первом курсе есть студент с ID = 17 который не сдавал экзамены вообще. Запрос

```
select
       student_id
from EXAM_MARKS
where
       STUDENT_ID = 17
```

вернет пустое множество. NOT EXISTS на пустом множестве вернет TRUE. Пример: запрос

```
select * from students where not exists (select 1 from EXAM_MARKS where 1 = 2)
```

вернет все множество студентов. По этой причине в список студентов УСПЕШНО СДАВШИХ попадет и студент, не сдававший экзамены вообще.

Формальные правила оценки истинности предикатов, использующих параметры ANY и ALL

Рассмотрим *<expression> <op>* ***ALL | ANY*** *(query)*

- Если определен параметр **ALL** или **ANY** и все результаты сравнения значения выражения слева и каждого значения, полученного из подзапроса, являются TRUE, истинностное значение равно TRUE.

- Если результат выполнения подзапроса не содержит строк и определен параметр **ALL**, результат равен TRUE. Если же определен параметр **ANY**, результат равен FALSE.
- Если определен параметр **ALL** и результат сравнения значения выражения слева хотя бы с одним значением, полученным из подзапроса, является FALSE, истинностное значение равно FALSE.
- Если определен параметр **ANY** и хотя бы один результат сравнения значения выражения слева и значения, полученного из подзапроса, является TRUE, истинностное значение равно TRUE.
- Если определен параметр **ANY** и каждое сравнение значения выражения и значений, полученных из подзапроса, равно FALSE, истинностное значение тоже равно FALSE.
- В любом другом случае результат будет равен UNKNOWN.

Оператор NOT IN со значением null

Подготовим тестовые данные. Подробнее о создании таблиц и том как в табицы добавляются данные мы будем говорить на следующих занятиях.

```
if exists(select 1 from INFORMATION_SCHEMA.TABLES where table_name='CITIES')
drop table CITIES

create table CITIES (city [varchar](50) null)

insert into CITIES
select 'Киев' union all
select 'Львов' union all
select 'Ужгород' union all
select null;

select * from CITIES;
go
```

Для **MySQL**

```
create table CITIES (city varchar(50) null);

insert into CITIES
select 'Киев' union all
select 'Львов' union all
select 'Ужгород' union all
select null;

select * from CITIES;
```

Определим при помощи оператора IN, находится ли Киев в таблице `CITIES`

```
select  'Киев найден' as colname  where 'Киев' in (select city from CITIES);
```

Выполнение запроса показывает, что оператор IN отработал как мы и ожидали (мы получили результат **Киев найден**).

Запрос

```
select 'Харьков найден' as colmame  where 'Харьков' in (select city from CITIES);
```

ожидаемо не возвращает нам ничего т.к. город Харьков отсутствует в таблице городов. Тогда запрос

```
select 'Харьков не найден' as colmame where 'Харьков' not in (select city from CITIES);
```

должен был бы вернуть нам значение **Харьков не найден**, однако он снова не дает нам никакого результата.
Попробуем выполнить такой запрос

```
select 'Харьков не найден' as colmame where 'Харьков' not in ('Киев','Львов','Ужгород');
```

Результатом запроса будет **Харьков не найден**. Это связано с тем, что оператор сравнивает каждый город в списке; предыдущий запрос логически эквивалентен следующему запросу:

```
select 1 as 'Харьков не найден'
where
       'Харьков'<>'Киев'
       and 'Харьков'<>'Львов'
       and 'Харьков'<>'Ужгород'
```

Поэтому, если в список городов мы добавим null, то запрос приобретет вид:

```
select 1 as 'Харьков не найден'
where
       'Харьков'<>'Киев'
       and 'Харьков'<>'Львов'
       and 'Харьков'<>'Ужгород'
       and 'Харьков'<> null
```

... и поскольку по умолчанию 'Харьков'<> null дает значение UNKNOWN (ни истина, ни ложь), никакая строка не возвращается, потому что каждое условие должно быть истинным, чтобы оператор AND возвращал TRUE-результат. Тот же контр-интуитивный результат происходит с оператором IN, как в этом примере:

```
select city from CITIES where city in (select city from CITIES);
```

Значение NULL исчезает из результата поскольку NULL <> NULL

Подчищаем за собой. Удаляем таблицу `CITIES`

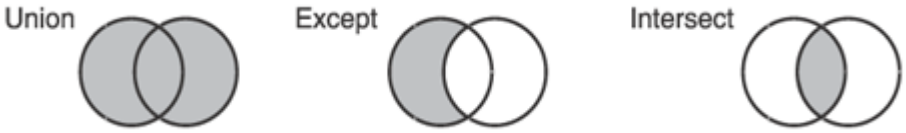
```
drop table CITIES
```

Оператор объединения UNION, EXCEPT, INTERSECT

UNION объединяет строки из двух или более наборов результатов в единый результирующий набор.

EXCEPT оценивает два набора результатов и возвращает все строки из первого набора, которые также не содержатся во втором наборе.

INTERSECT вычисляет набор результатов, содержащий общие строки из двух наборов результатов.



Операторы UNION, INTERSECT и EXCEPT могут быть объединены в одном выражении UNION.

Оператор UNION принимает две таблицы и строит их них третью (к которой снова может быть применен оператор UNION). Эти таблицы должны быть совместимы по объединению, т.е. содержать одинаковое количество столбцов, и каждый столбец первой таблицы должен быть того же типа

данных (или автоматически приводиться к нему), что и находящийся на том же месте столбец второй таблицы, т.е. строки первой и второй таблицы должны иметь одинаковую структуру.

Другими словами: оператор UNION позволяет выбрать с помощью SELECT строки из двух (или более) подобных наборов результатов и объединить их в один набор. При этом:

- Каждый оператор SELECT в UNION должен иметь одинаковое количество столбцов
- Столбцы также должны иметь похожие типы данных
- Столбцы в каждом операторе SELECT также должны следовать в одинаковом порядке

Оператор UNION автоматически применяет distinct к результирующей выборке, т.е. дубликаты исключаются.

Пример: выбрать объединенный список городов из таблицы студентов и лекторов

```
select city from students
union
select city from lecturers
```

В результате будет выбрано 24 строки

Для того, что бы выбрать ВСЕ записи применяется оператор UNION ALL

```
select city from students
union all
select city from lecturers
```

Выбирается 70 строк.
Выбрать список преподавателей из Киева, в фамилии которых вторая буква ‘о’ и которые проживают в Киеве:

```
select
       SURNAME
       ,CITY
from LECTURERS
where SURNAME like '_o%'
INTERSECT
select
       SURNAME
       ,CITY
from LECTURERS
where CITY = 'Киев'
```

Выбрать список преподавателей из Киева, в фамилии которых вторая буква ‘о’ и которые ***не проживают*** в Киеве:

```
select
       SURNAME
       ,CITY
from LECTURERS
where SURNAME like '_o%'
EXCEPT
select
       SURNAME
       ,CITY
from LECTURERS
where CITY = 'Киев'
```

Операция сортировки ORDER BY не может быть применена к отдельным операторам SELECT в UNION, а применяется только к результату:

```
select city from students
union all
select city from lecturers
order by city
```

Операции фильтрации и группировки могут применяться к отдельным операторам SELECT

```
select city from students
where course = 2
union all
select city from lecturers
order by city
```

Порядок выполнения

Если на порядок выполнения не влияют скобки, то операторы UNION и UNION ALL выполняются слева направо. Поскольку оператор UNION ассоциативен и коммутативен, то порядок объединения таблиц на результат не влияет.