MySQL

Подзапросы (дополнительные материалы)

Подготовка к работе (создание таблиц)

```
-- создание старую базу данных (если она существует)
DROP DATABASE IF EXISTS bookings;

-- создание базы данных резерваций
CREATE DATABASE bookings DEFAULT CHARSET utf8mb4;

-- переключение на новую базу данных
USE bookings;
```

```
-- создание таблицы доступных квартир
CREATE TABLE apartments (
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    country_iso CHAR(2)  NOT NULL,
    city VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL,
    rooms INT UNSIGNED NOT NULL,
    has_internet BOOLEAN NOT NULL,
    has_tv BOOLEAN NOT NULL,
    has_kitchen BOOLEAN NOT NULL,
    has_shower BOOLEAN NOT NULL,
    animals_allowed BOOLEAN NOT NULL,
    price_per_day DECIMAL(10,2) NOT NULL
) ENGINE=InnoDB;

-- создание таблицы клиентов
CREATE TABLE clients (
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    phone VARCHAR(255) NOT NULL UNIQUE,
    email VARCHAR(255) NOT NULL UNIQUE
) ENGINE=InnoDB;
```

```
-- создание таблицы резерваций
CREATE TABLE bookings (
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    apartment_id BIGINT UNSIGNED,
    client_id BIGINT UNSIGNED,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL,
    price_total DECIMAL(10,2) NOT NULL,
    price_per_day DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (apartment_id) REFERENCES apartments(id) ON UPDATE CASCADE ON DELETE SET NULL,
    FOREIGN KEY (client_id) REFERENCES clients(id) ON UPDATE CASCADE ON DELETE SET NULL
) ENGINE=InnoDB;

-- создание таблицы отзывов
CREATE TABLE reviews (
    id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    booking_id BIGINT UNSIGNED NOT NULL,
    rating INT UNSIGNED CHECK(rating >= 1 AND rating <= 10),
    FOREIGN KEY (booking_id) REFERENCES bookings(id) ON UPDATE CASCADE ON DELETE CASCADE
) ENGINE=InnoDB;
```

Подготовка к работе (заполнение таблиц данными)

```
-- заполнение таблицы доступных квартир
INSERT INTO apartments (
    id, country_iso, city, address, rooms, has_internet, has_tv, has_kitchen,
has_shower, animals_allowed, price_per_day
) VALUES
(1, 'LV', 'Riga', 'Brivibas st., 140', 2, TRUE, FALSE, TRUE, TRUE, FALSE, 100.0),
(2, 'LV', 'Riga', 'Maskavas st., 19',  1, FALSE, TRUE, FALSE, TRUE, FALSE, 50.0),
(3, 'LT', 'Vilnius', 'Laisves pr., 75', 3, TRUE, TRUE, TRUE, TRUE, TRUE, 300.0),
(4, 'LT', 'Vilnius', 'Konstitucijos pr., 34', 1, FALSE, FALSE, FALSE, FALSE, TRUE, 20.0),
(5, 'EE', 'Tallinn', 'Tonismagi st., 10', 2, TRUE, FALSE, TRUE, TRUE, TRUE, 170.0),
(6, 'EE', 'Tallinn', 'Magdaleena st., 3', 2, TRUE, FALSE, FALSE, TRUE, FALSE, 140.0);

-- заполнение таблицы клиентов
INSERT INTO clients (
    id, name, phone, email
) VALUES
(1, 'John Smith', '+371 123456789', 'jsmith@gmail.com'),
(2, 'Anna Bowman', '+371 987654321', 'abowman@gmail.com'),
(3, 'Ivan Ivanov', '+371 543216789', 'iivanov@gmail.com'),
(4, 'Irina Petrova', '+371 678954321', 'ipetrova@gmail.com');
```

```
-- заполнение таблицы резерваций
INSERT INTO bookings (
    id, apartment_id, client_id, start_date, end_date, price_total, price_per_day
) VALUES
(1, 2, 2, '2023-02-02', '2023-02-04', 150.0, 50.0),
(2, 3, 4, '2023-02-05', '2023-02-06', 500.0, 250.0),
(3, 1, 3, '2023-02-11', '2023-02-11', 100.0, 100.0),
(4, 4, 1, '2023-02-17', '2023-02-21', 100.0, 20.0),
(5, 1, 3, '2023-03-07', '2023-03-09', 300.0, 100.0),
(6, 2, 4, '2023-03-22', '2023-03-29', 400.0, 50.0),
(7, 5, 3, '2023-04-01', '2023-04-03', 510.0, 170.0),
(8, 1, 1, '2023-04-19', '2023-04-24', 600.0, 100.0),
(9, 3, 2, '2023-04-28', '2023-04-29', 500.0, 250.0);

-- заполнение таблицы отзывов
INSERT INTO reviews (
    id, booking_id, rating
) VALUES
(1, 1, 3),
(2, 2, 5),
(3, 3, 4),
(4, 4, 1),
(5, 5, 1),
(6, 6, 2),
(7, 7, 2),
(8, 8, 4),
(9, 9, 5);
```

Содержимое таблиц (1)

Таблица **apartments**:

id	country_iso	city	address	rooms	has_internet	has_tv	has_kitchen	has_shower	animals_allowed	price_per_day
1	LV	Riga	Brivibas st., 140	2	1	0	1	1	0	100.0
2	LV	Riga	Maskavas st., 19	1	0	1	0	1	0	50.0
3	LT	Vilnius	Laisves pr., 75	3	1	1	1	1	1	300.0
4	LT	Vilnius	Konstitucijos pr., 34	1	0	0	0	0	1	20.0
5	EE	Tallinn	Tonismagi st., 10	2	1	0	1	1	1	170.0
6	EE	Tallinn	Magdaleena st., 3	2	1	0	0	1	0	140.0

Таблица **clients**:

id	name	phone	email
1	John Smith	+371 123456789	jsmith@gmail.com
2	Anna Bowman	+371 987654321	abowman@gmail.com
3	Ivan Ivanov	+371 543216789	iivanov@gmail.com
4	Irina Petrova	+371 678954321	ipetrova@gmail.com

Содержимое таблиц (2)

Таблица **bookings**:

id	apartment_id	client_id	start_date	end_date	price_total	price_per_day
1	2	2	2023-02-02	2023-02-04	150.00	50.00
2	3	4	2023-02-05	2023-02-06	500.00	250.00
3	1	3	2023-02-11	2023-02-11	100.00	100.00
4	4	1	2023-02-17	2023-02-21	100.00	20.00
5	1	3	2023-03-07	2023-03-09	300.00	100.00
6	2	4	2023-03-22	2023-03-29	400.00	50.00
7	5	3	2023-04-01	2023-04-03	510.00	170.00
8	1	1	2023-04-19	2023-04-24	600.00	100.00
9	3	2	2023-04-28	2023-04-29	500.00	250.00

Таблица **reviews**:

id	booking_id	rating
1	1	3
2	2	5
3	3	4
4	4	1
5	5	1
6	6	2
7	7	2
8	8	4
9	9	5

Введение в подзапросы

Подзапрос - это дополнительный запрос, включенный в другой (основной) запрос (с целью получения дополнительных данных для фильтрации, либо для того, чтобы на лету создавать значения динамических столбцов). Подзапрос всегда заключен в круглые скобки. Время выполнения подзапроса зависит от контекста - он может выполняться как до основного запроса (один раз), так и каждый раз при получении каждой строки основного запроса.

Подзапрос может быть выполнен разными способами - в простейшем случае он может быть подставлен вместо того значения, по которому мы проводим фильтрацию (например, **... WHERE id = (*подзапрос*) ...**). Кроме того, существует ряд функций, которые существуют специально для выполнения подзапросов:

ФУНКЦИЯ	ПОЯСНЕНИЕ
IN	проверяемая строка основного запроса попадет в результат, **если сравниваемое значение (или значения) присутствует в результате подзапроса**
NOT IN	проверяемая строка основного запроса попадет в результат, **если сравниваемое значение (или значения) отсутствует в результате подзапроса**
ANY	**хотя бы одно** возвращенное подзапросом значение должно удовлетворять условию фильтрации - иначе проверяемая строка исключается из результата
ALL	**каждое** возвращенное подзапросом значение должно удовлетворять условию фильтрации - иначе проверяемая строка исключается из результата
EXISTS	если запрос вернул **хотя бы что-то**, то проверяемая строка гарантированно попадет в результат (сравнение в данном случае не применяется)
NOT EXISTS	только если запрос не вернул **ничего**, проверяемая строка попадет в результат (сравнение в данном случае не применяется)

Простейший подзапрос - подстановка вместо фильтруемого значения (1)

Данный подзапрос поддерживает все виды сравнения (**=**, **!=**, **>**, **>=**, **<**, **<=**), однако он должен возвращать 1 строку и то количество столбцов, которое используется при сравнении - другие варианты вызовут ошибку! Далее идут примеры с 1 столбцом:

```
-- сравнение на строгое равенство (=)
SELECT address, city, country_iso FROM apartments WHERE id = (SELECT
apartment_id FROM bookings ORDER BY start_date DESC LIMIT 1);

-- сравнение на строгое неравенство (<> или !=)
SELECT address, city, country_iso FROM apartments WHERE id <> (SELECT
apartment_id FROM bookings ORDER BY start_date DESC LIMIT 1);

-- сравнение на больше (>)
SELECT id, apartment_id, client_id, start_date, end_date FROM bookings WHERE
price_total > (SELECT AVG(price_total) FROM bookings);

-- сравнение на меньше или равно (<=)
SELECT id, apartment_id, client_id, start_date, end_date FROM bookings WHERE
price_total <= (SELECT AVG(price_total) FROM bookings);
```

Простейший подзапрос - подстановка вместо фильтруемого значения (2)

Продолжим рассматривать простейшие подзапросы и приведем пример, когда сравнивается несколько столбцов - этот вид подзапроса практически не используется, т.к. он обладает определенной спецификой. В этом случае применяется сравнение, идентичное лексикографическому. Если мы делаем строгое сравнение (=), то для прохождения фильтрации каждый столбец должен быть равен столбцу подзапроса на той же позиции. Если же мы сравниваем на больше (>, >=) или меньше (<, <=), то при равенстве столбцов сравнение будет продолжаться либо до конца, либо до того момента, когда какой-либо из столбцов будет больше или меньше столбца из подзапроса.

```
-- сравнение по нескольким столбцам (строгое равенство)
SELECT id, apartment_id, client_id, start_date, end_date
FROM bookings WHERE (start_date, end_date) = (SELECT MAX(start_date),
MAX(end_date) FROM bookings);

-- сравнение по нескольким столбцам (сравнение на меньше)
SELECT id, apartment_id, client_id, start_date, end_date
FROM bookings WHERE (start_date, end_date) < (SELECT MAX(start_date),
MAX(end_date) FROM bookings);
```

Подзапрос с применением функции IN (фильтрация по одному столбцу)

Как уже было сказано, применение функции **IN** позволяет оставить в результате основной выборки только те строки, значения столбцов которых присутствуют в результате подзапросе. Важно знать, что в этом случае подзапрос может вернуть более одной строки, но количество столбцов, которое используется при сравнении, должно быть идентичным количеству столбцов подзапроса. Кроме того, в контексте функции **IN** используется только строгое сравнение, сравнение на больше или меньше запрещено. Применим наши знания, чтобы получить те квартиры, которые были сданы в аренду хотя бы один раз:

```
/*
   подзапрос вернет разрешенный набор значений - для каждой строки
   основного запроса будет проверяться, находится ли id этой строки в этом
   наборе или нет - если не находится, то строка будет исключена из результата
*/
SELECT id, country_iso, city, address FROM apartments WHERE id IN (SELECT
apartment_id FROM bookings);
```

Подзапрос с применением функции IN (фильтрация по нескольким столбцам)

Теперь применим функцию **IN** в контексте сравнения двух столбцов - узнаем, какие квартиры не поменяли свою цену с тех пор, как были арендованы (в сравнении будут участвовать сразу два столбца таблицы **apartments** - **id** и **price_per_year**):

```
SELECT id, country_iso, city, address, price_per_day FROM apartments
WHERE (id, price_per_day) IN (SELECT apartment_id, price_per_day FROM
bookings);
```

Подзапрос с применением конструкции NOT IN

Очевидно, что конструкция **NOT IN** обратна применению функции **IN**, т.е. она позволяет оставить в результате основной выборки только те строки, значения столбцов которых **отсутствуют** в результате подзапросе. Все остальные ограничения и особенности с количеством столбцов и строк полностью идентичны. Повторим два предыдущих запроса, но уже с конструкцией **NOT IN**:

```
-- квартиры, которые ни разу не сдавались
SELECT id, country_iso, city, address FROM apartments WHERE id NOT IN
(SELECT apartment_id FROM bookings);

-- квартиры, которые поменяли свою цену после того, как были арендованы
SELECT id, country_iso, city, address, price_per_day FROM apartments
WHERE (id, price_per_day) NOT IN (SELECT apartment_id, price_per_day FROM
bookings);
```

Подзапрос с применением функции ANY

Функция **ANY** очень похожа на функцию **IN** (хотя бы одно значение результата подзапроса должно соответствовать условию), однако она умеет осуществлять не только строгое сравнение (=), но и другие виды сравнения (>, >=, <, <= и т.д.). Кроме того, ANY умеет сравнивать только со значением одного столбца. Попробуем использовать ANY, чтобы получить только тех пользователей, которые арендовали жилье на сумму, большую 500 евро (хотя бы один раз):

```
SELECT c.name FROM clients c WHERE 500 < ANY(SELECT b.price_total FROM
bookings b WHERE b.client_id = c.id);
```

Подзапрос с применением функции ALL

Функция **ALL** является усиленной версией функции **ANY** - теперь не одно значение результата подзапроса должно соответствовать условию, а абсолютно все значения. **ALL** также умеет осуществлять все виды сравнения (>, >=, <, <= и т.д.) и умеет сравнивать только со значением одного столбца. Однако надо помнить, что если подзапрос с ALL вообще не вернет результат, то условие будет признано выполненным (0 результатов отработали правильно 0 раз). Попробуем использовать **ALL**, чтобы получить только те квартиры, которые получили оценки больше 3 (в контексте всех случаев аренды):

```
SELECT a.id, a.country_iso, a.city, a.address FROM apartments a WHERE 3 < ALL(
    SELECT
    r.rating
    FROM bookings b
    JOIN reviews r ON b.id = r.booking_id
    WHERE b.apartment_id  = a.id
) AND a.id IN (SELECT apartment_id FROM bookings);
```

Подзапрос с применением функции EXISTS

EXISTS вообще не требует значения для сравнений - подзапрос, использующий эту функцию будет считаться прошедшим фильтрацию в том случае, если будет найдена хотя бы одна строка. Это значит, что во время подзапроса **EXISTS** не будет обходить всю таблицу - она прекратит выполнение как только найдет первую строку. Следовательно, эта функция в некоторых случаях может ускорить выборку по сравнению с использованием других функций (например **IN**, которая обходит всю таблицу). Применим **EXISTS**, чтобы получить тех клиентов, которые снимали квартиру в Таллине:

```
SELECT c.name FROM clients c WHERE EXISTS (
    SELECT * FROM bookings b
    JOIN apartments a ON b.apartment_id = a.id AND a.city = 'Tallinn'
    WHERE b.client_id = c.id
);
```

Подзапрос с применением функции NOT EXISTS

Как можно понять из названия, конструкция **NOT EXISTS** является абсолютным антиподом функции **EXISTS** - условие подзапроса будет истинным, если в подзапросе не вернется ни одной строки. Во всем остальном (отсутствие значений для сравнения и т.д.) **NOT EXISTS** повторяет поведение **EXISTS**. Применим эту конструкцию, чтобы получить имена клиентов, которые ни разу не ставили единицу за аренду квартиры:

```
SELECT c.name FROM clients c WHERE NOT EXISTS (
    SELECT *
    FROM bookings b
    JOIN reviews r ON b.id = r.booking_id
    WHERE b.client_id = c.id AND r.rating = 1
);
```