нетология

Реляционные базы данных: **SQL. Часть 2.**

Олег
Гежин

Олег Гежин

Инженер-программист, системный администратор УГМК

План занятия

1. JOIN
2. UNION/EXCEPT
3. Агрегатные функции
4. Группировка данных
5. Подзапросы
6. Условия
7. Итоги
8. Домашнее задание

JOIN

JOIN

В SQL JOIN'ы используются для соединения нескольких таблиц и получения из них данных. Существуют следующие типы JOIN:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN
- CROSS JOIN

JOIN

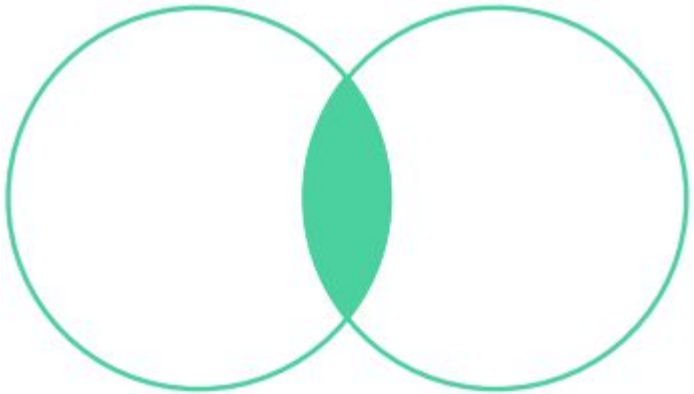
В LEFT OUTER JOIN, RIGHT OUTER JOIN и FULL OUTER JOIN ключевое слово OUTER можно опустить, оно не обязательно для использования.

Также при использовании INNER JOIN можно опустить ключевое слово INNER.

При работе с JOIN желательно использовать алиасы, для удобства чтения/написания запросов и указания, из каких таблиц какие столбцы нужно получать.

INNER JOIN

INNER JOIN возвращает данные по строкам, содержащим одинаковые значения.
Если смотреть на таблицы как на множества строк, то результат их выполнения можно представить на следующей диаграмме Венна:



INNER JOIN

Нужно вывести названия фильмов и имена актеров, которые снимались в этих фильмах.

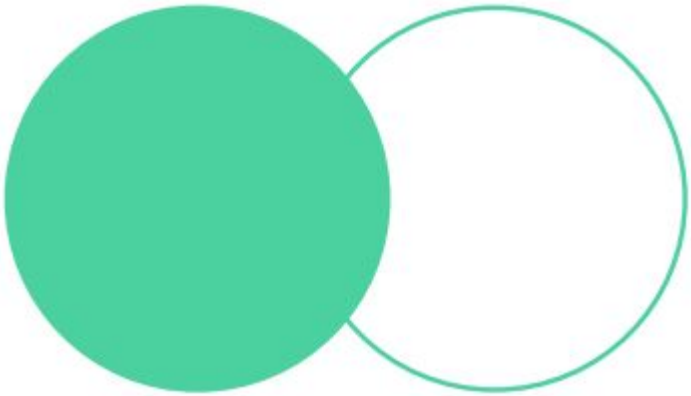
```
SELECT f.title, CONCAT(a.last_name, ' ', a.first_name) AS actor_name
FROM film f
INNER JOIN film_actor fa ON fa.film_id = f.film_id
INNER JOIN actor a ON a.actor_id = fa.actor_id;
```

```
title              |actor_name      |
-------------------+----------------+
SPLASH GUMP        |GUINESS PENELOPE|
VERTIGO NORTHWEST  |GUINESS PENELOPE|
WESTWARD SEABISCUIT|GUINESS PENELOPE|
WIZARD COLDBLOODED |GUINESS PENELOPE|
ADAPTATION HOLES   |WAHLBERG NICK   |
APACHE DIVINE      |WAHLBERG NICK   |
BABY HALL          |WAHLBERG NICK   |
```

В данном случае можно использовать INNER JOIN без потери данных, так как в таблицах есть необходимые ограничения.

LEFT JOIN

LEFT JOIN возвращает все данные из левой таблицы.
Если по ним есть совпадения в правой, они обогащаются соответствующими данными, иначе туда записывается специальное значение NULL.



LEFT JOIN

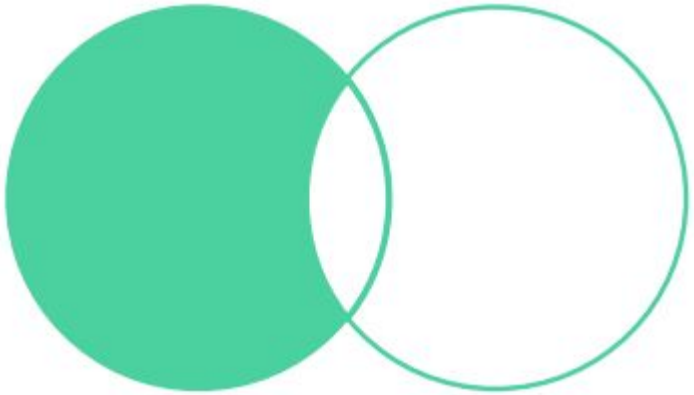
Нужно получить данные по всем пользователям и добавить информацию по городам, в которых они живут.

```
SELECT CONCAT(c.last_name, ' ', c.first_name), c2.city
FROM customer c
LEFT JOIN address a ON a.address_id = c.address_id
LEFT JOIN city c2 ON c2.city_id = a.city_id;
```

```
CONCAT(c.last_name, ' ', c.first_name)|city                |
--------------------------------------+--------------------+
SMITH MARY                            |Sasebo              |
JOHNSON PATRICIA                      |San Bernardino      |
WILLIAMS LINDA                        |Athenai             |
JONES BARBARA                         |Myingyan            |
BROWN ELIZABETH                       |Nantou              |
```

LEFT JOIN

Чтобы получить только те строки, которые не содержат данных в правой таблице, можно использовать оператор WHERE.



LEFT JOIN

Нужно получить все фильмы, которые не брали в аренду.

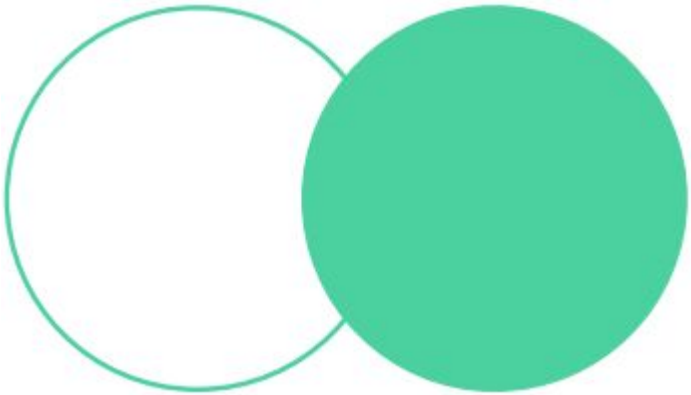
```
SELECT f.title
FROM film f
LEFT JOIN inventory i ON i.film_id = f.film_id
LEFT JOIN rental r ON r.inventory_id = i.inventory_id
WHERE r.rental_id IS NULL;
```

```
title                |
---------------------+
ALICE FANTASIA       |
APOLLO TEEN          |
ARGONAUTS TOWN       |
ARK RIDGEMONT        |
ARSENIC INDEPENDENCE |
BOONDOCK BALLROOM    |
```

Здесь получаем 100% данных по всем фильмам, обогащаем через LEFT JOIN данными по аренде. Фильмы, которые не брали в аренду дополняются значениями NULL, по которым и фильтруем в конце запроса.

RIGHT JOIN

RIGHT JOIN — это обратная версия LEFT JOIN.
Возвращает все данные из правой таблицы.
Если по ним есть совпадения в левой, они обогащаются соответствующими данными, иначе туда записывается специальное значение NULL.



RIGHT JOIN

Нужно получить список всех городов и добавить информацию по пользователям, которые живут в этих городах.

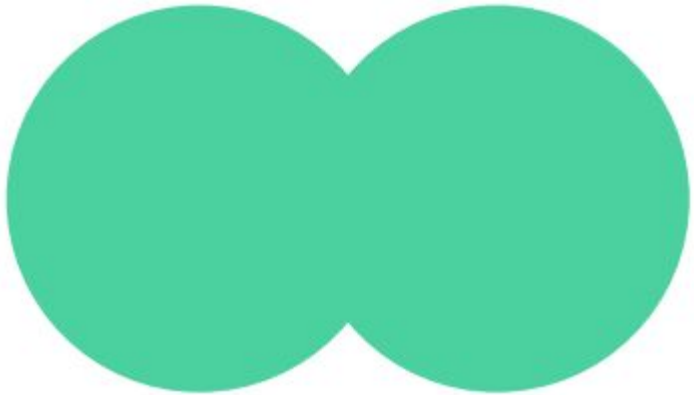
```
SELECT CONCAT(c.last_name, ' ', c.first_name), c2.city
FROM customer c
RIGHT JOIN address a ON a.address_id = c.address_id
RIGHT JOIN city c2 ON c2.city_id = a.city_id;
```

```
CONCAT(c.last_name, ' ', c.first_name)|city                 |
--------------------------------------+---------------------+
SANCHEZ JULIE                         |A Corua (La Corua)   |
MYERS PEGGY                           |Abha                 |
MILNER TOM                            |Abu Dhabi            |
TALBERT GLEN                          |Acua                 |
THRASHER LARRY                        |Adana                |
DOUGLASS SEAN                         |Addis Abeba          |
```

FULL JOIN

FULL JOIN не поддерживается MySQL.
Рассмотрим его синтаксис в других СУБД и как реализовать в MySQL.

FULL JOIN позволяет получить сопоставление по всем строкам в обеих таблицах. То есть получаем все данные из левой и правой таблиц, а там, где сопоставлений нет — добавляются значения NULL.



FULL JOIN

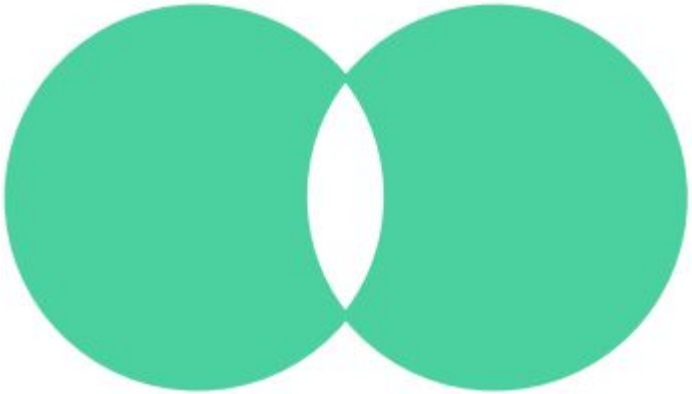
Нужно получить данные по всем арендам и платежам по этим арендам (пример выполнен в PostgreSQL)

```
SELECT r.rental_id, r.rental_date, p.payment_id, p.payment_date, p.amount
FROM rental r
FULL JOIN payment p ON p.rental_id = r.rental_id;
```

rental_id	rental_date	payment_id	payment_date	amount
587	2005-05-28 12:05:33			
1035	2005-05-31 05:01:09			
16040	2005-08-23 22:19:33	24553	2007-03-23 20:47:59.996577	11.99
11479	2005-08-02 22:18:13	24866	2007-03-02 20:46:39.996577	11.99
14763	2005-08-21 23:34:00	23757	2007-03-21 22:02:26.996577	11.99

FULL JOIN

Чтобы получить список уникальных строк из обеих таблиц, можно также воспользоваться оператором WHERE.



FULL JOIN

Нужно найти записи по арендам и платежам, по которым нет пересечения (пример выполнен в PostgreSQL).

```
SELECT r.rental_id, r.rental_date, p.payment_id, p.payment_date, p.amount
FROM rental r
FULL JOIN payment p ON p.rental_id = r.rental_id
WHERE r.rental_id IS NULL OR p.payment_id IS NULL;
```

rental_id	rental_date	payment_id	payment_date	amount
		17507	2007-02-20 17:31:48.996577	7.99
		17503	2007-02-15 22:25:46.996577	7.99
251	2005-05-26 14:35:40			
2024	2005-06-17 13:00:51			
1101	2005-05-31 14:13:59			
599	2005-05-28 14:05:57			

FULL JOIN

Реализация FULL JOIN в MySQL с помощью оператора UNION. Нужно получить данные по всем арендам и платежам по этим арендам.

```
SELECT r.rental_id, r.rental_date, p.payment_id, p.payment_date, p.amount
FROM rental r
LEFT JOIN payment p ON p.rental_id = r.rental_id
UNION
SELECT r.rental_id, r.rental_date, p.payment_id, p.payment_date, p.amount
FROM rental r
RIGHT JOIN payment p ON p.rental_id = r.rental_id;
```

rental_id	rental_date	payment_id	payment_date	amount
1	2005-05-24 22:53:30	3504	2005-05-24 22:53:30	2.99
2	2005-05-24 22:54:33	12377	2005-05-24 22:54:33	2.99
3	2005-05-24 23:03:39	11032	2005-05-24 23:03:39	3.99
4	2005-05-24 23:04:41	8987	2005-05-24 23:04:41	4.99

CROSS JOIN

CROSS JOIN — это Декартово произведение, когда каждая строка левой таблицы сопоставляется с каждой строкой правой таблицы. В результате получается таблица со всеми возможными сочетаниями строк обеих таблиц.

Нужно получить все возможные пары городов и убрать зеркальные варианты А-Б, Б-А

```
SELECT c.city, c2.city
FROM city c
CROSS JOIN city c2
WHERE c.city > c2.city;
```

=

```
SELECT c.city, c2.city
FROM city c, city c2
WHERE c.city > c2.city;
```

```
city                      |city              |
--------------------------+------------------+
Ziguinchor                |A Corua (La Corua)|
Zhoushan                  |A Corua (La Corua)|
Zhezqazghan               |A Corua (La Corua)|
```

UNION / EXCEPT

UNION / EXCEPT

Если при работе с JOIN соединение данных происходит «слева» или «справа», то при работе с операторами UNION или EXCEPT работа происходит «сверху» и «снизу».

Создадим две таблицы и внесем в них данные:

```
CREATE TABLE table_1 (
    color_1 VARCHAR(10) NOT NULL
);
CREATE TABLE table_2 (
    color_2 VARCHAR(10) NOT NULL
);
INSERT INTO table_1
VALUES('white'), ('black'), ('red'), ('green');
INSERT INTO table_2
VALUES('black'), ('yellow'), ('blue'), ('red');
```

```
color_1|    color_2|
-------+    -------+
white  |    black  |
black  |    yellow |
red    |    blue   |
green  |    red    |
```

UNION

При объединении данных через оператор UNION в результате будет список уникальных значений для двух таблиц:

```
SELECT color_1 FROM table_1
UNION
SELECT color_2 FROM table_2;
```

```
color_1|
-------+
white  |
black  |
red    |
green  |
yellow |
blue   |
```

Обязательное условие при работе с операторами UNION или EXCEPT — количество столбцов и их типы данных в таблицах сверху и снизу должно быть одинаковым.

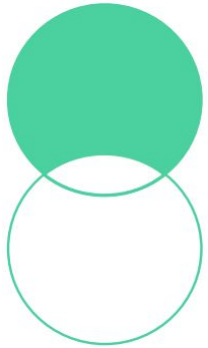
UNION ALL

При объединении данных через оператор UNION ALL в результате будет список всех значений для двух таблиц:

```
SELECT color_1 FROM table_1
UNION ALL
SELECT color_2 FROM table_2;
```

```
color_1|
-------+
white  |
black  |
red    |
green  |
black  |
yellow |
blue   |
red    |
```

EXCEPT



При использовании оператора EXCEPT из значений, полученных в верхней части запроса, будут вычтены значения, которые совпадут со значениями, полученными в нижней части запроса (Запрос выполнен в PostgreSQL):

```
SELECT color_1 FROM table_1
EXCEPT
SELECT color_2 FROM table_2;
```

```
color_1|
-------+
white  |
green  |
```

EXCEPT

Оператор EXCEPT не поддерживается MySQL, но можно такой же результат получить следующим запросом:

```
SELECT color_1
FROM table_1
WHERE color_1 NOT IN (
    SELECT color_2
    FROM table_2
);
```

```
color_1|
-------+
white  |
green  |
```

Агрегатные функции

Агрегатные функции

Агрегация — когда данные группируются по ключу, в качестве которого выступает один или несколько атрибутов, и внутри каждой группы вычисляются некоторые статистики.

- SUM — возвращает общую сумму числового столбца,
- COUNT — возвращает количество строк, соответствующих заданному критерию,
- AVG — возвращает среднее значение числового столбца,
- MIN — возвращает наименьшее значение выбранного столбца,
- MAX — возвращает наибольшее значение выбранного столбца.

Агрегатные функции

Посчитаем, сколько фильмов в базе начинается на букву A:

```
SELECT COUNT(1)
FROM film
WHERE LOWER(LEFT(title, 1)) = 'a';
```

```
COUNT(1)|
--------+
      46|
```

Так как функция COUNT возвращает количество строк, полученных в результате запроса, то аргументом можно передать любое значение, главное, чтобы оно соответствовало смыслу задачи.

Агрегатные функции

В одном запросе получим информацию по количеству платежей, общей сумме платежей, среднему платежу, максимальному и минимальному платежу по каждому пользователю:

```
SELECT customer_id, COUNT(payment_id), SUM(amount),
    AVG(amount), MIN(amount), MAX(amount)
FROM payment
GROUP BY customer_id;
```

customer_id	COUNT(payment_id)	SUM(amount)	AVG(amount)	MIN(amount)	MAX(amount)
1	32	118.68	3.708750	0.99	9.99
2	27	128.73	4.767778	0.99	10.99
3	26	135.74	5.220769	0.99	10.99
4	22	81.78	3.717273	0.99	8.99
5	38	144.62	3.805789	0.99	9.99
6	28	93.72	3.347143	0.99	7.99
7	33	151.67	4.596061	0.99	8.99
8	24	92.76	3.865000	0.99	9.99

Группировка данных

Группировка данных

GROUP BY — агрегирующий оператор, с помощью которого можно формировать данные по группам и уже в рамках этих групп получать значения с помощью агрегатных функций.

Группировать можно как по одному атрибуту, так и по нескольким. При этом важно помнить, что все значения указанные в SELECT, которые не указаны внутри агрегатных функций, должны быть указаны в операторе GROUP BY.

GROUP BY

В одном запросе получим информацию по количеству платежей и общей сумме платежей по каждому пользователю на каждый месяц:

```
SELECT customer_id, MONTH(payment_date), COUNT(payment_id), SUM(amount)
FROM payment
GROUP BY customer_id, MONTH(payment_date);
```

customer_id	MONTH(payment_date)	COUNT(payment_id)	SUM(amount)
1	5	2	3.98
1	6	7	31.93
1	7	12	50.88
1	8	11	31.89
2	5	1	4.99
2	6	1	2.99
2	7	14	75.86
2	8	11	44.89
3	5	2	4.98

GROUP BY

Если при использовании агрегации и группировки данных нужно вывести несколько столбцов из одной таблицы, то вместо указания всех этих столбцов в GROUP BY можно использовать Функциональную Зависимость.

GROUP BY

В примере ниже, вместо указания в GROUP BY столбцов title, release_year и lenght можно указать первичный ключ таблицы film – film_id:

```
SELECT f.title, f.release_year, f.length, COUNT(fa.actor_id)
FROM film f
JOIN film_actor fa ON fa.film_id = f.film_id
GROUP BY f.film_id;
```

title	release_year	length	COUNT(fa.actor_id)
ACADEMY DINOSAUR	2006	86	10
ACE GOLDFINGER	2006	48	4
ADAPTATION HOLES	2006	50	5
AFFAIR PREJUDICE	2006	117	5

HAVING

Вспоминая логический порядок инструкции SELECT:
оператор WHERE фильтрует данные до группировки, а чтобы отфильтровать сгруппированные данные, используется оператор HAVING. Найдем пользователей, которые совершили более 40 аренд:

```
SELECT CONCAT(c.last_name, ' ', c.first_name), COUNT(r.rental_id)
FROM rental r
JOIN customer c ON r.customer_id = c.customer_id
GROUP BY c.customer_id
HAVING COUNT(r.rental_id) > 40;
```

CONCAT(c.last_name, ' ', c.first_name)	COUNT(r.rental_id)
SANDERS TAMMY	41
SHAW CLARA	42
HUNT ELEANOR	46
DEAN MARCIA	42
SEAL KARL	45

Подзапросы

Подзапросы

Подзапрос — это SELECT, результаты которого используются в другом SELECT. Подзапросы нужны для разделения логики в основном запросе.

Подзапросы могут использоваться в любой части запроса, в зависимости от этой части запроса подзапросы могут возвращать:

- отдельное значение,
- таблицу,
- одномерный массив.

Если подзапрос возвращает таблицу, подзапросу обязательно задается алиас.

Подзапросы

Нужно получить процентное отношение платежей по каждому месяцу к общей сумме платежей:

```
SELECT MONTH(payment_date),
    COUNT(payment_id) / (SELECT COUNT(1) FROM payment) * 100
FROM payment
GROUP BY MONTH(payment_date);
```

MONTH(payment_date)	COUNT(payment_id) / (SELECT COUNT(1) FROM payment) * 100
5	7.2092
6	14.4059
7	41.8157
8	35.4352
2	1.1340

Подзапросы

Нужно получить фильмы из категорий, начинающихся на букву C:

```
SELECT f.title, c.name
FROM film f
JOIN film_category fc ON fc.film_id = f.film_id
JOIN category c ON c.category_id = fc.category_id
WHERE c.category_id IN (
    SELECT category_id
    FROM category
    WHERE name LIKE 'C%')
ORDER BY f.title;
```

```
title                   |name    |
------------------------+--------+
AIRPLANE SIERRA         |Comedy  |
ALICE FANTASIA          |Classics|
ANTHEM LUKE             |Comedy  |
ARIZONA BANG            |Classics|
BACKLASH UNDEFEATED     |Children|
BEAR GRACELAND          |Children|
BEAST HUNCHBACK         |Classics|
```

Подзапросы

Получим отношение количества платежей к количеству аренд по каждому сотруднику:

```
SELECT CONCAT(s.last_name, ' ', s.first_name), cp / cr
FROM staff s
JOIN (
    SELECT staff_id, COUNT(payment_id) AS cp
    FROM payment
    GROUP BY staff_id) t1 ON s.staff_id = t1.staff_id
JOIN (
    SELECT staff_id, COUNT(rental_id) AS cr
    FROM rental
    GROUP BY staff_id) t2 ON s.staff_id = t2.staff_id;
```

```
CONCAT(s.last_name, ' ', s.first_name)|cp / cr|
--------------------------------------+-------+
Hillyer Mike                          | 1.0021|
Stephens Jon                          | 0.9985|
```

Условия

CASE

Выражение CASE в SQL представляет собой общее условное выражение, напоминающее операторы if/else в других языках программирования. Типы данных всех выражений результатов должны приводиться к одному выходному типу.

CASE

В запросе мы проверяем, что если пользователь купил более чем на 200 у. е., то он хороший клиент, если менее чем на 200, то не очень хороший, в остальных случаях — «средний».

```
SELECT customer_id, SUM(amount),
    CASE
         WHEN SUM(amount) > 200 THEN 'Good user'
         WHEN SUM(amount) < 200 THEN 'Bad user'
         ELSE 'Average user'
    END AS good_or_bad
FROM payment
GROUP BY customer_id
ORDER BY SUM(amount) DESC
LIMIT 5;
```

```
customer_id|SUM(amount)|good_or_bad|
-----------+-----------+-----------+
        526|     221.55|Good user  |
        148|     216.54|Good user  |
        144|     195.58|Bad user   |
        178|     194.61|Bad user   |
        137|     194.61|Bad user   |
```

IFNULL

Функция IFNULL позволяет возвращать альтернативное значение, если выражение возвращает NULL.

Нужно получить список всех пользователей и сумму их платежа за 18.06.2005, вместо значений NULL нужно проставить 0.

```
SELECT CONCAT(c.last_name, ' ', c.first_name) AS user,
    IFNULL(SUM(p.amount), 0)
FROM customer c
LEFT JOIN (
    SELECT *
    FROM payment
    WHERE DATE(payment_date) = '2005-06-18') p
ON p.customer_id = c.customer_id
GROUP BY c.customer_id
```

user	amount
SMITH MARY	5.98
JOHNSON PATRICIA	0.00
WILLIAMS LINDA	0.00
JONES BARBARA	0.00
BROWN ELIZABETH	0.00
DAVIS JENNIFER	0.99
MILLER MARIA	2.99
WILSON SUSAN	0.00

COALESCE

Функция COALESCE позволяет возвращать первое значение из списка, которое не равно NULL.

Выведем в результат первый не NULL результат разницы между датой аренды и датой возврата, текущей даты и даты возврата, текущей даты и даты аренды.

```
SELECT rental_id,
    COALESCE(DATEDIFF(return_date, rental_date), DATEDIFF(NOW(), return_date),
        DATEDIFF(NOW(), rental_date)) AS diff
FROM rental
```

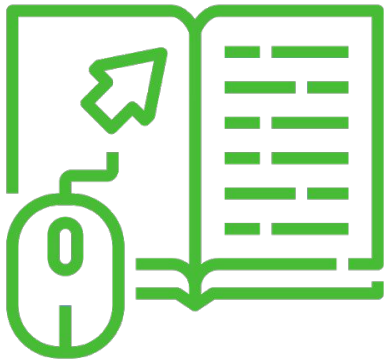
```
rental_id|diff|
---------+----+
        1|   2|
        2|   4|
        3|   8|
        4|  10|
        5|   9|
        6|   3|
```

Итоги

Итоги

В данной лекции мы:

- Научились соединять таблицы с помощью различных типов JOIN.
- Разобрали, как работают операторы UNION и EXCEPT.
- Посмотрели на работу агрегатных функций и группировку данных.
- Научились использовать подзапросы.
- Разобрали условия CASE.



Домашнее задание

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Олег Гежин