



# Оконные функции

Оконные функции

Типы оконных функций

Partition by

Order by

## Оконные функции




Окно — это некоторое выражение, описывающее набор строк, которые будет обрабатывать функция, и порядок этой обработки.

**Как работают оконные функции:**

- разбивают набор данных на окна с одинаковыми значениями колонок, указанных в partition by
- если указан order by сортируют данные в указанном порядке в рамках окна
- если указан rows between применяют ограничение на окно
- выполняют вычисления над строками из окна в указанном порядке с учетом границ
- не группируют строки в одну результирующую, а оставляют исходные и добавляют информацию
- похожи на обычные функции, но могут получать доступ к «соседним» строкам

- используются только в списке SELECT и в предложении ORDER BY

<название функции>(<выражение>) over ( <окно> <сортировка> <границы окна> )	sum(value) over ( partition by ... order by ... rows between ... and ... )
--	--

 Ключевое слово **OVER** задает окно и является признаком оконной функции. Если в запросе есть OVER, то это оконная функция. И наоборот, в оконную функцию ключевое слово OVER должно быть включено. Окно заданное пустыми скобками () после OVER – это весь набор данных, в таком случае окном являются все строки результата запроса.

## Типы оконных функций

- **Агрегатные: COUNT, SUM, AVG, MIN, MAX**

Могут вызываться и как оконные, и как обычные агрегатные.

Вычисляют соответствующее значение на основании значения выражения на всех строках которые попали в окно

Агрегатные оконные функции не допускают использования DISTINCT в списке аргументов.

- **Ранжирующие: ROW\_NUMBER, RANK, DENSE\_RANK**

Встроенные оконные функции и должны вызываться именно как оконные, то есть при вызове обязательно использовать ключевое слово OVER

- **Функции значений: LEAD, LAG, FIRST\_VALUE, LAST\_VALUE**

Возвращают значение выражения, вычисленного для какой-либо строки из своей группы. Аналогичных агрегатных функций нет, обязательно использовать ключевое слово OVER

## Partition by

Разбивает набор данных на окна с одинаковыми значениями колонок, указанных в partition by

```

select
  user_id,
  event_date,
  amount,
  sum(amount) over (
    partition by user_id
  )
from payments

```

user_id	event_date	amount	sum
1	2021-01-01	10	30
1	2021-01-02	10	30
1	2021-01-03	10	30
2	2021-01-02	20	40
2	2021-01-03	20	40
3	2021-01-03	30	30

В отличие от обычных агрегатных функций, оконные функции не группируют строки в одну результирующую строку, а оставляют количество строк в наборе данных неизменным, добавляя некоторую информацию.

```

select
    user_id,
    amount,
    sum(amount) over (
        partition by user_id
    )
from payments

```

user_id	amount
1	10
1	10
1	10
2	20
2	20
3	30

user_id	amount	sum
1	10	30
1	10	30
1	10	30
2	20	40
2	20	40
3	30	30

```

se
    user_id,
    sum(amount)
from payments
group by user_id

```

user_id	amount
1	10
1	10
1	10
2	20
2	20
3	30

user_id	sum
1	30
2	40
3	30



## Пример:

Дана таблица employee (id, name, salary, manager\_id)

Нужно каждому сотруднику сопоставить сумму зарплат всех подчиненных того же менеджера.

```

SELECT name, salary, manager_id,
       SUM(salary) OVER (PARTITION BY manager_id)
       AS manager_total
FROM employee;

```

Получились окна: 1, 3, NULL. Все строки каждой части равноправны. Результат запроса будет таким:

Name	Salary	Manager_id	Manager_total
Сотрудник Авито #2	3000	1	7000
Сотрудник Авито #3	4000	1	7000
Сотрудник Авито #4	1000	3	4500
Сотрудник Авито #5	1500	3	4500
Сотрудник Авито #6	2000	3	4500
Сотрудник Авито #1	5000		5000

В поле **manager\_total** для каждой строки получена сумма в разрезе менеджера. Исходные данные строк же остались неизменными: они разбиты на окна (или группы, выделены цветом), по значению **manager\_id**.

Каждая строка получила доступ к соседним строкам в своего окна.

👉 Решение через оконную функцию бывает значительно короче других вариантов и может выполняться быстрее.

## Order by

Сортирует данные в рамках окна.

Значение агрегатной функции вычисляется от начала окна до текущей строки.  
По умолчанию - range between unbounded preceding and current row.

```

select
  user_id,
  event_date,
  amount,
  sum(amount) over (
    partition by user_id
    order by event_date
    rows between unbounded preceding
    and current row
  )
from payments

```

user_id	event_date	amount	sum
1	2021-01-01	10	10
1	2021-01-02	10	20
1	2021-01-03	10	30
2	2021-01-02	20	20
2	2021-01-03	20	40
3	2021-01-03	30	30

Для изменения границ внутри окна используются ключевые слова

**rows between ... and ...** задает смещение в количестве строк

**range between ... and ...** задает смещение как дельту от значения текущей строки в колонке из сортировки

Для того чтобы указать в какую сторону от текущей строки смещать границу используются ключевые слова **preceding** и **following**

Начало и конец окна задаются ключевым словом **unbounded**

Текущая строка - ключевым словом **current row**

Например: rows between 1 following and 5 following

range between interval '1 day' preceding and current row

rows between current row and unbounded following

👉 Если набор колонок в order by содержит повторяющиеся значения, порядок сортировки таких дубликатов не определен. При разных запусках такого запроса могут получаться разные результаты

```
select
  user_id,
  event_date,
  amount,
  sum(amount) over (
    partition by user_id
    order by event_date
  )
from payments
```

user_id	event_date	amount
1	2021-01-01	10
1	2021-01-02	20
1	2021-01-02	30
1	2021-01-03	50

user_id	event_date	amount	sum
1	2021-01-01	10	✓ 10
1	2021-01-02	20	? 30
1	2021-01-02	30	60
1	2021-01-03	50	✓ 110

user_id	event_date	amount	sum
1	2021-01-01	10	✓ 10
1	2021-01-02	30	? 40
1	2021-01-02	20	60
1	2021-01-03	50	✓ 110

Когда используются несколько оконных функций, все оконные функции, имеющие в своих определениях одинаковые PARTITION BY и ORDER BY, обрабатывают данные за один проход. Поэтому они увидят один порядок сортировки, даже если ORDER BY не определяет порядок однозначно.

Если PARTITION BY или ORDER BY отличается - никаких гарантий не дается. В таких случаях между проходами вычислений оконных функций обычно требуется дополнительный этап сортировки, и эта сортировка может не сохранять порядок строк

👉 ORDER BY указанный в определении окна функции не имеет отношения к сортировке результата. Если нужно, чтобы результаты сортировались определенным образом, явно добавьте предложение ORDER BY в конце запроса

### 💻 Пример:

Дана таблица payments (user\_id, event\_date, amount)

Нужно на каждую дату каждому пользователю сопоставить сумму платежей к указанной дате.

```
select user_id, event_date, amount,  
       sum(amount) over (  
         partition by user_id  
         order by event_date  
         rows between unbounded preceding and current row  
       ) cumulative_amount  
from payments
```

В данном случае данные разбиваются на окна по пользователям: 1, 2 и сортируются по event\_date

При вычислении поля cumulative\_amount для строки N учитываются все строки того же окна у которых event\_date находится в границах от начала окна до строки N

Наличие ORDER BY и так задает нужные нам границы окна, но для наглядности они прописаны явно


Результат запроса будет выглядеть так:

user_id	event_date	amount	cumulative_amount
1	2021-01-01	10	10
1	2021-01-02	10	20
1	2021-01-03	10	30
2	2021-01-02	20	20
2	2021-01-03	20	40

 **Пример онлайн**




### DB Fiddle - SQL Database Playground

An online SQL database playground for testing, debugging and sharing SQL snippets.

 <https://www.db-fiddle.com/f/vE8UCsRYVaarLTFcfPGyVS/0>

---

## Вырожденные случаи

-  Отсутствие конструкции PARTITION BY означает, что все строки результирующего набора образуют одно единственное окно.
  -  Отсутствие конструкции ORDER BY для агрегирующей оконной функции означает что при вычислении значения для конкретной строки в функцию попадут значения всех строк окна без учета сортировки
  -  Отсутствие конструкции ORDER BY для других оконных функций приведет к ошибке. Но можно сортировать по константе (=
-



# Группировка и оконные функции

Если в запросе есть агрегатные функции, предложения GROUP BY или HAVING, оконные функции видят не исходные строки, полученные из FROM/WHERE, а сгруппированные.

указанные ниже запросы аналогичны:

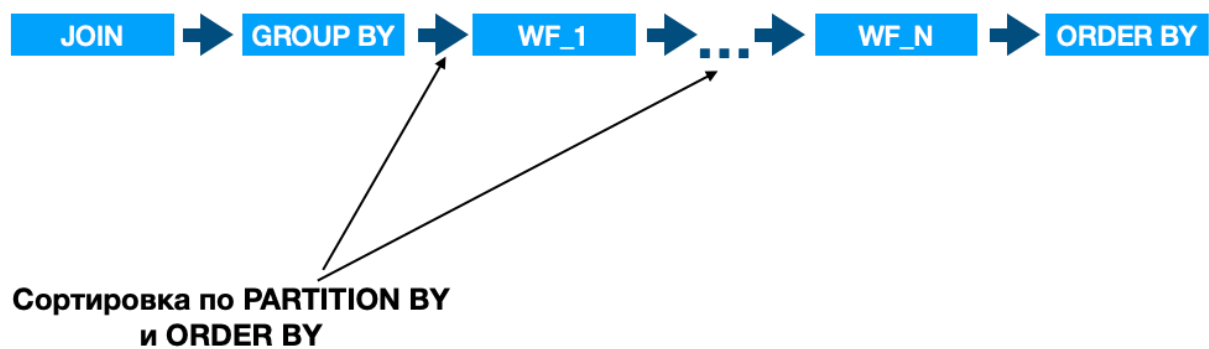
```
select user_id, event_date,  
sum(min(amount)) over ( partition by  
user_id order by event_date ) from  
payments group by 1,2 order by 1,2
```

```
select user_id, event_date, sum(amount) over (  
partition by user_id order by event_date ) from (  
select user_id, event_date, sum(amount) amount  
from payments group by 1,2 ) sq order by 1,2
```

## Порядок вычисления оконных функций

- После FROM, JOIN, WHERE
- После GROUP BY/HAVING
- До финальных ORDER BY, DISTINCT, LIMIT
- Можно иметь несколько оконных функций с разными окнами





Для фильтрации по значению в оконных функциях используется подзапрос

```
select user_id, event_date, amount, total
from (
  select user_id, event_date, amount,
         sum(amount) over (partition by user_id) total
  from payments
) sq where total >= 40
```



# Ранжирующие функции: ROW\_NUMBER, RANK, DENSE\_RANK

Ранжирующие функции используются для нумерации строк в порядке сортировки.

Они отличаются поведением при наличии дубликатов значений в колонке сортировки

**ROW\_NUMBER** — Дубликаты сортируются произвольным образом (1, 2, 3, 4, 5, ...)

**RANK** — Дубликаты получают одинаковый номер. Строка следующая за дубликатом получает тот же номер что и в случае ROW\_NUMBER (1, 2, 2, 2, 5, ...)

**DENSE\_RANK** — Дубликаты получают одинаковый номер. Строка следующая за дубликатом - номер на единицу больше (1, 2, 2, 2, 3, ...)

```
select user_id, event_date,
       row_number() over (
         partition by user_id
         order by event_date),
       rank() over (
         partition by user_id
         order by event_date),
       dense_rank() over (
         partition by user_id
         order by event_date)
from payments
```

user_id	event_date	row_number	rank	dense_rank
1	2021-01-01	1	1	1
1	2021-01-02	2	2	2
1	2021-01-02	3	2	2
1	2021-01-02	4	2	2
1	2021-01-03	5	5	3



## Пример:

Дана таблица payments (user\_id, event\_date, amount)

Нужно пронумеровать операции для каждого пользователя в отдельности

```
SELECT
  ROW_NUMBER() OVER(PARTITION BY user_id ORDER BY event_date) as rn,
  user_id, event_date, amount
FROM payments
ORDER BY user_id, event_date
```

Предложение OVER, с которым используется функция ROW\_NUMBER, задает правила нумерации строк.

Конструкция PARTITION BY задает окна, для которых выполняется независимая нумерация. Окно определяется равенством значений в списке столбцов, перечисленных в этой конструкции.

Предложение ORDER BY внутри OVER указывает порядок нумерации, но не имеет отношения к порядку вывода строк запроса.

rn	user_id	event_date	amount
1	1	2021-01-01	10
2	1	2021-01-02	10
3	1	2021-01-03	10
1	2	2021-01-02	20
2	2	2021-01-03	20




## Пример посложнее:

Дана таблица sequence. В ней содержится последовательность чисел с пропусками. Нужно написать запрос, который выведет границы последовательностей без пропусков.

```

SELECT
  MIN(n) AS left, MAX(n) AS right
FROM (
  SELECT
    n, n - ROW_NUMBER() OVER(ORDER BY n) AS grp
  FROM sequence
) t
GROUP BY grp
ORDER BY 1

```

N		Left	Right
1			
2		1	2
5		5	5
10		10	12
11		15	15
12		17	17
15			
17			

Для решения нужно найти значение, которое объединит последовательность без пропусков в одну группу.

Сначала каждому n добавить его порядковый номер в таблице rn

Заметим, что пока пропусков нет, значение разницы между n и rn постоянна.

Если вычесть из n текущее значение row\_number() – получается разбить последовательность на группы подпоследовательностей без пропусков. После этого данные группируются и находятся граничные значения.

```

SELECT
  MIN(n) AS left, MAX(n) AS right
FROM (
  SELECT
    n,
    ROW_NUMBER() OVER (ORDER BY n) AS RN
    n-ROW_NUMBER() OVER (ORDER BY n) AS grp
  FROM sequence
) t
GROUP BY grp
ORDERBY 1

```

N		N	RN		N	GRP		GRP	MIN(N)	MAX(N)
1		1	1		1	0				
2		2	2		2	0		0	1	2
5		5	3		5	2		2	5	5
10	→	10	4	→	10	6	→	6	10	12
11		11	5		11	6		8	15	15
12		12	6		12	6		9	17	17
15		15	7		15	8				
17		17	8		17	9				

## Функции значения

Возвращают значение выражения, вычисленного для какой-либо строки из соответствующего окна

**FIRST\_VALUE** - значение выражения для первой строки из окна

**LAST\_VALUE** - значение выражения для последней строки из окна. Границы по умолчанию для окна этой функции - все окно.

**LEAD** - значение выражения для следующей строки из окна

**LAG** - значение выражения для предыдущей строки из окна

Если значение не может быть найдено, например, предыдущее значение для первой строки, будет выведен null.

Некоторое поведение функций можно изменить, читайте документацию по используемой системе

Один и тот же результат можно получить используя разные функции и разные сортировки. Старайтесь использовать сортировку по возрастанию, как наиболее интуитивно понятную. Старайтесь использовать одинаковые определения окон в разных функциях одного запроса.

Приведенные ниже примеры возвращают одинаковый результат

```
with src(n) as (
  select generate_series(1,5)
)
select
  n,
  lead(n) over (
    order by n asc
  ) as next
from src
order by 1
```

N	NEXT
1	2
2	3
3	4
4	5
5	

```
with src(n) as (
  select generate_series(1,5)
)
select
  n,
  lag(n) over (
    order by n desc
  ) as next
from src
order by 1
```



## Пример

Дана таблица stocks\_example (stock\_id, event\_date, value)

Для каждой акции выведите изменение ее цены в процентах по отношению к предыдущему дню

```
select
  stock_id, event_date, value, lag_value,
  100 * (value - lag_value) / lag_value
from (
  select stock_id, event_date, value,
    lag(value) over (
      partition by stock_id
      order by event_date
    ) as lag_value
  from stocks_example
)sq order by 1,2
```

stock_id	event_date	value	lag_value	diff
1	2021-01-01	100.00		
1	2021-01-02	102.85	100.00	2.85
1	2021-01-03	110.37	102.85	7.31
2	2021-01-02	89.35		
2	2021-01-03	87.60	89.35	-1.96

## Примечания

📌 Оконные функции не изменяют количество строк запроса. Если вы используете `distinct` вместе с оконными функциями, скорее всего вы делаете что-то не то и нужно пересмотреть логику запроса.

Найдите значение первой операции каждого пользователя

<p>Плохо.</p> <p><code>distinct</code> - дорогая операция.</p> <pre>select distinct   user_id,   first_value(amount) over (     partition by user_id     order by event_date   ) from payments</pre>	<p>Хорошо.</p> <pre>select   user_id, amount from (   select     user_id, amount,     row_number() over (       partition by user_id       order by event_date     ) as rn   from payments ) sq where rn = 1 order by 1</pre>
--	---

📌 Значение оконных функций сильно зависит от порядка сортировки. Старайтесь сортировать по набору колонок с уникальными значениями, чтобы избежать неопределенного поведения.

Дано: таблица `events` (`event_time`, `event_type`, `event_number`)

Нужно найти первую операцию за день.

Данные	Плохо. В таблице есть строки с одинаковым event_time Результат запроса не определен однозначно	Хорошо.												
<table border="1"> <thead> <tr> <th>event_time</th><th>event_type</th><th>event_number</th></tr> </thead> <tbody> <tr> <td>2021-01-01 12:00:00</td><td>buy</td><td>1</td></tr> <tr> <td>2021-01-01 13:00:00</td><td>sell</td><td>3</td></tr> <tr> <td>2021-01-01 12:00:00</td><td>sell</td><td>2</td></tr> </tbody> </table>	event_time	event_type	event_number	2021-01-01 12:00:00	buy	1	2021-01-01 13:00:00	sell	3	2021-01-01 12:00:00	sell	2	<pre>select   event_type from (   select     event_type,     row_number() over (       order by event_date     ) as rn   from events ) sq where rn = 1</pre>	<pre>select   event_type from (   select     event_type,     row_number() over (       order by event_date,       event_number     ) as rn   from events ) sq where rn = 1</pre>
event_time	event_type	event_number												
2021-01-01 12:00:00	buy	1												
2021-01-01 13:00:00	sell	3												
2021-01-01 12:00:00	sell	2												

### Вопрос на подумать:

можете объяснить почему функция last\_value возвращает именно такой результат?





# Практика: Оконные функции

## Задача 0

### Условие:

Дана таблица

```
d8_training_log (training_date, player_id, training_result);
```

training_date	players
2020-01-30	4
2020-01-24	2
2020-01-11	2

### Вопрос:

Вывести топ 3 дня, когда тренировалось больше всего игроков.

## Задача 1

### Условие:

Дана таблица

```
d8_training_log (training_date, player_id, training_result);
```

player_id	max_result	min_trainig_date_with
1	1	2020-01-30
2	5	2020-01-11
3	8	2020-01-30
4	3	2020-01-26
5	10	2020-01-27

### Вопрос:

Для каждого игрока выведите максимальный результат (training\_result) за тренировку и минимальную дату, когда этот игрок получил свой максимальный результат без использования join.

## Задача 2

### Условие:

Дана таблица

```
d8_tournament_log (match_date, player_id, opponent_id, match_result, score);
```

player_id	score	max_date
1	266	2020-02-02
2	249	2020-02-01
3	249	2020-01-31
4	230	2020-02-02
5	138	2020-01-31

### Вопрос:

Выведите последнюю даты игры и суммарное кол-во заработанных очков (score) для каждого игрока. Отсортируйте по ID игрока.

## Задача 3

#### Условие:

Дана таблица

```
d8_tournament_log (match_date, player_id, opponent_id, match_result, score);
```

	match_date	123 player_id	123 sum
1	2020-01-31	1	164
2	2020-02-01	1	194
3	2020-02-02	1	266
4	2020-01-13	2	95
5	2020-01-25	2	150
6	2020-01-27	2	234
7	2020-02-01	2	249

#### Вопрос:

Для каждого игрока подсчитайте накопленный score ко дню каждого матча.

Обратите внимание что в один день могло быть две игры.

### Задача 4

#### Условие:

```
d8_player (player_id, player_name, registration_date);
```

ABC player_name	123 diff
Васильев Василий Васильевич	0
Григорьев Григорий Григорьевич	14
Иванов Иван Иванович	19
Петров Петр Петрович	15
Федоров Федор Федорович	0

#### Вопрос:

Каждому игроку сопоставьте число дней, прошедших от регистрации первого игрока до его регистрации без использования join.

### Задача 5

#### Условие:

```
d8_training_log (training_date, player_id, training_result);
```

123 player_id	training_date	123 numeric
1	2020-01-30	1
2	2020-01-11	0,29
2	2020-01-18	0,29
2	2020-01-20	0,24
2	2020-01-24	0,06
2	2020-01-25	0,06
2	2020-01-28	0,06

#### Вопрос:

На каждую тренировочную игру выведите какую долю составляет ее результат от суммарного этого игрока.

### Задача 6

#### Условие:

```
d8_training_log (training_date, player_id, training_result);
```

#### Вопрос:

Для каждого игрока подсчитайте накопленный training result ко дню каждой тренировки. Используйте `rows between`.

match_date	player_id	sum
2020-01-31	1	164
2020-02-01	1	194
2020-02-02	1	266
2020-01-13	2	95
2020-01-25	2	150
2020-01-27	2	234
2020-02-01	2	249

## Задача 7

**Условие:**

```
d8_training_log (training_date, player_id, training_result);
```

**Вопрос:**

Для каждого игрока выведите сумму очков, которые он получил с текущей даты включительно и за следующие 3 дня.

match_date	player_id	score
2020-01-31	1	266
2020-02-01	1	102
2020-02-02	1	72
2020-01-13	2	95
2020-01-25	2	139
2020-01-27	2	84
2020-02-01	2	15

## Задача 8

**Условие:**

```
d8_training_log (training_date, player_id, training_result);
```

**Вопрос:**

Для каждого игрока выведите сумму очков, которые он получил с текущей даты включительно и за следующие 3 тренировки.

	match_date	player_id	sum
1	2020-01-31	1	266
2	2020-02-01	1	102
3	2020-02-02	1	72
4	2020-01-13	2	249
5	2020-01-25	2	154
6	2020-01-27	2	99
7	2020-02-01	2	15

## Задача 9

**Условие:**

```
d8_player (player_id int, player_name text, registration_date date);
```

```
d8_player_city (player_id int, city text, actual_date date);
```

player_id	city
1	Владивосток
2	[NULL]
3	Москва
4	[NULL]
5	Владивосток

**Вопрос:**

Для каждого игрока найдите последний актуальный город.

Какой последний город у игрока с id 3?

## Задача 10

### Условие:

```
d8_tournament_log (match_date, player_id, opponent_id, match_result, score);
```

123 player_id	12 total_result	12 total_score	12 dense_rank
2	4	249	1
1	2	266	2
3	2	249	2
5	2	138	2
4	0	230	3

### Вопрос:

Найдите игроков с наибольшим количеством побед.

Выведите игроков занявших призовые 1, 2 и 3 место. Несколько игроков могут занимать одно место.

⚠ прежде чем открывать ответы, попробуй решить задачи самостоятельно

### ▼ Ответы

#### ▼ Ответ: задача 0

```
select distinct training_date, count(player_id) over(partition by training_date) players
from (select distinct training_date, player_id
      from d8_training_log) t
order by 2 desc
limit 3
```

#### ▼ Ответ: задача 1

```
with t as (
select training_date, player_id, training_result, max(training_result) over (partition by player_id) max_result
from d8_training_log
order by player_id, training_date
)
select player_id, max_result, min(training_date) min_trainig_date_with
from t
where max_result=training_result
group by 1,2
```

#### ▼ Ответ: задача 2

```
select distinct player_id, sum(score) over (partition by player_id) score, max(match_date) over(partition by player_id) max_da
from d8_tournament_log
order by 1
```

#### ▼ Ответ: задача 3

```
with src as (
select match_date, player_id, sum(score) score
from d8_tournament_log
group by 1,2
)
select match_date, player_id, sum(score) over (partition by player_id order by match_date)
from src
order by player_id, match_date

или

select distinct match_date, player_id, sum(score) over (partition by player_id)
from d8_tournament_log
order by player_id, match_date
```

▼ Ответ: задача 4

```
select player_name, registration_date - min(registration_date) over () diff
from d8_player
order by player_name
```

▼ Ответ: задача 5

```
select player_id, training_date, (training_result::numeric(5,2) / sum(training_result) over (partition by player_id))::numeric
from d8_training_log
order by player_id, training_date
```

▼ Ответ: задача 6

```
with t as
  (select match_date, player_id, sum(score) over(partition by player_id order by match_date rows between unbounded preceding
    from d8_tournament_log
    order by player_id, match_date)
  select distinct match_date, player_id, max(score) over(partition by player_id, match_date) score
  from t
  order by 2,1
```

▼ Ответ: задача 7

```
select distinct match_date, player_id, sum(score) over(partition by player_id order by match_date range between current row an
  from d8_tournament_log
  order by player_id, match_date
```

▼ Ответ: задача 8

```
with t as
  (select match_date, player_id, sum(score) score
  from d8_tournament_log
  group by player_id, match_date)
select match_date, player_id, sum(score) over(partition by player_id order by match_date rows between current row and 3 follow
from t
order by 2,1
```

▼ Ответ: задача 9

```
with city as (
  select player_id, city
  from (
    select player_id, city, row_number() over (partition by player_id order by actual_date desc) rn
    from d8_player_city
    order by 1
  ) _ where rn = 1
)
select p.player_id, c.city
from d8_player p
left join city c on p.player_id = c.player_id
order by 1
```

▼ Ответ: задача 10

```
with src as (
  select player_id, sum((match_result>0)::int) total_result, sum(score) total_score
  from d8_tournament_log
  group by 1
)
select *
from (
  select player_id, total_result, total_score,
    dense_rank() over (order by total_result desc)
  from src
) _ where dense_rank <= 3
order by total_result desc, total_score desc
```

