

ТЕХНОАТОМ

# Смешанное занятие №2

## Аналитические запросы

## MySQL

Ставровская Елена



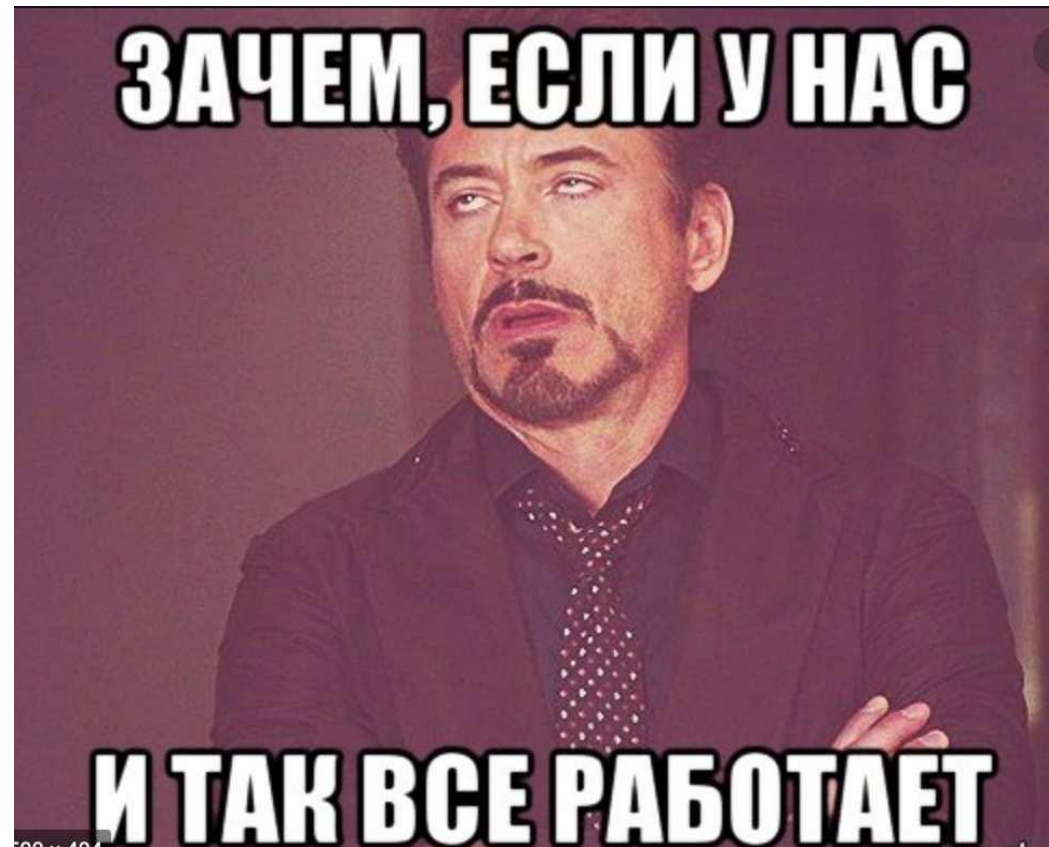


## Содержание занятия

1. Ключи и индексы в реляционной базе данных;
2. Обобщенное табличное выражение (СТЕ);
3. Рекурсивное обобщённое табличное выражение;
4. Оконные функции.
5. GROUP\_CONCAT()

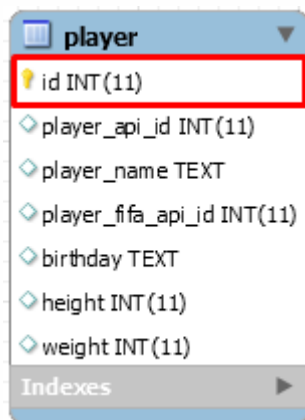
---

Ключи в реляционной базе данных. Какие бывают и зачем нужны?



# Первичный ключ (Primary key)

Это поле или комбинация полей, которые однозначно определяют запись (кортеж таблицы). Очень часто в качестве ключа берется порядковый номер. В таблице не может быть двух записей с одинаковым значением первичного ключа.



player
id INT(11)
player_api_id INT(11)
player_name TEXT
player_fifa_api_id INT(11)
birthday TEXT
height INT(11)
weight INT(11)
Indexes

	id	player_api_id	player_name	player_fifa_api_id	birthday	height	weight
▶	1	505942	Aaron Appindangoye	218353	1992-02-29 00:00:00	183	187
	2	155782	Aaron Cresswell	189615	1989-12-15 00:00:00	170	146
	3	162549	Aaron Doran	186170	1991-05-13 00:00:00	170	163
	4	30572	Aaron Galindo	140161	1982-05-08 00:00:00	183	198
	5	23780	Aaron Hughes	17725	1979-11-08 00:00:00	183	154
	6	27316	Aaron Hunt	158138	1986-09-04 00:00:00	183	161
	7	564793	Aaron Kuhl	221280	1996-01-30 00:00:00	173	146
	8	30895	Aaron Lennon	152747	1987-04-16 00:00:00	165	139
	9	528212	Aaron Lennox	206592	1993-02-19 00:00:00	190	181
	10	101042	Aaron Meijers	188621	1987-10-28 00:00:00	175	170
	11	23889	Aaron Mokoena	47189	1980-11-25 00:00:00	183	181
	12	231592	Aaron Mooy	194958	1990-09-15 00:00:00	175	150
	13	163222	Aaron Muirhead	213568	1990-08-30 00:00:00	188	168
	14	40719	Aaron Niguez	183853	1989-04-26 00:00:00	170	143
	15	75489	Aaron Ramsey	186561	1990-12-26 00:00:00	178	154
	16	597948	Aaron Splaine	226014	1996-10-13 00:00:00	173	163
	17	161644	Aaron Taylor-Sindair	213569	1991-04-08 00:00:00	183	176
	18	23499	Aaron Wilbraham	2335	1979-10-21 00:00:00	190	159
	19	120919	Aatif Chahechouhe	187939	1986-07-02 00:00:00	175	150
	20	46447	Abasse Ba	156626	1976-07-12 00:00:00	188	185
	21	167027	Abdelaziz Barrada	192274	1989-06-19 00:00:00	178	161

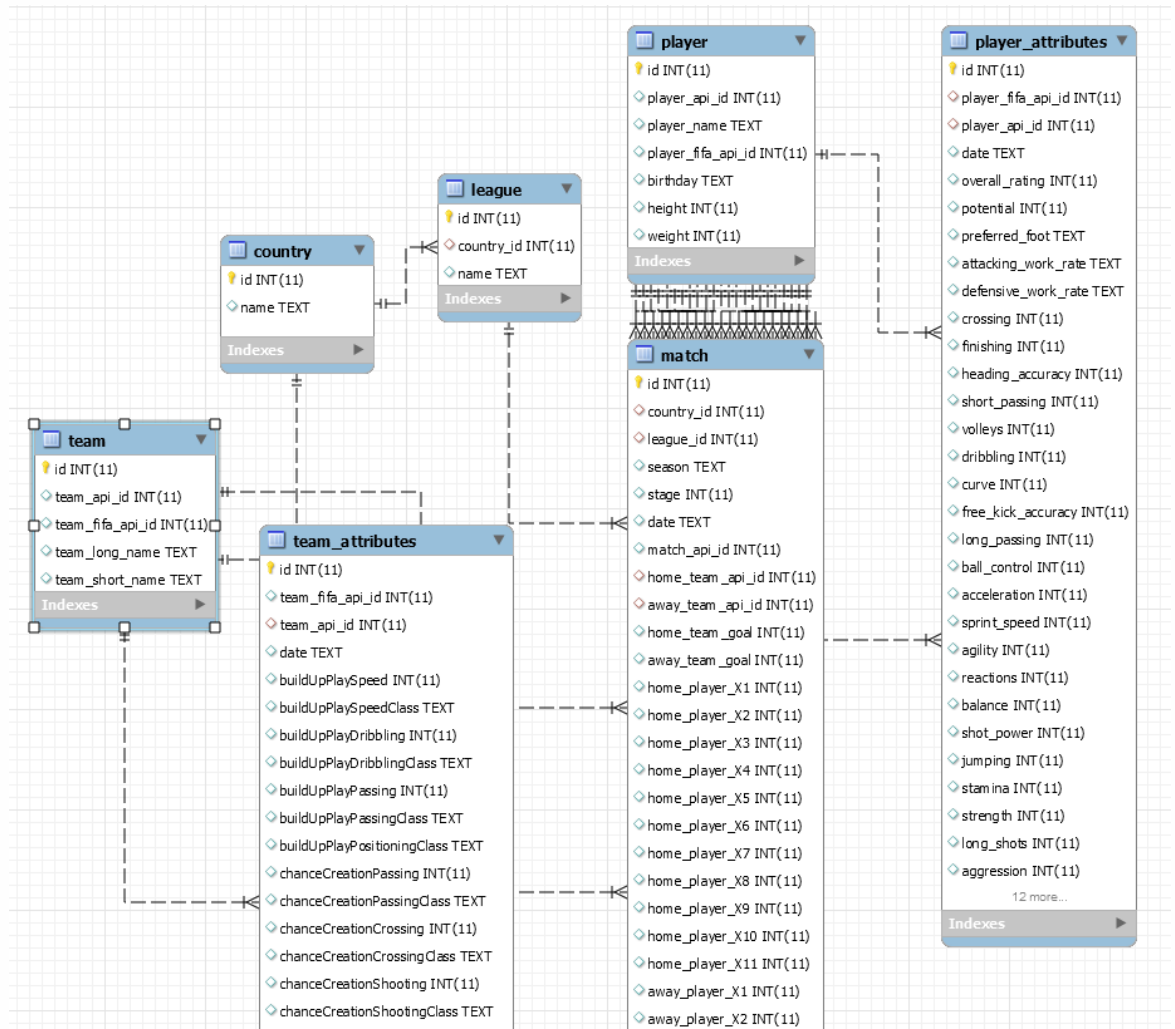


## Что может быть первичным ключом, а что нет?

- Фамилия клиента;
- Дата заказа;
- Модель телефона;
- Уникальный идентификатор клиента;
- Номер паспорта;
- Серия паспорта;
- Номер телефона.

# Ключи обеспечивают:

- однозначную идентификацию записей таблицы;
- ускорение выполнения запросов к БД;
- установление связи между отдельными таблицами БД



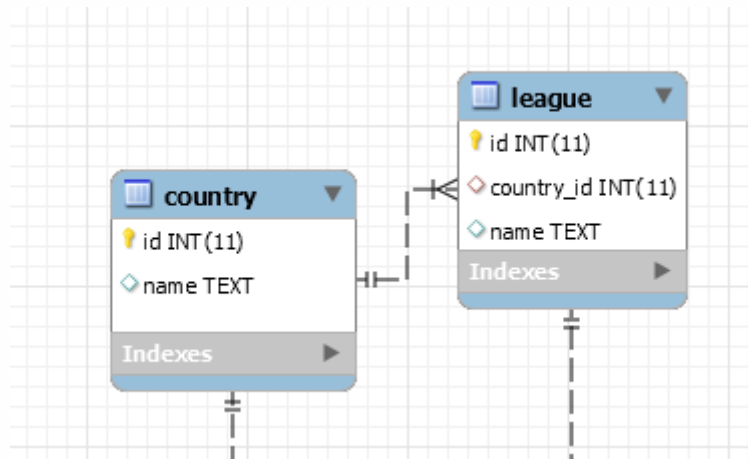


## Первичный ключ может быть простым и составным

- Простой первичный ключ - состоит из единственного поля таблицы, значения которого уникальны для каждой записи (уникальный идентификатор клиента, девайса и так далее);
- Составной первичный ключ - составлен из нескольких полей, совокупность значений которых гарантирует уникальность (номер + серия паспорта и так далее).

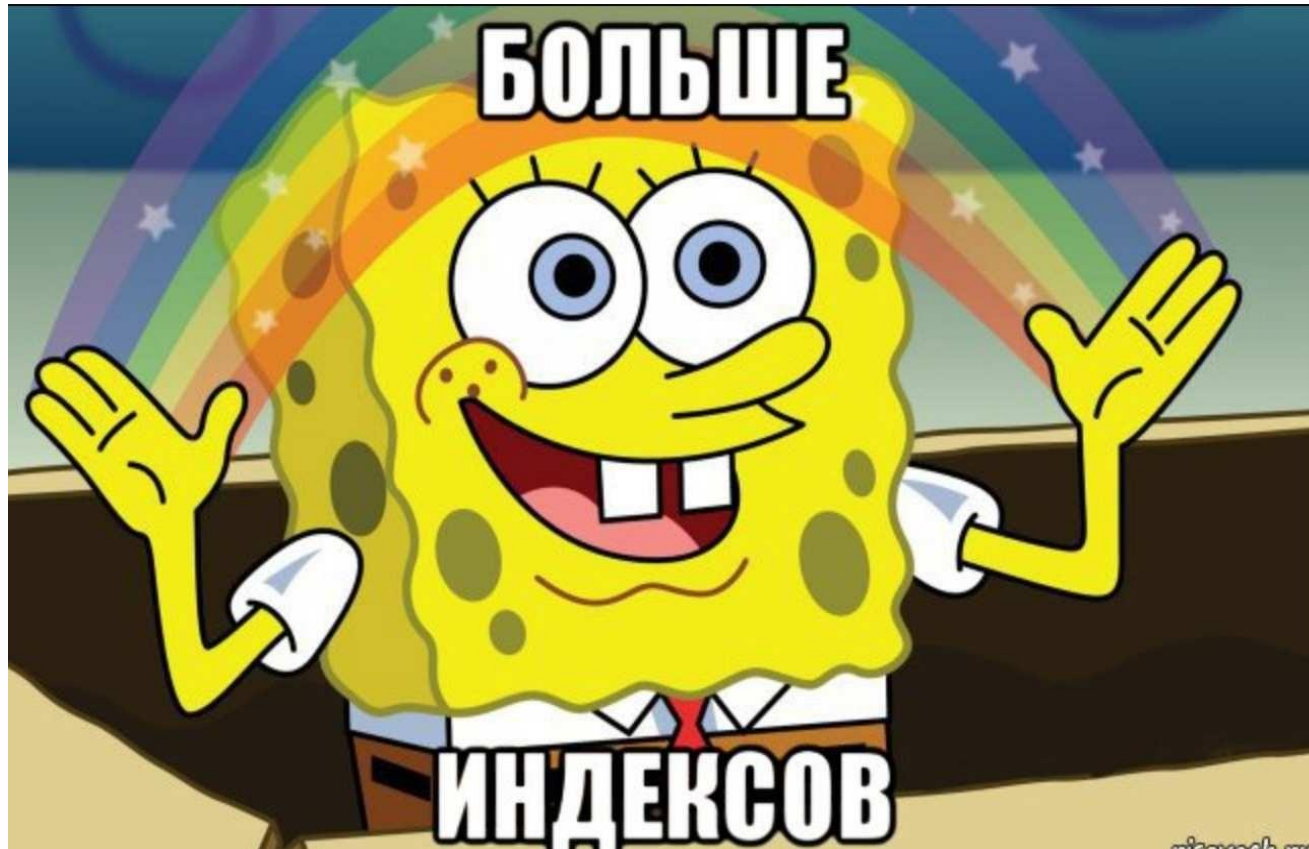
# Внешний ключ (Foreign key)

Множество колонок подчиненной таблицы, соответствующее ключу главной таблицы.



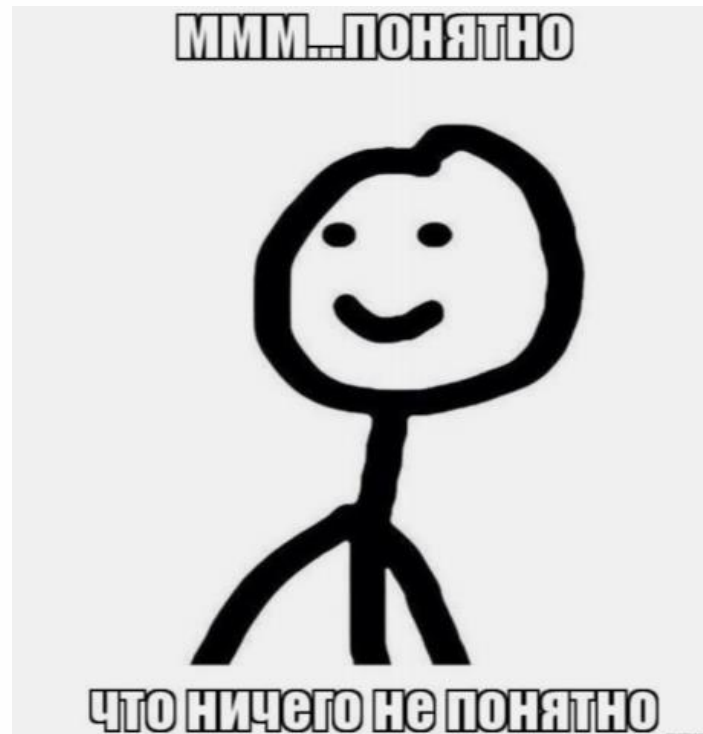


## Индексы



# Индексы

Это специальные структуры в базах данных, которые позволяют ускорить поиск и сортировку по определенному полю или набору полей в таблице.





# Индексы

**Индекс** – это системная структура данных, в которой размещается обязательно упорядоченный перечень значений какого-либо ключа со ссылками на те кортежи отношения, в которых эти значения встречаются.

Самая часто встречающаяся, но до сих пор работающая аналогия - классификатор книг, содержащий информацию о том на какой полке, какая книга; указатель в книгах, обозначающий на какой странице, какая глава.

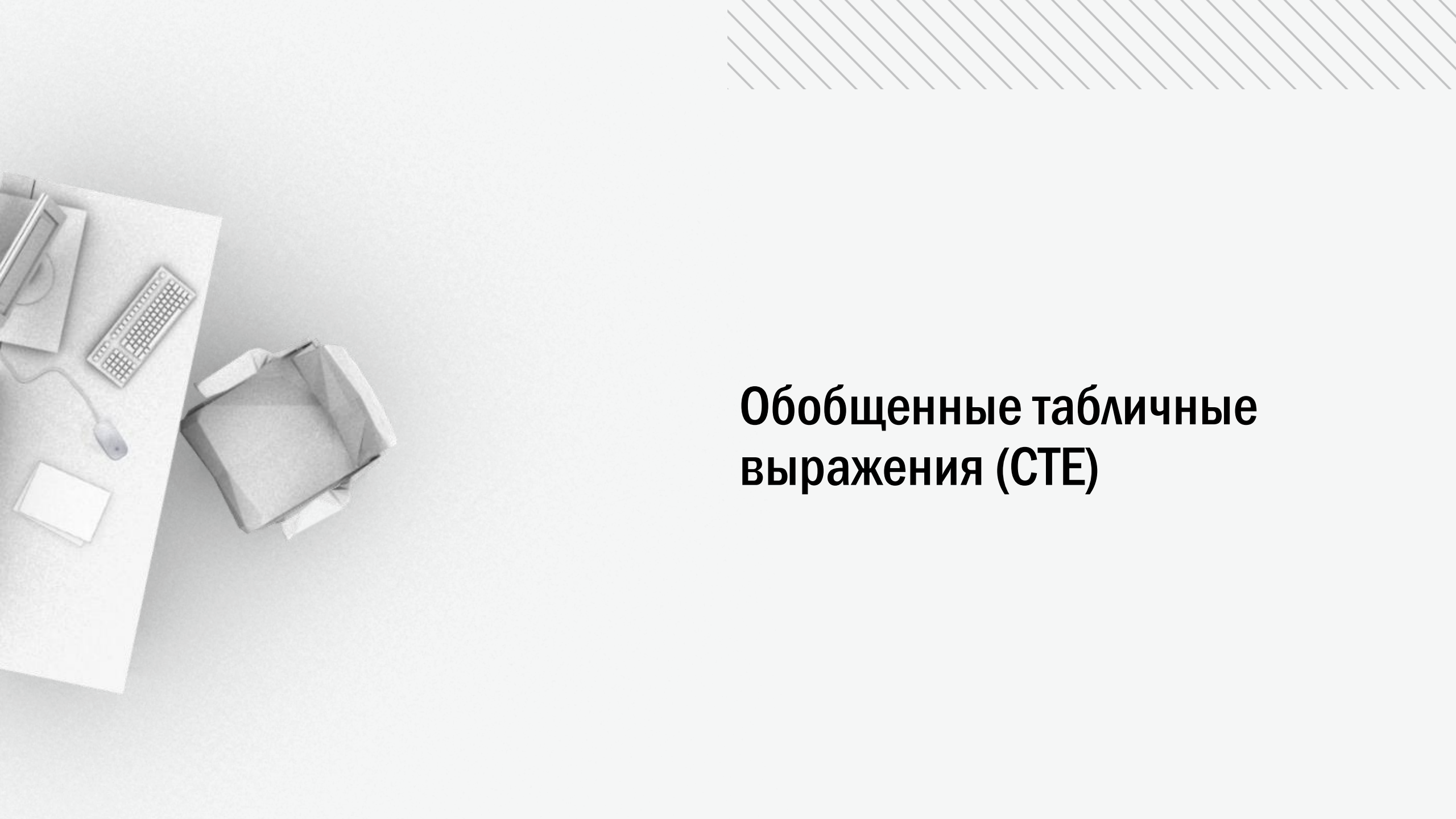
При наличии индекса у поля, с которым наш запрос работает, база сначала смотрит в каких кортежах эти значения есть, это позволяет не сканировать таблицу целиком.

---

## Совет

Делайте **EXPLAIN** своих запросов и вы начнете замечать, как ваша база выполняет запрос при наличии индексов и без.





## Обобщенные табличные выражения (СТЕ)

# Обобщенные табличные выражения

**Common Table Expression (CTE)** или **обобщенное табличное выражение (ОТВ)** – это временные результирующие наборы (т.е. результаты выполнения SQL запроса), которые не сохраняются в базе данных в виде объектов, но к ним можно обращаться.

**Базовый синтаксис:**

```
WITH common_table_expression_name  
AS  
    ( CTE_query_definition )
```

- **common\_table\_expression\_name** - это имя обобщенного табличного выражения;
- **CTE\_query\_definition** — запрос , к результирующему набору которого, мы и будем обращаться через обобщенное табличное выражение



## Когда удобно использовать СТЕ?

1. Для замены представлений (VIEW), например, в тех случаях, когда нет необходимости сохранять в базе SQL запрос представления;
2. Для повышения читаемости кода, когда запрос разделяется на логические блоки;
3. Позволяет многократно ссылаться на один результирующий набор данных.

---

## Пример 1. Оставим для работы только матчи с победами на домашнем поле

```
with home_win as(  
  SELECT *  
  from `match`  
  where home_team_goal > away_team_goal  
)  
select * FROM home_win;
```




## Пример 2. СТЕ может быть много

```
WITH seasons AS
(
    SELECT season, min(date) as start_date,
               max(date) as end_date
    FROM `match`
    GROUP BY season
),
team_speed AS
(
    SELECT team.team_api_id, team_long_name,
           date, buildUpPlaySpeed
    FROM team_attributes, team
    WHERE team_attributes.team_api_id=team.team_api_id
)
SELECT season, team_api_id, team_long_name,
       buildUpPlaySpeed
FROM seasons, team_speed
WHERE team_speed.date >= seasons.start_date
AND team_speed.date <= seasons.end_date
ORDER BY team_long_name, season;
```



## Задание



Вывести названия команд, которые не играли в сезоне 2008/2009 (Последний пример из прошлой лекции). Перепишите этот запрос с помощью СТЕ

# СТЕ бывают:

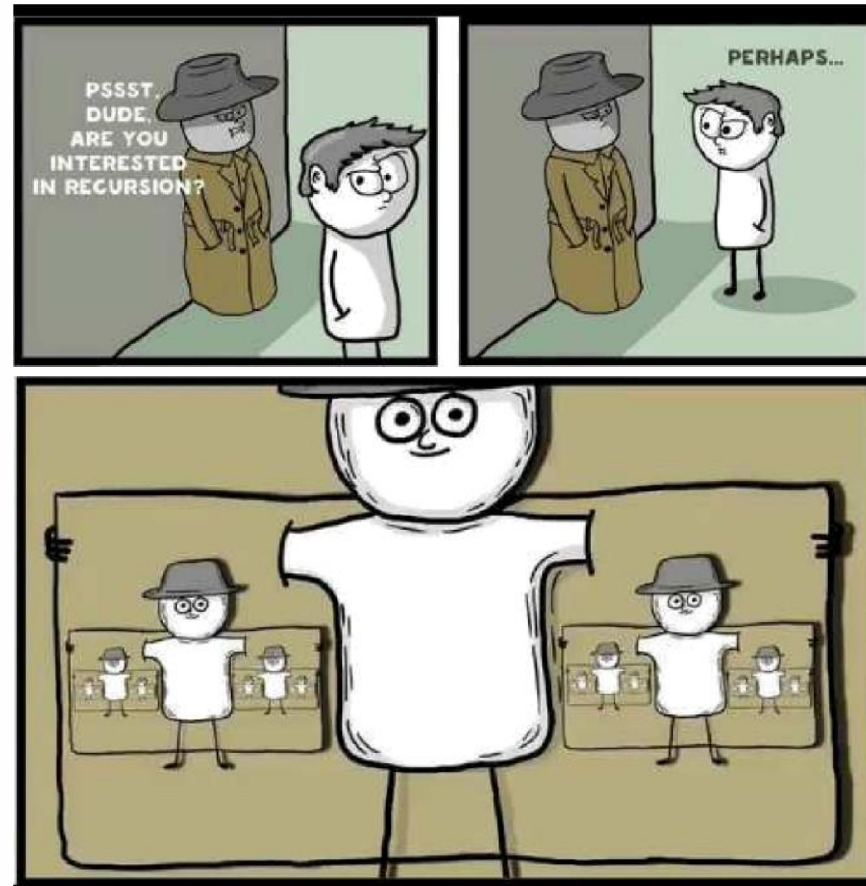
1. Простые;
2. Рекурсивные.



# Рекурсивные СТЕ

Все по классике.

Рекурсивный запрос будет повторяться определенное количество раз, до тех пор, пока не будет истинно заданное условие. (и бесконечность тут тоже не предел...)



---

## Пример теоретический

```
with recursive cte(n)
as (
  select 1
  union all
  select n + 1 from cte where n < 10
)
select * from cte;
```

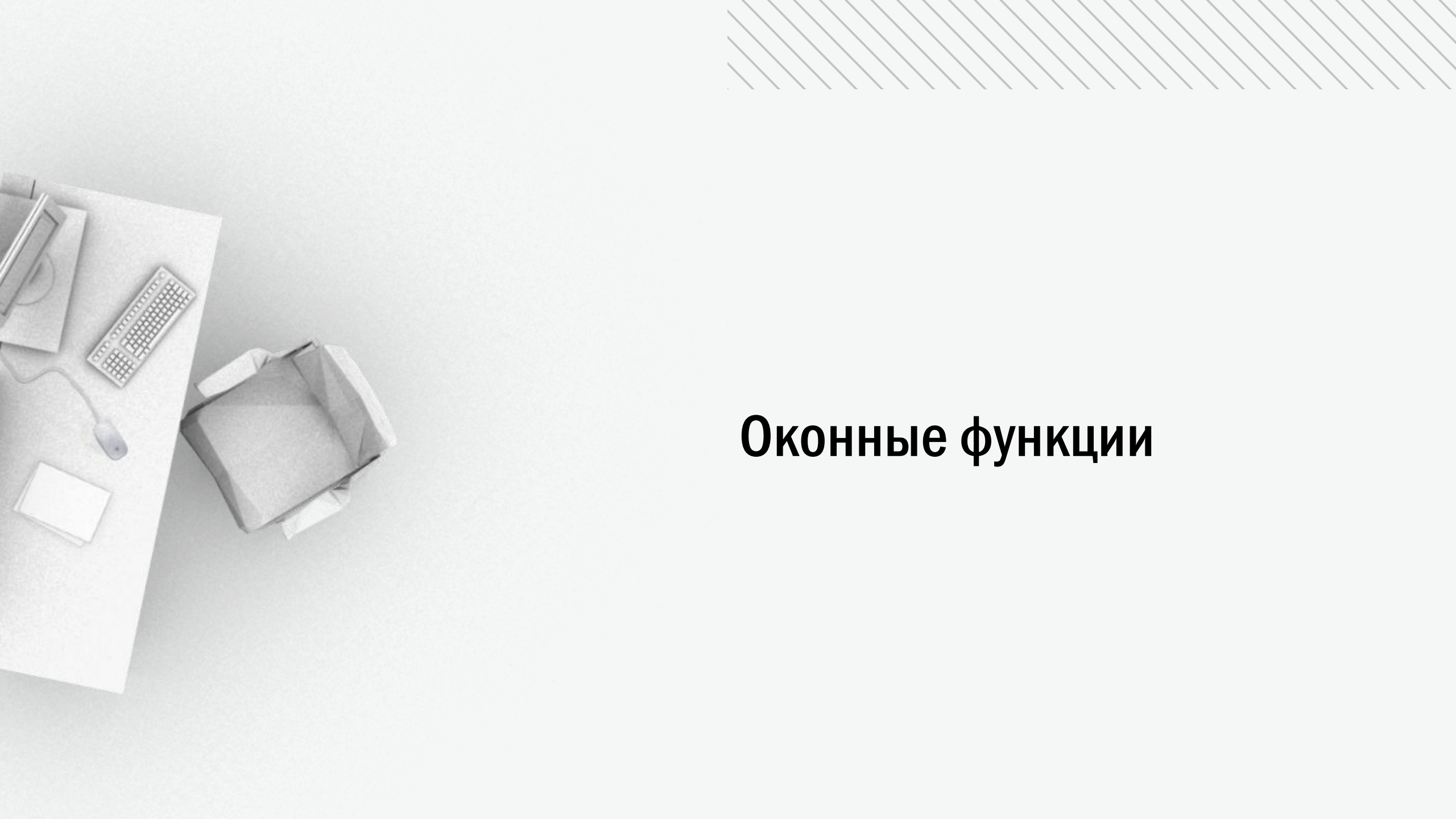
## Пример: Вывести последовательность ростов от минимального до максимального с шагом 10

```
with recursive cte as (  
    select  
        min(height) as height_threshold  
    from player  
    union all  
    select  
        height_threshold + 10  
    from cte  
    where height_threshold + 10 <= (select max(height) from player)  
)  
select height_threshold  
from cte;
```

---

# Задание

Вывести все степени 2, меньше 100



# Оконные функции

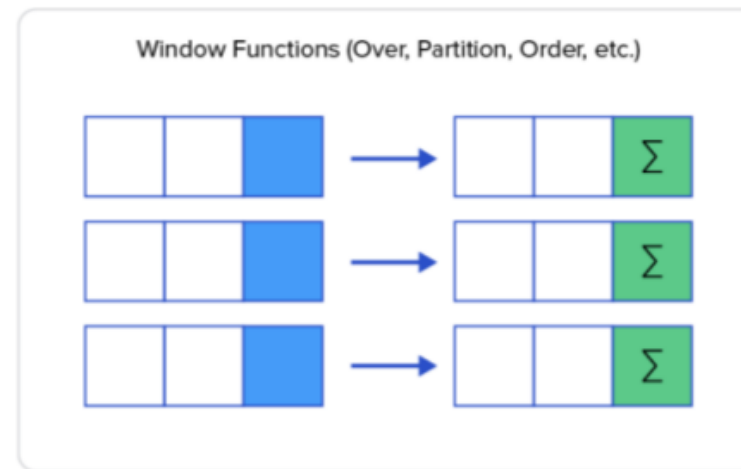
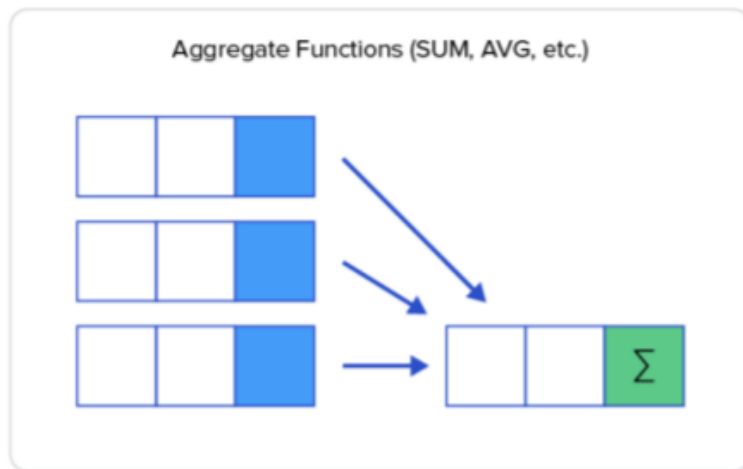


# Как работают оконные функции?

Окно – набор строк, в рамках которого происходит вычисление.

Оконная функция позволяет разбивать весь набор данных на такие окна.

Основное преимущество использования оконных функций над регулярными агрегатными функциями заключается в следующем: оконные функции не приводят к группированию строк в одну строку вывода, строки сохраняют свои отдельные идентификаторы, а агрегированное значение добавляется к каждой строке.



---

## Кратко

Оконные функции позволяют нам получить агрегированные по какому-либо признаку значения в изначальной таблице, а не в итоговом агрегате :)





## Оконные функции. OVER()

Предложение OVER- определяет оконную функцию и всегда должно быть включено в выражение. OVER() описывает набор строк, которые будет обрабатывать функция и порядок этой обработки.

Причем окно может быть просто задано пустыми скобками (), т.е. окном являются все строки результата запроса.

## По умолчанию контекст over () - все строки

```
select
  id,
  country_id,
  season,
  count(*) over ()
from `match`;
```

id	country_id	season	count(*) over ()
1	1	2008/2009	25979
2	1	2008/2009	25979
3	1	2008/2009	25979
4	1	2008/2009	25979
5	1	2008/2009	25979
6	1	2008/2009	25979
7	1	2008/2009	25979
8	1	2008/2009	25979
9	1	2008/2009	25979
10	1	2008/2009	25979
11	1	2008/2009	25979
12	1	2008/2009	25979
13	1	2008/2009	25979
14	1	2008/2009	25979
15	1	2008/2009	25979
16	1	2008/2009	25979

# Пример

```
select
    id,
    country_id,
    season,
    row_number() over ()
from `match`;
```

id	country_id	season	row_number() over ()
1	1	2008/2009	1
2	1	2008/2009	2
3	1	2008/2009	3
4	1	2008/2009	4
5	1	2008/2009	5
6	1	2008/2009	6
7	1	2008/2009	7
8	1	2008/2009	8
9	1	2008/2009	9
10	1	2008/2009	10
11	1	2008/2009	11
12	1	2008/2009	12
13	1	2008/2009	13
14	1	2008/2009	14
15	1	2008/2009	15
16	1	2008/2009	16


# Сортировка внутри окна

```
select
  player_api_id,
  crossing,
  finishing,
  row_number() over(order by crossing desc) as crossing_rating,
  row_number() over(order by finishing desc) as finishing_rating
from player_attributes;
```

id	player_api_id	crossing	finishing	crossing_rating	finishing_rating
102492	30981	84	97	1644	1
102493	30981	84	97	1645	2
134056	38567	68	97	42110	3
102494	30981	84	97	1646	4
102495	30981	84	97	1647	5
102496	30981	84	97	1648	6
102497	30981	84	97	1649	7
104176	30709	68	96	45909	8
170037	30626	76	95	13420	9
170038	30626	75	95	17278	10
102498	30981	85	95	984	11
33334	30893	83	95	2354	12
104175	30709	68	95	45908	13
97190	36784	52	95	117386	14
155164	34602	70	95	33849	15
33335	30893	83	95	2355	16
155165	34602	70	95	33850	17
33336	30893	83	95	2356	18
55890	30853	69	95	36717	19
33337	30893	83	95	2357	20
33331	30893	82	95	3596	21
33332	30893	82	95	3597	22
33333	30893	82	95	3598	23



## Задание



Вывести рейтинг команд по показателю `buildUpPlaySpeed` (чем больше тем лучше) за 2010 год.  
Какое значение показателя `defencePressure` у команды на 10-м месте?



## PARTITION BY

Позволяет определить с какими конкретно строками мы будем работать. Если **PARTITION BY** отсутствует, то мы работаем со всеми строками таблицы. Можно рассматривать как аналог группировки.



# Рейтинг команд по скорости перемещения и шансам создания паса в рамках класса скорости в сезоне 2009/2010

```
select
team_api_id,
    buildUpPlaySpeedClass,
    buildUpPlaySpeed,
    chanceCreationPassing,
    row_number() over(partition by buildUpPlaySpeedClass order by buildUpPlaySpeed
desc) as buildUpPlaySpeed_rating,
    row_number() over(partition by buildUpPlaySpeedClass order by chanceCreationPassing
desc) as chanceCreationPassing_rating
from team_attributes
where date > '2009-07-01'
and date < '2010-05-31'
order by buildUpPlaySpeedClass;
```

id	team_api_id	buildUpPlaySpeedClass	buildUpPlaySpeed	chanceCreationPassing	buildUpPlaySpeed_rating	chanceCreationPassing_rating
766	8689	Balanced	35	80	1167	1
171	8658	Balanced	56	77	387	2
172	8658	Balanced	56	77	388	3
1107	8686	Balanced	53	77	540	4
1021	8696	Balanced	53	73	538	5
557	8429	Balanced	48	73	805	6
202	8559	Balanced	57	72	360	7
203	8559	Balanced	57	72	361	8
254	8191	Balanced	54	72	504	9
527	7878	Balanced	52	72	559	10
253	8191	Balanced	48	72	797	11
526	7878	Balanced	48	72	804	12
792	9864	Balanced	45	72	961	13
791	9864	Balanced	41	72	1023	14
391	8398	Balanced	64	71	139	15
731	8581	Balanced	62	71	194	16

## Пример. Определим класс реакции игроков и посчитаем рейтинг внутри класса

with cte as

```
(  
  select player_api_id,  
    avg(reactions) as avg_reactions,  
    avg(balance) as avg_balance,  
    avg(ball_control) as avg_ball_control  
  from player_attributes  
  group by player_api_id
```

```
),
```

```
cte2 as (  
  select *,  
    case when cte.avg_reactions > 50 then 'heigh_reactive' else 'low_active' end as reaction_class  
  from cte
```


```
)
```

```
select *,  
  row_number() over (partition by cte2.reaction_class order by cte2.avg_reactions desc) as reaction_rating  
from cte2;
```

player_api_id	avg_reactions	avg_balance	avg_ball_control	reaction_class	reaction_rating
30981	92.5385	92.2308	95.7692	heigh_reactive	1
39854	91.7273	87.4545	93.3636	heigh_reactive	2
30894	91.1667	86.8000	85.8000	heigh_reactive	3
34602	89.7000	77.0000	83.4000	heigh_reactive	4
30955	89.3200	86.0400	92.9600	heigh_reactive	5
30729	89.2222	60.3333	73.7778	heigh_reactive	6
38817	88.9474	88.7895	87.7368	heigh_reactive	7
30893	88.1600	74.3200	93.9600	heigh_reactive	8
30872	86.9615	75.2308	84.9231	heigh_reactive	9
30731	86.8222	77.1333	92.0222	heigh_reactive	10
30834	86.6400	87.1600	89.5200	heigh_reactive	11
30829	86.4167	78.7778	86.5833	heigh_reactive	12
30909	86.2500	77.3333	86.0417	heigh_reactive	13
40636	86.0750	78.2500	86.0000	heigh_reactive	14
30924	86.0417	89.2917	91.4583	heigh_reactive	15



## Задание



Посчитать средний потенциал (`potential`) для каждого игрока среди игроков, для которых указано значение категории предпочитаемой ноги (`preferred_foot`). Вывести рейтинг игроков по среднему потенциалу в рамках категории предпочитаемой ноги (`preferred_foot`). Какой `player_api_id` у игроков с рейтингом 3 в каждом классе?

# Функции LAG() и LEAD()

Эти функции возвращают значение выражения, вычисленного для предыдущей строки (LAG) или следующей строки (LEAD) результирующего набора соответственно.

Пример LAG()

Посчитаем приращение реакции игроков

```
select
    player_api_id,
    date,
    reactions,
    reactions - lag(reactions) over (partition by player_api_id order by date) as reaction_diff
from player_attributes;
```

---

## Задание

Посчитать приращение `buildUpPlaySpeed` у команд. Какие приращения были у команды с `team_api_id = 8455`?



**GROUP\_CONCAT()**

# Сразу к примеру

Для каждой команды выведем все классы buildUpPlaySpeedClass, которые ей когда-либо присваивали

```
with cte as (  
    select distinct team.team_api_id , team.team_long_name, buildUpPlaySpeedClass  
    from team  
    join team_attributes on team.team_api_id = team_attributes.team_api_id  
)  
select team_api_id , team_long_name,  
       group_concat(buildUpPlaySpeedClass)  
from cte  
group by team_api_id, team_long_name;
```

team_api_id	team_long_name	group_concat(buildUpPlaySpeedClass)
1601	Ruch Chorzów	Slow,Balanced
1773	Oud-Heverlee Leuven	Balanced
1957	Jagiellonia Białystok	Slow,Balanced
2033	S.C. Olhanense	Balanced
2182	Lech Poznań	Balanced,Fast
2183	P. Warszawa	Slow,Balanced
2186	Cracovia	Slow,Balanced
4087	Évian Thonon Gaillard FC	Balanced
4170	US Boulogne Cote D'Opale	Balanced
6269	Novara	Balanced
6351	KAS Eupen	Balanced
6391	GFC Ajaccio	Balanced
6403	FC Paços de Ferreira	Slow,Balanced
6413	PEC Zwolle	Balanced
6421	Leixões SC	Slow
6433	Go Ahead Eagles	Balanced

## Сколько команд имеют такое сочетание классов buildUpPlaySpeedClass за историю своего существования (в рамках нашей базы)

```
with cte as (  
    select distinct team_api_id, buildUpPlaySpeedClass  
    from team_attributes  
)  
cte2 as (  
    select team_api_id ,  
           group_concat(buildUpPlaySpeedClass)  
    from cte  
    group by team_api_id  
)  
select buildUpPlaySpeedClasses, count(*)  
from cte2  
group by buildUpPlaySpeedClasses  
order by count(*) desc;
```

buildUpPlaySpeedClasses	count(*)
Balanced	123
Fast,Balanced	49
Slow,Balanced	48
Balanced,Fast	36
Balanced,Slow	10
Slow,Fast,Balanced	7
Slow,Balanced,Fast	5
Balanced,Fast,Slow	3
Slow	2
Fast	2
Fast,Balanced,Slow	2
Slow,Fast	1



# Задание

Как исправить предыдущий запрос, чтобы не было перестановок?

	buildUpPlaySpeedClasses	count(*)
►	Balanced	123
	Balanced,Fast	85
	Balanced,Slow	58
	Balanced,Fast,Slow	17
	Slow	2
	Fast	2
	Fast,Slow	1

ВМЕСТО

buildUpPlaySpeedClasses	count(*)
Balanced	123
Fast,Balanced	49
Slow,Balanced	48
Balanced,Fast	36
Balanced,Slow	10
Slow,Fast,Balanced	7
Slow,Balanced,Fast	5
Balanced,Fast,Slow	3
Slow	2
Fast	2
Fast,Balanced,Slow	2
Slow,Fast	1



## Задание

Для каждого игрока вывести в одном столбце все значения категории `attacking_work_rate`, которые ему когда-либо присваивали. Какая последовательность значений соответствует игроку с идентификатором `player_api_id=5610`?

**СПАСИБО  
ЗА ВНИМАНИЕ**

