Использование оконных функций

Наборы или курсоры?

- Часто решения для запроса данных на основе SQL делят на два вида:
  - основанные на наборах записей
  - основанные на курсорах с итеративным проходом по записям
- Наиболее эффективно – обрабатывать наборы записей:
  - таблица представляет собой набор неупорядоченных данных
  - сохраняются реляционные принципы обработки
- Язык SQL никак не связан с конкретной физической реализацией в ядре СУБД
  - Задача физического уровня - определить, как выполнить логический запрос, и максимально быстро вернуть правильный результат

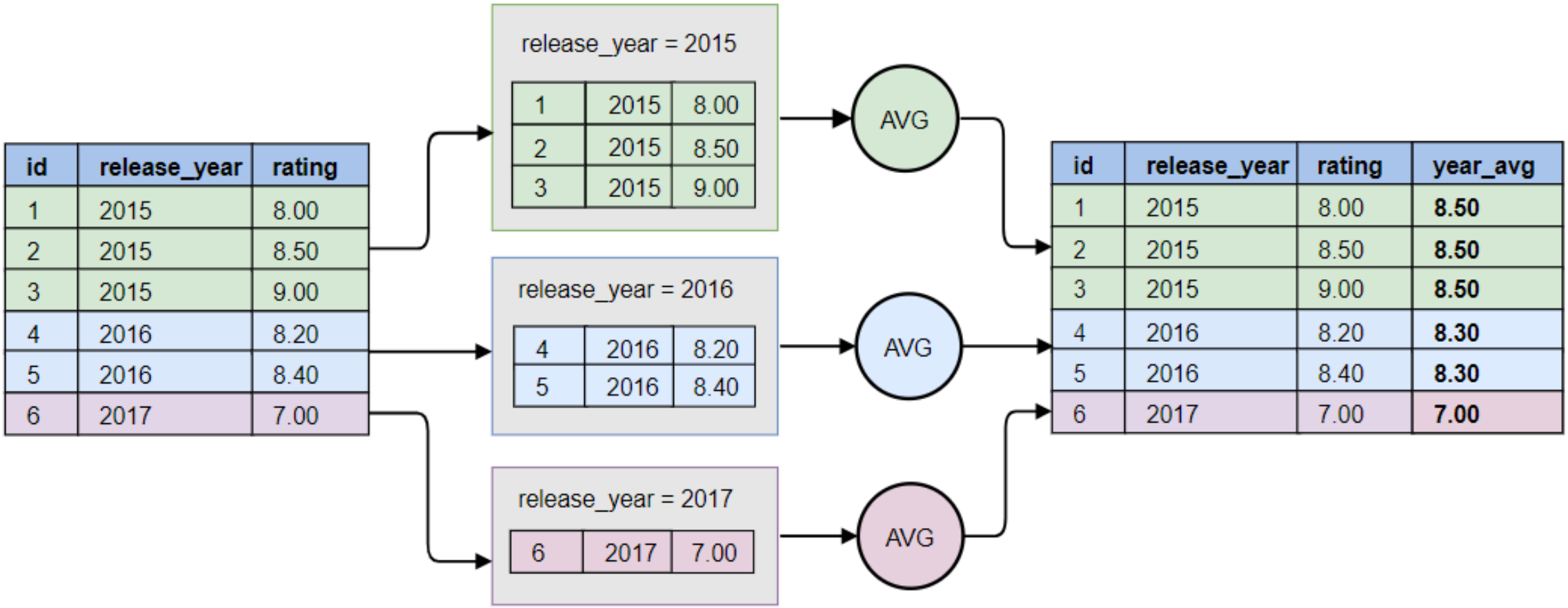
Зачем нужны оконные функции

- Оконные функции — это мощнейший инструмент для анализа данных, который с легкостью помогает решать множество задач
- Отличие от агрегатных функций:
  - групповые запросы формируют информацию в виде агрегатов, но при этом теряются детали (уменьшают количество строк)
  - оконные функции позволяют использование как детальных данных, так и агрегатов (не уменьшают количество строк)
  - результаты работы оконных функций просто добавляются к результирующей выборке в виде нового столбца
- Этот функционал очень полезен для построения аналитических отчетов, расчета скользящего среднего и нарастающих итогов, а также для расчетов различных моделей атрибуции

Задачи, решаемые с помощью оконных функций

- Ранжирование (всевозможные рейтинги)
- Сравнение со смещением (соседние элементы и границы)
- Агрегация (сумма и среднее)
- Скользящие агрегаты (сумма и среднее в динамике)

Проблема


id
release_year
rating
1
2015
8.00
2
2015
8.50
3
2015
9.00
4
2016
8.20
5
2016
8.40
6
2017
7.00
release_year = 2015
1
2015
8.00
2
2015
8.50
3
2015
9.00
release_year = 2016
4
2016
8.20
5
2016
8.40
release_year = 2017
6
2017
7.00
AVG
AVG
AVG
id
release_year
rating
year_avg
1
2015
8.00
8.50
2
2015
8.50
8.50
3
2015
9.00
8.50
4
2016
8.20
8.30
5
2016
8.40
8.30
6
2017
7.00
7.00

Синтаксис

```
Название_функции([arg1, arg2,..])
                 [FILTER (WHERE условие_фильтра )]
                 OVER  имя_окна |( определение_окна )
```

определение_окна

```
(
    [PARTITION BY выражение_для_группировки]
    [ORDER BY выражение_для_сортировки [ASC|DESC]
    [NULLS {FIRST | LAST }]
    [определение_фрэйма]
)
```

OVER()

- При использовании инструкции OVER() без предложений окном будет весь набор данных

	empid	ord_year	total
1	1	2 006	$38,789.00
2	2	2 006	$22,834.70
3	3	2 006	$19,231.80
4	4	2 006	$53,114.80
5	5	2 006	$21,965.20
6	1	2 007	$97,533.58
7	2	2 007	$74,958.60
8	3	2 007	$111,788.61
9	4	2 007	$139,477.70
10	5	2 007	$32,595.05
11	1	2 008	$65,821.13
12	2	2 008	$79,955.96
13	3	2 008	$82,030.89
14	4	2 008	$57,594.95
15	5	2 008	$21,007.50

GrandTotal: **$918,699.47**

```
SELECT empid, ord_year,
       SUM(total) OVER()
                  AS "GrandTotal"
FROM public."WindowFunctions";
```

	empid	ord_year	GrandTotal
1	1	2 006	$918,699.47
2	2	2 006	$918,699.47
3	3	2 006	$918,699.47
4	4	2 006	$918,699.47
5	5	2 006	$918,699.47
6	1	2 007	$918,699.47
7	2	2 007	$918,699.47
8	3	2 007	$918,699.47
9	4	2 007	$918,699.47
10	5	2 007	$918,699.47
11	1	2 008	$918,699.47
12	2	2 008	$918,699.47
13	3	2 008	$918,699.47
14	4	2 008	$918,699.47
15	5	2 008	$918,699.47

Предложение PARTITION BY

- Предложение PARTITION BY делит строки на несколько групп или разделов, к которым применяется оконная функция
  - Определяет столбец, по которому будет производиться разделение набора строк на окна
- Предложение PARTITION BY является необязательным
- Если вы пропустите предложение PARTITION BY, оконная функция будет рассматривать весь результирующий набор как один раздел

PARTITION BY

	empid	ord_year	total
1	1	2 006	$38,789.00
2	2	2 006	$22,834.70
3	3	2 006	$19,231.80
4	4	2 006	$53,114.80
5	5	2 006	$21,965.20
6	1	2 007	$97,533.58
7	2	2 007	$74,958.60
8	3	2 007	$111,788.61
9	4	2 007	$139,477.70
10	5	2 007	$32,595.05
11	1	2 008	$65,821.13
12	2	2 008	$79,955.96
13	3	2 008	$82,030.89
14	4	2 008	$57,594.95
15	5	2 008	$21,007.50

```
SELECT empid, ord_year,
       SUM(total)
       OVER(PARTITION BY ord_year)
       AS "TotalByYear"
FROM public.windowfunctions
WHERE empid <=5;
```

	empid	ord_year	TotalByYear
1	1	2 006	$155,935.50
2	2	2 006	$155,935.50
3	3	2 006	$155,935.50
4	4	2 006	$155,935.50
5	5	2 006	$155,935.50
6	1	2 007	$456,353.54
7	2	2 007	$456,353.54
8	3	2 007	$456,353.54
9	4	2 007	$456,353.54
10	5	2 007	$456,353.54
11	1	2 008	$306,410.43
12	2	2 008	$306,410.43
13	3	2 008	$306,410.43
14	4	2 008	$306,410.43
15	5	2 008	$306,410.43

Предложение ORDER BY

- Предложение ORDER BY указывает порядок строк в каждом разделе, к которому применяется оконная функция
  - Сортирует строки внутри окна для вычисления нарастающего итога
- Предложение ORDER BY использует параметр **NULLS FIRST** или **NULLS LAST**, чтобы указать, должны ли значения, допускающие значение NULL, быть первыми или последними в результирующем наборе
  - По умолчанию используется опция **NULLS LAST**
- Предложение ORDER BY является обязательным для некоторых функций

ORDER BY

	empid	ord_year	total
1	1	2 006	$38,789.00
2	2	2 006	$22,834.70
3	3	2 006	$19,231.80
4	4	2 006	$53,114.80
5	5	2 006	$21,965.20
6	1	2 007	$97,533.58
7	2	2 007	$74,958.60
8	3	2 007	$111,788.61
9	4	2 007	$139,477.70
10	5	2 007	$32,595.05
11	1	2 008	$65,821.13
12	2	2 008	$79,955.96
13	3	2 008	$82,030.89
14	4	2 008	$57,594.95
15	5	2 008	$21,007.50

```
SELECT empid, ord_year,
       SUM(total)
       OVER(ORDER BY ord_year)
          AS RunningTotalByYear
FROM public."WindowFunctions";
```

	ord_year	Total
1	2 006	$155,935.50
2	2 007	$456,353.54
3	2 008	$306,410.43

	empid	ord_year	runningtotalbyyear
1	1	2 006	$155,935.50
2	2	2 006	$155,935.50
3	3	2 006	$155,935.50
4	4	2 006	$155,935.50
5	5	2 006	$155,935.50
6	1	2 007	$612,289.04
7	2	2 007	$612,289.04
8	3	2 007	$612,289.04
9	4	2 007	$612,289.04
10	5	2 007	$612,289.04
11	1	2 008	$918,699.47
12	2	2 008	$918,699.47
13	3	2 008	$918,699.47
14	4	2 008	$918,699.47
15	5	2 008	$918,699.47

PARTITION BY с ORDER BY

- Разделение набора строк на окна и упорядочивание строк внутри каждого окна

	empid	ord_year	total
1	1	2 006	$38,789.00
2	2	2 006	$22,834.70
3	3	2 006	$19,231.80
4	4	2 006	$53,114.80
5	5	2 006	$21,965.20
6	1	2 007	$97,533.58
7	2	2 007	$74,958.60
8	3	2 007	$111,788.61
9	4	2 007	$139,477.70
10	5	2 007	$32,595.05
11	1	2 008	$65,821.13
12	2	2 008	$79,955.96
13	3	2 008	$82,030.89
14	4	2 008	$57,594.95
15	5	2 008	$21,007.50

```
SELECT empid, ord_year,
       SUM(total)
       OVER(PARTITION BY empid
            ORDER by ord_year)
              AS "TotalByYear"
FROM public."WindowFunctions";
```

	empid	TotalByEmpid
1	1	$192,107.60
2	2	$166,537.76
3	3	$202,812.82
4	4	$232,890.87
5	5	$68,792.29

	empid	ord_year	TotalByYear
1	1	2 006	$35,764.51
2	1	2 007	$128,912.61
3	1	2 008	$192,107.60
4	2	2 006	$21,757.06
5	2	2 007	$92,201.20
6	2	2 008	$166,537.76
7	3	2 006	$18,223.96
8	3	2 007	$126,250.09
9	3	2 008	$202,812.82
10	4	2 006	$49,945.12
11	4	2 007	$178,754.93
12	4	2 008	$232,890.87
13	5	2 006	$18,383.92
14	5	2 007	$49,100.40
15	5	2 008	$68,792.29

FILTER

- Предложение FILTER определяет какие строки будут подаваться на вход оконной функции
  - подаются только те входные строки, для которых условие_фильтра вычисляется как истинное; другие строки отбрасываются
  - **предложение FILTER допускается только для агрегирующих оконных функций**

```
SELECT empid, ord_year,
SUM(total) filter (where ord_year > 2006)
OVER(PARTITION BY empid ORDER by
ord_year)
AS "TotalByYear"
FROM public."WindowFunctions";
```

	empid	ord_year	TotalByYear
1	1	2 006	[NULL]
2	1	2 007	$97,533.58
3	1	2 008	$163,354.71
4	2	2 006	[NULL]
5	2	2 007	$74,958.60
6	2	2 008	$154,914.56
7	3	2 006	[NULL]
8	3	2 007	$111,788.61
9	3	2 008	$193,819.50

Именованное окно

- Предложение **WINDOW** определяет подмножество строк в текущем разделе, к которому применяется оконная функция
- Это подмножество строк называется **именованное окно**

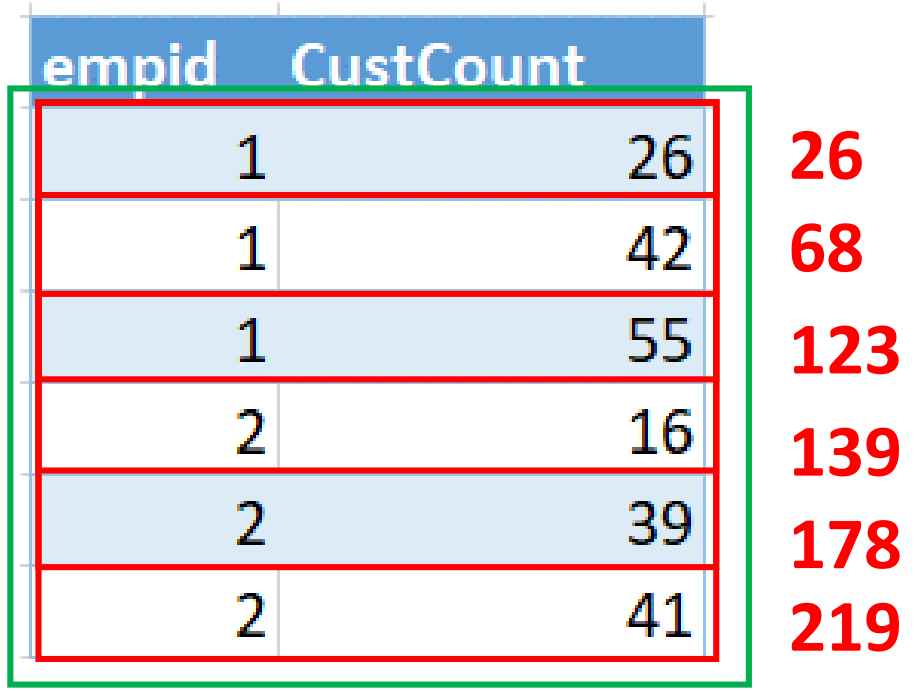
```
SELECT empid, ord_year,
       SUM(total)  OVER(PARTITION BY ord_year) AS "TotalByYear",
       avg(total::numeric) OVER(PARTITION BY ord_year)::money as "Average"
FROM public."WindowFunctions";
```

```
SELECT empid, ord_year,
       SUM(total)  over w as "TotalByYear",
       avg(total::numeric) over w::money as "Average"
FROM public."WindowFunctions"
WINDOW w AS (PARTITION BY ord_year);
```

Определение фрэйма (рамки)

```
{ROWS|RANGE} «начало» [исключение_рамки]
{ROWS|RANGE} BETWEEN «начало» AND «конец» [исключение_рамки]
```

- Фрэйм определяет *рамку окна*, ограничивающую подмножество строк текущего раздела
  - оконная функция работает с рамкой, а не со всем разделом
- Определение фрэйма (режимы):
  - **ROWS** позволяет ограничить строки в окне, указывая фиксированное количество строк, предшествующих или следующих за текущей
  - **RANGE** работает с набором строк имеющих одинаковый ранг в инструкции ORDER BY
- Необходимо наличие **ORDER BY**



Задание границ фрэйма

- Для ограничения строк ROWS или RANGE можно использовать следующие ключевые слова:
  - ***UNBOUNDED PRECEDING*** — указывает, что фрэйм начинается с первой строки группы;
  - ***UNBOUNDED FOLLOWING*** – с помощью данной инструкции можно указать, что фрэйм заканчивается на последней строке группы;
  - ***CURRENT ROW*** – инструкция указывает, что фрэйм начинается или заканчивается на текущей строке;
  - «Число» ***PRECEDING*** – определяет число строк перед текущей строкой **(не допускается в предложении RANGE)**;
  - «Число» ***FOLLOWING*** — определяет число строк после текущей строки **(не допускается в предложении RANGE)**.

- Если **конец_рамки** не задан, подразумевается **CURRENT ROW**

ROWS

	ord_year	CustCount
1	2 006	27
2	2 006	40
3	2 006	43
4	2 006	70
5	2 006	97
6	2 007	53
7	2 007	102
8	2 007	156
9	2 007	184
10	2 007	218
11	2 008	37
12	2 008	94
13	2 008	99
14	2 008	105
15	2 008	119

ROWS позволяет ограничить строки в окне путем задания границ фрэйма

```
SELECT ord_year, "CustCount",
       SUM("CustCount")
       OVER(PARTITION BY ord_year
            ORDER BY "CustCount"
            ROWS BETWEEN CURRENT ROW
            AND 1 FOLLOWING) AS "SUM"
FROM public."WindowFunctions"
order by 1,2;
```

	ord_year	CustCount	SUM
1	2 006	27	67
2	2 006	40	83
3	2 006	43	113
4	2 006	70	167
5	2 006	97	97
6	2 007	53	155
7	2 007	102	258
8	2 007	156	340
9	2 007	184	402
10	2 007	218	218
11	2 008	37	131
12	2 008	94	193
13	2 008	99	204
14	2 008	105	224
15	2 008	119	119

RANGE

	empid	CustCount
1	1	70
2	1	119
3	1	156
4	2	40
5	2	99
6	2	102
7	3	43
8	3	94
9	3	184
10	4	97
11	4	105
12	4	218
13	5	27
14	5	37
15	5	53

RANGE работает с набором строк, имеющих одинаковый ранг в инструкции **ORDER BY**

```
SELECT empid, "CustCount",
       SUM("CustCount")
       OVER(ORDER BY empid
                RANGE BETWEEN
                        UNBOUNDED PRECEDING
                        AND CURRENT ROW)
       AS "SUM"
FROM public."WindowFunctions"
order by 1,2;
```

	empid	CustCount	SUM
1	1	70	345
2	1	119	345
3	1	156	345
4	2	40	586
5	2	99	586
6	2	102	586
7	3	43	907
8	3	94	907
9	3	184	907
10	4	97	1 327
11	4	105	1 327
12	4	218	1 327
13	5	27	1 444
14	5	37	1 444
15	5	53	1 444

Исключение рамки

- Исключение рамки позволяет исключить из рамки строки, которые окружают текущую строку, даже если они должны быть включены согласно указаниям, определяющим начало и конец рамки
- Исключение рамки может быть следующим:
  - ***EXCLUDE CURRENT ROW*** — исключает из рамки текущую строку
  - ***EXCLUDE GROUP*** — исключает из рамки текущую строку и родственные ей согласно порядку сортировки
  - ***EXCLUDE TIES*** — исключает из рамки все родственные строки для текущей, но не собственно текущую строку
  - ***EXCLUDE NO OTHERS*** — явно выражает поведение по умолчанию — не исключает ни текущую строку, ни родственные ей

Обработка оконных функций

- Если запрос содержит оконные функции, эти функции вычисляются после каждой группировки, агрегатных выражений и фильтрации HAVING:
  - оконные функции видят не исходные строки, полученные из FROM/WHERE, а сгруппированные
- При использовании нескольких оконных функций с одинаковым определением окна (PARTITION BY и ORDER BY) все функции обрабатывают данные за один проход
  - они увидят один порядок сортировки, даже если ORDER BY не определяет порядок однозначно
- Оконные функции требуют предварительно отсортированных данных, так что результат запроса будет отсортирован согласно тому или иному предложению PARTITION BY/ORDER BY оконных функций
  - Чтобы результаты сортировались определённым образом, необходимо явно добавить предложение ORDER BY на верхнем уровне запроса

Порядок выполнения запроса

- Взять нужные таблицы (**FROM**) и соединить их при необходимости (**JOIN**)
- Отфильтровать строки (**WHERE**)
- Сгруппировать строки (**GROUP BY**)
- Отфильтровать результат группировки (**HAVING**)
- Сформировать конкретные столбцы результата (**SELECT**)
- Рассчитать значения оконных функций (**FUNCTION() OVER WINDOW**)
- Отсортировать то, что получилось (**ORDER BY**)

Оконные функции

Типы оконных функций

- Агрегатные функции
- Ранжирующие функции
- Функции смещения
- Аналитические функции

Агрегатные функции

- **SUM** — возвращает сумму значений в столбце
- **COUNT** — вычисляет количество значений в столбце
- **AVG** — определяет среднее значение в столбце
- **MAX** — определяет максимальное значение в столбце
- **MIN** — определяет минимальное значение в столбце

- Значения **NULL** не учитываются
- *Использование ключевого слова **DISTINCT** не допускается с предложением **OVER***


```
SELECT empid, ord_year
    , SUM(Total) OVER(PARTITION BY empid) AS GrandTotal
    , AVG(Total::numeric) OVER(PARTITION BY empid)::money AS AvgTotal
    , MIN(Total) OVER(PARTITION BY empid) AS MinTotal
    , MAX(Total) OVER(PARTITION BY empid) AS MaxTotal
    , COUNT(Total) OVER(PARTITION BY empid) AS CountTotal
FROM public."WindowFunctions";
```

	empid	ord_year	grandtotal	avgtotal	mintotal	maxtotal	counttotal
1	1	2 008	$202,143.71	$67,381.24	$38,789.00	$97,533.58	3
2	1	2 007	$202,143.71	$67,381.24	$38,789.00	$97,533.58	3
3	1	2 006	$202,143.71	$67,381.24	$38,789.00	$97,533.58	3
4	2	2 007	$177,749.26	$59,249.75	$22,834.70	$79,955.96	3
5	2	2 006	$177,749.26	$59,249.75	$22,834.70	$79,955.96	3
6	2	2 008	$177,749.26	$59,249.75	$22,834.70	$79,955.96	3
7	3	2 006	$213,051.30	$71,017.10	$19,231.80	$111,788.61	3
8	3	2 008	$213,051.30	$71,017.10	$19,231.80	$111,788.61	3
9	3	2 007	$213,051.30	$71,017.10	$19,231.80	$111,788.61	3
10	4	2 006	$250,187.45	$83,395.82	$53,114.80	$139,477.70	3

Ранжирующие функции

- **ROW_NUMBER()** –возвращает номер строки в окне
- **RANK()** —возвращает ранг каждой строки
  - Строки которые имеют одинаковые значения в столбцах, по которым выполняется сортировка, получают одинаковый ранг **с пропуском следующего значения**
- **DENSE_RANK()** —возвращает ранг каждой строки
  - В отличие от функции RANK, она для одинаковых значений возвращает ранг, **не пропуская следующий**
- **NTILE(n)** – позволяет разделить упорядоченные строки в секции на заданное (n) количество ранжированных групп. Возвращает для каждой строки номер группы
- **Наличие ORDER BY – обязательно!**


```sql
SELECT productname, unitprice, categoryid
    , ROW_NUMBER() OVER w1 AS RowNumber
    , RANK() OVER w2 AS RankPrice
    , DENSE_RANK() OVER w2 AS DenseRankPrice
    , NTILE(3) OVER w1 AS GroupNumber
FROM "Production"."Products"
WINDOW  w1 as (Order BY productid),
        w2 AS (PARTITION BY categoryid ORDER BY unitprice Desc);
```

	productname	unitprice	categoryid	rownumber	rankprice	denserankprice	groupnumber
1	Product QDOMO	$263.50	1	38	1	1	2
2	Product ZZZHR	$46.00	1	43	2	2	2
3	Product RECZE	$19.00	1	2	3	3	1
4	Product NEVTJ	$18.00	1	35	4	4	2
5	Product HHYDP	$18.00	1	1	4	4	1
6	Product LSOFL	$18.00	1	39	4	4	2
7	Product JYGFE	$18.00	1	76	4	4	3
8	Product TOONT	$15.00	1	70	8	5	3
9	Product SWNJY	$14.00	1	34	9	6	2
10	Product XLXQF	$14.00	1	67	9	6	3

ROW_NUMBER

- Позволяет:
  - задать нумерацию, которая будет отличаться от порядка сортировки строк результирующего набора
  - создать "несквозную" нумерацию, т.е. выделить группы из общего множества строк и пронумеровать их отдельно для каждой группы
  - использовать одновременно несколько способов нумерации, так как нумерация не зависит от сортировки строк основного запроса

Функции смещения

- **LAG** или **LEAD** – функция LAG обращается к данным из предыдущей строки окна, а LEAD к данным из следующей строки
  - Имеют три параметра:
    - столбец, значение которого необходимо вернуть
    - количество строк для смещения (по умолчанию 1)
    - значение, которое необходимо вернуть если после смещения возвращается значение NULL
- **FIRST_VALUE** или **LAST_VALUE** — позволяют получить первое и последнее значение в окне
  - В качестве параметра принимают столбец, значение которого необходимо вернуть
- **NTH_VALUE(column_name, n)** - возвращает значение из n-й строки указанного столбца в упорядоченном разделе результирующего набора

PostgreSQL

```
SELECT
    p.productid, p.productname,p.unitprice, p.categoryid,
    NTH_VALUE(productname, 2)
    OVER(PARTITION BY categoryid
        ORDER BY unitprice DESC
        RANGE BETWEEN UNBOUNDED PRECEDING AND
            UNBOUNDED FOLLOWING )
FROM  "Production"."Products" as p;
```

	productid	productname	unitprice	categoryid	nth_value
1	38	Product QDOMO	$263.50	1	Product ZZZHR
2	43	Product ZZZHR	$46.00	1	Product ZZZHR
3	2	Product RECZE	$19.00	1	Product ZZZHR
4	35	Product NEVTJ	$18.00	1	Product ZZZHR
5	1	Product HHYDP	$18.00	1	Product ZZZHR
6	39	Product LSOFL	$18.00	1	Product ZZZHR
7	76	Product JYGFE	$18.00	1	Product ZZZHR
8	70	Product TOONT	$15.00	1	Product ZZZHR

второй самый дорогой продукт в каждой категории

```
SELECT empid, ord_year,
       LAG(total,1,0::money) OVER w AS PrevYear,
       total,
       LEAD(total,1,0::money) OVER w AS NextYear
FROM public."WindowFunctions"
window w as (Partition by empid Order BY ord_year);
```

	empid	ord_year	prevyear	total	nextyear
1	1	2 006	$0.00	$38,789.00	$97,533.58
2	1	2 007	$38,789.00	$97,533.58	$65,821.13
3	1	2 008	$97,533.58	$65,821.13	$0.00
4	2	2 006	$0.00	$22,834.70	$74,958.60
5	2	2 007	$22,834.70	$74,958.60	$79,955.96
6	2	2 008	$74,958.60	$79,955.96	$0.00
7	3	2 006	$0.00	$19,231.80	$111,788.61
8	3	2 007	$19,231.80	$111,788.61	$82,030.89
9	3	2 008	$111,788.61	$82,030.89	$0.00

```
SELECT empid, ord_year, Total,
       FIRST_VALUE(total) OVER w1 AS "FirstValue",
       LAST_VALUE(total)  OVER w1 AS "LastValueErr",
       LAST_VALUE(total) OVER(PARTITION BY empid ORDER BY ord_year
                              ROWS BETWEEN UNBOUNDED PRECEDING AND
                              UNBOUNDED FOLLOWING) AS "LastValue1",
       FIRST_VALUE(total) OVER w2 AS "LastValue2"
FROM public."WindowFunctions"
window w1 as (PARTITION BY empid ORDER BY ord_year),
       w2 as (PARTITION BY empid ORDER BY ord_year desc);
```

	empid	ord_year	total	FirstValue	LastValueErr	LastValue1	LastValue2
1	1	2 006	$38,789.00	$38,789.00	$38,789.00	$65,821.13	$65,821.13
2	1	2 007	$97,533.58	$38,789.00	$97,533.58	$65,821.13	$65,821.13
3	1	2 008	$65,821.13	$38,789.00	$65,821.13	$65,821.13	$65,821.13
4	2	2 006	$22,834.70	$22,834.70	$22,834.70	$79,955.96	$79,955.96
5	2	2 007	$74,958.60	$22,834.70	$74,958.60	$79,955.96	$79,955.96
6	2	2 008	$79,955.96	$22,834.70	$79,955.96	$79,955.96	$79,955.96

Диапазоном по умолчанию является
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

Аналитические функции

- Возвращают информацию о распределении данных и используются для статистического анализа
  - **CUME_DIST** — возвращает относительный ранг текущей строки в окне
  - **PERCENT_RANK** — возвращает относительный ранг текущей строки (ранг-1) / (всего строк — 1) в окне
- Функции распределения рангов вычисляют относительный ранг текущей строки в секции окна, который выражается числом от 0 до 1 (процент)

CUME_DIST и PERCENT_RANK

- Вычисляются по формулам:

```
PERCENT_RANK = (RANK – 1)/(COUNT -1)
```

```
CUME_DIST = RANK/COUNT
```

```
WITH t
AS ( SELECT categoryname, AVG(unitprice::numeric) AS
     FROM "Production"."Products" AS p JOIN
          "Production"."Categories" AS c
           ON p.categoryid = c.categoryid
     GROUP BY categoryname)

SELECT categoryname, "AvgPrice",
       RANK() OVER(ORDER BY "AvgPrice") AS "Rnk",
       PERCENT_RANK() OVER(ORDER BY "AvgPrice") * 100.0 AS "PR",
       CUME_DIST() OVER(ORDER BY "AvgPrice") * 100.0 AS "CD"
FROM t ;
```

	categoryname	AvgPrice	Rnk	PR	CD
1	Grains/Cereals	20,25	1	0	12,5
2	Seafood	20,6825	2	14,2857142857143	25
3	Condiments	23,0625	3	28,5714285714286	37,5
4	Confections	25,16	4	42,8571428571429	50
5	Dairy Products	28,73	5	57,1428571428571	62,5
6	Produce	32,37	6	71,4285714285714	75
7	Beverages	37,9791	7	85,7142857142857	87,5
8	Meat/Poultry	54,0066	8	100	100

- https://leafo.net/guides/postgresql-calculating-percentile.html