Кафедра ЦТ
Институт информационных технологий
РТУ МИРЭА

Дисциплина «Разработка баз данных»

Практическая работа №4. Аналитические запросы: оконные функции и построение сводных таблицв POSTGRES PRO

Постановка задачи: основываясь на индивидуальной схеме данных, составьте необходимые запросы:

Задание №1: использование ранжирующих функций

Для каждой **основной «родительской» сущности** в вашей БД *(например, производитель, категория)* определить **три** наиболее значимых по некоторому **числовому признаку** **«дочерних» сущности** *(например, три самых дорогих товара)*.

В результирующей таблице должны быть указаны идентификатор **группы**, идентификатор **дочерней сущности**, её **числовой признак** и **ранг**. Для расчёта ранга использовать функцию **RANK**() или **DENSE_RANK**().

Задание №2: использование агрегатных оконных функций

Для ключевой **сущности**, имеющей **транзакции во времени** *(например, товар, услуга)*, рассчитать **нарастающий итог** *(кумулятивную сумму)* по **некоторому показателю** *(например, объем продаж, количество заказов)* с **разбивкой по временным периодам** *(месяцам или годам)*.

Отчёт должен содержать **идентификатор сущности** (название/id/…), временной **период**, **сумму** за период и **кумулятивную сумму** с начала наблюдений.

(продолжение на следующем слайде)

Практическая работа №4. Аналитические запросы: оконные функции и построение сводных таблицв POSTGRES PRO

Постановка задачи: основываясь на индивидуальной схеме данных, составьте необходимые запросы:

Задание №3: использование функции смещения

Провести **сравнительный анализ общих показателей по периодам**. Для **каждого периода** *(например, месяца)*, **начиная со второго**, необходимо вывести **общий показатель** за текущий период и аналогичный показатель за **предыдущий период** в одной строке. Это позволит наглядно оценить динамику. Необходимо **использовать функцию LAG()**.

Задание №4: построение сводной таблицы

Создать **сводный отчет**, который агрегирует некоторый **числовой показатель** для **основной сущности по категориям**, представленным в виде столбцов. **Строки** в отчете должны представлять **основные сущности**, а **столбцы — категории**.

Задачу необходимо **решить двумя способами**:

1. С использованием условной агрегации (комбинация SUM и CASE).
2. С использованием функции crosstab из расширения tablefunc.

(начало на предыдущем слайде)

Подготовка базы данных

Задание №1 (ранжирование)

В базе данных должны быть как **минимум две таблицы** с отношением **«один-ко-многим»** *(например, Категории и Товары)*.

В **«дочерней» таблице** *(например, Товары)* **обязательно** должен присутствовать **числовой столбец**, по которому можно проводить ранжирование *(например, цена, рейтинг, количество_на_складе)*.

Для наглядного результата необходимо **создать не менее 3 записей в «родительских» таблицах**, для каждой из которых будет **не менее 3-4 дочерних записей**.

Подготовка базы данных

Задания №2, №3, №4 (анализ по времени и сводные таблицы)

Эти задачи требуют наличия в базе данных **таблицы** с **транзакционными или историческими данными** *(например, таблиц «Продажи» или «Заказы»)*.

В этой таблице **должен быть столбец** с типом данных **DATE** или **TIMESTAMP** (например, дата_продажи, дата_заказа).

Также **необходим числовой столбец** для проведения вычислений *(например, сумма_продажи, количество)*.

Для корректного выполнения заданий необходимо, чтобы в этой **таблице** было **не менее 10-15 записей**, причем **даты в этих записях** должны охватывать **несколько разных периодов** *(например, несколько месяцев или кварталов одного года)*.

Это позволит корректно рассчитать **нарастающие итоги**, **сравнить периоды** и построить информативную **сводную таблицу**.

Оконные функции

Зачем нужны оконные функции?

- В одной строке – и данные и метрика.
  «Покажи покупку и её **накопительный** итог/место в **топе**/сравнение с **прошлым месяцем**».
- Меньше «обходных путей». Множество возможностей **без** дополнительных **JOIN** и **без потери строк**.

Таблица "Покупки" - группировка

ID_покупателя	итог
1	600
2	260

Таблица "Покупки" - оконная функция

ID_покупки	ID_покупателя	сумма покупки	накопительный итог
1	1	100	100
2	1	120	220
3	1	80	300
4	1	300	600
5	2	60	60
6	2	200	260

Оконные функции – окно

Окно – это правило, определяющее, **какие** **строки участвуют в расчёте значения функции** для **текущей строки**.

Оно задаётся строго внутри **OVER**(...):

- **PARTITION BY** – **границы** раздела: определяет, какие **строки** относятся к какому **разделу**
  *(**похоже** на **GROUP BY**, но только разделяет строки таблицы на разделы, не «схлопывая» их).*
- **ORDER BY** – **порядок** внутри раздела (ось «кто был **раньше**, кто **позже**»).
- [**ROWS** | **RANGE** | **GROUPS***] – **рамка (FRAME) окна**: определяет, какую ***часть*** текущего **раздела** (сформированного **PARTITION BY**) мы рассматриваем прямо сейчас.

```
<функция>(...) OVER (
    [PARTITION BY ...]
    [ORDER BY ...]
    [ROWS | RANGE ...] -- FRAME
)
```

**Есть не во всех СУБД*

Оконные функции – окно (схема работы)

Таблица "Покупки"

ID_строки	ID_покупателя	дата	стоимость	сумма
1	1	01.01.2025	100	100
2	1	10.01.2025	120	220
3	2	20.01.2025	70	70
4	2	05.02.2025	50	120
5	2	05.02.2025	180	300
6	2	10.02.2025	40	340
7	2	10.02.2025	80	420
8	2	10.02.2025	60	480
9	2	17.02.2025	90	570
10	2	20.02.2025	30	600
11	3	01.01.2025	60	
12	3	11.01.2025	200	

Раздел покупателя 1

Раздел покупателя 2 *(обрабатывается)*

Раздел покупателя 3

FRAME (рамка)

Активная строка

```
PARTITION BY ID_покупателя ORDER BY дата, ID_строки
```

* По умолчанию используется рамка **«Нарастающий итог до текущей»**

Оконные функции – разделы

PARTITION BY – формирует **разделы**, отделяя одни строки от других по **выбранному столбцу**.

Любая **оконная функция** для **текущей строки** видит **только** **строки её раздела**.

- Это **НЕ GROUP BY**: строки **не «схлопываются»** – мы лишь **ограничиваем «пул строк»** для окна.
- Если **PARTITION BY** **опущен**, получаем один **глобальный раздел** из всех строк.

```
ФУНКЦИЯ() OVER (
    PARTITION BY ID_покупателя
) AS название_столбца;
```

Таблица "Покупки"

	ID_покупки	ID_покупателя	сумма покупки
Покупатель 1	1	1	100
	2	1	120
	3	1	80
Покупатель 2	4	2	60
	5	2	200

Оконные функции – порядок

ORDER BY – задаёт **последовательность строк внутри** каждого **раздела**.

Нужен, чтобы **различать** «*предыдущая/следующая*», «*раньше/позже*».

Обязателен для:

- **Рангов** (**ROW_NUMBER**, **RANK**, **DENSE_RANK**)
- **Кумулятивов** (нарастающих/скользящих)
- **Позиционирования строк** **LAG** *(предыдущий)* / **LEAD** *(следующий)*

Таблица "Покупки"

ID_покупки	ID_покупателя	сумма покупки
1	1	80
2	1	100
4	1	120
3	1	120
5	2	60
6	2	200

При **равных значениях** ***«ключа сортировки»*** *(выбранного столбца)* **необходим «tie-breaker»** *(второй столбец)*.

```
ФУНКЦИЯ() OVER (
    PARTITION BY ID_покупателя
    ORDER BY "сумма покупки", ID_записи DESC
) AS название_столбца;
```

Оконные функции – порядок (ранги)

Ранги (**ROW_NUMBER**, **RANK**, **DENSE_RANK**)

Формирование рангов возможно **только по столбцам**, значения которых были **упорядочены** с помощью **ORDER BY**!

Полезны, чтобы **задать номера** строкам **внутри группы**, позволяя сформировать «топ-N строк», определить, кто «первый» / «последний в группе», решить проблему дублей, создать пагинацию и т.д.

- **ROW_NUMBER()** – **позиция** строки в разделе. Выдаёт 1,2,3… строго по порядку.
- **RANK()** – № места **с пропусками после дублей**. При равных значениях даёт одинаковый ранг, следующее место перескакивает (1,1,3).
- **DENSE_RANK()** – № места **без пропусков**. При равенствах место одно и то же, следующее – на 1 больше (1,1,2).

Таблица "Покупки"

ROW_NUMBER()	ID_покупки	ID_покупателя	сумма покупки	RANK()	DENSE_RANK()
1	1	1	120	1	1
2	2	1	120	1	1
3	3	1	80	3	2
1	4	2	200	1	1
2	5	2	60	2	2
3	6	2	60	2	2

Оконные функции – порядок (позиционирование

Позиционирование строк

LAG(), **LEAD**() – функции, необходимые для сравнения **текущей** строки **с соседями** **на основании** **порядка**.

Позволяет считать дельты, темпы роста, флаги изменений, интервалы между событиями – **без JOIN**.

- **LAG(колонка [, шаг смещения [, default]])** *(предыдущая)*
- **LEAD(колонка [, шаг смещения [, default]])** *(следующая)*

```
SELECT ID_покупателя, "сумма покупки",
    LAG("сумма покупки", 1, 0) OVER (
        PARTITION BY ID_покупателя
        ORDER BY ID_покупки
    ) AS "сумма предыдущей покупки"
FROM "Покупки";
```

Таблица "Покупки"

ID_покупки	ID_покупателя	LAG(сумма покупки)	сумма покупки	LEAD(сумма покупки)
1	1	0	120	120
2	1	120	120	80
3	1	120	80	0
4	2	0	200	60
5	2	200	60	60
6	2	60	60	0

Оконные функции – рамка

[**ROWS** | **RANGE** | **GROUPS**] – уточняет, **какую часть текущего раздела видит оконная функция** при **обработке текущей строки** (важен **порядок строк**, заданный оператором **ORDER BY**).

- **Без ORDER BY** точная рамка бессмысленна, т.к. фактически **«окном»** будет **весь раздел**.
- Если **ORDER BY** есть, но **FRAME*** не указан, по умолчанию будет использован нарастающий итог: *«с начала раздела до текущей группы»*

Варианты рамок:

- **ROWS** – **позиционная**, по количеству **строк** *(выбираем **дополнительные строки** относительно **текущей строки**)*;
- **RANGE** – **диапазон значений по ключу сортировки**. Берёт **все строки**, у которых **значение ключа** попадает в интервал относительно **текущего значения** (например, «+7 дней **до**»). **Дубликаты** строки включает **всегда**;
- **GROUPS** – окно по **группам одинаковых значений** *(дополнительные строки выбираются относительно **всех** строк со значением **как у текущей** строки)*.

*** FRAME** – это **название группы** операторов [**ROWS** | **RANGE** | **GROUPS**], это **НЕ оператор**.

Оконные функции – рамка (ROWS)

Позиционирование по количеству строк

ROWS выбирает **N строк** **относительно** **текущей строки** (в прошлое/будущее).

Дубликаты ключа сортировки **не схлопываются**: каждая строка с той же датой/ценой/… считается **отдельной строкой**, и будет ли она включена в окно – зависит только **от её позиции**.

В примере **окном** является
текущая СТРОКА + 2 предшествующих СТРОКИ

```
SELECT
  ID_покупателя, дата, стоимость,
  SUM(сумма) OVER (
    PARTITION BY ID_покупателя
    ORDER BY дата, ID_строки
    ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
) AS сумма
FROM Покупки
WHERE ID_покупателя = 2;
```

Таблица "Покупки"

ID_строки	ID_покупателя	дата	стоимость	сумма
3	2	20.01.2025	70	70
4	2	05.02.2025	50	120
5	2	05.02.2025	180	300
6	2	10.02.2025	40	270
7	2	10.02.2025	80	300
8	2	10.02.2025	60	180
9	2	17.02.2025	90	230
10	2	20.02.2025	30	180

Рамка (ROWS)

Активная строка

Оконные функции – рамка (RANGE)

Диапазон по значению ключа сортировки

RANGE берёт **все строки**, **значение ключа** которых попадает **в интервал** **относительно** **текущего значения** (даты/числа).

Все **дубликаты** – «***пиры***» *(строки с тем же значением ключа, что у текущей строки)* **входят** **всегда**.

В примере **окном** является
текущая СТРОКА + все за 14 дней до неё + дубли

```
SELECT
  ID_покупателя, дата, стоимость,
  SUM(сумма) OVER (
    PARTITION BY ID_покупателя
    ORDER BY дата
    RANGE BETWEEN INTERVAL '14 days'
PRECEDING AND CURRENT ROW
  ) AS сумма
FROM Покупки
WHERE ID_покупателя = 2;
```

Таблица "Покупки"

ID_строки	ID_покупателя	дата	стоимость	сумма
3	2	20.01.2025	70	70
4	2	05.02.2025	50	230
5	2	05.02.2025	180	230
6	2	10.02.2025	40	410
7	2	10.02.2025	80	410
8	2	10.02.2025	60	410
9	2	17.02.2025	90	500
10	2	20.02.2025	30	300

Оконные функции – рамка (GROUPS)

Счёт по «позициям» ключа (группам пиров)

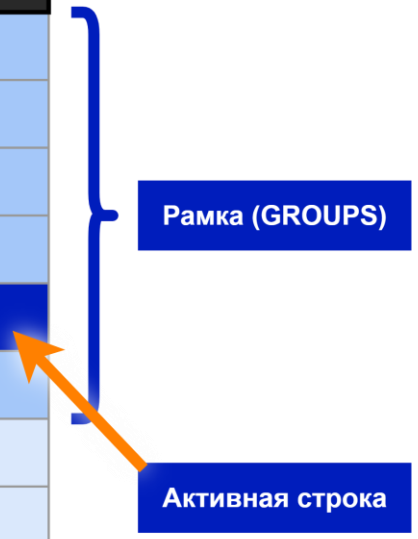
GROUPS двигает рамку **группами одинаковых значений ключа** (**ORDER BY**): «N групп назад/вперёд», где **группа** = **все строки** с тем же **значением ключа** (одна дата, одна цена и т.п.).

В примере **окном** является
текущая ГРУППА + 2 предшествующих ГРУППЫ

```
SELECT
  ID_покупателя, дата, стоимость,
  SUM(сумма) OVER (
    PARTITION BY ID_покупателя
    ORDER BY дата
    GROUPS BETWEEN 2 PRECEDING AND
CURRENT ROW
  ) AS сумма
FROM Покупки
WHERE ID_покупателя = 2;
```

Таблица "Покупки"

ID_строки	ID_покупателя	дата	стоимость	сумма
3	2	20.01.2025	70	70
4	2	05.02.2025	50	300
5	2		180	300
6	2	10.02.2025	40	480
7	2	10.02.2025	80	480
8	2	10.02.2025	60	480
9	2	17.02.2025	90	500
10	2	20.02.2025	30	300



Оконные функции – рамка (варианты)

Здесь в качестве **FRAME** выступает одно из значений: [**ROWS** | **RANGE** | **GROUPS**]

- **Нарастающий итог** ***(до текущей)***
  `[FRAME] BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW` ***(значение по умолчанию, с RANGE)***
- **Последние N** ***единиц (строк/групп)*** **+ текущая** ***единица*** ***(скользящее окно)***
  `[FRAME] BETWEEN N PRECEDING AND CURRENT ROW`
- **Окно «вперёд»** ***(от текущей)***
  `[FRAME] BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`
- **Только текущая строка**
  `[FRAME] BETWEEN CURRENT ROW AND CURRENT ROW` *(или шорткат – [**FRAME**]* ***CURRENT ROW****)*
- **Весь раздел**
  `[FRAME] BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`
- **Центрированное окно** ***(±N соседей)***
  `[FRAME] BETWEEN N PRECEDING AND N FOLLOWING`

* Более подробно **все варианты** расписаны в **методичке**

Pivot

Pivot – это превращение «***длинной***» таблицы *(категория в строках)*
в «**широкую**» *(когда категории становятся столбцами)*.

Таблица "Покупки"

ID_продукта	месяц	сумма
1	январь	100
1	февраль	120
2	март	100
2	март	120
2	январь	100
3	январь	300
3	март	50
3	февраль	210

Таблица "Покупки"

ID_продукта	январь	февраль	март	итого
1	100	120	0	220
2	100	0	220	320
3	300	210	50	560

Pivot

Зачем нужен Pivot?

1. Посмотреть **соседние показатели** (сравнение по категориям/периодам «в линию»).
2. Подготовить **отчёт/дашборд**, где категории – это колонки.
3. **Упростить фильтрацию/сортировку** по нескольким категориям одновременно.

Когда **Pivot** **не нужен**?

- Аналитика «по времени» с трендами/скользящими окнами – часто лучше в длинном формате с использованием оконных функций/визуализации.
- Если категорий слишком много (сотни/тысячи), «широкий» вид становиться неудобным, поэтому данным лучше оставаться «длинными», а для удобства просмотра – пользоваться фильтрами.

ВАЖНО! Pivot почти всегда требует **агрегации** по ячейкам (SUM/COUNT/AVG), т.к. при его использовании с высокой вероятностью будут дубликаты.

Pivot – условная агрегация

Pivot с помощью **условной агрегации**.

Стандартный и универсальный метод, работающий в любой СУБД.

Для **каждого будущего столбца** **применяем агрегирующую функцию** (**SUM**) **в комбинации с условием** (**CASE**).

Плюсы: простота и универсальность, полный контроль логики.

Минусы: много кода при большом числе столбцов; список колонок фиксированный.

```sql
SELECT id_продукта,
    SUM(CASE WHEN месяц = 'январь' THEN сумма END) AS "январь",
    SUM(CASE WHEN месяц = 'февраль' THEN сумма END) AS "февраль",
    SUM(CASE WHEN месяц = 'март' THEN сумма END) AS "март",
    SUM(сумма) AS "итого"
FROM продажи
GROUP BY id_продукта
ORDER BY id_продукта;
```

Pivot – Crosstab

Crosstab – это мощная функция из специального расширения **PostgreSQL** **tablefunc**, предназначенная для создания сводных таблиц.

Crosstab принимает на вход **SQL-запрос**, который обязательно должен **возвращать ровно три столбца в строгом порядке**:

1. **row_name** – идентификатор строки (что станет строками в итоговой таблице).
2. **category** – категория (что станет столбцами).
3. **value** – значение (что окажется в ячейках).

Критически важное правило: **результаты** этого запроса должны быть **отсортированы** сначала **по первому столбцу**, а затем **по второму** (**ORDER BY 1, 2**). Нарушение этого правила – самая частая причина ошибок.

Pivot – Crosstab

Crosstab – структура вызова.

Функция принимает **на вход** два **текстовых аргумента**:

- **source_sql** – исходный запрос, возвращающий **3 столбца** (***row_name***, ***category***, ***value***).
- **categories_sql** *(опционально)* – запрос, возвращающий **список уникальных категорий** в том порядке, в котором они должны стать столбцами.

```
SELECT * FROM crosstab(
    $$ SELECT id_продукта, месяц, SUM(сумма)
        FROM продажи
        GROUP BY id_продукта, месяц
        ORDER BY 1,2
    $$,
    $$ SELECT m FROM (VALUES ('январь'),('февраль'),('март')) v(m)$$
) AS ct(
    id_продукта int, "январь" numeric, "февраль" numeric, "март" NUMERIC
);
```

Спасибо за внимание