Наследование и оконные функции в PostgreSQL

chatgpt.com/c/6826e505-eaf8-8006-b016-08882e820f8d

Наследование в Postgresql

Наследование в PostgreSQL — это возможность создавать таблицы, которые "наследуют" структуру другой таблицы (аналогично наследованию классов в объектно-ориентированном программировании). Эта функциональность уникальна среди реляционных СУБД и может быть полезна для определённых паттернов моделирования данных.

📌 Синтаксис

```sql
CREATE TABLE дочерняя_таблица (
    -- дополнительные столбцы
) INHERITS (родительская_таблица);
```

🧱 Пример

```
sql
```

```
-- Родительская таблицаCREATETABLE animals (
    id SERIAL PRIMARY KEY,
    name TEXT,
    species TEXT
);

-- Дочерняя таблицаCREATETABLE dogs (
    breed TEXT
) INHERITS (animals);
```

Теперь `dogs`содержит поля `id`, `name`, `species`и `breed`.

🔍 Особенности

- **Вставка в дочернюю таблицу**не влияет на родительскую.
- **Запрос к родительской таблице**по умолчанию включает строки из всех дочерних таблиц:

  ```
  sql
  ```

  ```
  SELECT*FROM animals;
  ```

  вернёт и строки из `dogs`, и из других потомков.

  Чтобы получить только строки из родительской таблицы:

  ```
  sql
  ```

  ```
  SELECT*FROMONLY animals;
  ```

⚠️ Ограничения

- **Ограничения (constraints)**: `CHECK`, `NOT NULL`, `FOREIGN KEY`не наследуются автоматически.
- **Индексы и первичные ключи**не наследуются.
- **Триггеры и правила**не наследуются.

- **Нельзя одновременно унаследовать от нескольких таблиц** (нет множественного наследования).

🎯 Когда использовать

- Разделение по типу: например, `transactions`, `income_transactions`, `expense_transactions`.
- Логическое разделение с возможностью обращаться к общей родительской таблице.
- Простейшая реализация *table partitioning* (в старых версиях PostgreSQL <10, до появления `PARTITION BY`).

⚡ Альтернатива: секционирование

С PostgreSQL 10 и выше рекомендован более мощный механизм — **секционирование таблиц** (`PARTITION BY`). Оно имеет более высокую производительность, лучше поддерживается оптимизатором и более предсказуемо.

📚 Итог

Поддерживается наследованием	Поддерживается секционированием
Структура таблицы	Да
Ограничения, индексы	Нет
Прозрачная агрегация данных	Да
Производительность	Умеренная
Оптимизация планов	Ограничена

Хочешь пример с `INSERT`/`SELECT`, или переписать модель на секционирование?

секционирование таблиц подробно

Секционирование (partitioning) в PostgreSQL — это способ физического разделения большой таблицы на более мелкие части (секции), называемые *partitions*, с целью повышения производительности, масштабируемости и управляемости.

🧱 Что такое секционирование

Секционирование позволяет:

- Работать с одной логической таблицей, данные которой хранятся в разных физических таблицах.
- Автоматически направлять `INSERT`в нужную секцию.
- Улучшать производительность `SELECT`, `UPDATE`, `DELETE`, если используется partition pruning (отсечение ненужных секций).

📌 Синтаксис (PostgreSQL ≥10)

1. Создание секционированной таблицы

```
sql



CREATETABLE sales (
    id SERIAL,
    sale_date DATENOTNULL,
    amount NUMERIC
) PARTITIONBYRANGE (sale_date);
```

2. Добавление секций

```
sql



CREATETABLE sales_2024 PARTITIONOF sales
    FORVALUESFROM ('2024-01-01') TO ('2025-01-01');

CREATETABLE sales_2025 PARTITIONOF sales
    FORVALUESFROM ('2025-01-01') TO ('2026-01-01');
```

🧩 Виды секционирования

Тип секционирования	Описание
`RANGE`	Диапазоны значений (например, по дате)
`LIST`	Наборы значений (например, по региону)
`HASH`	Хэш-функция от значения (равномерное распределение)
`COMPOSITE`	Комбинация нескольких типов

Примеры

Секционирование по списку:

```
sql
```

```
CREATETABLE users (
    id INT,
    region TEXT
) PARTITIONBY LIST (region);

CREATETABLE users_ru PARTITIONOF users FORVALUESIN ('RU');
CREATETABLE users_us PARTITIONOF users FORVALUESIN ('US');
```

Секционирование по хэшу:

```
sql
```

```
CREATETABLE logs (
    id BIGINT,
    msg TEXT
) PARTITIONBY HASH (id);

CREATETABLE logs_0 PARTITIONOF logs FORVALUESWITH (MODULUS 4, REMAINDER 0);
CREATETABLE logs_1 PARTITIONOF logs FORVALUESWITH (MODULUS 4, REMAINDER 1);
```

⚙️ Что важно учитывать

✔️ Преимущества

- **Быстрые запросы**: благодаря отсечению ненужных секций (`partition pruning`).
- **Управляемость**: можно отдельно управлять таблицами-секциями (архивация, удаление).
- **Параллельность**: запросы обрабатываются параллельно по секциям.

⚠️ Ограничения и особенности

- **Первичные ключи**работают только с `UNIQUE` + `NOT NULL`по `PARTITION KEY`.
- **Индексы и ограничения**создаются отдельно для каждой секции.
- **В `INSERT`значения должны явно попадать в одну из секций**, иначе ошибка.

- `FOREIGN KEY`**на секционированную таблицу не поддерживается** (но можно сделать на секции вручную).

📈 Пример полного цикла

sql

```
-- Шаг 1: Создаём таблицу со секционированием по датеCREATETABLE events (
    id SERIAL PRIMARY KEY,
    event_date DATENOTNULL,
    description TEXT
) PARTITIONBYRANGE (event_date);

-- Шаг 2: Создаём секцииCREATETABLE events_2023 PARTITIONOF events
    FORVALUESFROM ('2023-01-01') TO ('2024-01-01');

CREATETABLE events_2024 PARTITIONOF events
    FORVALUESFROM ('2024-01-01') TO ('2025-01-01');

-- Шаг 3: Вставка автоматически попадёт в нужную секциюINSERTINTO events (event_date,
description) VALUES ('2024-03-10', 'Spring event');
```

🔎 Как узнать, куда попали данные?

sql

```
SELECT tableoid::regclass, *FROM events;
-- Покажет имя секции, где реально хранятся данные
```

🏴 Администрирование

- Добавление секции: `CREATE TABLE ... PARTITION OF ... FOR VALUES ...`
- Удаление: `DROP TABLE секция`
- Переименование: `ALTER TABLE секция RENAME TO ...`

- Индексы: создаются отдельно:

```sql
CREATE INDEX ON events_2024 (event_date);
```

🚀 Итог

Секционирование полезно, если:

- У вас миллионы/миллиарды строк.
- Часто обращаетесь по `WHERE date`, `region`, `id`.
- Нужно быстро архивировать, удалять или загружать данные по блокам.

Хочешь пример автоматической генерации секций или функцию для создания новых разделов по времени?

Оконные функции Postgresql

Оконные функции в PostgreSQL — мощный инструмент для анализа строк в пределах логического окна (набора строк, связанных по определённому признаку). Они позволяют выполнять агрегатные и аналитические расчёты без группировки всего результата, то есть **оставляя доступ к каждой строке**.

📌 Общий синтаксис

```sql
<функция>() OVER (
    [PARTITIONBY ...]
    [ORDERBY ...]
    [ROWS|RANGEBETWEEN ...]
)
```

🔍 Части окна

Компонент	Назначение
`PARTITION BY`	Делит строки на подмножества (как `GROUP BY`, но без склеивания строк)
`ORDER BY`	Задает порядок обработки строк в каждом окне
`ROWS BETWEEN`	Задает "рамку" — какие строки относительно текущей будут участвовать

🧠 Часто используемые оконные функции

Функция	Назначение
`ROW_NUMBER()`	Номер строки в рамках окна
`RANK()`	Ранг с пропусками (например: 1, 1, 3)
`DENSE_RANK()`	Ранг без пропусков (например: 1, 1, 2)
`NTILE(n)`	Разбивает окно на `n` групп по порядку
`LAG(expr, N)`	Значение `N` строкой выше
`LEAD(expr, N)`	Значение `N` строкой ниже
`FIRST_VALUE()`	Первое значение в окне
`LAST_VALUE()`	Последнее значение в окне
`SUM()`, `AVG()`	Агрегаты в пределах окна

📈 Примеры

🔢 Пронумеровать строки по отделу

```
sql
```

```
SELECT*,
       ROW_NUMBER() OVER (PARTITIONBY department ORDERBY salary DESC) AS row_num
FROM employees;
```

🧮 Сумма зарплат по отделу без группировки

```
sql
```

```
SELECT*,
       SUM(salary) OVER (PARTITIONBY department) AS total_dept_salary
FROM employees;
```

📉 Отставание по зарплате от предыдущего сотрудника

```
sql
```

```
SELECT*,
       salary -LAG(salary) OVER (PARTITIONBY department ORDERBY salary) AS diff_from_prev
FROM employees;
```

📊 Скольжение по 3 строкам (скользящее окно)

```
sql
```

```
SELECT*,
       AVG(salary) OVER (
           ORDERBY hire_date
           ROWSBETWEEN2 PRECEDING ANDCURRENTROW
       ) AS avg_last_3
FROM employees;
```

⚙️ ROWSvs RANGE

- `ROWS BETWEEN 1 PRECEDING AND CURRENT ROW`: строго 2 строки — текущая и предыдущая.
- `RANGE BETWEEN 100 PRECEDING AND CURRENT ROW`: строки в пределах значения (например, даты ± 100 дней) — используется реже.

💡 Применения оконных функций

- Ранжирование лидеров, топ-N по категориям.
- Поиск изменений между строками (`LAG`, `LEAD`).
- Кумулятивные суммы (`SUM(...) OVER (...)`).
- Подсчёт процентов (`RANK() / COUNT()`).
- Анализ поведения пользователя по сессиям.
- Расчёт скользящих средних.

🔪 Пример из жизни

```
sql
```

```
SELECT
    user_id,
    event_time,
    LAG(event_time) OVER (PARTITIONBY user_id ORDERBY event_time) AS prev_event_time,
    event_time -LAG(event_time) OVER (PARTITIONBY user_id ORDERBY event_time) AS time_diff
FROM user_events;
```

👉 Анализ времени между действиями одного пользователя.

Хочешь пример из своей задачи или генерацию нескольких оконных функций в одном запросе?

Полный список "оконные функции"

В PostgreSQL оконные функции делятся на три основные группы:

✔️ 1. Нумерация и ранжирование

Функция	Описание
`ROW_NUMBER()`	Порядковый номер строки в окне
`RANK()`	Ранг с пропусками (1, 1, 3, ...)
`DENSE_RANK()`	Ранг без пропусков (1, 1, 2, ...)
`NTILE(n)`	Делит окно на `n`равных групп, возвращает номер группы

✔️ 2. Смещения (LAG/LEAD) и значения

Функция	Описание
`LAG(expr [, offset [, default]])`	Значение предыдущей строки (по умолчанию `offset = 1`)
`LEAD(expr [, offset [, default]])`	Значение следующей строки
`FIRST_VALUE(expr)`	Значение первой строки в окне
`LAST_VALUE(expr)`	Значение последней строки в окне
`NTH_VALUE(expr, n)`	`n`-тое значение в окне

✔️ 3. Агрегатные функции как оконные

Обычные агрегаты можно использовать как оконные функции с `OVER(...)`:

Функция	Описание
`SUM(expr)`	Сумма значений в окне
`AVG(expr)`	Среднее значение

Функция	Описание
`MIN(expr)`	Минимальное значение
`MAX(expr)`	Максимальное значение
`COUNT(expr)`	Количество непустых значений
`COUNT(*)`	Количество строк
`STDDEV(expr)`	Стандартное отклонение
`VARIANCE(expr)`	Дисперсия
`BOOL_AND(expr)`	Логическое И по всем строкам
`BOOL_OR(expr)`	Логическое ИЛИ
`ARRAY_AGG(expr)`	Агрегация значений в массив
`STRING_AGG(expr, delimiter)`	Склеивание строк через разделитель

✔️ 4. Статистические и специальные

Функция	Описание
`CUME_DIST()`	Кумулятивное распределение (значение от 0 до 1)
`PERCENT_RANK()`	Процентный ранг
`PERCENTILE_CONT(f)`	Непрерывная квантиль (для медиан и т. п.)
`PERCENTILE_DISC(f)`	Дискретная квантиль
`MODE() WITHIN GROUP`	Мода — наиболее часто встречающееся значение
`MEDIAN() WITHIN GROUP`	Медиана (если включена через расширения)

🧩 Поддержка оконной рамки

Многие функции поддерживают опции **рамки окна**:

```
sql
```

```
OVER (
  PARTITIONBY ...
  ORDERBY ...
  ROWSBETWEEN ... AND ...
)
```

Типы рамок:

- `ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`
- `RANGE BETWEEN INTERVAL '1 day' PRECEDING AND CURRENT ROW`
- `GROUPS BETWEEN ...`

🔍 Пример всего вместе

```
sql
```

```
SELECT
  department,
  employee,
  salary,
  SUM(salary) OVER (PARTITIONBY department) AS dept_total,
  ROW_NUMBER() OVER (PARTITIONBY department ORDERBY salary DESC) AS rownum,
  LAG(salary) OVER (PARTITIONBY department ORDERBY salary) AS prev_salary,
  RANK() OVER (PARTITIONBY department ORDERBY salary DESC) AS salary_rank
FROM employees;
```

Хочешь, я сгенерирую рабочий пример таблицы и запросов с оконными функциями для анализа?

Полезно ли это обсуждение на данный момент?