

Рекурсивни заявки (Recursive Queries)

Въведение

Рекурсивните заявки са мощен инструмент в SQL, използван за работа с йерархични или рекурсивни структури от данни, като организационни йерархии, графи и дървета. Те се реализират чрез **CTE (Common Table Expressions)** и **рекурсивни CTE-та**.

Какво представляват рекурсивните заявки и защо се използват?

Рекурсивните заявки позволяват на SQL да изпълнява итеративни операции върху данни, без да се използват цикли в приложната логика. Те са особено полезни в случаи, когато работим с йерархични данни (като структура на компания), графи (намиране на пътища между възли) или когато се изисква генериране на числови последователности.

Основни сценарии за използване:

1. **Йерархични структури** – намиране на всички подчинени на даден мениджър.
2. **Графи** – намиране на най-кратък път между възли.
3. **Генериране на последователности** – алтернативен метод за `generate_series`.
4. **Агрегации и трансформации** – при обработка на рекурсивни зависимости между записи.
5. **Разглеждане на връзки в социални мрежи** – намиране на приятели на приятели.

Как работят рекурсивните CTE?

Рекурсивният CTE има две основни части:

1. **Анкерна (базова) част** – това е първата заявка, която задава началния набор от резултати.
2. **Рекурсивна част** – тя използва рекурсивно самия CTE, докато не се достигне до условие за спиране.

Синтаксисът изглежда така:

```
WITH RECURSIVE cte_name AS (  
  -- Анкерна част  
  базова_заявка  
  UNION ALL
```

```

-- Рекурсивна част
рекурсивна_заявка
)
SELECT * FROM cte_name;

```

Пример 1: Йерархична структура на служители

Представете си таблица **employees**, която съдържа следните данни:

id	name	manager_id
1	Иван	NULL
2	Петър	1
3	Анна	1
4	Мария	2
5	Георги	2

За да намерим йерархията на подчинение, можем да използваме рекурсивен CTE:

```

WITH RECURSIVE EmployeeHierarchy AS (
  -- Анкерна част: намираме всички служители без мениджър (топ ниво)
  SELECT id, name, manager_id, 1 AS level
  FROM employees
  WHERE manager_id IS NULL

  UNION ALL

  -- Рекурсивна част: добавяме служителите, чиито мениджъри вече са включени
  SELECT e.id, e.name, e.manager_id, eh.level + 1
  FROM employees e
  JOIN EmployeeHierarchy eh ON e.manager_id = eh.id
)
SELECT * FROM EmployeeHierarchy;

```

Пример 2: Генериране на последователност от числа

Ако искаме да генерираме числа от 1 до 10 без използване на **generate_series**, можем да направим това с рекурсивен CTE:

```

WITH RECURSIVE numbers AS (

```

```

SELECT 1 AS num
UNION ALL
SELECT num + 1 FROM numbers WHERE num < 10
)
SELECT * FROM numbers;

```

Пример 3: Търсене в граф (намиране на път между възли)

Ако имаме таблица `graph_edges`, която представлява граф със свързани възли:

from_node e	to_node e
A	B
B	C
C	D
A	E
E	F
F	G

Искаме да намерим всички пътища от `A` до `D`:

```


WITH RECURSIVE path AS (
  SELECT from_node, to_node, 1 AS depth FROM graph_edges WHERE from_node = 'A'
  UNION ALL
  SELECT g.from_node, g.to_node, p.depth + 1
  FROM graph_edges g
  JOIN path p ON g.from_node = p.to_node
  WHERE p.depth < 5 -- Ограничаваме дълбочината
)
SELECT * FROM path WHERE to_node = 'D';

```





Предимства и недостатъци

Предимства:

- ✓ Улесняват работата с йерархични данни и графи.
- ✓ Позволяват изчисления, които иначе изискват многократни заявки или сложен код.
- ✓ Работят директно в SQL без

нужда от външни скриптове.  Ефективни за обработка на графи и сложни зависимости.

Недостатъци:

 Могат да бъдат бавни при големи набори от данни.  Трудни за отстраняване на грешки и оптимизация.  В някои бази данни има ограничения върху дълбочината на рекурсията.  Ако не са правилно оптимизирани, могат да доведат до неефективно използване на ресурси.

Ограничения и оптимизация

- **Ограничаване на дълбочината:** Ако рекурсията не е контролирана, може да доведе до безкраен цикъл. Затова е добра практика да използваме **ограничение на дълбочината** (например `WHERE level < 10`).
- **Производителност:** Рекурсивните заявки могат да бъдат бавни при големи набори от данни. Индексирането на колоните, използвани в `JOIN`, може да подобри производителността.
- **Алтернативи:** Понякога йерархични данни могат да се обработват по-ефективно чрез материализирани изгледи или графови бази от данни (като Neo4j).

Заклучение

Рекурсивните заявки в SQL са мощен инструмент за работа с йерархични и итеративни структури. Те могат да бъдат полезни за изграждане на организационни диаграми, намиране на пътища в графи и други сложни операции. Разбирането и правилното им използване ще ти помогне да се справяш по-ефективно с подобни задачи.