

Избавляемся от рекурсивных запросов в Postgresql при работе с графом

1. Постановка задачи
2. Уточняем требования (синтетическая задача)
 - 2.1 Обеспечить запись вершин и связей дерева в PostgreSQL
 - 2.2 Обеспечить доступ по id вершины к ней и ее потомкам
- 3 Проблема наивного решения
- 4 Решение проблемы
- 5 Цена решения
- 6 Альтернативный способ решения

<https://github.com/itatsiy/pg-graph-demo>

1. Постановка задачи

Обеспечить хранение и доступ данных образующих **древовидную структуру**.

Дерево - это граф в котором только один узел **не имеет входящих ребер** совсем, а остальные узлы имеют строго **по одному входящему узлу**.



2. Уточняем требования (синтетическая задача)

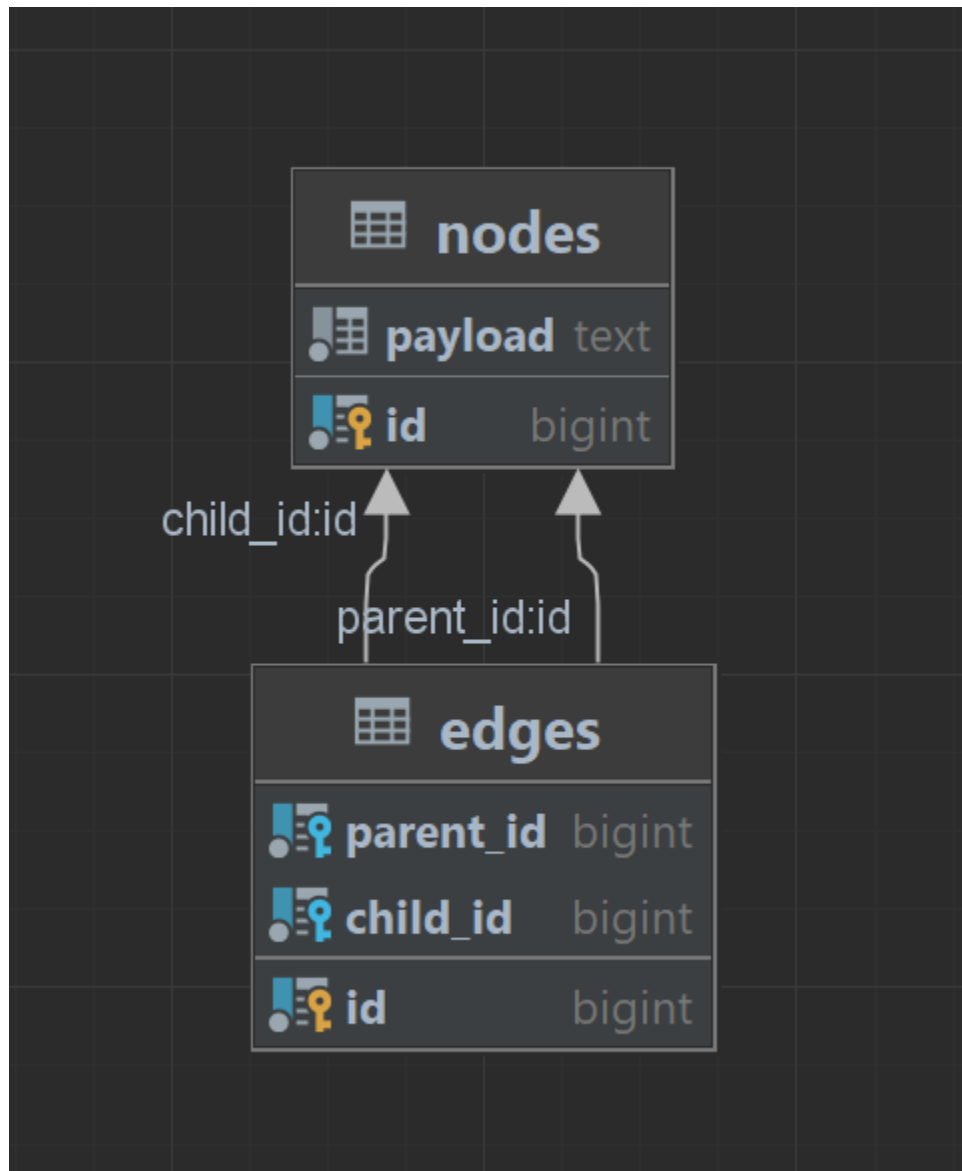
2.1 Обеспечить запись вершин и связей дерева в PostgreSQL

Пример входных данных:

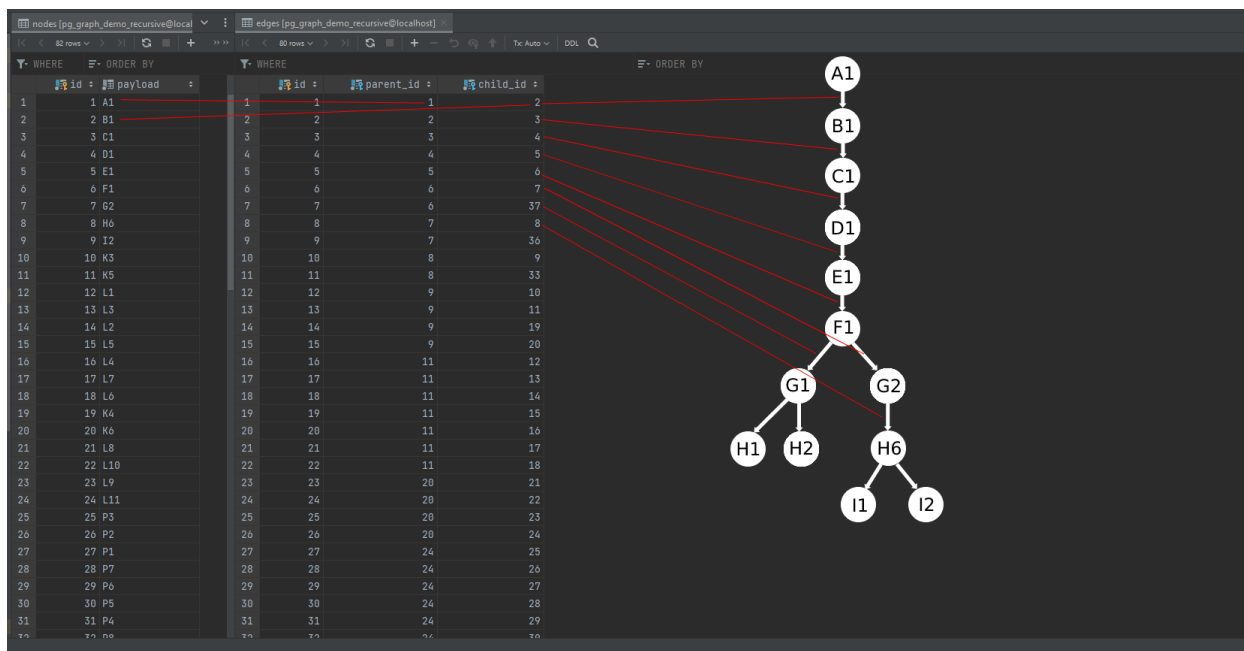
A1->B1->C1->D1->E1->F1->G1->H1
A1->B1->C1->D1->E1->F1->G1->H2
A1->B1->C1->D1->E1->F1->G1->H3
A1->B1->C1->D1->E1->F1->G1->H4
A1->B1->C1->D1->E1->F1->G2->H5
A1->B1->C1->D1->E1->F1->G2->H6->I1->K1
A1->B1->C1->D1->E1->F1->G2->H6->I1->K2
A1->B1->C1->D1->E1->F1->G2->H6->I2->K3
A1->B1->C1->D1->E1->F1->G2->H6->I2->K4
A1->B1->C1->D1->E1->F1->G2->H6->I2->K5->L1
A1->B1->C1->D1->E1->F1->G2->H6->I2->K5->L2

A1->B1->C1->D1->E1->F1->G2->H6->I2->K5->L3
A1->B1->C1->D1->E1->F1->G2->H6->I2->K5->L4
A1->B1->C1->D1->E1->F1->G2->H6->I2->K5->L5
A1->B1->C1->D1->E1->F1->G2->H6->I2->K5->L6
A1->B1->C1->D1->E1->F1->G2->H6->I2->K5->L7
A1->B1->C1->D1->E1->F1->G2->H6->I2->K6->L8
A1->B1->C1->D1->E1->F1->G2->H6->I2->K6->L9
A1->B1->C1->D1->E1->F1->G2->H6->I2->K6->L10
A1->B1->C1->D1->E1->F1->G2->H6->I2->K6->L11->P1
A1->B1->C1->D1->E1->F1->G2->H6->I2->K6->L11->P2
A1->B1->C1->D1->E1->F1->G2->H6->I2->K6->L11->P3
A1->B1->C1->D1->E1->F1->G2->H6->I2->K6->L11->P4
A1->B1->C1->D1->E1->F1->G2->H6->I2->K6->L11->P5
A1->B1->C1->D1->E1->F1->G2->H6->I2->K6->L11->P6
A1->B1->C1->D1->E1->F1->G2->H6->I2->K6->L11->P7
A1->B1->C1->D1->E1->F1->G2->H6->I2->K6->L11->P8
где '[A-Z][0-9]+' - вершина, '->' связь.

В схему базы данных:



Пример хранения:



2.2 Обеспечить доступ по id вершины к ней и ее потомкам

/api/v1/nodes/1

```
{
  "id": 1,
  "payload": "A1",
  "children": [
    {
      "id": 2,
      "payload": "B1",
      "children": [
        {
          "id": 3,
          "payload": "C1",
          "children": [
            {
              "id": 4,
              "payload": "D1",
              "children": [
                {
                  "id": 5,
                  "payload": "E1",
                  "children": [
                    {
                      "id": 6,
                      "payload": "F1",
                      "children": [
                        {
                          "id": 7,
```

```
"payload": "G2",
"children": [
  {
    "id": 8,
    "payload": "H6",
    "children": [
      {
        "id": 9,
        "payload": "I2",
        "children": [
          {
            "id": 10,
            "payload": "K3",
            "children": []
          },
          {
            "id": 11,
            "payload": "K5",
            "children": [
              {
                "id": 12,
                "payload": "L1",
                "children": []
              },
              {
                "id": 13,
                "payload": "L3",
                "children": []
              },
              {
                "id": 14,
                "payload": "L2",
                "children": []
              },
              {
                "id": 15,
                "payload": "L5",
                "children": []
              },
              {
                "id": 16,
                "payload": "L4",
                "children": []
              },
              {
                "id": 17,
                "payload": "L7",
                "children": []
              },
              {
                "id": 18,
                "payload": "L6",
                "children": []
              }
            ]
          }
        ]
      }
    ]
  }
]
```

```

    ]
  },
  {
    "id": 19,
    "payload": "K4",
    "children": []
  },
  {
    "id": 20,
    "payload": "K6",
    "children": [
      {
        "id": 21,
        "payload": "L8",
        "children": []
      },
      {
        "id": 22,
        "payload": "L10",
        "children": []
      },
      {
        "id": 23,
        "payload": "L9",
        "children": []
      },
      {
        "id": 24,
        "payload": "L11",
        "children": [
          {
            "id": 25,
            "payload": "P3",
            "children": []
          },
          {
            "id": 26,
            "payload": "P2",
            "children": []
          },
          {
            "id": 27,
            "payload": "P1",
            "children": []
          },
          {
            "id": 28,
            "payload": "P7",
            "children": []
          },
          {
            "id": 29,
            "payload": "P6",
            "children": []
          }
        ]
      }
    ]
  }
]

```

```

    },
    {
      "id": 30,
      "payload": "P5",
      "children": []
    },
    {
      "id": 31,
      "payload": "P4",
      "children": []
    },
    {
      "id": 32,
      "payload": "P8",
      "children": []
    }
  ]
}
]
}
]
},
{
  "id": 33,
  "payload": "I1",
  "children": [
    {
      "id": 34,
      "payload": "K2",
      "children": []
    },
    {
      "id": 35,
      "payload": "K1",
      "children": []
    }
  ]
}
]
},
{
  "id": 36,
  "payload": "H5",
  "children": []
}
]
},
{
  "id": 37,
  "payload": "G1",
  "children": [
    {
      "id": 38,
      "payload": "H1",

```


Наивное решение запускает череду каскадных запросов, что не обеспечивает условно-константное время отклика.

Так доступ к дереву:

A1->B1->C1->D1->E1->F1->G1->H1->I1->J1->K1

по медиане 7.9 ms, а к дереву вида:

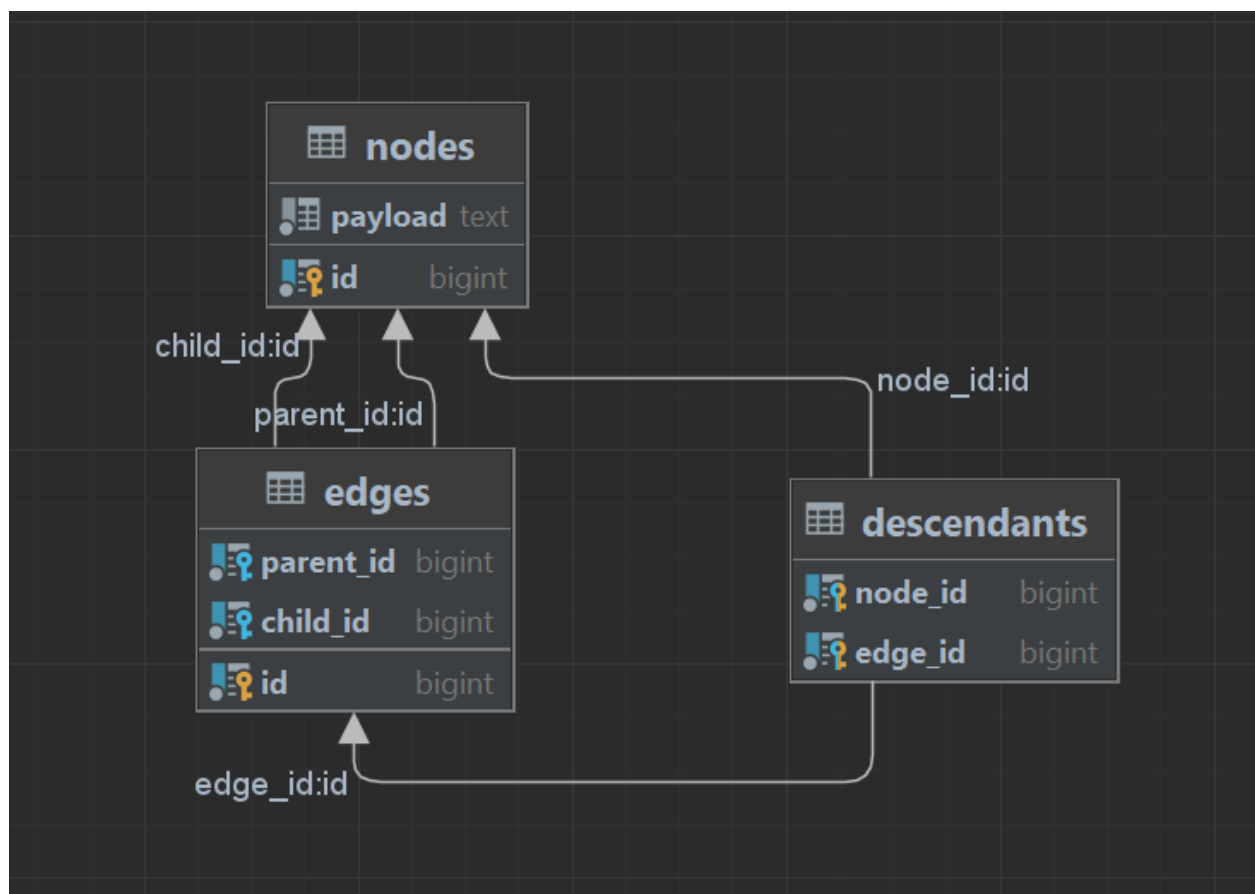
A1->B1->C1->D1->E1->F1->G1->H1->I1->J1->K1->L1->M1->N1->O1->P1->Q1->R1->S1->T1->U1->V1->W1->X1->Y1->Z1

по медиане 16 ms.

Не обеспечена согласованность данных: **возможно нарушить структуру дерева**, т.к. нет проверок на циклы.

4 Решение проблемы

Решением проблемы может послужить новая таблица:



descendants - таблица в которой хранятся все потомки вершины.

Пример денормализации дерева вида: A1->B1->C1

pg_gr...ursive.public.nodes [pg_graph_demo_nonrecursive@localhost]	nonrecursive.NodeEntity.java	edges [pg_graph_demo_nonrecursive@localhost]
3 rows	2 rows	
WHERE	ORDER BY	WHERE
id	payload	id
1	A1	parent_id
2	B1	child_id
3	C1	

descendants [pg_graph_demo_nonrecursive@localhost]	tmp.txt
3 rows	1
WHERE	ORDER BY
node_id	edge_id
1	1
2	1
3	2

A1->B1->C1

The screenshot displays a database interface with three tables and their data:

- nodes** table:

id	payload
1	A1
2	B1
3	C1
- edges** table:

id	parent_id	child_id
1	1	2
2	2	3
- descendants** table:

node_id	edge_id
1	1
2	1
3	2

Handwritten green annotations show the following connections:

- From **nodes** table, row 2 (B1), to **edges** table, row 1 (parent_id 1).
- From **edges** table, row 1 (child_id 2), to **descendants** table, row 2 (node_id 2).
- From **edges** table, row 2 (child_id 3), to **descendants** table, row 3 (node_id 3).

A text box labeled **tmp.txt** contains the path: `A1->B1->C1`.


```
All Vus finished. Total time: 7 minutes, 4 seconds
Summary report @ 19:42:17(+0300)
-----
http_codes.200: ..... 10000
http_request_rate: ..... 24/sec
http_requests: ..... 10000
http_response_time: .....
min: ..... 37
max: ..... 165
median: ..... 41.7
p95: ..... 45.2
p99: ..... 48.9
http_responses: ..... 10000
users.completed: ..... 1
users.created: ..... 1
users.created_by_name.0: ..... 1
users.failed: ..... 0
users.session_length: .....
min: ..... 423609.2
max: ..... 423609.2
median: ..... 420997.3
p95: ..... 420997.3
p99: ..... 420997.3

All Vus finished. Total time: 2 minutes, 21 seconds
Summary report @ 19:37:37(+0300)
-----
http_codes.200: ..... 10000
http_request_rate: ..... 72/sec
http_requests: ..... 10000
http_response_time: .....
min: ..... 12
max: ..... 103
median: ..... 13.1
p95: ..... 15
p99: ..... 15
http_responses: ..... 10000
users.completed: ..... 1
users.created: ..... 1
users.created_by_name.0: ..... 1
users.failed: ..... 0
users.session_length: .....
min: ..... 138883.7
max: ..... 138883.7
median: ..... 140132.7
p95: ..... 140132.7
p99: ..... 140132.7
```

Прирост по чтению:

```
All Vus finished. Total time: 26 seconds
Summary report @ 20:26:16(+0300)
-----
http_codes.200: ..... 1000
http_request_rate: ..... 34/sec
http_requests: ..... 1000
http_response_time: .....
min: ..... 23
max: ..... 28
median: ..... 24.8
p95: ..... 25.8
p99: ..... 26.8
http_responses: ..... 1000
users.completed: ..... 1
users.created: ..... 1
users.created_by_name.0: ..... 1
users.failed: ..... 0
users.session_length: .....
min: ..... 25205.7
max: ..... 25205.7
median: ..... 25091.6
p95: ..... 25091.6
p99: ..... 25091.6

All Vus finished. Total time: 4 seconds
Summary report @ 20:27:28(+0300)
-----
http_codes.200: ..... 1000
http_request_rate: ..... 427/sec
http_requests: ..... 1000
http_response_time: .....
min: ..... 1
max: ..... 4
median: ..... 2
p95: ..... 2
p99: ..... 3
http_responses: ..... 1000
users.completed: ..... 1
users.created: ..... 1
users.created_by_name.0: ..... 1
users.failed: ..... 0
users.session_length: .....
min: ..... 2352.2
max: ..... 2352.2
median: ..... 2369
p95: ..... 2369
p99: ..... 2369
```

Доступ к дереву:

A1->B1->C1->D1->E1->F1->G1->H1->I1->J1->K1

по медиане 2 ms, и к дереву вида:

A1->B1->C1->D1->E1->F1->G1->H1->I1->J1->K1->L1->M1->N1->O1->P1->Q1->R1->S1->T1->U1->V1->W1->X1->Y1->Z1

по медиане 2 ms.

Более того доступ к дереву в тестовом примере, также 2 ms.

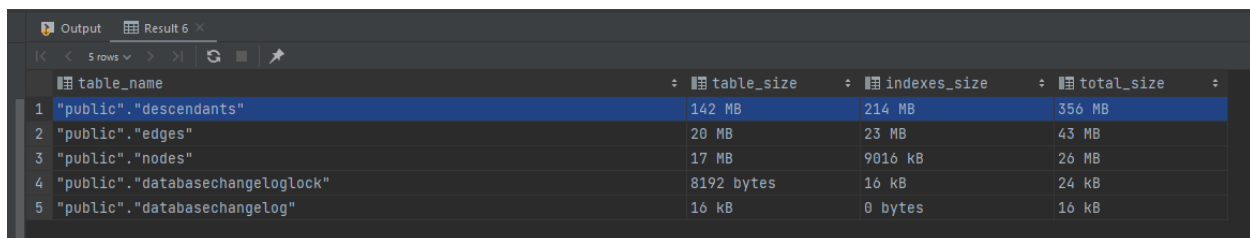
5 Цена решения

```
SELECT
  table_name,
  pg_size_pretty(table_size) AS table_size,
  pg_size_pretty(indexes_size) AS indexes_size,
  pg_size_pretty(total_size) AS total_size
```

```

FROM (
    SELECT
        table_name,
        pg_table_size(table_name) AS table_size,
        pg_indexes_size(table_name) AS indexes_size,
        pg_total_relation_size(table_name) AS total_size
    FROM (
        SELECT ('"' || table_schema || '"."' || table_name || '"') AS table_name
        FROM information_schema.tables
        WHERE table_schema = 'public'
    ) AS all_tables
    ORDER BY total_size DESC
) AS pretty_sizes;

```



The screenshot shows a database interface with a query result. The result is a table with 5 rows and 4 columns: table_name, table_size, indexes_size, and total_size. The rows are ordered by total_size in descending order.

	table_name	table_size	indexes_size	total_size
1	"public"."descendants"	142 MB	214 MB	356 MB
2	"public"."edges"	20 MB	23 MB	43 MB
3	"public"."nodes"	17 MB	9016 kB	26 MB
4	"public"."databasechangelock"	8192 bytes	16 kB	24 kB
5	"public"."databasechangelog"	16 kB	0 bytes	16 kB

В зависимости от глубины дерева оптимизация будет умножать расходы по ПЗУ в разы.

6 Альтернативный способ решения

```

CREATE EXTENSION LTREE;
DROP TABLE edges;
ALTER TABLE nodes ADD COLUMN path LTREE NOT NULL DEFAULT '';
CREATE INDEX ix__nodes__path ON nodes USING gist (path);

```

7 Итоги

