睿创神脑
RUICHUANGSHENNAO

PolarDB
PostgreSQL

PostgreSQL
递归查询

睿创神脑
RUICHUANGSHENNAO

PolarDB PostgreSQL

Objectives

- 语法
- 造数
- 层级查询
- 最短路径查询

睿创神脑
RUICHUANGSHENNAO

递归语法

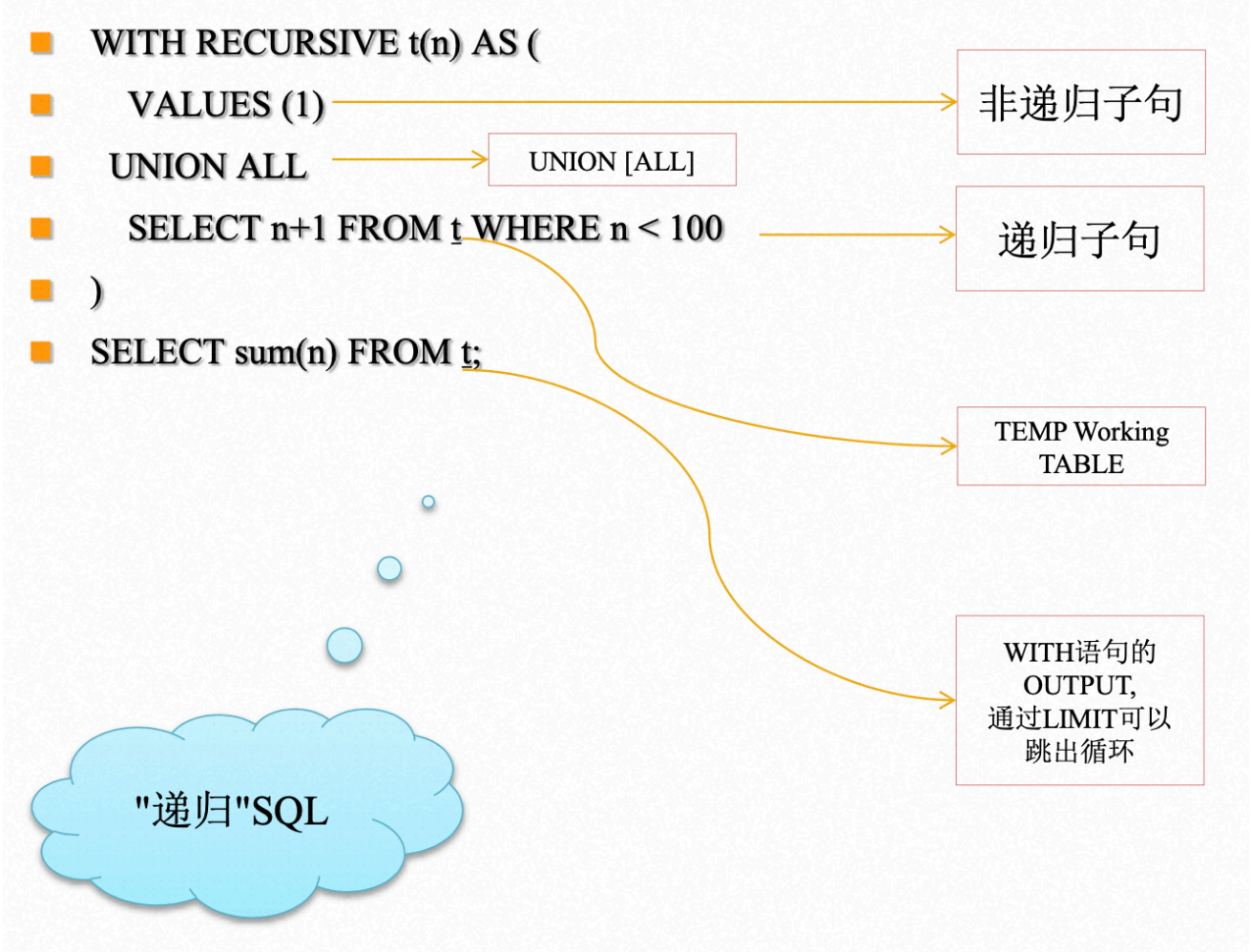
PolarDB PostgreSQL

```
WITH regional_sales AS (
        SELECT region, SUM(amount) AS total_sales
        FROM orders
        GROUP BY region
    ), top_regions AS (
        SELECT region
        FROM regional_sales
        WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales)
    )
SELECT region,
       product,
       SUM(quantity) AS product_units,
       SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions)
GROUP BY region, product;
```

睿创神脑
RUICHUANGSHENNAO

PolarDB PostgreSQL

递归语法

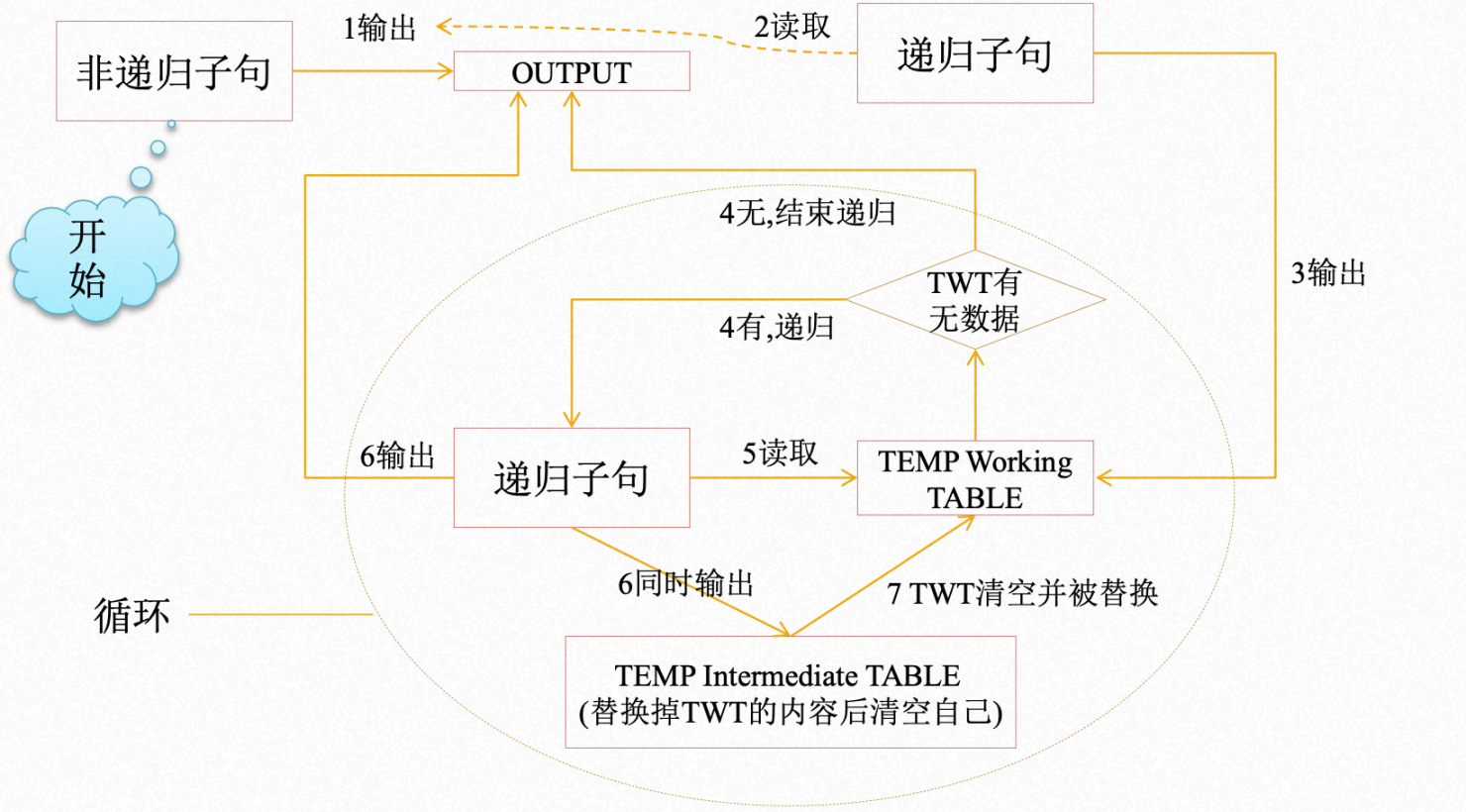


睿创神脑
RUICHUANGSHENNAO

递归语法

PolarDB PostgreSQL

- UNION ALL 去重复(去重复时NULL 视为等同)
- 图中所有输出都涉及UNION [ALL]的操作, 包含以往返回的记录和当前返回的记录

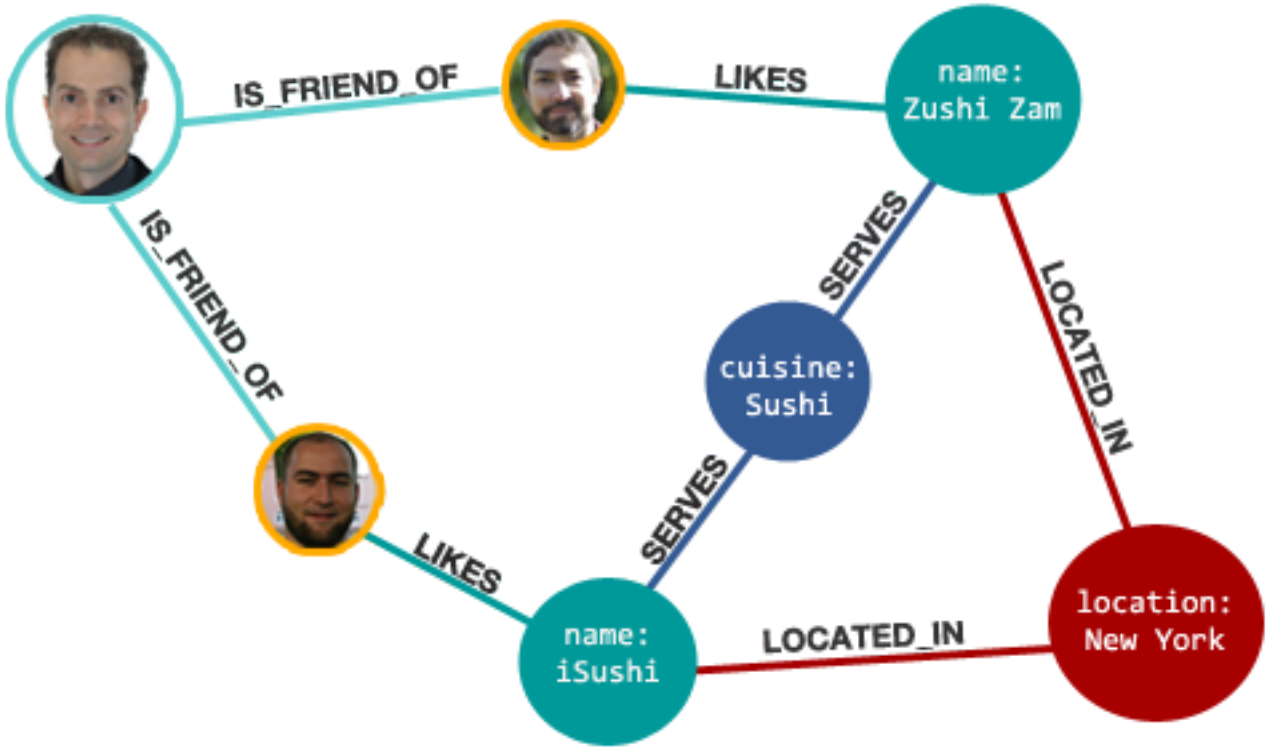


睿创神脑
RUICHUANGSHENNAO

造数

PolarDB PostgreSQL

https://github.com/digoal/blog/blob/master/201801/20180102_04.md



睿创神脑
RUICHUANGSHENNAO

造数

PolarDB PostgreSQL

https://github.com/digoal/blog/blob/master/201801/20180102_04.md

创建1000万用户，每5万作为一个有牵连的群体，平均每个用户牵连500个用户，形成50亿的大规模关系网。

在此基础上，演示如下

1、如何实现N度搜索，边的属性查看，以及最短路径搜索等需求。

2、如何去除循环点，如何控制深度，如何展示路径等。

睿创神脑
RUICHUANGSHENNAO

造数

PolarDB PostgreSQL

- create table a(
- c1 int, -- 点1
- c2 int, -- 点2
- prop jsonb, -- 点1,2对应的边的属性，使用JSON存储，包括权重，关系等等。
- primary key (c1,c2) -- 主键
-);
-
- create index idx_a_2 on a(c1, COALESCE(((prop ->> 'weight'::text))::float8, 0));

睿创神脑
RUICHUANGSHENNAO

造数

PolarDB PostgreSQL

- vi test.sql
-
- \set id random(1,10000000)
- insert into a select :id, ((width_bucket(:id,1,10000000,2000)-1)*50000 + (random()*50000)::int) from generate_series(1,1000) on conflict (c1,c2) do nothing;

- pgbench -M prepared -n -r -P 5 -f ./test.sql -c 50 -j 50 -t 100000

睿创神脑
RUICHUANGSHENNAO

N度搜索

PolarDB PostgreSQL

```
### N度搜索
N度搜索，如上SQL，输入sg.depth <= N。

例如，搜索ROOT=31208的三度数据。

```
WITH RECURSIVE search_graph(
  c1,   -- 点1
  c2,   -- 点2
  prop, -- 边的属性
  depth, -- 深度，从1开始
  path -- 路径，数组存储
) AS (
        SELECT   -- ROOT节点查询
          g.c1,   -- 点1
          g.c2,   -- 点2
          g.prop,   -- 边的属性
          1 as depth,   -- 初始深度=1
          ARRAY[g.c1] as path -- 初始路径
        FROM a AS g
        WHERE
          c1 = 31208   -- ROOT节点=?
      UNION ALL
        SELECT   -- 递归子句
          g.c1,   -- 点1
          g.c2,   -- 点2
          g.prop,   -- 边的属性
          sg.depth + 1 as depth,   -- 深度+1
          path || g.c1 as path   -- 路径中加入新的点
        FROM a AS g, search_graph AS sg   -- 循环 INNER JOIN
        WHERE
          g.c1 = sg.c2   -- 递归JOIN条件
          AND (g.c1 <> ALL(sg.path))   -- 防止循环
          AND sg.depth <= 3   -- 搜索深度=?
)
SELECT * FROM search_graph;   -- 查询递归表，可以加LIMIT输出，也可以使用游标
```
```

睿创神脑
RUICHUANGSHENNAO

最短路径

PolarDB PostgreSQL

例如搜索 1 到 100 的最短路径。

```
WITH RECURSIVE search_graph(
  c1,    -- 点1
  c2,    -- 点2
  prop, -- 边的属性
  depth, -- 深度, 从1开始
  path  -- 路径, 数组存储
) AS (
        SELECT     -- ROOT节点查询
          g.c1,    -- 点1
          g.c2,    -- 点2
          g.prop,    -- 边的属性
          1 as depth,         -- 初始深度=1
          ARRAY[g.c1] as path  -- 初始路径
        FROM a AS g
        WHERE
          c1 = 1          -- ROOT节点=?
      UNION ALL
        SELECT      -- 递归子句
          g.c1,     -- 点1
          g.c2,     -- 点2
          g.prop,           -- 边的属性
          sg.depth + 1 as depth,     -- 深度+1
          path || g.c1 as path    -- 路径中加入新的点
        FROM a AS g, search_graph AS sg   -- 循环 INNER JOIN
        WHERE
          g.c1 = sg.c2          -- 递归JOIN条件
          AND (g.c1 <> ALL(sg.path))          -- 防止循环
--          AND sg.depth <= ?     -- 搜索深度=? 也可以保留, 防止搜索太深影响性能, 比如深入10以后就不返回了
)
SELECT * FROM search_graph
  where c2 = 100    -- 最短路径的终点
  limit 1;        -- 查询递归表, 可以加LIMIT输出, 也可以使用游标
```

如果要控制深度, 比如5度以内搜不到就不搜了, 把搜索深度的条件再加进去即可。

睿创神脑
RUICHUANGSHENNAO

总结

PolarDB PostgreSQL

- 语法
- 造数
- 层级查询
- 最短路径查询

睿创神脑
RUICHUANGSHENNAO

练习

PolarDB PostgreSQL

- 1、设计一张表用来存储传感器上报的日志数据, 字段包括传感器ID, 上报时间, 上报数据(JSON字段存储).
- 2、写入1000个传感器的测试数据1440条, 总共144万条记录.
- 3、使用一般方法查询每个传感器的最新数据.
- 4、使用递归查询每个传感器的最新数据.
- 5、对比这个CASE中, 使用递归查询的性能相比一般方法的提升, 思考原因是什么?

睿创神脑
RUICHUANGSHENNAO

PolarDB PostgreSQL