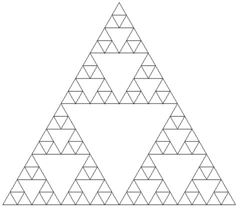
SQL Recursion, Window Queries

PG 7.8; 3.5 & 9.21

Dr. Chris Mayfield

Department of Computer Science
James Madison University

Apr 05, 2022



WITH clause

Basic syntax:
`WITH` R `AS` <definition of R> <query involving R>

For example:

- `Flights(airline, src, dst, departs, arrives)`

```
WITH den_flights AS (
  SELECT * FROM Flights
  WHERE src = 'DEN'
)
SELECT * FROM den_flights
ORDER BY departs;
```

Common table expressions

Define temporary tables that exist for one query

- WITH can involve SELECT, INSERT, UPDATE, or DELETE
- Can be attached to SELECT, INSERT, UPDATE, or DELETE

For example:

```
WITH moved_rows AS (
  DELETE FROM products
  WHERE "date" >= '2010-10-01'
    AND "date" <  '2010-11-01'
  RETURNING *
)
INSERT INTO products_log
SELECT * FROM moved_rows;
```

https://www.postgresql.org/docs/11/queries-with.html

Famous mathematician

Paul Erdős (1913–1996)

One of the most prolific mathematicians of the 20th century

- More than 1500 articles
- Over 500 collaborators
- *The Oddball's Oddball*

Tribute: Erdős number

https://en.wikipedia.org/wiki/Paul_Erd%C5%91s

Erdős numbers

```
WITH e1 AS (
  -- Erdos number is 1
  SELECT DISTINCT b.author
  FROM auth AS a
    -- same paper, but different author
    JOIN auth AS b ON a.dblp_key = b.dblp_key
                   AND a.author != b.author
  WHERE a.author = 'Paul Erdös'
)
-- Erdos number is 2
SELECT DISTINCT d.author
FROM e1
  -- first get all papers of e1 authors
  JOIN auth AS c ON e1.author = c.author
  -- same paper, but different author
  JOIN auth AS d ON c.dblp_key = d.dblp_key
                 AND c.author != d.author
-- excluding e0 and e1
WHERE d.author != 'Paul Erdös'
  AND d.author NOT IN (SELECT author FROM e1);
```

Recursive queries using CTE’s

Recursive relations in SQL

`RECURSIVE` modifer allows `WITH` queries to refer to their own output

```sql
-- Result is 5050
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

General form:

1. Non-recursive term
2. `UNION` or `UNION ALL`
3. Recursive term

Recursive query evaluation

1. Evaluate the non-recursive term
   - Include all rows in the query result
   - If `UNION`, eliminate duplicate rows
   - Also place them in a working table
2. While the working table is not empty
   - Evaluate the recursive term using working table
   - If `UNION`, eliminate duplicates of any previous row
   - Add rows to result and create intermediate table
   - Replace working table with the intermediate table

Strictly speaking, this process is iteration not recursion!

Recursive flight example

```
-- transitive closure of flights
WITH RECURSIVE Reaches(src, dst) AS
    SELECT src, dst
    FROM Flights
  UNION
    SELECT R1.src, R2.dst
    FROM Reaches AS R1, Reaches AS R2
    WHERE R1.dst = R2.src
)
-- all cities reachable from Denver
SELECT dst FROM Reaches
WHERE src = 'DEN';
```

More complex example using `depth` and `path`:
https://www.postgresql.org/docs/11/queries-with.html

More Advanced SQL

Analytical queries and Window functions

Analytical queries

Calculate a running total

- Show the cumulative salary within a department row by row

Find percentages within a group

- Show the percentage of the total salary paid to an individual

Compute a moving average

- Average the current row's value with the previous N rows

Perform ranking queries

- Show the relative rank of each salary within a department

Top-N queries

- Find the top *n* sales by region

Window functions

Perform a calculation across *related rows*

- Partition: which rows are related
- Order: how to sort each partition

Example:

```
-- sort by salary in each dept
SELECT depname, empno, salary,
  rank() OVER (PARTITION BY depname ORDER BY salary DESC)
FROM empsalary;
```

Window functions only allowed in `SELECT` and `ORDER BY` clauses

- Defined over output of `FROM`, `WHERE`, `GROUP BY`, and `HAVING`

Example OVER clauses

```
-- average salary in each dept
SELECT depname, empno, salary,
  avg(salary) OVER (PARTITION BY depname)
FROM empsalary;

-- running total of salaries
SELECT depname, empno, salary,
  sum(salary) OVER (ORDER BY salary)
FROM empsalary;

-- GROUP BY without grouping
SELECT depname, empno, salary,
  sum(salary) OVER ()
FROM empsalary;
```

https://www.postgresql.org/docs/11/tutorial-window.html

Other window functions

```
SELECT depname, empno, salary,
  sum(salary) OVER w, -- and other aggregate functions
  row_number() OVER w, -- from 1 to number of rows in w
  rank() OVER w, -- rows with same value get same rank
FROM empsalary
WINDOW w AS (PARTITION BY depname ORDER BY salary DESC);
```

- There are many more options for `OVER` clauses
  - https://www.postgresql.org/docs/11/sql-expressions.html#SYNTAX-WINDOW-FUNCTIONS
- List of general-purpose window functions
  - https://www.postgresql.org/docs/11/functions-window.html