RECURSIVE SQL

2 DEMOS

- Factorial "table" - not a real-world use-case, but easy
- Optimal mass transport transfers (simple version of imhd.sk)
  - real-world, but hard(er)

BREAKING NEWS: THERE IS NO RECURSION IN SQL

RECURSIVE SQL = ITERATIVE SQL

- Traditional recursion:
  - Start with unknown, back down until known value
- ~~Recursive~~ Iterative SQL
  - Start with known, build up the answer
  - Get ready for weird syntax

DEMO: FACTORIAL

```
with recursive fact(n, fact) as (
  select 0, 1
  union all
  select f.n+1, f.fact * (f.n+1) from fact f
)
select * from fact limit 10
```

#1

```
with recursive fact(n, fact) as (
  select 0, 1
  union all
  select f.n+1, f.fact * (f.n+1) from fact f
)
select * from fact limit 10
```

WORKING TABLE

WORKING TABLE / NEXT

0 1

OUTPUT TABLE

0 1

#2

```
with recursive fact(n, fact) as (
  select 0, 1
  union all
  select f.n+1, f.fact * (f.n+1) from fact f
)
select * from fact limit 10
```

WORKING TABLE

0 1

WORKING TABLE / NEXT

1 1

OUTPUT TABLE

0 1

1 1

#3

```
with recursive fact(n, fact) as (
  select 0, 1
  union all
  select f.n+1, f.fact * (f.n+1) from fact f
)
select * from fact limit 10
```

WORKING TABLE

1 1

WORKING TABLE / NEXT

2 2

OUTPUT TABLE

0 1
1 1
2 2

#4

```
with recursive fact(n, fact) as (
  select 0, 1
  union all
  select f.n+1, f.fact * (f.n+1) from fact f
)
select * from fact limit 10
```

WORKING TABLE

2	2

WORKING TABLE / NEXT

3	6

OUTPUT TABLE

0	1
1	1
2	2
3	6

UNION / UNION ALL

- Union = Append to output table
- Union all = Add to output table but avoid duplicate rows

DEMO: IMHD

```
with recursive hops(start_name, line, end_name, end_stop_id,
stops_visited, lines_used, duration) as (
  select s.name, c.line, e.name, e.id, array[s.name, e.name],
array[c.line], c.duration
    from connections c
    join stops s on c.start_stop_id = s.id
    join stops e on c.end_stop_id = e.id
   where s.name = 'Nad lúčkami'
     and c.line not like 'N%'

  union

  select h.end_name, c.line, e.name, e.id, h.stops_visited || e.name,
         case when c.line = h.line then h.lines_used else h.lines_used ||
c.line end, h.duration + c.duration
    from hops h
    join connections c on h.end_stop_id = c.start_stop_id
    join stops e on c.end_stop_id = e.id
    where c.line not like 'N%'
      and (c.line = h.line or not (c.line = any(h.lines_used)))
      and not (e.name = any(h.stops_visited))
      and h.end_name != 'Zochova'
      and array_length(h.lines_used, 1) <= 2
)
select h.stops_visited, h.lines_used, h.duration from hops h
where h.end_name = 'Zochova'
order by array_length(h.lines_used, 1) asc
```

BUT WHY?

- Recursive query
  - algorithm + data are collocated = fast processing
- Recursive algorithm pulling data from the database
  - need to pull data from db to app = slow(er) processing
- Graph database (e.g. Neo4j)
  - may be an overkill
  - need to synchronize data

WHAT YOU SHOULD NOW

- Write a simple recursive query ("1-2 cups of coffee"-level in labs)
- Understand when to use recursive SQL and when to use a different approach