


# **INTRODUCTION TO RELATIONAL DATABASE SYSTEMS**

## **DATENBANKSYSTEME 1 (INF 3131)**

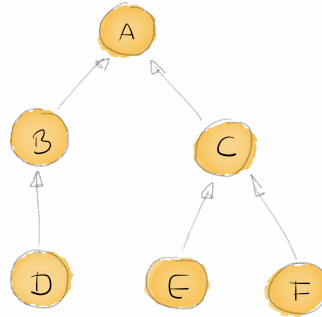
**Torsten Grust**  
**Universität Tübingen**  
Winter 2017/18

# LIMITS OF THE RELATIONAL ALGEBRA

- RA is **not a full-fledged programming language** by design:
  1. The **evaluation of any RA query will terminate**, regardless of the size/contents of the input relations.  
(*“RA does not loop forever”*)
  2. The **evaluation of any RA query will use a restricted amount of time and space only**.  
(*“RA can be evaluated efficiently”*)
    - An RA query that produces intermediate result relations of arity  $k$  can be evaluated in  $O(|D|^k)$  (where  $|D|$  denotes the database size).
    -  A problem like *“Compute all tuple subsets of relation  $R$ ”* thus is inexpressible in RA.
- Yet, there are a number of common, not too-far-fetched types of complex query problems that we would like a DBMS to compute.
  - $\Rightarrow$  Need **a query construct beyond RA** to tackle such problems.

# RECURSIVE QUERIES

- Recall the relational representation of **tree-shaped** data structures:



tree

node	parent
A	□
B	A
C	A
D	B
E	C
F	C

- Typical and useful **tree queries** are out of reach for RA:
  1. “Find all nodes under subtree root *C*.”
  2. “Find all nodes on the path from node *D* to the root.”

# RECURSIVE QUERIES

Find all nodes under subtree root  $\langle root \rangle$

tree

node	parent
A	$\square$
B	A
C	A
D	B
E	C
F	C

```
r := { (root:⟨root⟩) } # r will hold the result
t := r
while t ≠ ∅ do
  t := π[root←node](t ⋈[root = parent] tree)
  r := r ∪ t
end

return π[node←root](r) # “cosmetics”
```

# A RECURSIVE QUERY TEMPLATE

- The recursion pattern we have just seen turns out to be quite generally useful. Abstract over specific query parts to obtain a **iterative/recursive query template**:

```
iterate( $q$ ,  $r$ ):  
   $t := r$   
  while  $t \neq \emptyset$  do  
     $t := q(t)$   
     $r := r \uplus t$   
  end  
  return  $r$ 
```


```
recurse( $q$ ,  $r$ ):  
  if  $r \neq \emptyset$  then  
    return  $r \uplus \text{recurse}(q, q(r))$   
  else  
    return  $\emptyset$   
  end
```

# SQL: RECURSIVE QUERIES

- This iterative RA query template also is available in modern SQL dialects (since the SQL:1999 standard), expressed in terms of a **recursive common table expression**:

## WITH RECURSIVE (Recursive Common Table Expression)

```
WITH RECURSIVE
  <query_name>[ ( <column_name> [, ...] ) ] AS (
    <non-recursive SFW>      -- base case
    UNION ALL
    <recursive SFW>        -- may refer to <query_name>
  )
<final SFW>                -- may refer to <query_name>
```

-  In the <recursive SFW> block of a CTE, the following SQL constructs are ruled out:
  - **OUTER JOINs**, subqueries referring to <query\_name>, **GROUP BY**, aggregate functions, **ORDER BY**, **LIMIT/OFFSET**.

# SQL: RECURSIVE QUERIES

```
WITH RECURSIVE
  <query_name>[ ( <column_name> [, ...] ) ] AS (
    <non-recursive SFW>      -- base case
    UNION ALL
    <recursive SFW>        -- may refer to <query_name>
  )
<final SFW>                -- may refer to <query_name>
```

## - Semantics:

```
r := <non-recursive SFW>
t := r
while t ≠ ∅ do
  t := <recursive SFW>(t)
  r := r ⊔ t # ⊔: bag union
end
return <final SFW>(r)
```

# SQL: RECURSIVE QUERIES (SET SEMANTICS)

```
WITH RECURSIVE
  <query_name>[ ( <column_name> [, ...] ) ] AS (
    <non-recursive SFW>
    UNION [ DISTINCT ]      -- no UNION ALL: set semantics
    <recursive SFW>
  )
<final SFW>
```

- Set semantics

(avoids to add tuples to intermediate result  $r$  that have already been seen):

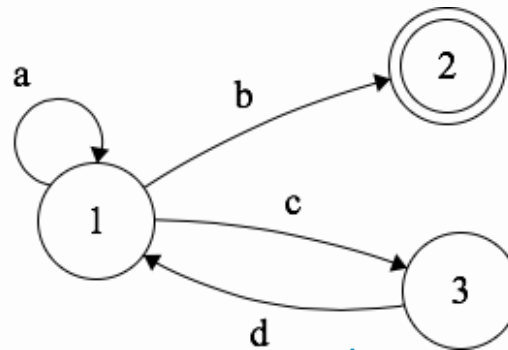
```
 $r := \langle \text{non-recursive SFW} \rangle$ 
 $t := r$ 
while  $t \neq \emptyset$  do
   $t := \text{DISTINCT}(\langle \text{recursive SFW} \rangle(t)) \setminus r$ 
   $r := r \cup t$  #  $\cup$ : disjoint union
end
return  $\langle \text{final SFW} \rangle(r)$ 
```



# SQL: RECURSIVE QUERIES

## Example: Run a Deterministic Finite State Automaton (DFA)

1. Encode the transition table of a DFA as a regular relational table.
2. Formulate a recursive common table expression that consumes/matches one character of input in each iteration.

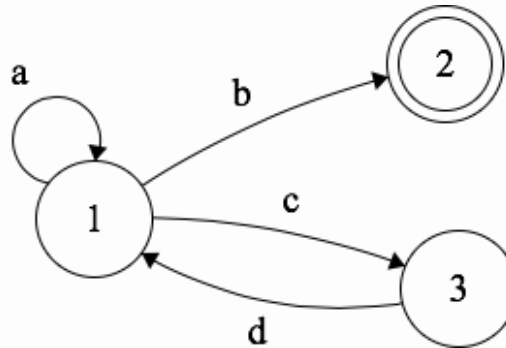


DFA for regular expression  $(a|cd)^*b$ , start state ①

- Note:
  - PostgreSQL:  
Extract first  $n$  characters of string  $s$ : `left(s,n)`; all but first character: `right(s,-1)`.

# SQL: RECURSIVE QUERIES

Example: Run a Deterministic Finite State Automaton (DFA)



fsm

source	trans	target	final
1	a	1	false
1	b	2	false
1	c	3	false
2	EOT	□	true
3	d	1	false

- Column **final**: Is the source state of this DFA edge an accepting state?
- **Quiz**: What would be the correct key for table **fsm**?

# SQL: RECURSIVE QUERIES

## Example: Iteratively Collapse Adjacent Time Intervals

calendar

appointment	start	stop
meeting	11:30	12:00
lunch	12:00	13:00
biking	18:30	□

attendees

appointment	person
meeting	Alex
meeting	Bert
meeting	Cora
lunch	Bert
lunch	Drew

- Recall that `calendar` ⋈ `attendees` can answer the “*Who is busy at at what times?*” question but that we end up with unpleasant **adjacent time intervals** (see `Bert`).
- Can we collapse/merge adjacent time intervals?

# SQL: RECURSIVE QUERIES

## Example: Iteratively Collapse Adjacent Time Intervals

busy

appointment	start	stop	person
meeting	11:30:00	12:00:00	Alex
meeting	11:30:00	12:00:00	Cora
meeting	11:30:00	12:00:00	Bert
lunch	12:00:00	13:00:00	Bert
lunch	12:00:00	13:00:00	Drew

- Helpful PostgreSQL built-in types and operators:
  - `tsrange(t1, t2)`: Construct the time range with start time `t1`, end time `t2`
  - `r1 -|- r2`: Are time ranges `r1`, `r2` directly adjacent?
  - `r1 @> r2`: Does time range `r1` cover `r2`? (also true for `r1 = r2`)
  - `least(x,y)`, `greatest(x,y)`: return the minimum/maximum of `x`, `y`