Managing recursive, tree-like data structures with **Firebird**

Frank Ingermann

Welcome to this session !

…say Sparkies I and III

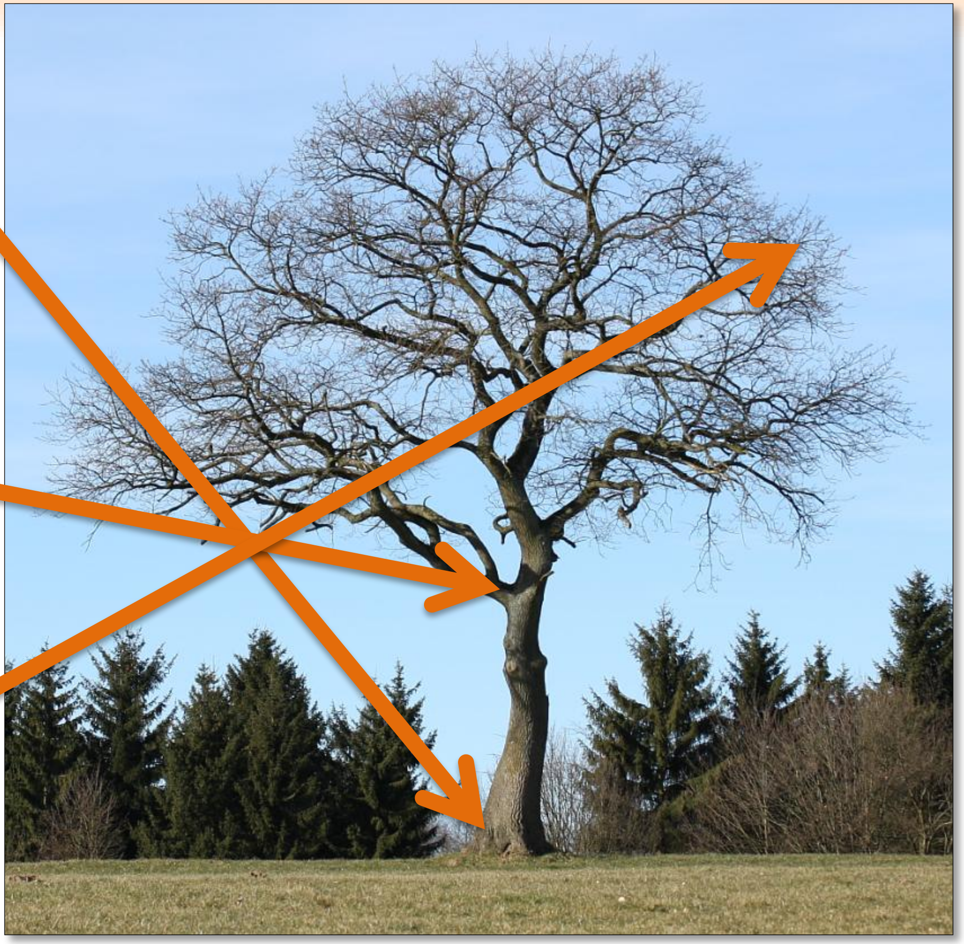
This session is about

Session overview

- **Short intro to Trees in DBs**
- Part 1: **Recursive StoredProcs**
- Part 2: **Nested Sets**
- Part 3: **Recursive CTEs**
- Part 4: **„real-*world*“ examples**

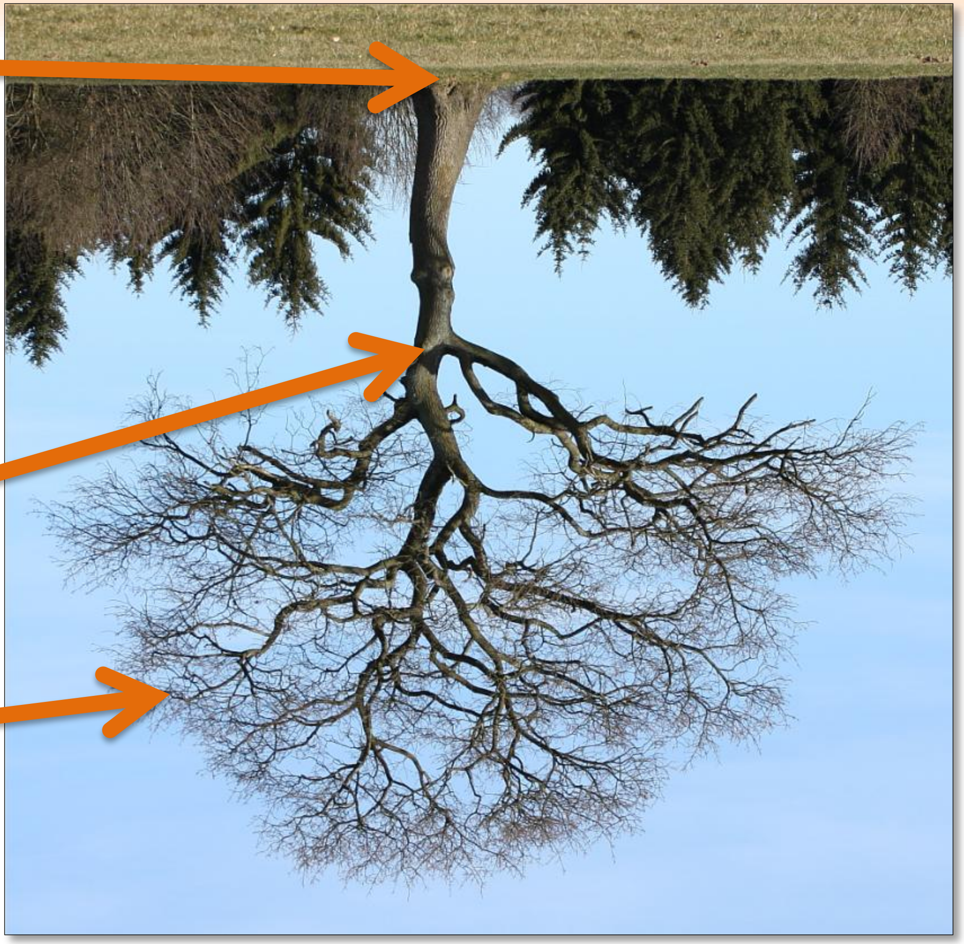
What is a tree?

- It has a single **Root**
- It has *forks* or *branches* (**Nodes**)
- Branches end up in **Leafs**
  (most of the time…)



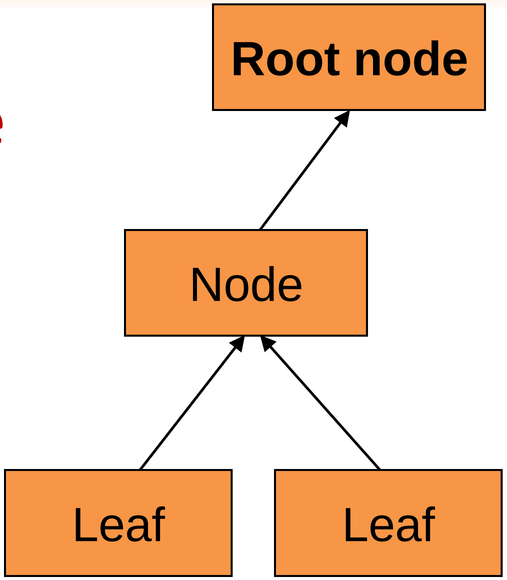
What is a tree?

- It has a <u>single</u> **Root**
- It has *forks* or *branches* (**Nodes**)
- Branches end up in **Leafs**
  (most of the time…)



Tree terms: Root, Nodes, Leafs

- **ROOT node**
  - „upper end“, has **no** parent node
- **NODE(s)**
  - Can have 0..1 **PARENT** node
  - Can have 0..**n** **CHILD** nodes
- **LEAF node(s)**
  - A node with **no** child nodes („lower end“)



- Leafs and nodes can have ***siblings***
  (same parent node = „brothers/sisters“)

Relations of nodes in trees

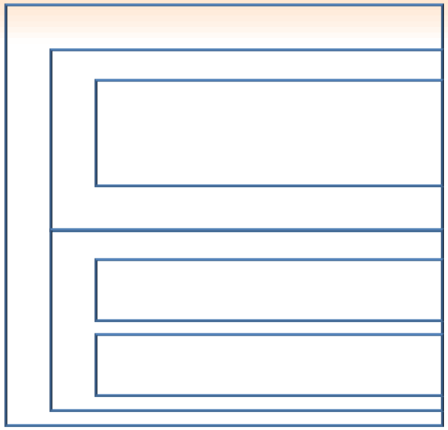
- **Owner** or **Containing** relation

  e.g. ***File System***:

  – each ***file*** is „**owned**" by the ***directory*** it's in

  – each ***file*** can only be in ***one directory***

  – deleting the ***directory*** deletes all ***files*** in it



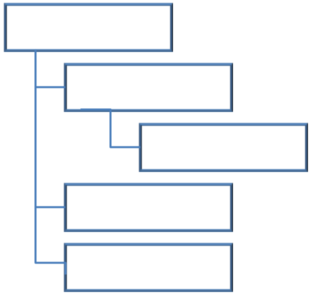
- **Referencing** relation *(links)*

  e.g. ***Recipe Database***:

  – each ***recipe*** can ***reference*** 0..n ***sub-recipes***

  – One ***sub-recipe*** can be referenced by **many** ***master recipes***

  – deleting a ***master recipe*** will **not** delete its ***sub-recipes***



- *A* ***node*** *can* ***reference*** *a* ***node*** *in* ***another tree***

Tree types

- ***„homogeneous“*** trees:

  all nodes: **same** type

  (SQL**: all node data comes from **one table)

- ***„heterogeneous“*** trees:

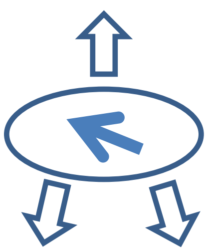
  nodes can have **different** data- or record types

  (SQL**: data can come from **various tables)

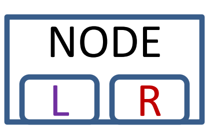
Strategies for storing trees

- **Store a *Parent ref. (PK/ID)* in each node/leaf**
  – Classic approach for N-trees *(each **child** knows it's **parent**)*
  – „unlimited" number of **children** for each **parent**

- **Store *all Child refs* (PKs) in *each parent* node**
  – Limited number of children *(one field for each Child ref.)*
  – good for ***binary search trees, B-trees***



- **Store *relations* of nodes in a *separate table***
  – Most flexible, but requires JOINs in each SELECT
  – allows „heterogeneous" trees
  – separates ***STRUCTURE*** from ***CONTENT (!!!)***



- **Store *„hints for traversal"* in nodes**
  – Does **not** use PKs or IDs at all (!) -> *nested sets*

Retrieving Trees from a DB

- ***Client-Side*** **recursion**
  - SELECT parent node
    - SELECT its child nodes one by one
      - For each child node: SELECT *its* child nodes one by one…
        - » For each child node: SELECT ***its*** child nodes one by one…
- ***Server-side*** **recursion**
  - Recursive Stored Procedures
  - Recursive CTEs
  - **entire tree** is returned by a ***single statement***
- ***„Neither-side“*** **recursion: Nested Sets**

Pros of Client-Side recursion

- **Client has full control**
  - **What** and **How** is traversed
  - **When to stop** traversal
  - Can **change** the **„What and How“** and **„When to stop“** *anytime* during traversal

like using a debugger in single-step mode

usually

Why we **don‘t want** *client-side* rec.:

a) SLOW *b) EXPENSIVE*

- Many ***Prepares*** on **Server side**
  (calculating plans etc. costs **Server time**)
- Many ***round-trips*** across the **network**
  (each TO-AND-FRO takes time!)
- Can not retrieve tree structures as simple,
  „flat“ result sets in „one go“
  (*client* cares about CONTENT, *server* about STRUCTURE)

Recursive

Stored

Procedures

Stored Procedures

- Can call *other* Stored Procedures

  (including ***themselves***)

- „**Direct**“ recursion:

  a procedure *directly* calls ***itself***

- „**Indirect**“ recursion:

  procedure **A** calls procedure **B**

  procedure **B** *recursively* calls **A**

Traversing trees with Selectable SPs

Recursive ***Top-Down*** SP outline:

- **SELECT parent** node‘s data, **SUSPEND**
- **FOR SELECT** <each **child** node of **parent**>:
  - **FOR SELECT** from „**self**“ SP with the current ***child*** as the new ***parent*** node, **SUSPEND**

Recursive SPs: Pros and Cons

- **Pros:**
  - Recursion on **Server** side, **few round-trips**
  - ***PRETTY FAST*** (pre-compiled to BLR)
  - Can handle ***all sorts of trees*** in ***all sorts of ways***
  - Full access to ***all PSQL*** features (!)
- **Cons:**
  - **Unflexible** (part of the DB‘s **metadata!**)
  - Client has **little control** and **no „insight“**

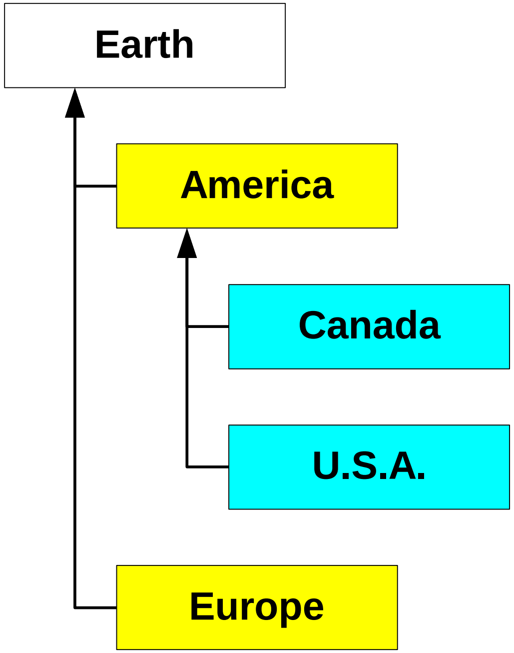
    (a SP is like a „black box, set in concrete“)
  - Can be **hard to maintain/change,** need **GRANTs**

Part 2

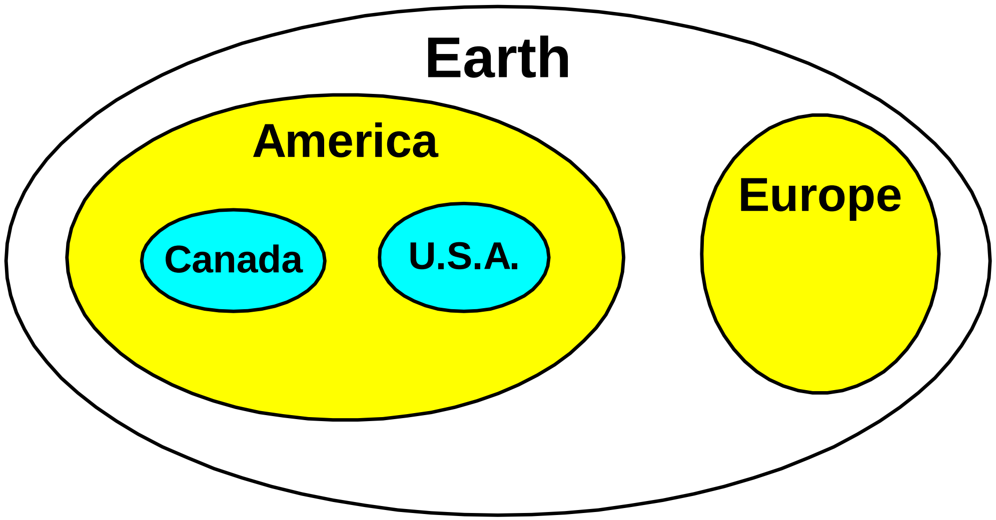
Nested Sets

Nested Sets: Intro

„classical“ tree:



same data as ***Nested Sets***:

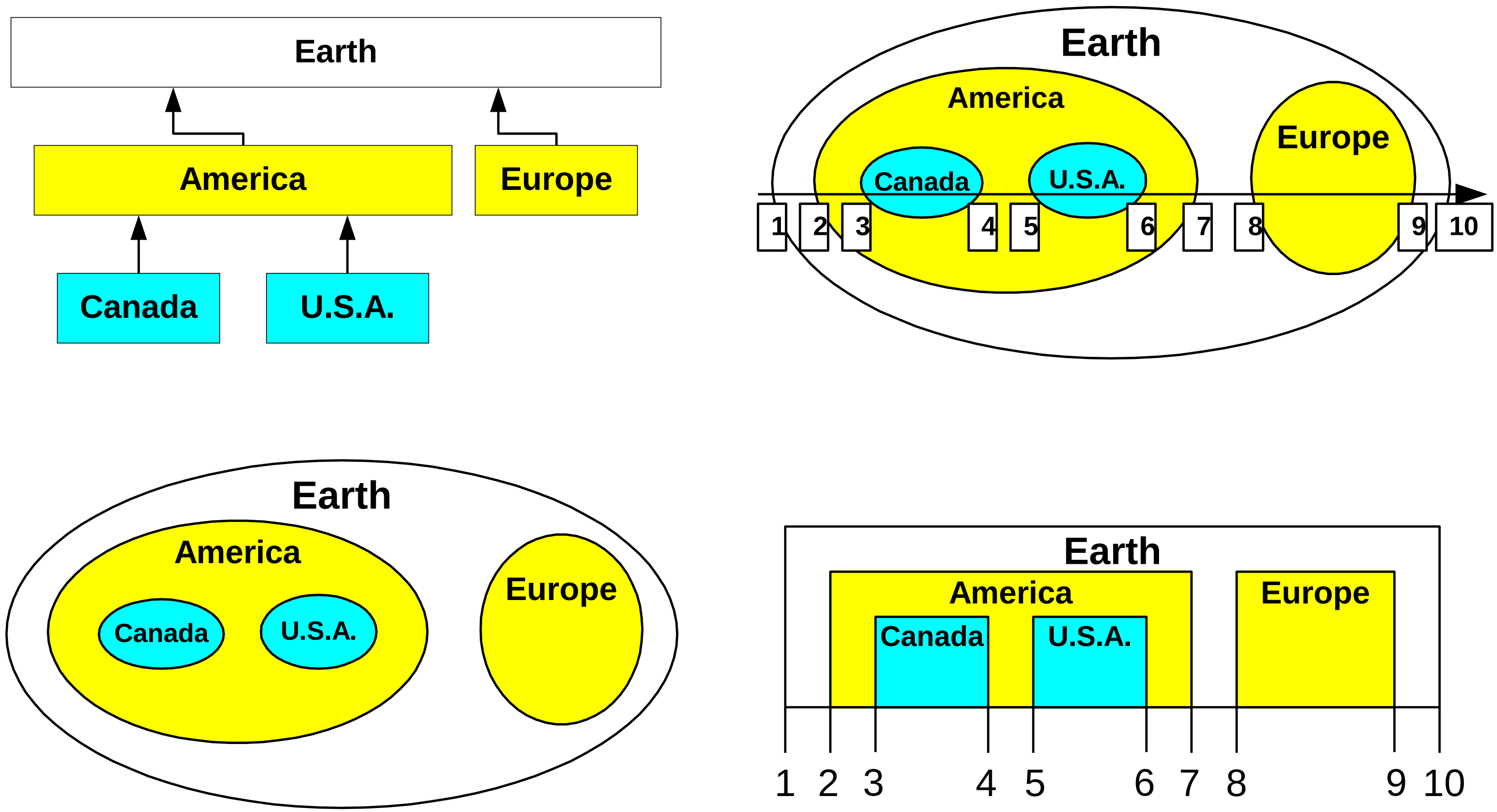


***Nested Sets** are all about*

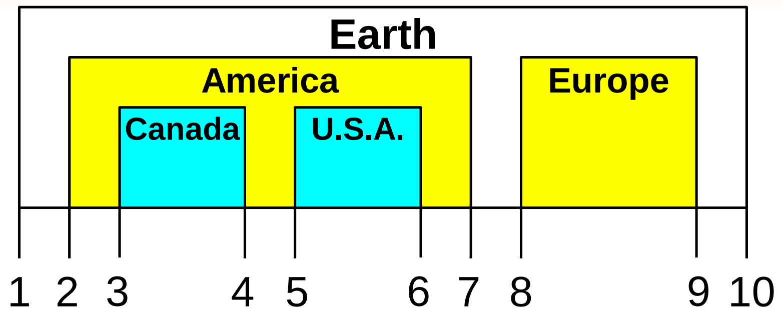
Containment !

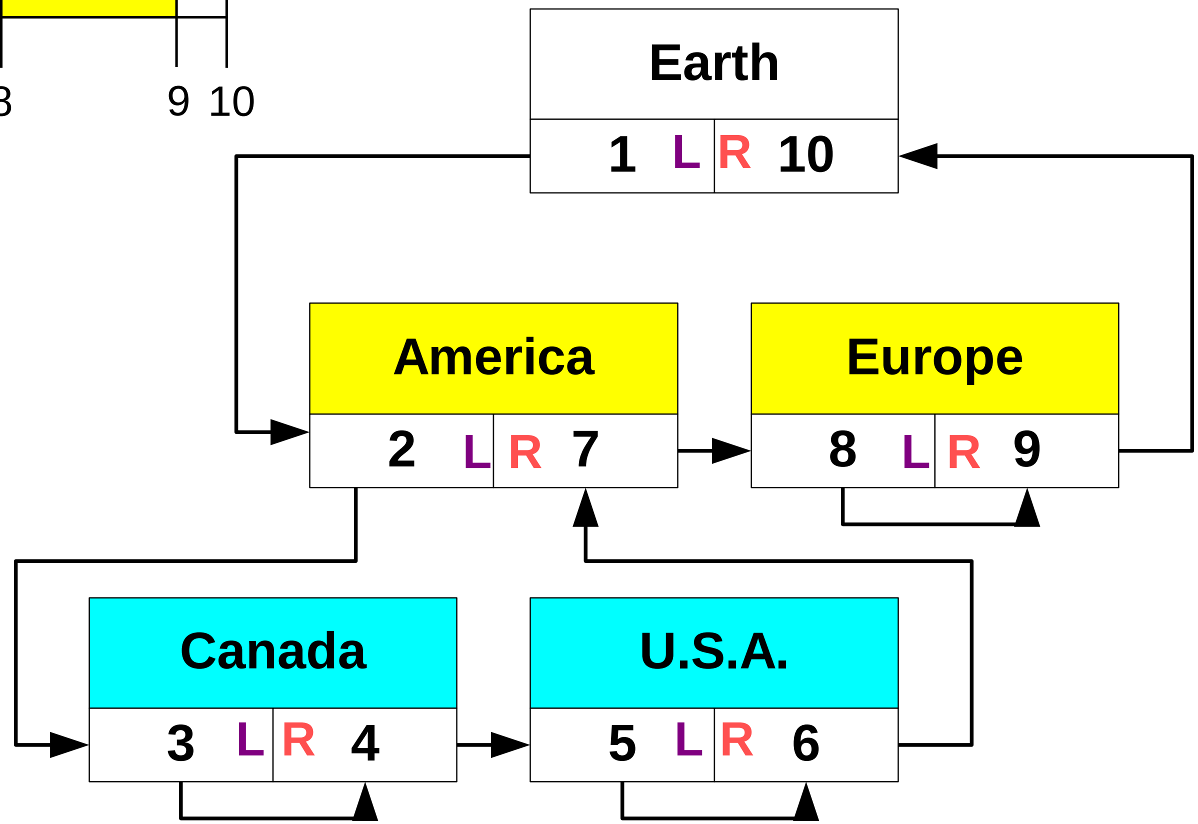
…and NO, this slide is NOT about fried eggs!

Nested Sets: different views

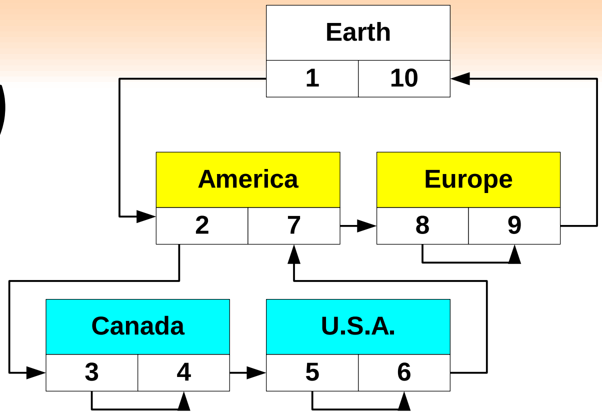


Nested Sets: L and R values





Nested Sets: Rules for L and R

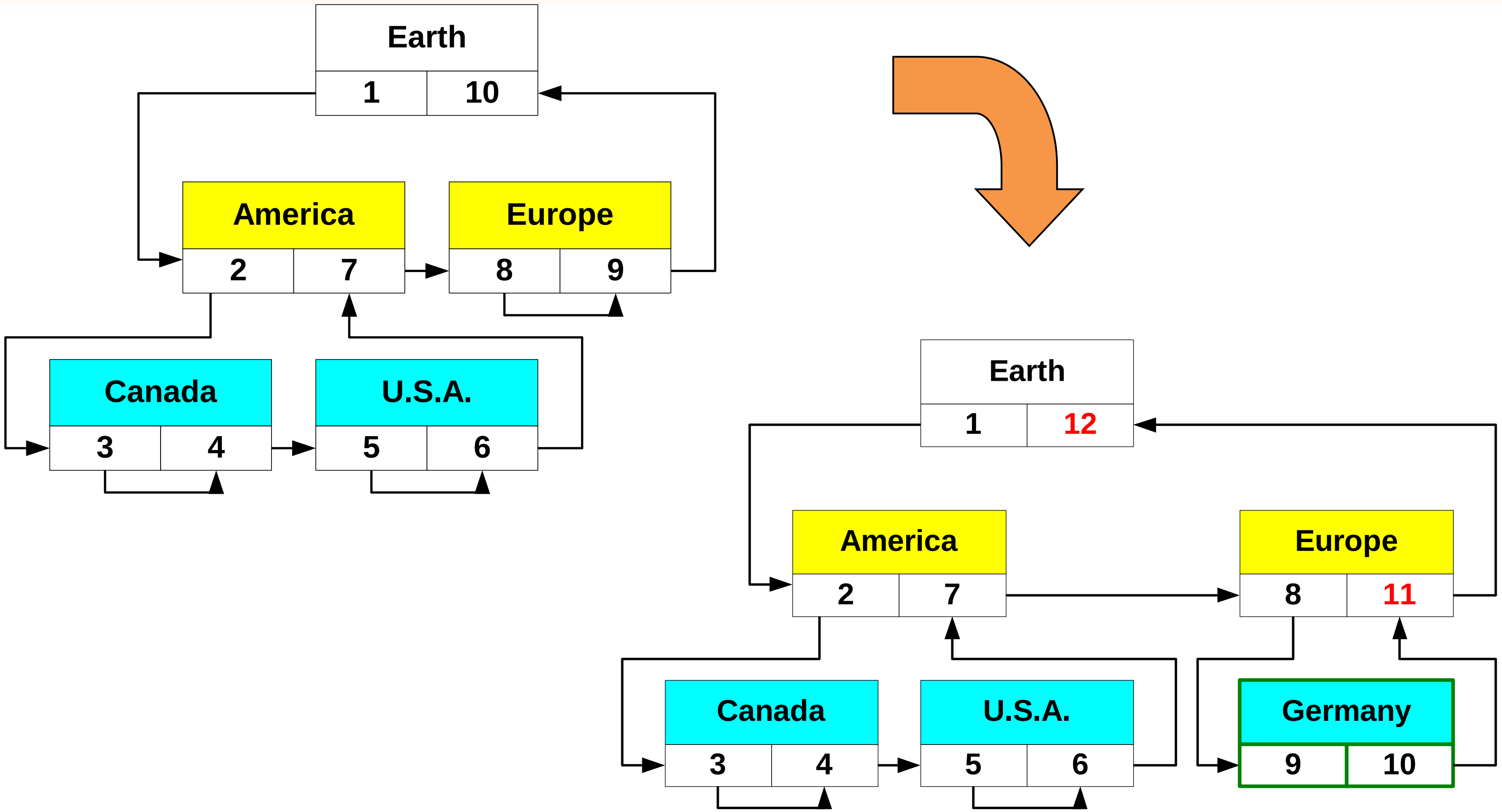


- L value of **ROOT == 1** *(ex def.)*
- L < R (for **all** nodes)
- L of each **parent** node < L of all it's **children**
- R of each **parent** node > R of all it's **children**
- **L == R – 1** for all **Leaf** nodes *if R=L+1: it has no childs!*
- Number of **Child** nodes **==** (**R** – **L** - 1) / 2

Nested Sets: Storage in DB

Name	*L*	*R*	*(R - L - 1) / 2*
Earth	1	10	4
America	2	7	2
Canada	3	4	0
U.S.A.	5	6	0
Europe	8	9	0

INSERTs in Nested Sets



Nested Sets: Pros and Cons

- **Pros:**
  - Good for static (read-only), **Owner/Containing** type trees
  - ***VERY FAST***, ***non-recursive traversal*** (index on „**L**“)
  - Can be mixed with „classic“ trees
- **Cons:**
  - **UPDATE**s/**INSERT**s/**DELETE**s are **VERY „expensive“**
  - No direct links between **child** and **parent** nodes
- **Depends:**
  - *Predefined order of child nodes* ***(Con? Pro?)***

Part 3

Recursive

CTEs

(Common Table Expressions)

Recursive CTEs: Pros and Cons

- **Cons:**
  - Client must ***know*** and ***understand*** tree structure
  - No full **PSQL** (just part of a **SELECT**)
  - No simple way to control the order of traversal **(yet)**

Recursive CTEs: Pros and Cons

- **Pros:** just about **everything else:**
  - **Server-side** recursion
  - **fast,** few round-trips
  - very **flexible** & dynamic
  - **transparent** to client
  - elegant + relatively easy *(once you get it ;-)*
  - no **Metadata** changes
  - no **GRANT…TO PROCEDURE**s required
  - Can be **used** in Stored Procedures

„normal“ CTEs: Intro

- **WITH <alias1> AS (<select_expression1>),**
  <alias2> AS (<select_expression2>)
  SELECT <...>
  FROM <alias1>
  JOIN <alias2> ON <join_condition>

*This is **one** **SELECT** you can send from a client „ad hoc“*

Recursive CTEs: Intro

Recursive CTEs can

recursively traverse tree structures with a

single „on the fly“ SELECT statement

from the client very efficiently

!

Recursive CTEs: basic structure

```
WITH RECURSIVE <cte_alias> AS (
    SELECT <parent data> -- root node's data

      UNION ALL

      SELECT <child data> -- children's data
        JOIN <cte_alias> ON <parent_link>
   ) -- DO    // for the Delphians

SELECT * FROM <cte_alias>
```

Traversing trees with recursive CTEs

```
WITH RECURSIVE  fs_tree AS (
    SELECT id, filename FROM filesys
    WHERE id_master = 0 -- condition for ROOT node

    UNION ALL

    SELECT ch.id, ch.filename FROM filesys ch   -- childs
     JOIN fs_tree pa ON ch.id_master  = pa.id )
                        -- ^^^ parent_link: p_l ^^^
SELECT * FROM fs_tree
```

Server processing of rec. CTEs I

What you send:

WITH RECURSIVE <x> AS

(SELECT *<parent>* -- *PA*

UNION ALL

SELECT *<child>* -- *CH*

JOIN <x> ON *P_L*)

SELECT * FROM <x>

Server Phase I: Preparation

„Analyse > *Transform* > PREPARE“:

- ***Transform* *PA*** (…)
- ***Transform* *CH:*** turn ***P_L*** into ***Params*** („un-recurse“/„flatten“ ***child*** select)

~~**JOIN <x> ON *CH.ID_Parent* = *PA.ID***~~

WHERE *CH.ID_Parent* = :ID *-- param*

- Prepare ***transformed PA***
- Prepare ***transformed CH***

Server processing of rec. CTEs II

What you get back *(Server Phase II: Execution)*

1. Execute ***PA*** („anchor query“)
2. **For each** result row **RR**: **SEND TO CLIENT**
3. **PUSH** result set **RS** to **stack**

   3.1 Execute ***CH*** with current ***params*** from **RR** **->** **RS2**

   3.2 **For each** result row **RR2** (if any): ***call*** **2.** with **RR2** *as* ***params***

   Back up one level, „unwind“

4. **POP RS** from **stack**, **goto** **2.** with next **RS** row

Recursion, one level down

Loop (same level)

Recursive results -> „flat“ result set

*this slide © **Vladyslav Khorsun***

- thanks, Vlad ! ☺

A

DEPT_NO	HEAD_DEPT	DEPARTMENT
000		Corporate Headquarters

Ch

DEPT_NO	HEAD_DEPT	DEPARTMENT
100	000	Sales and Marketing
600	000	Engineering
900	000	Finance

DEPT_NO	HEAD_DEPT	DEPARTMENT
180	100	Marketing
130	100	Field Office: East Coast
140	100	Field Office: Canada
110	100	Pacific Rim Headquarters
120	100	European Headquarters

DEPT_NO	HEAD_DEPT	DEPARTMENT
115	110	Field Office: Japan
116	110	Field Office: Singapore

DEPT_NO	HEAD_DEPT	DEPARTMENT
121	120	Field Office: Switzerland
123	120	Field Office: France
125	120	Field Office: Italy

DEPT_NO	HEAD_DEPT	DEPARTMENT
000		Corporate Headquarters
100	000	Sales and Marketing
180	100	Marketing
130	100	Field Office: East Coast
140	100	Field Office: Canada
110	100	Pacific Rim Headquarters
115	110	Field Office: Japan
116	110	Field Office: Singapore
120	100	European Headquarters
121	120	Field Office: Switzerland
123	120	Field Office: France
125	120	Field Office: Italy
600	000	Engineering
620	600	Software Products Div.
621	620	Software Development
622	620	Quality Assurance
623	620	Customer Support
670	600	Consumer Electronics Div.
671	670	Research and Development
672	670	Customer Services
900	000	Finance

Ordering Children in recursive CTEs

- **The Problem:**
  - Because of the **UNION**,
    you can‘t have an **ORDER BY** clause
    in the CTE‘s „Child“ SELECT
  - Since you can not **control**
    the order of child traversal,
    you **MUST** consider it to be **random (!)**

Ordering Children in recursive CTEs

- **Solution A *(Fb <x>)***

Use **DEPTH FIRST BY <columns>** clause

- Really **ORDERs** the Child select in the **UNION** (just using a different syntax)
- already returns the tree in the "right" order *during traversal*,
  no ordering of **result set** needed

(**but**: not yet implemented ☹)

Ordering Children in recursive CTEs

- **„Solution" B (Fb 3) :**

  Use a **Window Function**:

  with **rcte** as (

    select … from … UNION ALL select …,

     RANK() OVER(PARTITION BY PARENT_ID

              ORDER BY <sort col>)

- ***Looks* clever!**

  Only drawback: it doesn't work…(*)

  and if/when it does, that's **coincidence**!

  (*)NOTE: as of build 3.0.0.29631 this WILL actually work in Fb3 – Adriano has just committed a bugfix related to window functions in recursive CTEs. Thanks Adriano!

Ordering Children in recursive CTEs

- **Solution C:**

Use a **SELECTABLE SP** as Child Select

- Returns the Childs in a **defined** order **(!)**
- Unflexible for the client:
- ORDER is pre-defined in the SP…
- Columns are fixed…
- …see all other CONs of Recursive SPs!
- **Very clumsy** workaround

Ordering Children in recursive CTEs

- **Solution D:**

  Construct a **sort path**

  - Works *(kind of)* ok with Chars *(of limited length)*
  - Works *not so well* with numerical data
  - No index usage
  - orders **result set (after traversal)**
  - can take **LOTS** of reads
  - also a clumsy **workaround**
  - But: ***it works, and it's reliable!***

Part 4

„Real world“ CTE Examples

„Fun“ with recursive CTEs

Let‘s bake some

marble cake!

Chocolate icing

Chocolate cake mixture

Vanilla cake mixture

Shugga ***baby!***

- This cake
  has 5 *sub-recipes*
- Each has a different
  % of **sugar**

- Q1: What **%** of **sugar** is in the **entire cake** ?
- Q2: how much **sugar**,… do i need for 5 kg?
- Q3: How much **cake** can i bake,
  if i only have **<x>** [g] of **sugar** ??

that‘s about it…

Want some cake ???

☺