Query languages (NDBI049)

Recursion in SQL

Jaroslav Pokorný

MFF UK, Praha

jaroslav.pokorny@matfyz.cuni.cz

Content

1. **Introduction**
2. Creating recursive queries
3. Recursive calculation
4. Recursive searching
5. Logical hierarchies
6. Recursion termination
7. Conclusion

Recursion in SQL

- Intuitively: a query is *recursive*, if it is used in its own definition.
- This connection can be both direct and over more tables.
- Advantages: in certain cases the only effective way for obtaining the result
- Disadvantages: often worse readability a clarity

Where to use recursion in SQL

- effective for any data with hierarchical structure
  - relationships in tree structures
  - search in cyclic and acyclic graphs
- examples from practice:
  - search for connections in timetables
  - organizational structure of a company
  - bill of materials
  - components in a document management system, etc.

You can get around without recursion

- SQL before the SQL:99 standard did not contain a possibility to construct recursive queries,
- non-procedural solution: with adding certain „graph information“,
- procedural solution: use of cursors, cycles,
- others: ORACLE: proprietary solution + PL/SQL,
  - loss of efficiency and optimization
  - code is not so „elegant“

Application of recursion

- For graph traversal we obtain:
  - reachability

    Q1. Find all suborders of a given employee.
  - path enumerating

    Q2. Find the whole structure (all sub-products) for a given product.
  - path joining

    Q3. For a given product list all its components and including their amount.

Other advantages and disadvantages of recursion

- Advantages:
  - all work is specified in one query
  - It is possible to use a big part of the result

- Disadvantages
  - if only the small part of the result is really used
  - possibly endless recursion calls

Content

1. Introduction
2. **Creating recursive queries**
3. Recursive calculation
4. Recursive searching
5. Logical hierarchies
6. Recursion termination
7. Conclusion

Common Table Expression

- generalization of table expression in SQL:92
- declared by keyword WITH
- used as a substitute in nested queries
- from SELECT, INSERT, UPDATE, DELETE
- queries immediate after WITH keyword are called just once time

```
WITH [RECURSIVE] CTE [, CTE]…
CTE ::=name_CTE[(name_sl[,name_sl]…)] AS
        (CTE_query_definition)
```

Composition of aggregations – without CTE

Contributions(ID, forum, question)

Q4: Find the forum with the highest number of contributions

```
SELECT  COUNT(ID) AS number, forum
FROM  Contributions
GROUP BY  forum
HAVING  COUNT(ID) = (
      SELECT  MAX(number)
      FROM (SELECT  COUNT(ID) AS number, forum
              FROM   Contributions
              GROUP BY  forum)
```

Note: We are looking for MAX(COUNT(...))

Composition of aggregations – with CTE

Contributions(ID, forum, question)

```
WITH
   Amount_of_contrib(number, forum)
   AS ( SELECT COUNT(ID), forum)
              FROM Contributions
              GROUP BY forum )
SELECT number, forum
   FROM Amount_of_contrib
   WHERE number = (SELECT MAX(number)
                   FROM Ammount_of_contrib)
```

More CTEs in one query

```
WITH
  Amount_of_contrib(number, forum)
 AS (SELECT COUNT(ID), forum
      FROM Contributions
      GROUP BY forum ),
 Max_amount_of_contrib(number)
 AS (SELECT MAX(number)
      FROM Amount_of_contrib)
SELECT C1.*
 FROM Amount_of_contrib C1 INNER JOIN
      Max_amount_of_contrib C2  ON
                        C1.number = C2.number
```

Note: CTEs work in the same way as derived tables (given by SELECT behind FROM)

A movement to recursion

empID	name	function	supID
1	Novák	director	NULL
2	Srb	vice-director	1
3	Lomský	manager	2
4	Bor	manager	2

Q5.

```
WITH Superiors(name, supID, empID) AS
  (SELECT name, supID, empID
  FROM Employees
  WHERE function = 'manager'
  )
SELECT * FROM Superiors
```

name	supID	empID
Lomský	2	3
Bor	2	4

Recursive queries

- It is possible to refer R in CTE for table R
- the temporary table is created (exists only during query evaluation)
- three parts

  WITH

  anchoring (initialization subquery)

  UNION ALL

  recursive member

    - recursion runs when no further record is added or the recursion limit (MAXRECURSION) is not exceeded.
    - be careful to cycle occurrence in the recursive member

  SELECT

    - outer SELECT - returns the query result

Example

anchoring: executed only once

recursive member: repeatedly

join with the previous step

output

```
WITH RECURSIVE Superiors(name, supID, empID) AS
(SELECT name, supID, empID
        FROM Employees
        WHERE name = 'Nový'
        UNION ALL
SELECT E.name, E.supID, E.empID
        FROM Employees AS E
                INNER JOIN
                Superiors AS S
                ON S.supID = E.empID)
SELECT * FROM Superiors
```

name	supID	empID
Nový	11	13
Ryba	6	11
Rak	5	6
Syka	4	5
Bor	2	4
Srb	1	2
Novák	NULL	1

What was the query?

Q6.: Find all managers of employee Nový (including himself).

Example

anchoring: executed only once

recursive member: repeatedly

join with the previous step

output

```
WITH RECURSIVE Superiors(name, supID, empID) AS
(SELECT name, supID, empID
        FROM Employees
        WHERE name = 'Nový'
        UNION ALL
SELECT E.name, E.supID, E.empID
        FROM Employees AS E
        INNER JOIN
        Superiors AS S
        ON S.supID = E.empID)
SELECT * FROM Superiors
```

name	supID	empID
Nový	11	13
Ryba	6	11
Rak	5	6
Syka	4	5
Bor	2	4
Srb	1	2
Novák	NULL	1

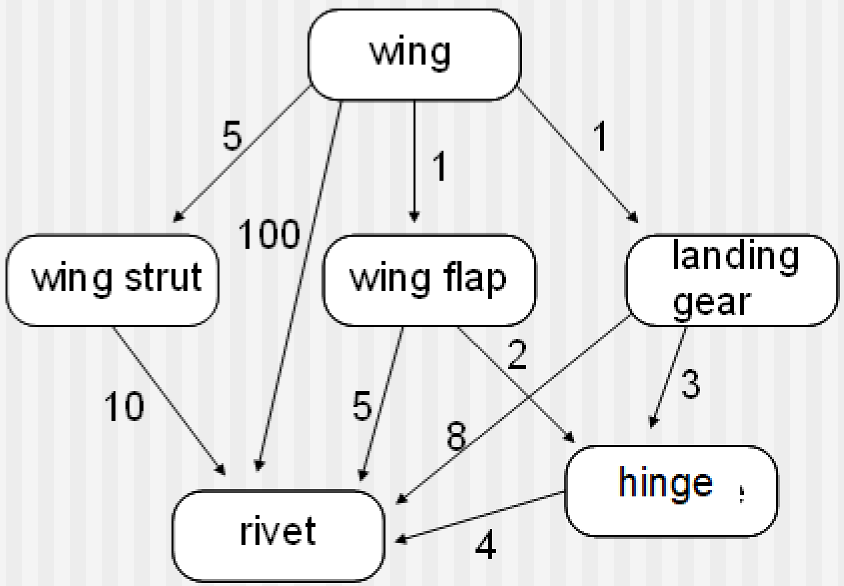
Restrictions of recursive queries

- It is not allowed to refer CTE in anchor
- Recursive part always self-refers CTE
  - SQL:99 supports only "linear" recursion: each FROM has at most one reference to recursively defined relation.
- Recursive part must not contain
  - SELECT DISTINCT
  - GROUP BY
  - HAVING
  - scalar aggregation
  - TOP
  - OUTER JOIN
- each column in recursive subquery has to be type-compatible with associated column in initialization subquery
  - type conversion – CAST can be used

Content

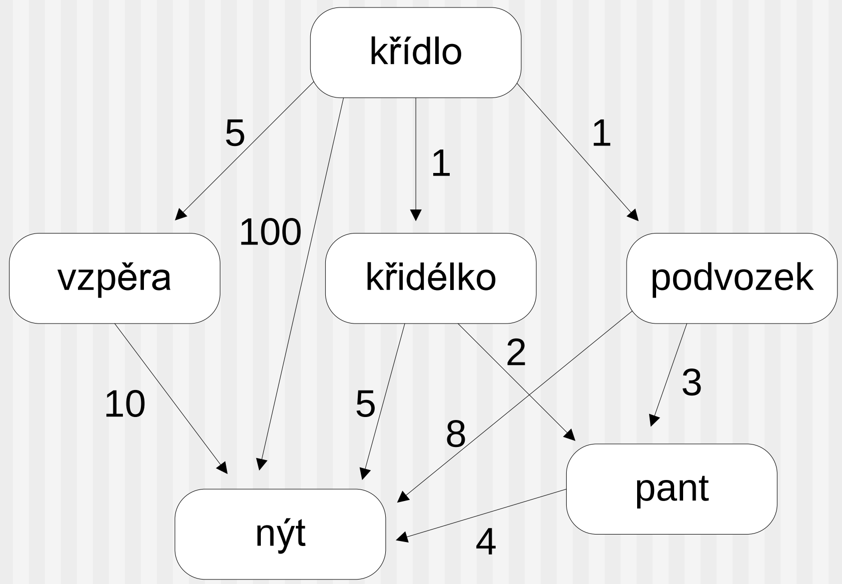
Recursive calculation

Q7. Which parts (including their amounts) are necessary to construct wing of a plain.



Recursive calculation (on Czech)

D7. Jaké součástky (včetně počtu) jsou potřeba pro výrobu křídla



Recursive calculation

- simplified storing in DB (relation Components) with quantities of particular parts in a part

Part	**Subpart**	**Qty**
wing strut	wing strut	5
wing	wing flap	1
wing	landing gear	1
wing	rivet	100
wing strut	rivet	10
wing flap	hinge	2
wing flap	rivet	5
landing gear	hinge	3
landing gear	rivet	8
hinge	rivet	4

Recursive calculation – queries

Q8. How many rivets are used to construct a wing plane?

Q9. List of all subparts for creating a wing plane including their amount.

Recursive calculation – solution

- What we have to be aware of?
  - recursion calling (graph walking)
  - to sum amounts of rivets in individual parts
  - amounts of individual sub-parts

Recursive calculation – Q8

- CTE

```
WITH RECURSIVE WingParts(Subpart, Qty)
   AS
 (( SELECT Subpart, Qty
    FROM Components
    WHERE Part = 'wing' )
   UNION ALL
  ( SELECT C.Subpart, W.Qty * C.Qty
    FROM WingParts W, Components C
    WHERE W.Subpart = C.Part ));
```

[initialization subquery]

[recursive subquery]

- result

Subpart	Qty	
wing strut	5	*directly*
wing flap	1	
landing gear	1	
rivet	100	
rivet	50	*from wing strut*
hinge	2	*from wing flap*
rivet	5	*from wing flap*
hinge	3	*from landing gear*
rivet	8	*from landing gear*
rivet	8	*from hinge of wing flap*
rivet	12	*from hinge of landing gear*

Recursive calculation – Q8

- finally we summarize particular quantities

```
WITH RECURSIVE WingParts(Subpart, Qty) AS
  (( SELECT Subpart, Qty
     FROM Components
     WHERE Part = ‘wing’ )
    UNION ALL
   ( SELECT C.Subpart, W.Qty * C.Qty
     FROM WingParts W, Components C
     WHERE W.Subpart = C.Part ))
SELECT sum(Qty) AS Qty
FROM WingParts
WHERE Subpart = ‘rivet’;
```

Result
Qty
183

Recursive calculation – Q9

- To solve Q9 it is enough to change only the result query

```
WITH RECURSIVE WingParts(Subpart, Qty) AS
  (( SELECT Subpart, Qty
     FROM Components
     WHERE Part = 'wing' )
    UNION ALL
   ( SELECT C.Subpart, W.Qty * C.Qty
     FROM WingParts W, Components C
     WHERE W.Subpart = K.Part ))
SELECT Subpart, sum(Qty) AS Qty
FROM WingParts
GROUP BY Subpart;
```

Result	
Subpart	Qty
wing strut	5
wing flap	1
landing gear	1
hinge	5
rivet	183

Syntax of tree traversal v Oracle 9i

```
SELECT columns FROM table
    [WHERE condition3]
    start WITH condition1
    CONNECT BY condition2
    [ORDER BY …]
```

- Rows satisfying the condition in start WITH are considered as root rows on the first level of nesting
- For each row at level *i*, direct descendants fulfilling condition in clause CONNECT BY at level *i*+1 are looked for recursively.
  - Ancestor row in the condition is denoted by the key word PRIOR

Syntax of tree traversal v Oracle 9i

- Finally, there are removed rows not satisfying the WHERE clause.
- If sorting is not defined, the order corresponds to the pre-order traversal.
- Each row contains the pseudocolumn LEVEL containing the row level in hierarchy.

Emp(empID, name, manager)

Oracle 9i vs. SQL:99

Inserts spaces in number 2*Level

- Oracle 9i:

```
SELECT LPAD(' ', 2*Level) || name, Level
FROM Emp
start WITH manager IS NULL
CONNECT BY manager = PRIOR empID;
```

- SQL:99

```
WITH RECURSIVE Emp1 AS (
     SELECT x.name AS name, 0 AS Level
     FROM Emp x WHERE manager IS NULL
   UNION ALL
     SELECT y.name, Level+1
     FROM Emp y JOIN Emp1 ON y.manager =
                 Emp1.empID)
     SELECT * FROM Emp1;
```

Oracle 9i vs. SQL:99

Effect of LPAD function

Data
Novák
Srb
Lomský
Bor

Recursion support in other DBMS

- Yes: IBB DB2, Microsoft SQL Server, PostgressSQL
- No: MySQL

Content

1. Introduction
2. Creating recursive queries
3. Recursive calculation
4. **Recursive searching**
5. Logical hierarchies
6. Recursion termination
7. Conclusion

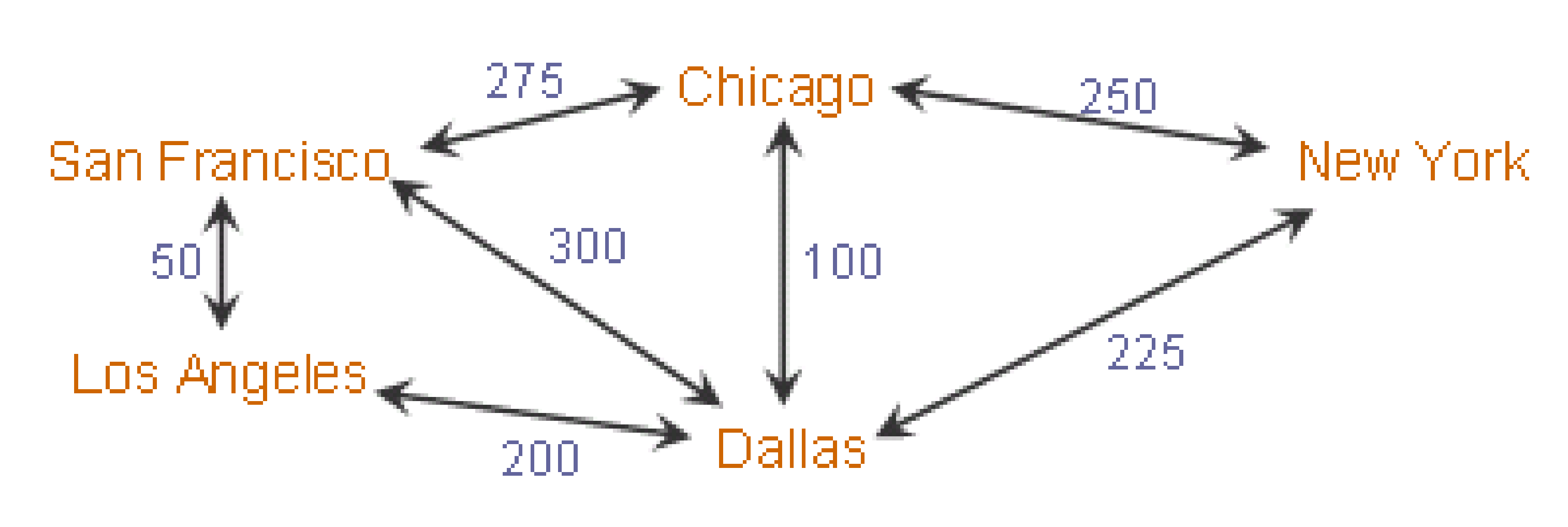
Recursive searching

- Effort to find the best solution based on certain criteria of the given problem.
- Example:

  Let us consider an airport departure system and a client who wants to travel from San Francisco to New York with the lowest cost.

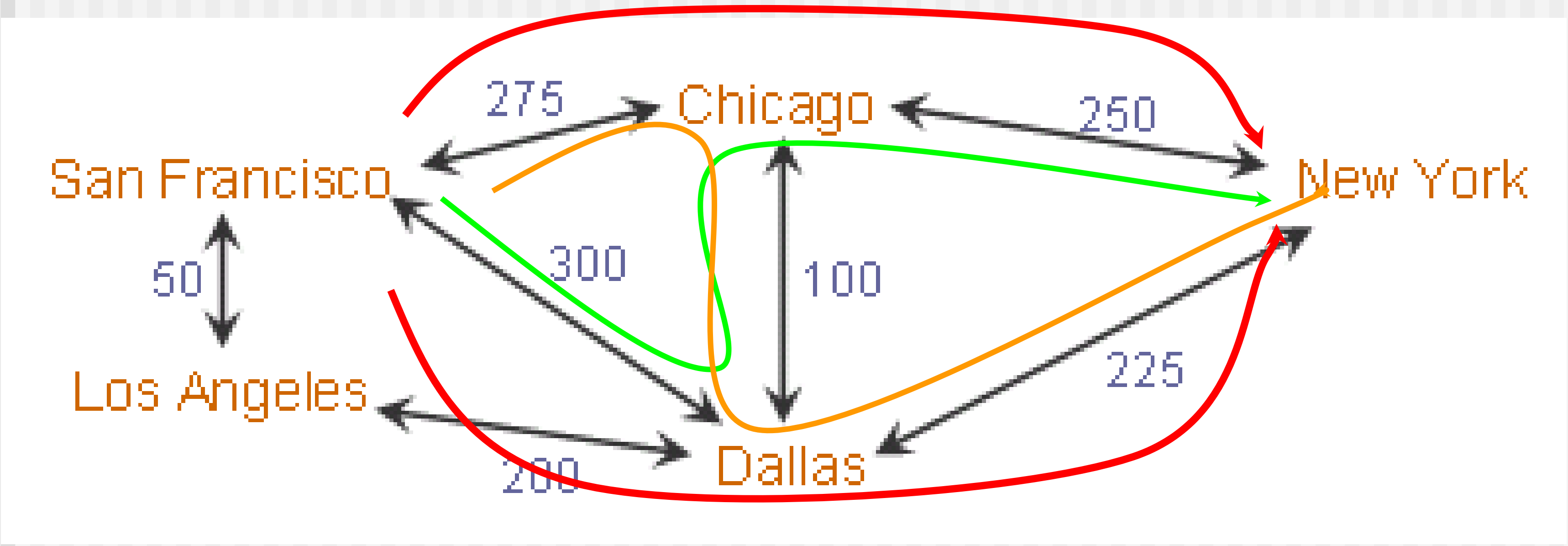
Recursive searching – example

- route map (including costs for the flight):



Recursive searching – example

- several possible paths (in different colours):



Recursive searching – example

- The table of Flights

flightno	start	destination	cost
xxx01	SF	CHG	275
xxx02	SF	DLS	300
…			

Q10. Find the lowest cost path from San Francisco to New York.

Problem: the flight map is not an acyclic graph – we have to solve the stopping of recursion.

Recursive searching – 1. solution

- Temporary table used in CTE is called Trips
  - the subquery with all directly (one-flight) reachable destinations from San Francisco will be the anchor of the query
  - the recursive part of the query will find others (two or more flights) destinations

Recursive searching – 1. solution

```
WITH RECURSIVE Trips (destination, route, totalcost) AS
   ((SELECT destination, destination, cost
     FROM Flights
     WHERE start = 'SF' )
UNION ALL
   (SELECT l.destination,
     v.route || ',' || l.destination, v.totalcost + l.cost
   FROM Trips v, Flights l
   WHERE v.destination = l.start))
SELECT route, totalcost
FROM Trips
WHERE destination = 'NY';
```

Where is the problems?

- We add a longer expression to the route column
- We are in endless loop.

Recursive searching – 1. solution + correction

- violation of the rule that the value in the column of the recursive subquery must not be longer in the corresponding column of the initialization subquery (anchor)

  Solution:
  - We change data type in both subqueries (initialization and recursive) to VARCHAR(50)
  - This is done by the CAST expression.
- function CAST

CAST (expression AS data_type)

  Examples:

  CAST (c1 + c2 AS Decimal(8,2))

  CAST (name||adress AS Varchar(255))

  string
  - longer is completed with spaces
  - shorter is cut and returns a warning

Recursive searching – 1. solution + correction

- looping problem

  Solution:

  - we will not take into account flights from the starting place, that is from San Francisco,
  - we will not take into account flights from the destination, that is from New Yorku
  - and we are interested in only flights that have a maximum of 2 legs

Recursive searching – final solution

```
WITH RECURSIVE Trips (destination, route, #flights, totalcost) AS
 ((SELECT destination,  CAST(destination AS Varchar(50)), 1, cost
    FROM Flights
    WHERE start = 'SF'
UNION ALL
  (SELECT l.destination, CAST(v.route || ',' || l.destination AS Varchar(50)),
    v. #flights + 1, v.totalcost + l.cost
   FROM Trips t, Flights f
   WHERE t.destination = f.start
    AND f.destination <> 'SF'
    AND f.start <> 'NY'
    AND t. #flights < 2))
SELECT route, totalcost
FROM Trips
WHERE destination = 'NY '  AND totalcost=(SELECT min(totalcost)
                                          FROM Trips
                                          WHERE destination='NY');
```

Result

route	totalcost
DLS, NY	525
CHG,NY	525

Content

Classification of hierarchies

- by graph properties
  - convergent
  - divergent
  - recursive
- by balance
  - balanced
    - all leafs on the same level
    - on each level different objects (e.g., geographical structure)
  - unbalanced
    - leafs at different levels
    - uniform objects (e.g. organizational structure)
- Problem: representation by relations

Divergent hierarchies

- each node except the root has exactly one parent
  Ex.: geographical hierarchies
    - continent, state, town, street
- implementation
  - Edge (PKEY, KEYO)
  - primary key KEYO
  - table with referential integrity PKEY$\subseteq$ KEYO

```
            AAA
             |
      +------+------+
      |      |      |
     BBB    CCC    DDD
                    |
                 +--+--+
                 |     |
                EEE   FFF
                       |
                       |
                      GGG
```

KEYO	PKEY	NUM	PRICE
AAA			$10
BBB	AAA	1	$21
CCC	AAA	5	$23
DDD	AAA	20	$25
EEE	DDD	44	$33
FFF	DDD	5	$34
GGG	FFF	5	$44

Convergent hierarchies

- Each object can have arbitrary number of ancestors and descendants

  Ex.: Departments of company
- Define the result of query

  Q11. How many descendants has “AAA”?
  - 6, 7, 8?
- Implementation
  - table of objects
  - table of relationships

```
          AAA
           |
   +-------+------+
   |       |      |
  BBB     CCC    DDD
           |      |
          +-+  +-+---+
            |  |     |
            EEE     FFF
                     |
                     |
                     G
```

OBJECTS

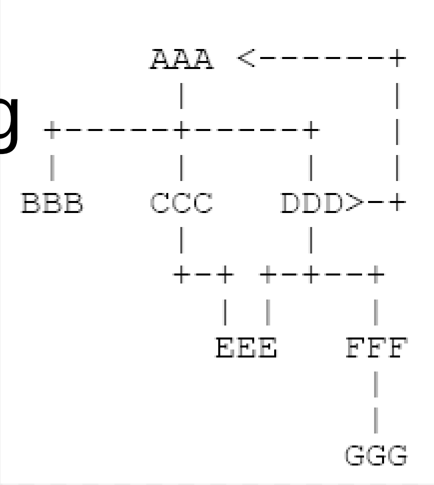
KEYO	PRICE
AAA	$10
BBB	$21
CCC	$23
DDD	$25
EEE	$33
FFF	$34
GGG	$44

RELATIONSHIPS

PKEY	CKEY	NUM
AAA	BBB	1
AAA	CCC	5
AAA	DDD	20
CCC	EEE	33
DDD	EEE	44
DDD	FFF	5
FFF	GGG	5

Recursive hierarchies

- similar to convergent
  - moreover: a node can be its ascendant (directly or undirectly)
  - Example: supervisor-subordinate vs. project manager and director as solver
- they cause cycling
- in practice, their use is mostly conflicting
- implementation
  - as convergent ones



Content

1. Introduction
2. Creating recursive queries
3. Recursive calculation
4. Recursive searching
5. Logical hierarchies
6. **Recursion termination**
7. Conclusion

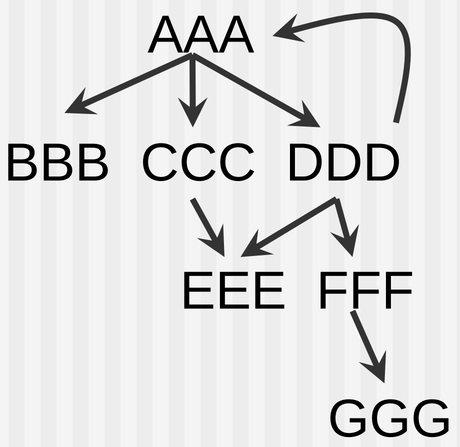
Recursion termination

- How remove cycling in recursive hierarchies?
- Possibilities of stopping the recursion
  - QB Server
    - V MS SQL after reaching the value MAXRECURSION (default 100)
  - after reaching a given level
  - to remember the path and omit already visited nodes

Problem: recursive hierarchies

table RH

PKEY	CKEY
AAA	BBB
AAA	CCC
AAA	DDD
CCC	EEE
DDD	AAA
DDD	FFF
DDD	EEE
FFF	GGG



Q12. Find all descendants AAA until level 4

Stopping after reaching n^{th} level (attribute LVL)

```
WITH RECURSIVE PARENT(CKEY, LVL) AS
(SELECT DISTINCT PKEY, 0
            FROM RH
            WHERE PKEY = 'AAA'
 UNION ALL
 SELECT H.CKEY, R.LVL+1
            FROM RH H, PARENT P
            WHERE P.CKEY = H.PKEY
            AND P.LVL + 1 < 4
)
SELECT CKEY, LVL
FROM PARENT;
```

N = 4

	CKEY	LVL
1	AAA	0
2	BBB	1
3	CCC	1
4	DDD	1
5	AAA	2
6	EEE	2
7	FFF	2
8	GGG	3
9	BBB	3
10	CCC	3
11	DDD	3
12	EEE	2

- What to do with duplicates in result?

Shift away the duplicates (using 2 CTE)

```
WITH RECURSIVE PARENT(CKEY, LVL) AS
(SELECT DISTINCT PKEY, 0
          FROM RH
          WHERE PKEY = 'AAA'
UNION ALL
SELECT H.CKEY, R.LVL+1
          FROM RH H, PARENT R
          WHERE P.CKEY = H.PKEY
          AND P.LVL + 1 <4
),
WITHOUT_DUPL(CKEY, LVL, NUM) AS
(SELECT CKEY, MIN(LVL), COUNT(*)
FROM PARENT
GROUP BY CKEY)

SELECT CKEY, LVL, NUM
FROM WITHOUT _DUPL
```

	CKEY	LVL	NUM
1	AAA	0	2
2	BBB	1	2
3	CCC	1	2
4	DDD	1	2
5	EEE	2	2
6	FFF	2	1
7	GGG	3	1

Ommiting already visited nodes

```
WITH PARENT (CKEY, LVL, PATH) AS
(SELECT DISTINCT PKEY, 0, VARCHAR(PKEY, 20)
        FROM RH
        WHERE PKEY = ‘AAA‘
 UNION ALL
 SELECT H.CKEY, P.LVL + 1,
      P.PATH || ‘>‘ || H.CKEY
   FROM RH H, PARENT R
   WHERE P.CKEY = H.PKEY
   AND
   LOCATE(H.CKEY || ‘>‘, P.PATH) = 0
)
SELECT CKEY, LVL, PATH
FROM PARENT;
```

returns the position of pattern in argument

Result

CKEY	LVL	PATH
AAA	0	AAA
BBB	1	AAA>BBB
CCC	1	AAA>CCC
DDD	1	AAA>DDD
EEE	2	AAA>CCC>EEE
EEE	2	AAA>DDD>EEE
FFF	2	AAA>DDD>FFF
GGG	3	AAA>DDD>FFF>GGG

Stack vs. recursion

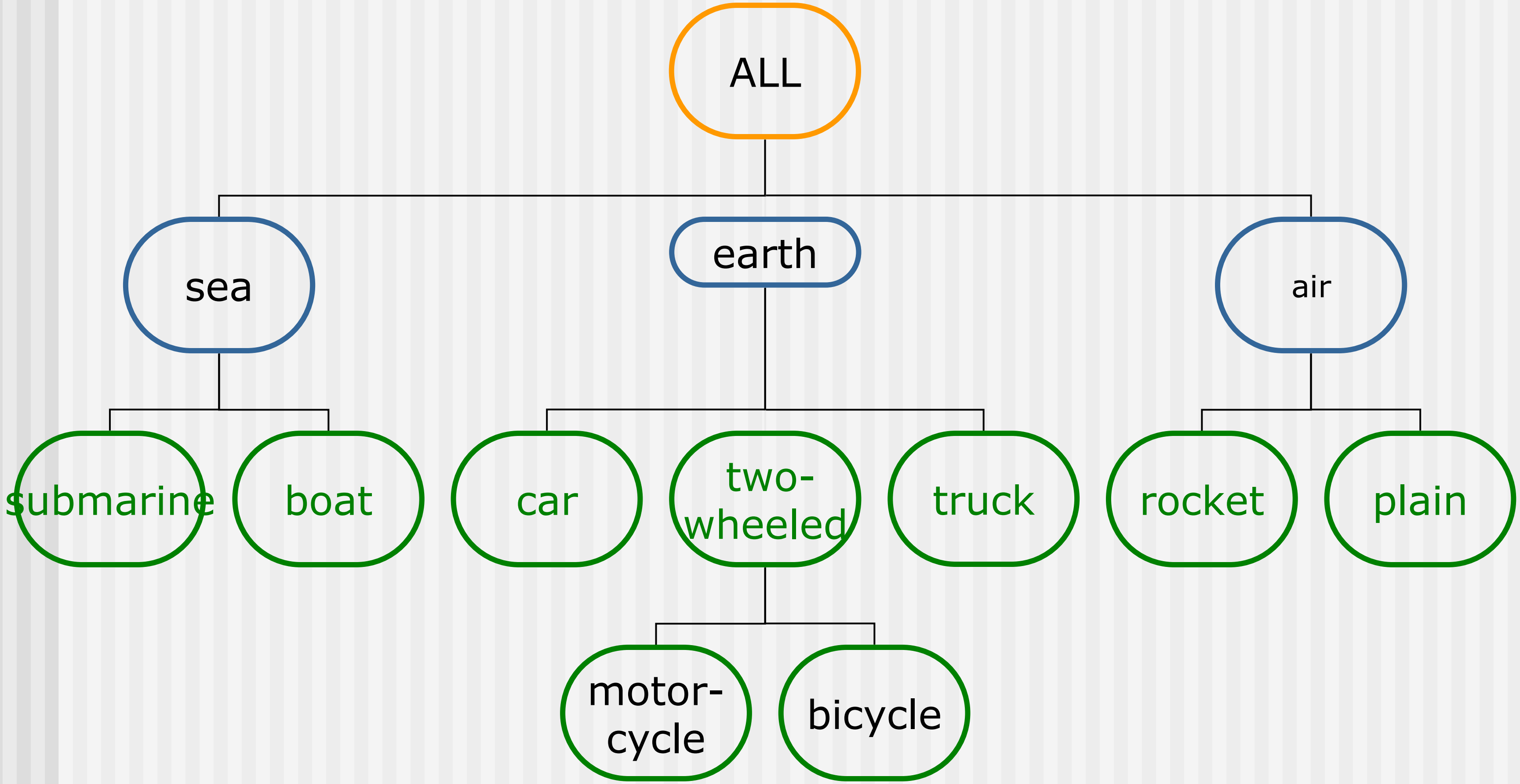
- Problem: how efectively implement recursion – opakování join může vést k tomu, že se věci počítají opakovaně
- Recursion can be simulated using a stack.
- Stack model is faster then CTE
  - Dá se použít only for querying hierarchical data

Vehicles(Id, parentID, name)

Example

Id	parentID	name
1	NULL	ALL
2	1	sea
3	1	earth
4	1	air
5	2	submarine
6	2	boat
7	3	car
8	3	two-wheeled
9	3	truck
10	4	rocket
11	4	plain
12	8	motorcycle
13	8	bicycle

Example


ALL
sea
earth
air
submarine
boat
car
two-
wheeled
truck
rocket
plain
motor-
cycle
bicycle

Ancestors without recursion (1)

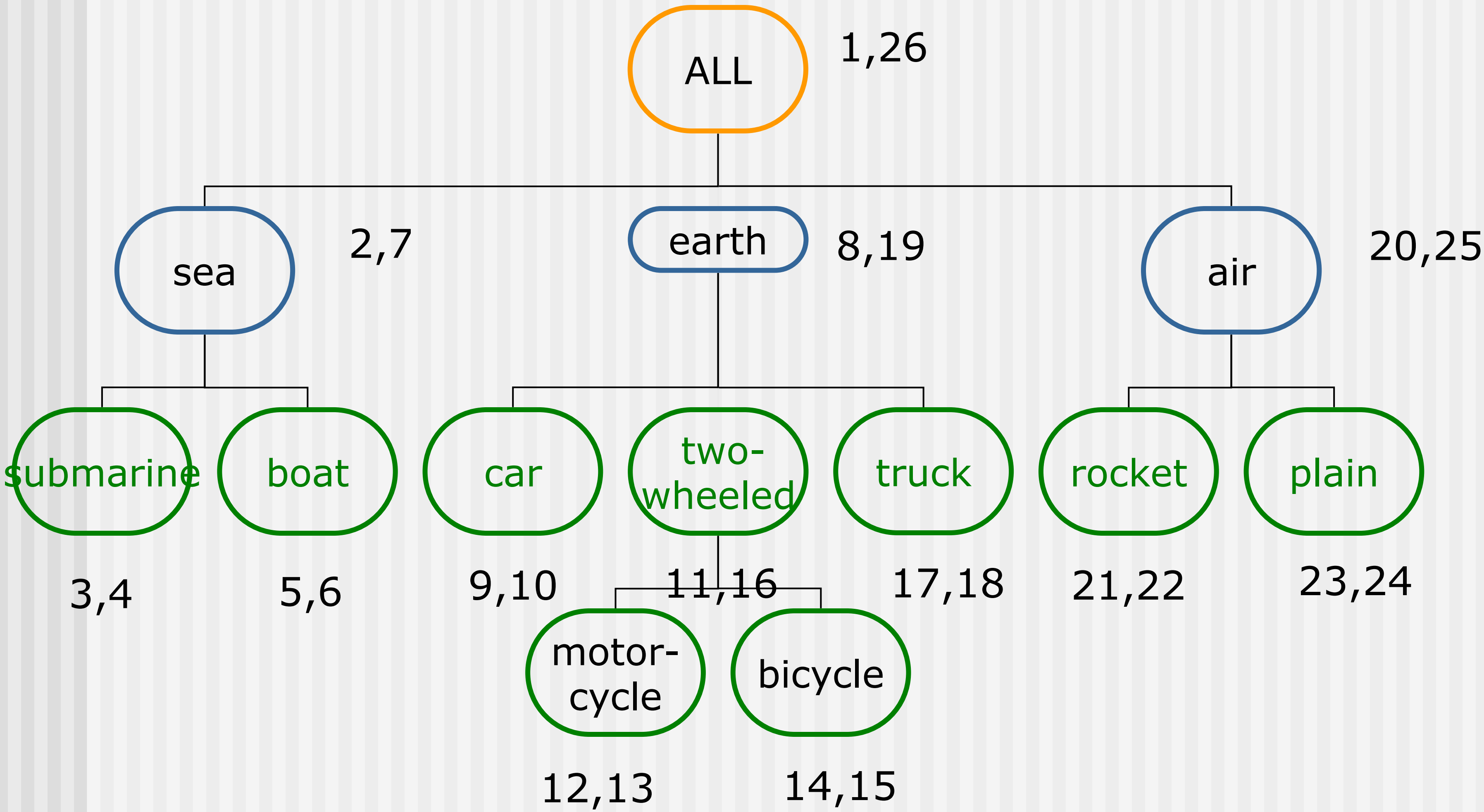
- Can recursion be removed? YES, using the stack.
- We add 2 new columns to the table Vehicles: R_bound and L_bound
- Their values are based on the numbering that occurs through the preorder tree traversal.

Ancestors without recursion (2)

- We fill the table with the data;
- For new columns:

```
UPDATE Vehicles SET L_bound = 1 , R_bound = 26 WHERE
   ID = 1
UPDATE Vehicles SET L_bound = 2 , R_bound = 7 WHERE
   ID = 2
…
UPDATE Vehicles SET L_bound = 12 , R_bound = 13
   WHERE ID = 12
UPDATE Vehicles SET L_bound = 14 , R_bound = 14
   WHERE ID = 13
```

Ancestors - without recursion (3)



Example

Id	parentID	name	L_bound	R_bound
1	NULL	ALL	1	26
2	1	sea	2	7
3	1	earth	8	19
4	1	air	20	25
5	2	submarine	3	4
6	2	boat	5	6
7	3	car	9	10
8	3	two-wheeled	11	16
9	3	truck	17	18
10	4	rocket	21	22
11	4	plain	23	24
12	8	motorcycle	12	13
13	8	bicycle	14	15

Ancestors - without recursion (4)

Query for ancestors of motorcycle uses intervals.

```
SELECT *
FROM Vehicles
WHERE R_bound > 12
  AND L_bound < 13
```

Example

Id	parentID	name	L_bound	R_bound
1	NULL	ALL	1	26
2	1	sea	2	7
3	1	earth	8	19
4	1	air	20	25
5	2	submarine	3	4
6	2	boat	5	6
7	3	car	9	10
8	3	two-wheeled	11	16
9	3	truck	17	18
10	4	rocket	21	22
11	4	plain	23	24
12	8	motorcycle	12	13
13	8	bicycle	14	15