



The slide features a background image of a smiling man in the foreground and a blurred group of four people behind him. The top of the slide has a decorative header with binary code and the IDUG Europe logo. The main content area is white with blue and black text. The IDUG logo at the bottom left consists of a blue circle with a white swoosh and the text 'IDUG The Worldwide DB2 User Community'. The session information at the bottom right is in a black sans-serif font.

Recursive queries are a powerful technique to traverse hierarchical data structures and to solve similar problems. This presentation revisits the concepts of recursive SQL, analyzes its performance impact, discusses and compares alternate design and query techniques, and delivers tips and tricks to optimize the performance of recursive queries.

# Agenda



- **What Is Recursive SQL?**
- **Performance Tips for Recursive Queries**
  - For example, DISTINCT for first result set
- **Alternate Data Structures**
  - Node Numbers
  - XML
- **Recursive Queries on Oracle and SQL Server**

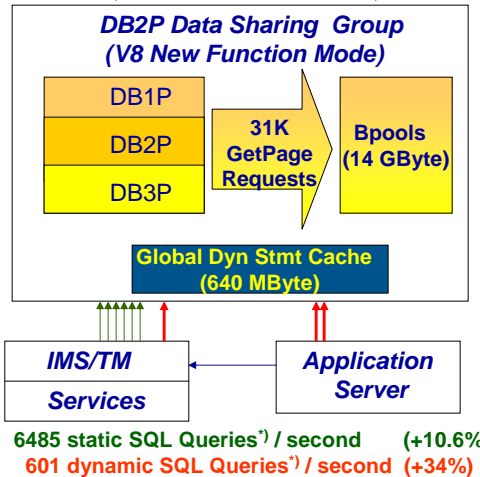
The following areas will be touched:

1. Recursive SQL concept and syntax
2. Limitations of recursive queries
3. Optimization and tuning of recursive queries
4. Comparison of alternate techniques
5. Case Studies

# Swiss Mobiliar: Key Facts at a Glance



(2008-12-02 @ 10:40 AM)



<sup>\*)</sup> 1 open / n fetch / 1 close = 1 Query

IDUG 2009 Europe

3

- Major insurer for all sectors.
- Leading property insurer.
- Leader in term insurance (private and occupational pensions).
- Reliable and expert partner.
- Mutual basis.
- Customers and employees share in our success.
- Continuity and a lasting relationship with policyholders.
- Committed to good corporate citizenship for the common good and the environment.

# Disclaimer



- The Information contained in this presentation has not been submitted to any formal Swiss Mobiliar or other review and is distributed on an 'as is' basis without any warranty either expressed or implied. The use of this information is the user's responsibility.
- The procedures, results and measurements presented in this paper were run in either the test and development environment or in the production environment at Swiss Mobiliar in Berne, Switzerland. There is no guarantee that the same or similar results will be obtained elsewhere. Users attempting to adapt these procedures and data to their own environments do so at their own risk. All procedures presented have been designed and developed for educational purposes only.

This presentation does not contain many notes pages. Therefore, if you are interested in more details, you can ask for an animated powerpoint document of this presentation. This might be particularly of interest if you are planning to teach this content to a class of application developers.

In order to get the powerpoint version, please mail to [thomas.baumann@mobi.ch](mailto:thomas.baumann@mobi.ch)

## Factorial Example: $n!$



- 1<sup>st</sup> approach: top-down calculation
- $n! := n * (n-1)!$ 
  - $3! := 3 * 2! := 3 * 2 * 1! := 3 * 2 * 1 * 0!$
  - $0!$  has been defined as 1
  - So,  $3! = 3 * 2 * 1 * 1 = 6$

# Factorial Example: $n!$



- 2<sup>nd</sup> approach: bottom-up calculation:
  - Initialization (“prime the pump”)
    - $0! := 1$  *Level 0*
  - Iteration (“pump more”, up to a limit)
    - $1! = 1 * 0!$ , so  $1! = 1$  *Level 1*
    - $2! = 2 * 1!$ , so  $2! = 2$  *Level 2*
    - $3! = 3 * 2!$ , so  $3! = 6$  *Level 3*
  - Selection (“use the pump”)
    - $2! = 2$ ,  $3! = 6$  etc.

# Factorial Example: n!



- SQL uses the 2<sup>nd</sup> approach:

- Initialization (“prime the pump”)

```
SELECT 1 as factorial, 0 as level  
FROM SYSIBM.SYSDUMMY1
```

- Iteration (“pump more”, up to a limit)

```
SELECT (level+1)* factorial as factorial  
      ,level+1 as level  
FROM -- result of most recent pump  
WHERE level <= 5
```

- Selection (“use the pump”)

```
SELECT level, factorial FROM -- pump  
WHERE level = 3
```

# Factorial Example: n!



- SQL Syntax:

WITH PUMP (factorial, level) AS (

```
SELECT 1 as factorial, 0 as level  
FROM SYSIBM.SYSDUMMY1
```

UNION ALL

```
SELECT (level+1)* factorial as factorial  
      ,level+1 as level  
FROM PUMP  
WHERE level <= 5
```

)

```
SELECT level, factorial FROM PUMP  
WHERE level = 3
```



## Example 2: Aston Martin



- SQL Syntax:

WITH PUMP (picture, name) AS (

```
SELECT Picture_BLOB, name
FROM CURRENT_CARS WHERE NAME LIKE ,DB%'
```

UNION ALL

```
SELECT Picture, H.name
FROM PUMP P, HISTORY_CARS H
WHERE H.NAME = „DB“!!STRIP(
DIGITS(INT(SUBSTR(P.NAME,3,1))-1),B,'0')
```

)

```
SELECT Picture FROM PUMP
WHERE NAME = 'DB2'
```

## Example 2: Aston Martin



- Performance Point:

executed  
at each level!

```
WITH PUMP (picture, name) AS (
```

```
SELECT Picture_BLOB, name  
FROM CURRENT_CARS WHERE NAME LIKE ,DB%`
```

```
UNION ALL
```

```
SELECT Picture, H.name  
FROM PUMP P, HISTORY_CARS H  
WHERE H.NAME = „DB“!!STRIP(  
DIGITS(INT(SUBSTR(P.NAME,3,1))-1),B,`0`)
```

```
)
```

```
SELECT Picture FROM PUMP  
ORDER BY NAME DESC
```

## Example 2: Aston Martin



- Performance Tip: Prepare Iteration

WITH PUMP (picture, name, next\_name) AS (

```
SELECT Picture_BLOB, name, „DB“!!STRIP(  
DIGITS(INT(SUBSTR(NAME,3,1))-1),B,‘0‘)  
FROM CURRENT_CARS WHERE NAME LIKE ‚DB%‘
```

UNION ALL

```
SELECT Picture, H.name, „DB“!!STRIP(  
DIGITS(INT(SUBSTR(P.NAME,3,1))-1),B,‘0‘)  
FROM PUMP P, HISTORY_CARS H  
WHERE H.NAME = P.NEXT_NAME
```

)

```
SELECT Picture FROM PUMP
```

# A Practical Example



- Swiss Mobiliar –  
Customer#Customer Relationship
  - Basic Performance Results
  - Performance Improvements

C97251_1	C97251_2
4711	7865
4711	9265
7865	8563

# Straight Forward Approach



With ZwRes (Partnernr, n) as (

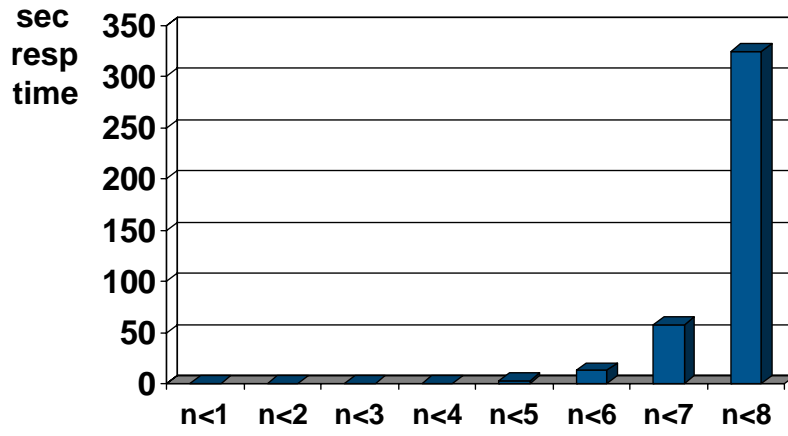
```
SELECT C97251_2, 1
      FROM DB2ZVIEW.VPARPPP1
      WHERE C97251_1= 10000050
```

UNION ALL

```
SELECT C97251_2, n+1
      FROM DB2ZVIEW.VPARPPP1, ZwRes
      WHERE C97251_1=Partnernr AND n < 7
      )
```

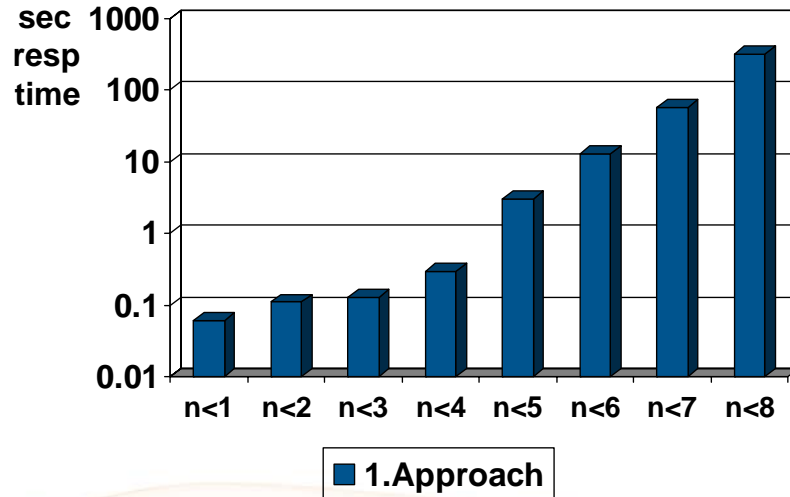
```
SELECT DISTINCT Partnernr
      FROM ZwRes
```

# Straight Forward Approach



■ 1.Approach

# Straight Forward Approach Logarithmic Scale



# DISTINCT in initialization



With ZwRes (Partnernr, n) as (

```
SELECT DISTINCT C97251_2, 1
   FROM DB2ZVIEW.VPARPPP1
   WHERE C97251_1= 10000050
```

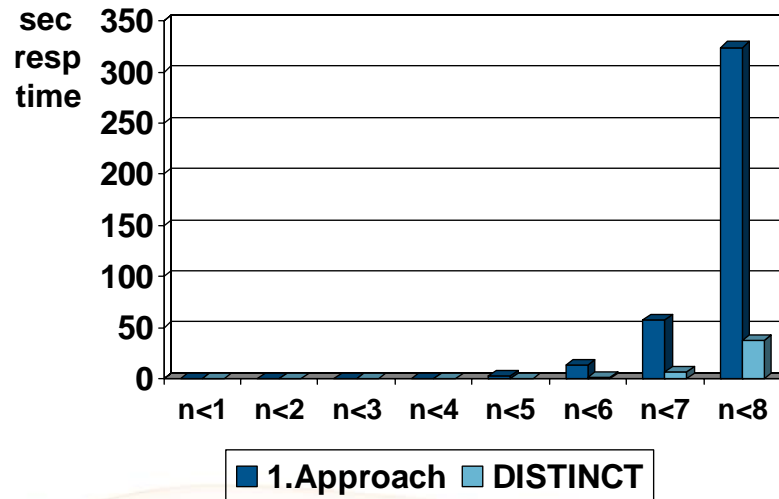
UNION ALL

```
SELECT C97251_2, n+1
   FROM DB2ZVIEW.VPARPPP1, ZwRes
   WHERE C97251_1=Partnernr AND n < 7
)
```

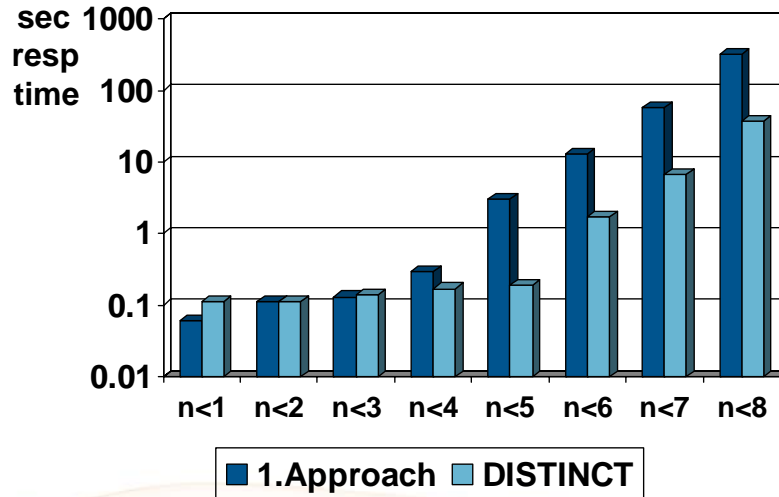
```
SELECT DISTINCT Partnernr
   FROM ZwRes
```



# DISTINCT in initialization



# DISTINCT in initialization Logarithmic Scale



## DISTINCT in iteration (1/4)



With ZwRes (Partnernr, n) as (

```
SELECT DISTINCT C97251_2, 1
  FROM DB2ZVIEW.VPARPPP1
  WHERE C97251_1= 10000050
```

UNION ALL

```
SELECT DISTINCT C97251_2, n+1
  FROM DB2ZVIEW.VPARPPP1, ZwRes
  WHERE C97251_1=Partnernr AND n < 7
)
```

```
SELECT DISTINCT Partnernr
  FROM ZwRes
```

42925(-342)[IBM][CLI Driver][DB2] SQL0342N  
The common table expression "ZWRES"  
cannot use SELECT DISTINCT and must use  
UNION ALL because it is recursive.  
SQLSTATE=42925

## DISTINCT in iteration (2/4)



```
With create view db2zview.vparppp_new as  
SELECT select distinct C97251_2, C97251_1  
FROM from db2zview.vparppp1  
WHERE C97251_1= 10000050
```

UNION ALL

```
SELECT C97251_2, n+1  
FROM DB2ZVIEW.VPARPPP_NEW ZwRes  
WHERE C97251_1=Partnernr AND n < 7  
)  
tablespace scan on VPARPPP1  
at each iteration
```

```
SELECT DISTINCT Partnernr  
FROM ZwRes
```

## DISTINCT in iteration (3/4): MQT



```
create table db2zview.vparppp_new
  (C97251_1 integer, C97251_2 integer);
create index db2zview.xparpppn
on db2zview.vparppp_new
(C97251_1, C97251_2);
```

EXEC SQL

```
DECLARE C1 CURSOR FOR
```

```
SELECT DISTINCT C97251_1, C97251_2
```

```
FROM DB2ZVIEW.VPARPPP1
```

ENDEXEC

```
LOAD DATA INCURSOR(C1) REPLACE
INTO TABLE DB2ZVIEW.VPARPPP_NEW
```

```
-- loads 1.8 m rows
```

```
-- consider MQT
```

## DISTINCT in iteration (3/4): MQT



With ZwRes (Partnernr, n) as (

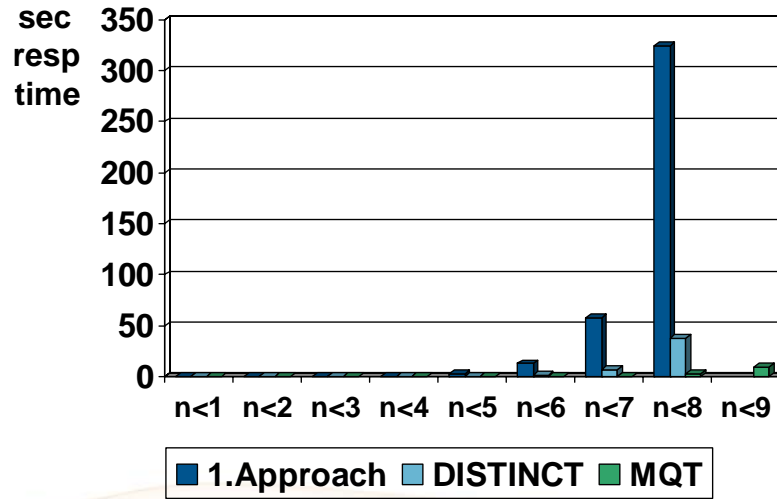
```
SELECT DISTINCT C97251_2, 1
  FROM DB2ZVIEW.VPARPPP1
  WHERE C97251_1= 10000050
```

UNION ALL

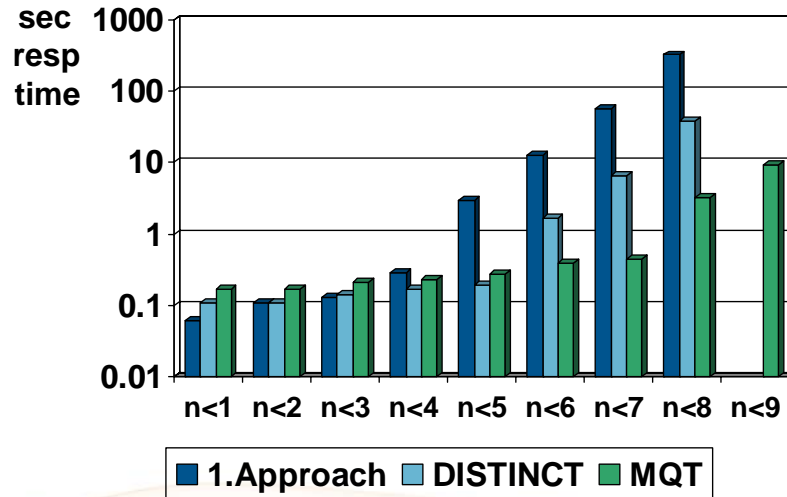
```
SELECT C97251_2, n+1
  FROM DB2ZVIEW.VPARPPP_NEW ZwRes
  WHERE C97251_1=Partnernr AND n < 1
)
```

```
SELECT DISTINCT Partnernr
  FROM ZwRes
```

# MQT



# MQT Logarithmic Scale





## DISTINCT in iteration (4/4)



With ZwRes (Partnernr, n) as (

```
SELECT DISTINCT C97251_2, 1
  FROM DB2ZVIEW.VPARPPP1
  WHERE C97251_1= 10000050
```

UNION ALL

```
SELECT C97251_2, max(n+1)
  FROM DB2ZVIEW.VPARPPP1, ZwRes
  WHERE C97251_1=Partnernr AND n < 7
  GROUP BY C97251_2
```

```
)
SELECT DISTINCT Partnernr
  FROM ZwRes
```

42836(-345)[IBM][CLI Driver][DB2] SQL0345N  
The fullselect of the recursive common table  
expression "ZWRES" must be the UNION of  
two or more fullselects and cannot include  
column functions, GROUP BY clause,  
HAVING clause, ORDER BY clause, or an  
explicit join including an ON clause.  
SQLSTATE=42836

# Different Query Approach



```
SELECT DISTINCT N.C97251_2
FROM DB2ZVIEW.VPARPPP1 N,
( SELECT DISTINCT N.C97251_2
  FROM DB2ZVIEW.VPARPPP1 N,
  ( SELECT DISTINCT C97251_2
    FROM DB2ZVIEW.VPARPPP1
    WHERE C97251_1= 10000050
  ) Level_0
  WHERE N.C97251_1=Level_0.C97251_2
) Level_1
WHERE N.C97251_1=Level_1.C97251_2
etc....
until
```

## Different Query Approach (cont.)



```
SELECT DISTINCT N.C97251_2
FROM DB2ZVIEW.VPARPPP1 N,
(SELECT DISTINCT N.C97251_2
 FROM DB2ZVIEW.VPARPPP1 N,
(SELECT DISTINCT N.C97251_2
 FROM DB2ZVIEW.VPARPPP1 N,
(SELECT DISTINCT N.C97251_2
 FROM DB2ZVIEW.VPARPPP1 N,
(SELECT DISTINCT N.C97251_2
 FROM DB2ZVIEW.VPARPPP1 N,
(SELECT DISTINCT N.C97251_2
 FROM DB2ZVIEW.VPARPPP1 N,
(SELECT DISTINCT C97251_2
 FROM DB2ZVIEW.VPARPPP1
 WHERE C97251_1= 10000050) Level_0
WHERE N.C97251_1=Level_0.C97251_2) Level_1
WHERE N.C97251_1=Level_1.C97251_2) Level_2
WHERE N.C97251_1=Level_2.C97251_2) Level_3
WHERE N.C97251_1=Level_3.C97251_2) Level_4
WHERE N.C97251_1=Level_4.C97251_2) Level_5
WHERE N.C97251_1=Level_5.C97251_2
```

IDUG 2009 Europe

27

The query above lists all customers which were added within the 6<sup>th</sup> iteration of the recursive query. In order to get all results, all these parts have to be set together using UNION:

```
SELECT DISTINCT C97251_2
FROM DB2ZVIEW.VPARPPP1
WHERE C97251_1= 10000050      -- level 0 only
```

UNION ...

```
SELECT DISTINCT N.C97251_2
FROM DB2ZVIEW.VPARPPP1 N RIGHT OUTER JOIN
(SELECT DISTINCT C97251_2
FROM DB2ZVIEW.VPARPPP1
WHERE C97251_1= 10000050) Level_0
ON N.C97251_1=Level_0.C97251_2  -- level 1 only
```

UNION ... -- level 2 only

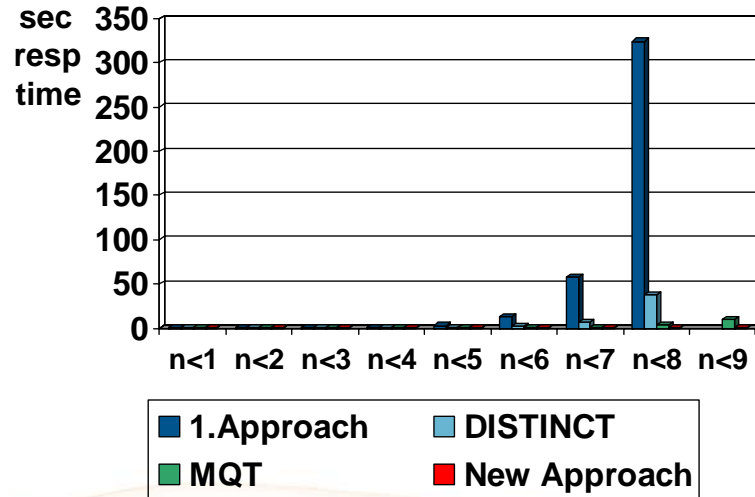
UNION ...

..

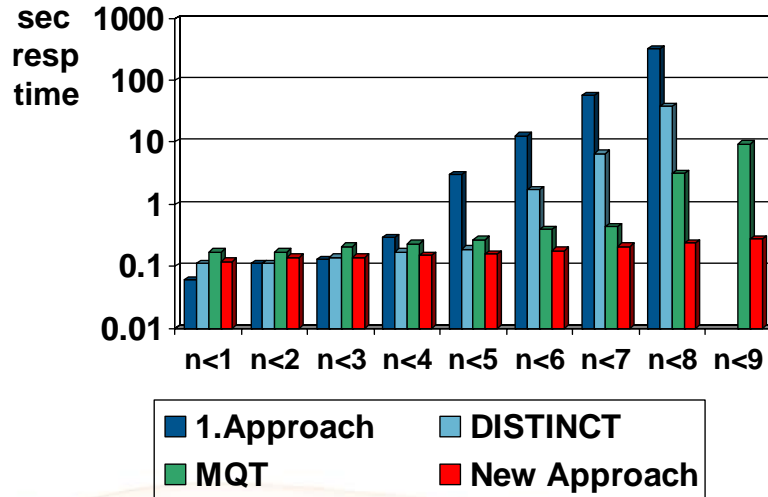
The query above -- level 5 only

Even the use of RIGHT OUTER JOIN instead of the inner joins does not make the UNIONS obsolete.

# Different Query Approach



# Different Query Approach Logarithmic Scale



# UNION



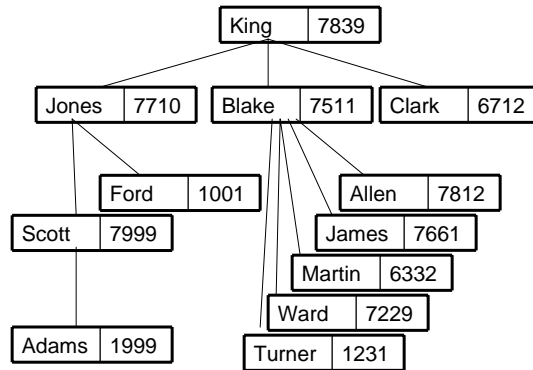
The initial NTE query selects the data found in the 5<sup>th</sup> iteration only.

Its correct form is by using UNION.

# Job Hierarchy Example

Employees		
name	empno	mgr
King	7839	<i>null</i>
Jones	7710	7839
Blake	7511	7839
Clark	6712	7839
Scott	7999	7710
Ford	1001	7710
Allen	7812	7511
James	7661	7511
Martin	6332	7511
Ward	7229	7511
Turner	1231	7511
Adams	1999	7999

List all employees which report directly or indirectly to "King"



# Job Hierarchy Example

```
WITH Pump (name, n, empno ) AS
```

Prime the pump

```
( SELECT e.name, 1, e.empno
  FROM employees e
  WHERE e.name='King'
```

```
UNION ALL
```

Pump more

```
SELECT e.name, n+1, e.empno
FROM employees e, Pump z
WHERE z.empno = e.mgr
   AND n < 5
```

Use the pump

```
)
SELECT name, n as level, empno
FROM Pump
```

Pump		
name	level	empno
King	1	7839
Jones	2	7710
Blake	2	7511
Clark	2	6712
Scott	3	7999
Ford	3	1001
Allen	3	7812
James	3	7661
Martin	3	6332
Ward	3	7229
Turner	3	1231
Adams	4	1999



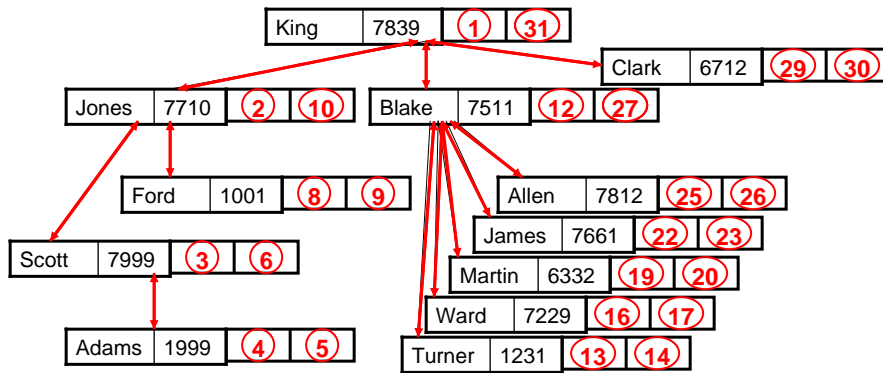
# Alternate Data Structures



- Hierarchical structures
  - Node Numbers
- Hierarchical structures with a fixed height
  - XML
- Performance Comparison

# Node Numbers

Traverse tree once and update "LEFT\_NODE" and "RIGHT NODE" columns



# Node Numbers



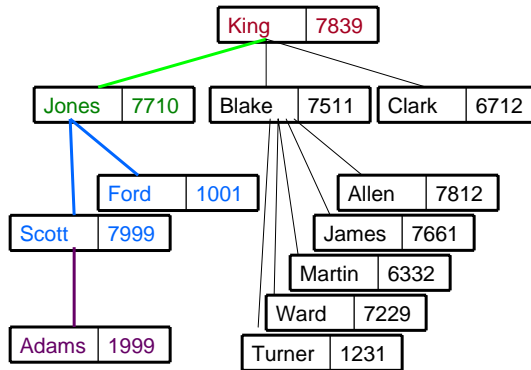
```
SELECT m.name, m.level, m.empno
       ,m.LNO, m.RNO
FROM employees p, employees m
WHERE p.name='King'
      AND m.LNO >= p.LNO
      AND m.LNO <= p.RNO
ORDER BY level, LNO
```

name	level	empno	LNO	RNO
King	1	7839	1	31
Jones	2	7710	2	10
Blake	2	7511	12	27
Clark	2	6712	29	30
Scott	3	7999	3	6
Ford	3	1001	8	9
Turner	3	1231	13	14
Ward	3	7229	16	17
Martin	3	6332	19	20
James	3	7661	22	23
Allen	3	7812	25	26
Adams	4	1999	4	5

# SQL/XML

Store hierarchy in XML data type

```
<level1>
  <name>King</name>
  <empno>7839</empno>
  <level2>
    <name>Jones</name>
    <empno>7710</empno>
    <level3>
      <name>Scott</name>
      <empno>7999</empno>
      <level4>
        <name>Adams</name>
        <empno>1999</empno>
      </level4>
    </level3>
    <level3>
      <name>Ford</name>
      <empno>1001</empno>
    </level3>
  </level2>
  ...
</level1>
```



# SQL/XML



```
SELECT XMLQUERY  
  ('$d//name/text()'  
   passing depthhierarchy as "d")  
FROM employees  
where xmlexists  
  ('$d/level1[name="King"]'  
   passing depthhierarchy as "d")
```

King  
Jones  
Scott  
Adams  
Ford  
Blake  
Turner  
Ward  
Martin  
James  
Allen  
Clark

## Alternate Data Structures



- Performance: No of getpages
  - Recursive SQL: 66 getpg
  - Node Numbers: 16 getpg
  - SQL/XML: 4 getpg
- Limits for Node Numbers and XML
  - pure hierarchical structures only

# Recursion Performance Challenges with Real Data



- **Fastest Path**
  - Find the fastest way from Berne, Switzerland to Denver, Colorado
- **Detect cycles in directed graph structures**
  - Identify circular definitions of referential integrity constraints in the DB2 catalog

# Fastest Path



- Find the fastest way from Berne, Switzerland to Rome, Italy

## Flüge von Zürich

Flights from Zurich

Flugtag Day	Abflug Depart.	an/ab Arr./Dep.	Ankunft Arrival	Flug-Nr Flight-No.	Über via	Kommentar Comment
<b>nach/to Cologne/Bonn (CGN) Continued</b>						
--345-7	1740		1845	4U769		nonstop
1-----	1825		1930	4U769		nonstop
-2-----	1835		1940	4U769		nonstop
-----6-	1955		2100	4U763		nonstop
-----7	2020		2125	4U763		nonstop
--345--	2040		2150	4U763		nonstop
-2-----	2040		2145	4U763		nonstop
						Bis 30.12
<b>nach/to Copenhagen (CPH)</b>						
1234567	0710		0905	LX1266		nonstop
1234567	1040		1225	SK802		nonstop
1234567	1220		1405	LX1270		nonstop
1234567	1710		1855	SK604		nonstop
1234567	1725		1920	LX1272		nonstop
12345-7	2010		2200	SK610		nonstop
<b>nach/to Dallas/Fort Worth (DFW)</b>						
12345-7	0840	1435/1630	1802	DL67/DL67	ATL	Ab 8.3
12345-7	0940	1435/1630	1802	DL67/DL67	ATL	Bis 7.3
<b>nach/to Dar es Salaam (DAR)</b>						
122-56-	0935		2105	LX292		direct/1 stop
<b>nach/to Delhi (DEL)</b>						
1234567	1230		0035+1	LX146		nonstop
<b>nach/to Denver (DEN)</b>						
1234567	1255	1800/1830	2003	LX8/LX3248	ORD	-06.00
<b>nach/to Djerba (DJE)</b>						
---4-6-	1100		1345	TU485		nonstop
<b>nach/to Doha (DOH)</b>						
---45--	1105		1855	QR062		nonstop
--3--7	2130		0520+1	QR064		nonstop

## FLIGHTS (

Dep CHAR(03),

Arr CHAR(03),

DepTime Time,

ArrTime Time)

ZRH CGN 1740 1845

ZRH CGN 1825 1930

...

ZRH ORD 1255 1600

...

DL 0101500  
00 0100010  
00 0100010

01015010 01000101 01000101 01000110 01000011 01000101 01000000 010001  
01000101 01011500 01010000 01000101 01010010 01000100 01000101 010115  
01000101 01011500 01010000 01000101 01010010 01000101 01000101 010115



# Fastest Path



```
WITH Pump (Dep, Arr, Path, ArrTime, level) AS
```

```
( SELECT Dep, Arr, Dep||Arr as Path,  
  ArrTime, 1 as level  
  FROM FLIGHTS  
  WHERE Dep='BRN' AND  
         DepTime > current time
```

```
UNION ALL
```

```
SELECT f.Dep, f.Arr,  
  p.Path||f.Arr as Path, f.Artime,  
  p.level+1  
FROM Pump p, Flights f  
WHERE p.Arr=f.Dep  
      AND p.ArrTime + 1 hours < f.DepTime  
      AND level < 4
```

```
)
```

```
SELECT Path, Artime  
FROM Pump  
WHERE Arr = 'ROM'  
ORDER BY ArrTime
```

# Fastest Path



- Number of Paths

- Level1: 20 paths (Berne is a very small airport)
- Level2: 2563 paths
- Level3: 1,040,578 paths
- Level4: 422,4 M paths

# Fastest Path



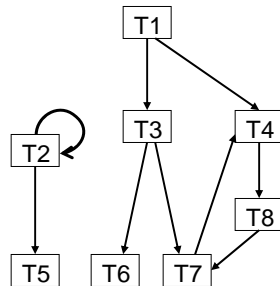
- Performance Improvements

- “Promising Paths” only
  - Needs GROUP BY or similar **in second step**  
MIN(ArrTime) GROUP BY Arr
  - again, use Nested Table Expression rather than recursive query
- Discard Paths
  - Transfer Time < 4 hours
- Additional Exit Criteria (other than just *level*)
  - ArrivalTime < 9 PM

# Detecting Cycles



SYSIBM.SYSRELS



Cycles:

T2→T2

T4→T8→T7→T4

(and T8→T7→T4→T8 etc.)

```
WITH PUMP (LEVEL, STARTTAB, ENDTAB, PATH) AS
(SELECT 1, REFTBNAME, TBNAME ,
STRIP(REFTBNAME)!!STRIP(TBNAME)
FROM SYSIBM.SYSRELS
```

UNION ALL

```
SELECT PUMP.LEVEL+1, PUMP.STARTTAB,
CHILD.TBNAME, PUMP.PATH!!CHILD.TBNAME
FROM PUMP, SYSIBM.SYSRELS CHILD
WHERE PUMP.ENDTAB = CHILD.REFTBNAME
AND CHILD.TBNAME <> CHILD.REFTBNAME
AND PUMP.STARTTAB <> PUMP.ENDTAB
AND PUMP.LEVEL < 30)
```

```
SELECT DISTINCT LEVEL, STARTTAB, PATH
FROM PUMP
WHERE STARTTAB=ENDTAB
```

Remark: TBCREATOR and REFTBCREATOR have been omitted in the query above. Please reintroduce these components as follows:

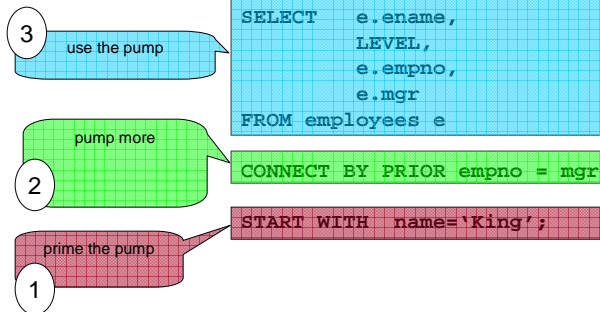
```
WITH PUMP (LEVEL, STARTCR, STARTTAB, ENDCR, ENDTAB,
PATH) AS
(SELECT 1, REFTBCREATOR,
REFTBNAME, CREATOR, TBNAME ,
STRIP(REFTBNAME)!!STRIP(TBNAME)
FROM SYSIBM.SYSRELS
--WHERE CREATOR='...' AND REFTBCREATOR='...'
UNION ALL
SELECT PUMP.LEVEL+1, PUMP.STARTCR, PUMP.STARTTAB,
CHILD.CREATOR,CHILD.TBNAME,PUMP.PATH!!CHILD.TBNAME
FROM PUMP, SYSIBM.SYSRELS CHILD
WHERE PUMP.ENDTAB = CHILD.REFTBNAME
AND PUMP.ENDCR = CHILD.REFTBCREATOR
AND (CHILD.TBNAME <> CHILD.REFTBNAME
OR CHILD.CREATOR <> CHILD.REFTBCREATOR)
AND (PUMP.STARTTAB <> PUMP.ENDTAB OR PUMP.STARTCR
<> PUMP.ENDCR)
AND PUMP.LEVEL < 20
--AND CHILD.CREATOR='...' AND CHILD.REFTBCREATOR='...'
)
SELECT DISTINCT LEVEL, STARTTAB, PATH FROM PUMP
WHERE STARTTAB=ENDTAB
```

# Recursive SQL @ Oracle

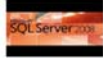


ORACLE  
DATABASE 11g

- Job hierarchy example



name	level	emp no	mgr
King	1	7839	null
Jones	2	7710	7839
Blake	2	7511	7839
Clark	2	6712	7839
Scott	3	7999	7710
Ford	3	1001	7710
Allen	3	7812	7511
James	3	7661	7511
Martin	3	6332	7511
Ward	3	7229	7511
Turner	3	1231	7511
Adams	4	1999	7999

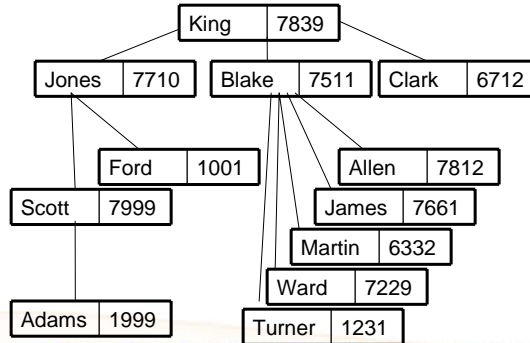


- Job hierarchy example

```
DECLARE @CurrentEmployee hierarchyid;
```

```
SELECT * FROM Employee
```

```
WHERE @CurrentEmployee.IsDescendantOf(Name) = 'King';
```



# Summary



- Use Alternate Data Structures for Hierarchical Queries
  - XML (or data structures such as node numbers)
- Use Alternate Queries If Peak Performance is Requested
- Limit First and Intermediate Result Sets
- Be aware that recursive SQL syntax is different among the products

Session: E02



## Recursive SQL From a Performance Perspective



# Thomas Baumann

Swiss Mobiliar, Switzerland  
thomas.baumann@mobi.ch

Since 1992, Thomas has been focusing on understanding how the DB2 database engine works. He has a master degree of computer sciences from ETH Zurich, Switzerland, and is currently working as data management team leader at Swiss Mobiliar Insurance in Berne, Switzerland. If he is not in his office trying how to get the most out of DB2, he is somewhere lecturing on optimization or performing an audit . Thomas is a member of the IDUG speakers hall of fame, and an internationally recognized expert in database auditing.