Recursive SQL from a Performance Perspective

par Thomas Baumann  La Mobilière
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
Swiss Mobiliar: Key Facts at a Glance

DB2P Data Sharing Group
(V8 New Function Mode)

DB1P
DB2P
DB3P

31K GetPage Requests

Bpools (14 GByte)

Global Dyn Stmt Cache (640 MByte)

IMS/TM Services

Application Server

6485 static SQL Queries\(^*\) / second (+10.6%)
601 dynamic SQL Queries\(^*\) / second (+34%)

\(^*\) 1 open / n fetch / 1 close = 1 Query
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.
Factorial Example: n!

- 1ˢᵗ 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\(^{nd} \)** approach: bottom-up calculation:
  - **Initialization** ("prime the pump")
    - \( 0! := 1 \) \textit{Level 0}
  - **Iteration** ("pump more", up to a limit)
    - \( 1! = 1 \times 0! \), so \( 1! = 1 \) \textit{Level 1}
    - \( 2! = 2 \times 1! \), so \( 2! = 2 \) \textit{Level 2}
    - \( 3! = 3 \times 2! \), so \( 3! = 6 \) \textit{Level 3}
  - **Selection** ("use the pump")
    - \( 2! = 2, \ 3! = 6 \) etc.
Factorial Example: n!

- SQL uses the 2\textsuperscript{nd} approach:
  - Initialization ("prime the pump")
    ```sql
    SELECT 1 as factorial, 0 as level
    FROM SYSIBM.SYSDUMMY1
    ```
  - Iteration ("pump more", up to a limit)
    ```sql
    SELECT (level+1)* factorial as factorial,
    ,level+1 as level
    FROM -- result of most recent pump
    WHERE level <= 5
    ```
  - Selection ("use the pump")
    ```sql
    SELECT level, factorial FROM -- pump
    WHERE level = 3
    ```
Factorial Example: n!

- SQL Syntax:

```sql
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:

```sql
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:

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

executed at each level!
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

<table>
<thead>
<tr>
<th>C97251_1</th>
<th>C97251_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4711</td>
<td>7865</td>
</tr>
<tr>
<td>4711</td>
<td>9265</td>
</tr>
<tr>
<td>7865</td>
<td>8563</td>
</tr>
</tbody>
</table>
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

![Graph showing response time for different n values]
Straight Forward Approach
Logarithmic Scale

1. Approach

sec
resp
time

0.01
0.1
1
10
100
1000

n<1 n<2 n<3 n<4 n<5 n<6 n<7 n<8

1. Approach
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

![Graph showing the comparison between 1. Approach and DISTINCT in initialization.](image-url)
DISTINCT in initialization
Logarithmic Scale

<table>
<thead>
<tr>
<th>n&lt;1</th>
<th>n&lt;2</th>
<th>n&lt;3</th>
<th>n&lt;4</th>
<th>n&lt;5</th>
<th>n&lt;6</th>
<th>n&lt;7</th>
<th>n&lt;8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>0.1</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
</tr>
</tbody>
</table>

sec 1000
resp 100
time 10

1. Approach
DISTINCT
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
Notes

- 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 distinct C97251_2, C97251_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 < 7

SELECT DISTINCT Partnernr 
FROM ZwRes 

tablespace scan on VPARPPP1 at each iteration
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

1. Approach

sec resp time

n<1 n<2 n<3 n<4 n<5 n<6 n<7 n<8 n<9

1. Approach  DISTINCT  MQT
MQT Logarithmic Scale

sec resp time

n<1 n<2 n<3 n<4 n<5 n<6 n<7 n<8 n<9

1. Approach  DISTINCT  MQT
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, \text{max}(n+1)
    FROM DB2ZVIEW.VPARPPP1, ZwRes
    WHERE C97251_1 = Partnernr AND n < 7
    GROUP BY C97251_2
)

SELECT DISTINCT Partnernr
    FROM ZwRes
Notes

• 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
Different Query Approach

```sql
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
The query above lists all customers which were added within the 6th 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 
...  -- level 5 only

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

1. Approach

- DISTINCT
- MQT
- New Approach

- sec
- resp
- time

n<1  n<2  n<3  n<4  n<5  n<6  n<7  n<8  n<9
Different Query Approach Logarithmic Scale

- Approach
- DISTINCT
- MQT
- New Approach

- sec
- resp
- time
UNION

The initial NTE query selects the data found in the 5th iteration only.

Its correct form is by using UNION.
Job Hierarchy Example

<table>
<thead>
<tr>
<th>Employees</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>empno</td>
</tr>
<tr>
<td>King</td>
<td>7839</td>
</tr>
<tr>
<td>Jones</td>
<td>7710</td>
</tr>
<tr>
<td>Blake</td>
<td>7511</td>
</tr>
<tr>
<td>Clark</td>
<td>6712</td>
</tr>
<tr>
<td>Scott</td>
<td>7999</td>
</tr>
<tr>
<td>Ford</td>
<td>1001</td>
</tr>
<tr>
<td>Allen</td>
<td>7812</td>
</tr>
<tr>
<td>James</td>
<td>7661</td>
</tr>
<tr>
<td>Martin</td>
<td>6332</td>
</tr>
<tr>
<td>Ward</td>
<td>7229</td>
</tr>
<tr>
<td>Turner</td>
<td>1231</td>
</tr>
<tr>
<td>Adams</td>
<td>1999</td>
</tr>
</tbody>
</table>

List all employees which report directly or indirectly to “King”
### Job Hierarchy Example

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

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

UNION ALL

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

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

<table>
<thead>
<tr>
<th>Pump</th>
<th>name</th>
<th>level</th>
<th>empno</th>
</tr>
</thead>
<tbody>
<tr>
<td>King</td>
<td>King</td>
<td>1</td>
<td>7839</td>
</tr>
<tr>
<td>Jones</td>
<td>Jones</td>
<td>2</td>
<td>7710</td>
</tr>
<tr>
<td>Blake</td>
<td>Blake</td>
<td>2</td>
<td>7511</td>
</tr>
<tr>
<td>Clark</td>
<td>Clark</td>
<td>2</td>
<td>6712</td>
</tr>
<tr>
<td>Scott</td>
<td>Scott</td>
<td>3</td>
<td>7999</td>
</tr>
<tr>
<td>Ford</td>
<td>Ford</td>
<td>3</td>
<td>1001</td>
</tr>
<tr>
<td>Allen</td>
<td>Allen</td>
<td>3</td>
<td>7812</td>
</tr>
<tr>
<td>James</td>
<td>James</td>
<td>3</td>
<td>7661</td>
</tr>
<tr>
<td>Martin</td>
<td>Martin</td>
<td>3</td>
<td>6332</td>
</tr>
<tr>
<td>Ward</td>
<td>Ward</td>
<td>3</td>
<td>7229</td>
</tr>
<tr>
<td>Turner</td>
<td>Turner</td>
<td>3</td>
<td>1231</td>
</tr>
<tr>
<td>Adams</td>
<td>Adams</td>
<td>4</td>
<td>1999</td>
</tr>
</tbody>
</table>
```
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
```
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>
  </level2>
  ...<level1>
```
SELECT XMLQUERY ('$d/name/text()'
    passing depthierarchy as "d")
FROM employees
WHERE xmlexists ('$d/level1[name="King"]'
    passing depthierarchy 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 Denver, Colorado

```sql
FLIGHTS(
  Dep CHAR(03),
  Arr CHAR(03),
  DepTime Time,
  ArrTime Time
)
```

- ZRH  CGN  1740  1845
- ZRH  CGN  1825  1930
- ...
- ZRH  ORD  1255  1600
- ...

43
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.Arrtime,
  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, Arrtime
FROM Pump
WHERE Arr = 'DEN'
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

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

• Job hierarchy example

```
SELECT e.ename, LEVEL, e.empno, e.mgr
FROM employees e
CONNECT BY PRIOR empno = mgr
START WITH name='King';
```

<table>
<thead>
<tr>
<th>name</th>
<th>level</th>
<th>empno</th>
<th>mgr</th>
</tr>
</thead>
<tbody>
<tr>
<td>King</td>
<td>1</td>
<td>7839</td>
<td>null</td>
</tr>
<tr>
<td>Jones</td>
<td>2</td>
<td>7710</td>
<td>7839</td>
</tr>
<tr>
<td>Blake</td>
<td>2</td>
<td>7511</td>
<td>7839</td>
</tr>
<tr>
<td>Clark</td>
<td>2</td>
<td>6712</td>
<td>7839</td>
</tr>
<tr>
<td>Scott</td>
<td>3</td>
<td>7999</td>
<td>7710</td>
</tr>
<tr>
<td>Ford</td>
<td>3</td>
<td>1001</td>
<td>7710</td>
</tr>
<tr>
<td>Allen</td>
<td>3</td>
<td>7812</td>
<td>7511</td>
</tr>
<tr>
<td>James</td>
<td>3</td>
<td>7661</td>
<td>7511</td>
</tr>
<tr>
<td>Martin</td>
<td>3</td>
<td>6332</td>
<td>7511</td>
</tr>
<tr>
<td>Ward</td>
<td>3</td>
<td>7229</td>
<td>7511</td>
</tr>
<tr>
<td>Turner</td>
<td>3</td>
<td>1231</td>
<td>7511</td>
</tr>
<tr>
<td>Adams</td>
<td>4</td>
<td>1999</td>
<td>7999</td>
</tr>
</tbody>
</table>
Recursive SQL @ SQLServer

• Job hierarchy example

DECLARE @CurrentEmployee hierarchyid;

SELECT * FROM Employee
WHERE @CurrentEmployee.IsDescendantOf(Name) = 'King';

```
<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>King</td>
<td>7839</td>
</tr>
<tr>
<td>Jones</td>
<td>7710</td>
</tr>
<tr>
<td>Blake</td>
<td>7511</td>
</tr>
<tr>
<td>Clark</td>
<td>6712</td>
</tr>
<tr>
<td>Ford</td>
<td>1001</td>
</tr>
<tr>
<td>Scott</td>
<td>7999</td>
</tr>
<tr>
<td>Allen</td>
<td>7812</td>
</tr>
<tr>
<td>James</td>
<td>7661</td>
</tr>
<tr>
<td>Martin</td>
<td>6332</td>
</tr>
<tr>
<td>Ward</td>
<td>7229</td>
</tr>
<tr>
<td>Turner</td>
<td>1231</td>
</tr>
</tbody>
</table>
```
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
Recursive SQL from a Performance Perspective

par Thomas Baumann  La Mobilière
thomas.baumann@mobi.ch