

## Take Your SQL to the Next Level – Harnessing the Power of SQL SELECTs

Kent Milligan  
*DB2 Center of Excellence*  
*IBM STG Lab Services*



Information Management

© 2011 IBM Corporation

## New Wiki for DB2 Enhancements via PTF

- Regularly check (or Subscribe) to the DB2 for i Updates Wiki!
  - Contains details on new PTFs that deliver new DB2 capabilities
  - Examples:
    - CANCEL\_SQL system stored procedure
    - PROGRAM NAME keyword for controlling SQL Triggers Program Name
    - SQL Query Engine 6.1 support for Logical File on FROM clause
  - Wiki URL:  
<https://www.ibm.com/developerworks/ibmi/techupdates/db2>
- The wiki is part of a new IBM i zone in IBM developerWorks  
<https://www.ibm.com/developerworks/ibmi/>

## Agenda

- Views
- Subqueries
- Common Table Expressions
- Derived Tables
- Other Advanced SQL Features

## Challenge – Simplicity & Speed



## Views

- Possible applications

- Rename columns
- Mask complexity from users
- Ensure proper data conversion or derivations
- Provide solutions that can't be solved without a view
- Control access to sensitive data
- Insulate applications from changes to the physical table

## Views – Mask Complexity

```
CREATE VIEW JoinView(orderyear, ordermonth, orderdate, country,
    last_name, part_name, supplier_name, quantity, revenue)
AS SELECT t.year,t.month,i.orderdt,c.country,c.cust
    p.part,s.supplier,i.quantity,i.revenue
FROM item_fact i
    INNER JOIN part_dim p ON (i.partid = p.partid)
    INNER JOIN time_dim t ON (i.orderdt = t.datekey)
    INNER JOIN cust_dim c ON (i.custid = c.custid)
    INNER JOIN supp_dim s ON (i.suppdt = s.suppdt)
WHERE t.year > 2005
```



```
SELECT * FROM JoinView
WHERE orderyear = 2009 AND ordermonth IN (10,11,12)
```

## Views – Solve “Unsolvable” Problems

- Provide solution for problems that cannot be solved with a single statement. Example: including detail and aggregate information in the same result row

```
CREATE VIEW column_length
(table_name, max_length, min_length, avg_length) AS
  SELECT table_name, MAX(length), MIN(length), AVG(length)
  FROM qsys2.syscolumns
  GROUP BY table_name
-----

SELECT table_name, column_name, ordinal_position, length,
       max_length, min_length, avg_length,
       length - avg_length
FROM qsys2.syscolumns S, column_length V
WHERE S.table_name = V.table_name
      AND table_name IN ('CUSTOMER_MASTER', 'ORDER_MASTER')
ORDER BY 1, 3
```

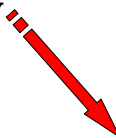
## Views – Control Data Access

- Remove sensitive columns from the view and remove user access to the table object

```
CREATE VIEW EmpDirectory
  AS SELECT firstname, lastname, phoneno,
         workdept, jobtitle
  FROM employee
```

## Views – Control Data Access

```
CREATE VIEW secureDept(division,deptname,lname,fname,salary) AS
  SELECT division, deptname, lname, fname,
         protectSal(deptno,salary) AS salary
FROM employee a, org b
WHERE a.deptno = b.deptno AND division IN
      (SELECT division FROM org c, staff d
       WHERE c.deptno = d.deptno AND
            emp_userid = SESSION_USER)
```



```
CREATE FUNCTION protectSal
(indept SMALLINT, insalary DECIMAL(7,2))
RETURNS DECIMAL(7,2)
LANGUAGE SQL
BEGIN
  DECLARE mymgrflag CHAR(1);
  SELECT '1' INTO mymgrflag FROM staff
  WHERE usrprf = SESSION_USER AND
         jobtitle='Mgr' AND dept=indept;
  IF mymgrflag = '1'
  THEN RETURN insalary ;
  ELSE RETURN NULL ;
END IF;
END
```

## Subqueries

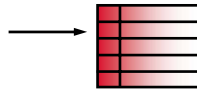
- Possible applications
  - Compare with value(s) dynamically extracted from other table(s)
    - Single value
    - List of values
    - Existence of a value
  - Providing values

## Subqueries = Join?

```
SELECT last_name
FROM employee
WHERE status='PT'
AND deptnum IN
  ( SELECT deptno FROM location WHERE floornum = 2)
```

### Step 1

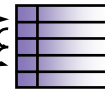
Select row  
from outer query



### Step 2

Probe into the  
inner query result set

Repeat  
Steps 1-2 for  
each row in  
outer query



*Join Approximation:*

```
SELECT last_name
FROM employee e, ( SELECT DISTINCT deptno FROM location WHERE floornum=2) d
WHERE ON e.deptnum = d.deptno AND status='PT'
```

## Subqueries - Non-correlated & Correlated

### ▪ Non-correlated

- Subquery (along with any of its inner) is autonomous
- Example:

```
SELECT last_name FROM employee
WHERE deptnum IN
  (SELECT deptno FROM location WHERE floornum = 3)
```

### ▪ Correlated

- Dependent on outer row for evaluation because of correlated reference
- Example:

```
SELECT last_name FROM employee x
WHERE salary >
  (SELECT AVG(salary) FROM employee y
   WHERE x.deptnum = y.deptnum )
```

## Subqueries – List of Values Comparison

- Additional operators (ALL, SOME, ANY) can be used to provide further stipulations on the list comparison

```
SELECT order_id FROM orders
WHERE order_qty >=
      ALL (SELECT a.order_qty FROM orders_archive)
```

## Subqueries – Existence Comparisons

- Conditionally process rows in outer Select statement based on the existence of rows returned by subquery
  - EXISTS
  - NOT EXISTS

```
SELECT contact_name, contact_phone FROM contact
WHERE EXISTS(
      SELECT '1' FROM customer
      WHERE customer_state = 'MN' AND
            contact.custid = customer.custid )
```

## Subqueries – Single Value Comparison

- The single value being compared is often the result of a column-function

```
SELECT deal_date FROM analytics_input
WHERE cust_id = 4
AND deal_date =
  (SELECT MIN(deal_date) FROM analytics_input
  WHERE deal_date BETWEEN
    (SELECT (MAX(import_date))- 10 days FROM scores)
    AND
    (SELECT (MAX(import_date)) FROM scores) )
```

## Subqueries – Update one table from another table

- Subqueries are not limited to SELECT statements

```
UPDATE employee e SET emprate =
  (SELECT cmprate FROM cpany c WHERE c.jobcode = e.empcode)
WHERE EXISTS (SELECT '1' FROM cpany cc
  WHERE cc.jobcode=e.empcode)
```

## Common Table Expressions (CTEs)

### ▪ Possible applications

- Breaking a report into logical steps
  - Improve readability
  - Reduce usage of physical work tables
- Dealing with legacy data models
- Recursive SQL
- Enabling a query to be re-used within a query

## CTEs – Logical Step-by-Step

- Break query into logical steps when query requires multiple SQL statements
  - Improved readability
  - Removes SQL view management, host variables also supported

```
WITH staff (deptno, empcount) AS
(SELECT deptno, COUNT(*) FROM employee
 WHERE division = :div_var GROUP BY deptno)
SELECT deptno, empcount FROM staff
WHERE empcount =
  (SELECT MAX(empcount) FROM staff)
```

## CTEs – Logical Step-by-Step & Work Tables

- CTE logical step processing can eliminate the need for temporary work tables

```

DECLARE GLOBAL TEMPORARY TABLE t1 AS
(SELECT shipdate, customer, phone, orderkey, linenumber
 FROM item_fact i, cust_dim c
 WHERE c.custkey=i.custkey AND discount=0.08) WITH DATA;

DECLARE GLOBAL TEMPORARY TABLE t2 AS
(SELECT customer, phone, orderkey, linenumber, year, quarter
 FROM t1, starlg.time_dim t
 WHERE t.datekey=shipdate ) WITH DATA;

...

```

## CTEs – Logical Step-by-Step & Work Tables (Continued)

- CTE provides the same step-by-step approach without the overhead of populating physical tables

```

WITH t1 AS
(SELECT shipdate, customer, phone, orderkey, linenumber
 FROM item_fact i, cust_dim c
 WHERE c.custkey = i.custkey AND discount=0.08),
     t2 AS
(SELECT customer, phone, orderkey, linenumber, year, quarter
 FROM t1, starlg.time_dim t
 WHERE t.datekey = shipdate)
SELECT * FROM t2;

```

### CTEs - Legacy Data Model Support

- Also useful when trying to join across data stored in a single table with different row types

```
WITH tablescans AS (SELECT qqjfld, qqcnt, qqrest, qqtotr
                    FROM dbmonout1 WHERE qqrid=3000)

SELECT SUM(qqi6) "Total Time", COUNT(*) "Times Run",
       a.qqcnt, INTEGER(AVG(b.qqrest)) "Est Rows Selected",
       INTEGER(AVG(b.qqtotr)) "Total Rows in Table", qq1000
FROM dbmonout1 a, tablescans b
WHERE qqrid=1000 AND a.qqjfld = b.qqjfld
      AND qqc21 IN ('OP','SI')
GROUP BY a.qqcnt, qq1000
```

Total Time	Times Run	QQUCNT	Est Rows Selected	Table Rows	QQ1000
200608	3	1011	113	6317	select distinct CATENTR
165480	3	1036	113	6317	select distinct CATENTR
91400	1	1040	113	6317	select distinct CATENTR
21568	1	1035	0	0	SELECT T1.ORDIADJUE

### CTEs: Recursive SQL

- Recursive Common Table Expressions

- Useful for navigating tables where rows are inherently related to other rows in same table
  - Bill of Materials
  - Organizational Hierarchies
  - ...

Example:

```
WITH emp_list (level, empid, name) AS
  (SELECT 1, empid, name FROM emp
   WHERE name = 'Carfino'
  UNION ALL
   SELECT o.level + 1, next_layer.empid, next_layer.name
   FROM emp as next_layer, emp_list o
   WHERE o.empid = next_layer.mgrid )
SELECT level, name FROM emp_list
```

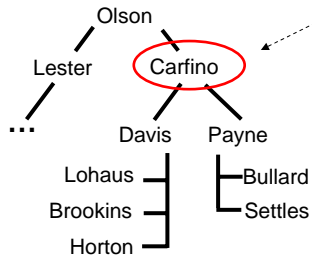
1 - Initializing the query

2 - Recursive reference to the next level

3 - Start the query & return final results

### CTEs: Recursive SQL Example – Initialization Phase

**SELECT 1, empid, name FROM emp  
WHERE name = 'Carfino'**

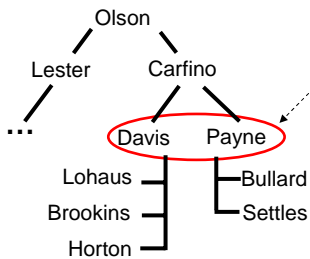


```

CREATE TABLE emp(
  empid INTEGER PRIMARY KEY,
  name VARCHAR(10),
  salary DECIMAL(9, 2),
  mgrid INTEGER)
  
```

### CTEs: Recursive SQL Example – Recursion Pass #1

**SELECT o.level + 1, next\_layer.empid, next\_layer.name  
FROM emp as next\_layer, emp\_list o  
WHERE o.empid = next\_layer.mgrid**

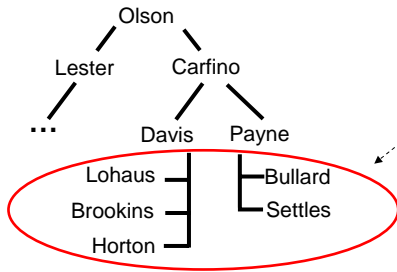


```

CREATE TABLE emp(
  empid INTEGER PRIMARY KEY,
  name VARCHAR(10),
  salary DECIMAL(9, 2),
  mgrid INTEGER)
  
```

CTEs: Recursive SQL Example – Recursion Pass #2

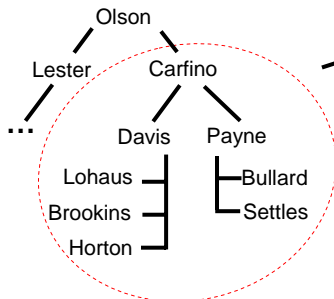
```
SELECT o.level + 1, next_layer.empid, next_layer.name
FROM emp as next_layer, emp_list o
WHERE o.empid = next_layer.mgrid
```



```
CREATE TABLE emp(
empid INTEGER PRIMARY KEY,
name VARCHAR(10),
salary DECIMAL(9, 2),
mgrid INTEGER)
```

CTEs: Recursive SQL Example – Final Results

```
WITH emp_list (level, empid, name) AS
SELECT 1, empid, name FROM emp
WHERE name = 'Carfino'
UNION ALL
SELECT o.level + 1, next_layer.empid, next_layer.name
FROM emp as next_layer, emp_list o
WHERE o.empid = next_layer.mgrid
SELECT level, name FROM emp_list
```



00001	NAME
1	Carfino
2	Payne
2	Davis
3	Settles
3	Bullard
3	Lohaus
3	Horton
3	Brookins

### CTEs: Recursive SQL Considerations

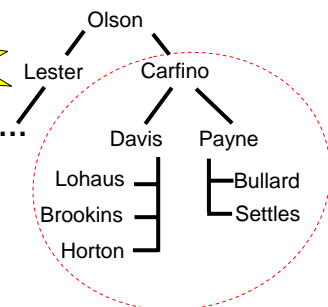
- Breadth First is default processing order, Search clause can be used to change processing order – if the Order By clause also specified
  - Search clause adds extra overhead, only use as required

```
WITH emp_list(level,empid, name) AS
(SELECT 1,empid, name FROM emp
 WHERE name = 'Carfino')
UNION ALL
SELECT level+1, next_layer.empid, next_layer.name
 FROM emp as next_layer, emp_list p
 WHERE p.empid = next_layer.mgrid)
SEARCH DEPTH FIRST BY empid SET seqcol
SELECT level,name FROM emp_list ORDER BY seqcol
```

Depth Search Output

00001	NAME
1	Carfino
2	Davis
3	Brookins
3	Lohaus
3	Horton
2	Payne
3	Bullard
3	Settles

### Recursive SQL Simplification - CONNECT BY



```
SELECT LEVEL, name
 FROM emp
START WITH name = 'Carfino'
CONNECT BY mgrid = PRIOR empid
```

```
WITH emp_list (level, empid, name) AS
SELECT 1, empid, name FROM emp
 WHERE name = 'Carfino'
UNION ALL
SELECT o.level+1, next_layer.empid, next_layer.name
 FROM emp as next_layer, emp_list o
 WHERE o.empid = next_layer.mgrid )
SELECT level, name FROM emp_list
```

1	Carfino
2	Davis
3	Brookins
3	Lohaus
3	Horton
2	Payne
3	Bullard
3	Settles

CTEs – NO Query Reuse Example

```

SELECT a.storename, SUM(a.totalsales) AS TOTALSALES,
      COUNT(DISTINCT a.transid) AS ALLTRANSACTIONS,
      COUNT(DISTINCT b.transid) AS COMPLETEDTRANSACTIONS,
#1 FROM sales A
      LEFT OUTER JOIN
#2      (SELECT DISTINCT storename, transid FROM sales
        WHERE transdate BETWEEN '20090101' AND '20090131'
        AND completed=1) b
      ON a.storename = b.storename
WHERE a.transdate BETWEEN '20090101' AND '20090131'
      AND a.storename IN
#3      (SELECT a.storename
        FROM sales C
        WHERE c.transdate BETWEEN '20090101' AND '20090131'
        AND c.custid=12345)
GROUP BY a.storename

```

## CTEs – Query Reuse Example

```

WITH sales_in_range AS
(SELECT storename, price, transid, completed, custid
 FROM sales
 WHERE transdate BETWEEN '20090101' AND '20090131')
SELECT a.storename, SUM(a.price) AS TOTALSALES,
      COUNT(DISTINCT a.transid) AS ALLTRANSACTIONS,
      COUNT(DISTINCT b.transid) AS COMPLETEDTRANSACTIONS
FROM sales_in_range a
      LEFT OUTER JOIN
      (SELECT DISTINCT storename, transid
        FROM sales_in_range
        WHERE completed=1) b
      ON a.storename = b.storename
WHERE a.storename IN
      (SELECT c.storename FROM sales_in_range c
        WHERE c.custid=12345)
GROUP BY a.storename

```

## Derived Tables

- Possible applications, similar to CTEs ...
  - Some differences...
    - Harder/Easier to read?!?
    - No support for recursive SQL
    - Less optimizer reuse of derived table data set

## Derived Tables – CTE Comparison #1

```
SELECT SUM(qqi6) "Total Time", COUNT(*) "Times Run",
       a.qqcnt, INTEGER(AVG(b.qqrest)) "Est Rows Selected",
       INTEGER(AVG(b.qqtotr)) "Total Rows in Table", qq1000
FROM dbmonout1 a, (SELECT qqjfld, qqcnt, qqrest, qqtotr
                   FROM qqpl.clitest WHERE qqrid=3000) b
WHERE qqrid=1000 AND a.qqjfld = b.qqjfld
       AND qqc21 IN ('OP','SI') GROUP BY a.qqcnt, qq1000
```

```
WITH tablescans AS (SELECT qqjfld, qqcnt, qqrest, qqtotr
                    FROM dbmonout1 WHERE qqrid=3000)
SELECT SUM(qqi6) "Total Time", COUNT(*) "Times Run",
       a.qqcnt, INTEGER(AVG(b.qqrest)) "Est Rows Selected",
       INTEGER(AVG(b.qqtotr)) "Total Rows in Table", qq1000
FROM dbmonout1 a, tablescans b
WHERE qqrid=1000 AND a.qqjfld = b.qqjfld
       AND qqc21 IN ('OP','SI')
GROUP BY a.qqcnt, qq1000
```

## Derived Tables – CTE Comparison #2

```
SELECT d1.deptno, d1.empcount FROM
  (SELECT deptno, COUNT(*) as empcount
   FROM employee GROUP BY deptno) d1
WHERE d1.empcount =
  (SELECT MAX(d2.empcount) FROM
   (SELECT deptno, COUNT(*) AS empcount
    FROM employee GROUP BY deptno) d2
  )

WITH staff (deptno, empcount) AS
  (SELECT deptno, COUNT(*) FROM employee
   GROUP BY deptno)
SELECT deptno, empcount FROM staff
WHERE empcount = (SELECT MAX(empcount) FROM staff)
```

## Other Advanced SQL Features

## Set Operators

- EXCEPT & INTERSECT set operators

- Return all rows that are in t1, but not t2

```
(SELECT cusnum FROM orders2008)
  EXCEPT DISTINCT
  (SELECT cusnum FROM orders2009)
```

- All rows that exist in both t1 & t2

```
(SELECT cusnum FROM orders2008)
  INTERSECT DISTINCT
  (SELECT cusnum FROM orders2009)
```

## Conditional Summary Processing

- CASE expression provides lots of flexibility to create queries with conditional summaries and calculations

```
SELECT storename, SUM(price) AS Total_Sales,
       SUM(CASE WHEN cardHolder='Y'
              THEN SUM(price) ELSE 0 END) AS Card_Sales
       SUM(CASE WHEN cardHolder='N'
              THEN SUM(price) ELSE 0 END) AS NonCard_Sales
FROM sales_trans
GROUP BY storename
```

StoreName	Total_Sales	Card_Sales	NonCard_Sales
BIG AL'S	1,750,000	1,100,000	650,000
...	...	...	...

“Super” Summary Processing - ROLLUP

```
SELECT country, region, SUM(sales)
FROM trans
GROUP BY ROLLUP (country, region)
```

GROUP BY  
country, NULL



Country	Region	Sum(Sales)
Canada	-	100,000
Canada	NW	100,000
U.S.A.	-	3,250,000
U.S.A.	NE	450,000
U.S.A.	NW	940,000
U.S.A.	SE	550,000
U.S.A.	SW	1,310,000
-	-	3,350,000

GROUP BY  
NULL, NULL



“Super” Summary Processing - CUBE

```
SELECT country, region, SUM(sales)
FROM trans
GROUP BY CUBE (country, region)
```

GROUP BY NULL, region



GROUP BY NULL, NULL



GROUP BY country, NULL



Country	Region	Sum(Sales)
-	NE	450000
-	NW	1040000
-	SE	550000
-	SW	1310000
-	-	3350000
Canada	-	100000
U.S.A.	-	3250000
Canada	NW	100000
U.S.A.	NE	450000
U.S.A.	NW	940000
U.S.A.	SE	550000
U.S.A.	SW	1310000

“Super” Summary Processing - GROUPING SETS

**SELECT country, region, store, SUM(sales)**  
**FROM trans**  
**GROUP BY GROUPING SETS ((country, region), (country, store))**

GROUP BY  
COUNTRY, REGION

GROUP BY  
COUNTRY, STORE

Country	Region	Store	Sum(Sales)
Canada	NW	-	100,000
U.S.A.	NE	-	450,000
U.S.A.	NW	-	940,000
U.S.A.	SE	-	550,000
U.S.A.	SW	-	1,310,000
Canada	-	Dougs	100,000
U.S.A.	-	Mariahs	350,000
U.S.A.	-	KMs	770,000
U.S.A.	-	Jennas	400,000
U.S.A.	-	Adrians	500,000
U.S.A.	-	Joshs	300,000
U.S.A.	-	TZs	200,000
U.S.A.	-	Maddies	210,000

Querying Legacy Date fields – “Date conversion table”

- Join to pre-populated table instead of converting legacy date fields to “real” SQL date formats



- Benefits
  - Performance may be faster than legacy conversion routines & user-defined functions
    - Pre-populated table is small
    - Assumes right indexes in place for optimizer to use
  - Applications get access to a wide variety of date format values for a single date value
  - SQL Views can be used to mask join complexity

### Querying Legacy Date fields – “Date conversion table”

ORDERS legacy file →

ORDER	CUST	ORDDAT	SHPDAT	SHPVIA	ORDSTS	ORDAMT	TOTLIN
1715810	H4541	2152005	6302005	UPS	1	933346.39	2
1563685	R1948	2202005	8042005	Mule Train	2	0.00	0
7195900	Q7881	2232005	5022005	Pick Up	2	0.00	0
8854635	S1511	2232005	9102005	Train	1	118086.88	2
6694902	X8863	2242005	7192005	Camel	2	0.00	0
8054679	F4327	2272005	9112005	Pneumatic Tube	2	0.00	0
527879	C3233	2282005	5022005	US Mail	1	997524.71	3
4011038	G1496	2282005	10162005	Train	1	865114.60	2
5417918	J3825	2282005	10022005	FedEx	1	84608.50	2
9456994	N2796	2282005	7012005	Train	2	0.00	0
3526155	Q7881	3012005	8062005	UPS	1	194660.67	2

Date conversion table ↓

JOIN

DC_DATE	DC_MDYY_DEC	DC_YEAR	DC_DOW	DC_DOY	DC_MM	DC_DD	DC_DAY_NAME	DC_WEEKEND	DC_FISCAL_YEAR	DC_FISCAL_QUARTER	DC_MONTH_NAME	DC_MONTH_ABRV	DC_SEASON
2005-02-18	2182005	2005	5	49	2	18	Friday	N	2005	2	February	FEB	Winter
2005-02-19	2192005	2005	6	50	2	19	Saturday	N	2005	2	February	FEB	Winter
2005-02-20	2202005	2005	7	51	2	20	Sunday	Y	2005	2	February	FEB	Winter
2005-02-21	2212005	2005	1	52	2	21	Monday	Y	2005	2	February	FEB	Winter
2005-02-22	2222005	2005	2	53	2	22	Tuesday	N	2005	2	February	FEB	Winter
2005-02-23	2232005	2005	3	54	2	23	Wednesday	N	2005	2	February	FEB	Winter
2005-02-24	2242005	2005	4	55	2	24	Thursday	N	2005	2	February	FEB	Winter
2005-02-25	2252005	2005	5	56	2	25	Friday	N	2005	2	February	FEB	Winter
2005-02-26	2262005	2005	6	57	2	26	Saturday	N	2005	2	February	FEB	Winter
2005-02-27	2272005	2005	7	58	2	27	Sunday	Y	2005	2	February	FEB	Winter
2005-02-28	2282005	2005	1	59	2	28	Monday	Y	2005	2	February	FEB	Winter
2005-03-01	3012005	2005	2	60	3	1	Tuesday	N	2005	2	March	MAR	Winter
2005-03-02	3022005	2005	3	61	3	2	Wednesday	N	2005	2	March	MAR	Winter
2005-03-03	3032005	2005	4	62	3	3	Thursday	N	2005	2	March	MAR	Winter

### Querying Legacy Date fields – “Date conversion table”

#### Steps for “Date Conversion Table” methodology

1. Create the date conversion table (Single SQL statement)
2. Populate the date conversion table (SQL stored procedure – example available)
3. Add date conversion table join to applications
4. Validate right indexes are in place for the join



Technique described in detail in: [DB2 Web Query Redbook \(SG24-7214-01\)](http://ibm.com/redbooks)  
<http://ibm.com/redbooks>

## Additional Information

- DB2 for i Websites
  - Home Page: [ibm.com/systems/i/db2](http://ibm.com/systems/i/db2)
  - DeveloperWorks Zone: [ibm.com/developerworks/db2/products/db2i5OS](http://ibm.com/developerworks/db2/products/db2i5OS)
  - Porting Zone: [ibm.com/partnerworld/i/db2porting](http://ibm.com/partnerworld/i/db2porting)
- Newsgroups & Forums
  - USENET: comp.sys.ibm.as400.misc, comp.databases.ibm-db2
  - DeveloperWorks: <https://www.ibm.com/developerworks/forums/forum.jspa?forumID=292>
  - System i Network DB2 Forum: <http://forums.systeminetwork.com/isnetforums/>
- Education Resources - Classroom & Online
  - [ibm.com/systemi/db2/gettingstarted.html](http://ibm.com/systemi/db2/gettingstarted.html)
  - [ibm.com/partnerworld/wps/training/i5os/courses](http://ibm.com/partnerworld/wps/training/i5os/courses)
- DB2 for i Publications
  - White Papers: [ibm.com/partnerworld/wps/whitepaper/i5os](http://ibm.com/partnerworld/wps/whitepaper/i5os)
  - Online Manuals: [ibm.com/systems/i/db2/books.html](http://ibm.com/systems/i/db2/books.html)
  - DB2 for i Redbooks (<http://ibm.com/redbooks>)
    - [Getting Started with DB2 Web Query for System i \(SG24-7214\)](#)
    - [OnDemand SQL Performance Analysis ... in V5R4 \(SG24-7326\)](#)
    - [Preparing for and Tuning the SQL Query Engine on DB2 for i5/OS \(SG24-6598\)](#)
    - [Modernizing iSeries Application Data Access \(SG24-6393\)](#)

# THANKS!