Tony's Top Db2 SQL Enhancements for Db2 V9-V12

by Tony Andrews, Themis Inc.
tandrews@themisinc.com

Twitter: 

Themis
Leaders in IT Education

Dave's Favorite of Tony's Top Db2 SQL Enhancements for Db2 V9-V12

by David Simpson, Themis Inc.
dsimpson@themisinc.com

Twitter: 

Themis
Leaders in IT Education

Questions?

- I will try my best to get to some questions towards the end of the webinar.
- You can submit questions by typing into the questions area of your webinar control panel.
- Any questions not answered due to time constraints can be answered afterward via an email.

Agenda - Objectives

- Highlight the many new SQL enhancements and features from V9 to V12or your five items
- Highlight the SQL enhancements that were especially helpful and efficient for developers
- Take away many great SQL examples

Following are some of the many application features from Db2 V8:

1) More Stage 1 predicates
2) Multi Row Fetch, Update, and Insert
3) Multiple Distincts
4) Expressions in the ‘Group By’
5) Common Table Expression
6) Dynamic Scrollable Cursors
7) Sequences versus Identity Columns
8) Materialized Query Tables (MQTs)
9) Recursive SQL
10) More efficient use of indexes. Forward and Backward scans
11) New XML functions and datatypes
12) New ‘Get Diagnostics’ for warning and error information
13) Select from an Insert statement
14) Scalar Fullselect within a ‘Select’, ‘Case’, Update, etc.

Following are some of the many application features in Db2 V9:

1) Set operations ‘Intersect’ and ‘Except’
2) Merge statement for ‘Upsert’ processing. Insert or Update
3) OLAP features for Ranking and Numbering of data
4) Native SQL Stored Procedures
5) ‘Instead of’ Triggers
6) New support and SQL for XML data
7) IBM Data Studio
8) Distinct sort avoidance with non unique indexes
9) Indexing on Expressions
10) Statistics on Views
11) Skipped locked data
12) Truncate statement

Following are some of the many application features in Db2 V9:

13) Optimizer Changes
14) Timestamp auto update for inserts and Updates
15) Optimistic locking
16) New DECFLOAT datatype
17) Select from Update or Delete getting old or new values
18) Fetch First, Order BY within subqueries
19) REOPT AUTO (Dynamic SQL)
20) Data Studio for Native Stored Procedures

Following are some of the many application features in Db2 V10:

1) Ranking, Moving Sum and AVG
2) Variable Inserts and Updates
3) Extender indicator values
4) Increased timestamp precision
5) Currently committed
6) Hash access design
7) SQL PL enhancements
8) XML Enhancements
9) Column Masking
10) Temporal Tables
11) Many Optimization improvements
12) Index included columns
13) Etc… Etc…. Etc ….

Following are some of the many application features in Db2 V11:

1) Global Variables
2) Transparent Archiving
3) SQL Grouping Sets and Rollups
4) SQL PL variable arrays
5) More SQL PL enhancements
6) More XML enhancements
7) Global Temp Table enhancements
8) New optimization features and improvements
9) Explain table additions
10) Etc… Etc…. Etc ….

Following are some of the many application features in Db2 V12:

1) Additional support for triggers
2) Pagination improvements
3) Additional support for arrays
4) MERGE improvements
5) Piece-wise deletes
6) Optimization improvements
7) Etc… Etc…. Etc ….

V9 MERGE Statement

```
MERGE INTO ITEM I
   USING (VALUES (1, 'WIDGIT') )
      AS NEWITEM (ITEMNO, ITEMNAME)
  ON I.ITEMNO = NEWITEM.ITEMNO
WHEN MATCHED THEN
   UPDATE SET ITEMNAME = NEWITEM.ITEMNAME
WHEN NOT MATCHED THEN
   INSERT (ITEMNO,ITEMNAME)
     VALUES (NEWITEM.ITEMNO, NEWITEM.ITEMNAME);
```

Set of values for a row

Establishes a match for an existing row

Update action if row is already present

Insert action if no row is found

V12 MERGE Enhancements

```
MERGE INTO EMP E1
USING (SELECT EMPNO, SALARY
       FROM EMP
       WHERE DEPTNO = 'A00') AS E2
ON (E1.EMPNO = E2.EMPNO)
  WHEN MATCHED AND E1.SALARY >= 50000 THEN
    UPDATE SET E1.SALARY = E1.SALARY * 500
  WHEN MATCHED AND E1.SALARY < 40000 THEN
    DELETE
  WHEN MATCHED AND E1.SALARY < 50000 THEN
    UPDATE SET E1.SALARY = E1.SALARY * 1.1
  ELSE
    IGNORE
```

Allows for a Select to specify input data
Allows for multiple conditions on WHEN
Allows for delete option

V12 MERGE Enhancement

Allows for source data to be different
Allows for an update or delete to affect more than 1 row

```
MERGE INTO EMPPROJACT EPA
USING (SELECT EMPNO, SALARY
        FROM EMP
        WHERE DEPTNO = 'A00') AS E
ON (EPA.EMPNO = E.EMPNO)
  WHEN MATCHED AND E.SALARY >= 50000 THEN
    UPDATE SET EPA.EMPTIME = 99
  WHEN MATCHED AND E.SALARY < 40000 THEN
    DELETE
  WHEN NOT MATCHED THEN
    INSERT (EMPNO, PROJNO, ACTNO)
      VALUES (E.EMPNO, 'P99999', '123')
```

V9 Final Table: Great for Auto Key Values

```
INSERT INTO CUSTOMER
        (CUSTNO, CUST_NAME, LOCATION)
        VALUES
        (NEXT VALUE FOR CUSTKEY,
                'Themis Inc.', 'NJ')
```

CUSTKEY = Sequence Object

```
SELECT CUSTNO FROM FINAL TABLE
(INSERT INTO CUSTOMER
        (CUSTNO, CUST_NAME, LOCATION)
        VALUES
        (NEXT VALUE FOR CUSTKEY,
                'Themis Inc.', 'NJ')
)
```

Identity Column - Review

```
CREATE TABLE CUSTOMER
(CUSTNO          INTEGER     NOT NULL
        GENERATED ALWAYS
        AS IDENTITY
        (START WITH 200
        INCREMENT BY 1
        CACHE 50)
,CUST_NAME       CHAR(20)    NOT NULL
,LOCATION        CHAR(2)     )
```

Sequence Info will be tied directly to CUSTNO Column

V9 Final Table: Great for Identity Column Usage

```
INSERT INTO CUSTOMER
        (CUST_NAME,
LOCATION)
        VALUES
        ('Themis Inc.', 'NJ')
```

No mention of CUSTNO Value is supplied by Db2

```
SELECT CUSTNO FROM FINAL
TABLE
(INSERT INTO CUSTOMER
        (CUST_NAME, LOCATION)
        VALUES
        ('Themis Inc.', 'NJ')
)
```

V9 SELECT from MERGE: To get ‘I’ or ‘U’ back

```
SELECT ITEMNAME, UPD_IND FROM FINAL TABLE
(MERGE INTO ITEM I
   INCLUDE (UPD_IND CHAR(1))
   USING (VALUES (1, 'SOCKET') )
      AS NEWITEM (ITEMNO, ITEMNAME)
  ON I.ITEMNO = NEWITEM.ITEMNO
 WHEN MATCHED THEN
   UPDATE SET ITEMNAME = NEWITEM.ITEMNAME,
              UPD_IND = 'U'
 WHEN NOT MATCHED THEN
   INSERT (ITEMNO,ITEMNAME,UPD_IND)
     VALUES (NEWITEM.ITEMNO,
NEWITEM.ITEMNAME,'I') )
```

ITEMNAME	UPD_IND
SOCKET	U

V9 Rank Function

Usage:

```
SELECT LASTNAME, SALARY,
   RANK() OVER (ORDER BY SALARY DESC) AS R
FROM EMP
```

LASTNAME	SALARY	R
HAAS	52750.00	1
HEMMINGER	46500.00	2
LUCCHESI	46500.00	2
THOMPSON	41250.00	4

V9 Dense Rank Function

Usage:

```
SELECT LASTNAME, SALARY,
   DENSE_RANK() OVER (ORDER BY SALARY DESC) AS R
FROM EMP
```

LASTNAME	SALARY	R
HAAS	52750.00	1
HEMMINGER	46500.00	2
LUCCHESI	46500.00	2
THOMPSON	41250.00	3

V9 Row Numbering

Usage:

```
SELECT LASTNAME, SALARY,
   ROW_NUMBER() OVER (ORDER BY SALARY DESC) AS R
FROM EMP
```

LASTNAME	SALARY	R
HAAS	52750.00	1
HEMMINGER	46500.00	2
LUCCHESI	46500.00	3
THOMPSON	41250.00	4

SQL Percentiles

It's easy to get ranking numbers, but how do we use those with percentiles (top 5%, bottom 10%, etc.)?

Percentiles:
1 You need to know total number in data
2 You need to know your rank within

Example:
1 There are 600 students in your class
2 You are ranked 120 based on GPA
3 (120/600)*100=20, AND 100-20=80
4 80^{th} percentile = you are in top 20%

Example 1: EMP table contains 33 employees. Retrieve those emps making top 5% of salaries.

```
--
--  THIS QUERY CALCULATES PERCENTILE IN THE CTE SO WE CAN THEN
--   ADD 'WHERE' LOGIC ON IT.
--
  WITH TEMP_RANK AS
   (SELECT EMPNO, LASTNAME, SALARY,
       SMALLINT(DENSE_RANK () OVER (ORDER BY SALARY ASC) ) AS RANK,
       (SMALLINT(DENSE_RANK () OVER (ORDER BY SALARY ASC) ) / 33.0)
         * 100 AS PERCENTILE
    FROM THEMIS90.EMP
    ORDER BY SALARY ASC)
  SELECT EMPNO, LASTNAME, SALARY, RANK,
       PERCENTILE
  FROM TEMP_RANK
  WHERE PERCENTILE >= 95
  ORDER BY PERCENTILE DESC
;
```

Could be a subquery that counts rows

SQL Percentiles

Example 1: EMP table contains 33 employees. Retrieve those emps making top 5% of salaries.

	EMPNO	LASTNAME	SALARY	RANK	PERCENTILE
1	000010	HAAS	52750.00	32	96.969696900
2	000011	HAAS	52750.00	32	96.969696900

Sampling of all data and their percentiles

	EMPNO	LASTNAME	SALARY	RANK	PERCENTILE
1	000010	HAAS	52750.00	32	96.969696900
2	000011	HAAS	52750.00	32	96.969696900
3	000110	LUCCHESI	46500.00	31	93.939393900
4	000020	THOMPSON	41250.00	30	90.909090900
5	000050	GEYER	40175.00	29	87.878787800
6	000030	KWAN	38250.00	28	84.848484800
7	000070	PULASKI	36170.00	27	81.818181800
8	000060	STERN	32250.00	26	78.787878700
9	000220	LUTZ	29840.00	25	75.757575700
10	000090	HENDERSON	29750.00	24	72.727272700
11	000120	O'CONNELL	29250.00	23	69.696969600
12	000240	MARINO	28760.00	22	66.666666600
13	000140	NICHOLLS	28420.00	21	63.636363600
14	000200	BROWN	27740.00	20	60.606060600

V9 Row Change Timestamp

```
CREATE TABLE EMP2
(EMPNO          CHAR(06)        NOT NULL
,LAST_NAME      CHAR(20)        NOT NULL
,FIRST_NAME     CHAR(20)        NOT NULL
,ZIP_CODE       CHAR(5)         NOT NULL
,BIRTH_DTE      DATE            NOT NULL
,UPD_TSP                TIMESTAMP NOT NULL
    GENERATED ALWAYS FOR EACH ROW
    ON UPDATE AS ROW CHANGE TIMESTAMP)
...
```

V9 Row Changed Timestamp

Now that the ROW CHANGE TIMESTAMP is defined for a column ….. you can code logic using the column. The following will get all rows that have

been changed in the last week.

```
SELECT …….
FROM EMP2
WHERE UPD_TSP  > CURRENT TIMESTAMP – 7 DAYS
```

RID Function

```
SELECT EMPNO, RID(EMP) AS RID, LASTNAME, SALARY
 FROM EMP
 WHERE EMPNO = '000030'
 ;
```

	EMPNO	RID	LASTNAME	SALARY
1	000030	8708	KWAN	38250.00

```
UPDATE EMP
SET SALARY = 100000.00
WHERE RID(EMP) = 8708
;
```

V9 Optimistic Locking in COBOL

```
EXEC SQL
       FETCH C1
       INTO :V-EMPNO,
            :v-UPD_TSP,
            :V-MY-RID,      ➔ BIGINT
            :V-UPD-TSP
END-EXEC.
...
EXEC SQL
      UPDATE EMP E
       SET LASTNAME = :NEW-LASTNAME
     WHERE RID(E) = :V-MY-RID
       AND UPD_TSP = :V-UPD-TSP
END-EXEC.
```

V10 Moving Sum / Avg

Example 1:

```
SELECT EMPNO, DEPTNO, SALARY,
  SUM (SALARY)
  OVER (
        ORDER BY SALARY ASC
        ROWS UNBOUNDED PRECEDING
       ) AS SUM_SAL
FROM EMP
```

EMPNO	DEPTNO	SALARY	SUM_SAL
000290	E11	15340.00	15340.00
000310	E11	15900.00	31240.00
000260	D21	17250.00	48490.00
000300	E11	17750.00	66240.00
000210	D11	18270.00	84510.00
000250	D21	19180.00	103690.00
000320	E21	19950.00	123640.00

V10 Moving Sum / Avg

Example 2:

```
SELECT DEPTNO, EMPNO, SALARY,
  SUM  (SALARY)
  OVER (PARTITION BY DEPTNO
        ORDER BY SALARY ASC
        ROWS UNBOUNDED PRECEDING
       ) AS SUM_SAL
FROM EMP
ORDER BY DEPTNO
```

	DEPTNO	EMPNO	SALARY	SUM_SAL
1	A00	000120	29250.00	29250.00
2	A00	000110	46500.00	75750.00
3	A00	000010	52750.00	128500.00
4	A00	000011	52750.00	181250.00
5	B01	000020	41250.00	41250.00
6	C01	000130	23800.00	23800.00
7	C01	000140	28420.00	52220.00
8	C01	000030	38250.00	90470.00

Timestamp Precision Enhancements

- Sub-second precision up to 12 decimal positions
- TIMESTAMP WITH TIMEZONE
- CURRENT TIMESTAMP

Timestamp Precision Enhancements

- Storage size can vary from 7 bytes (0 decimal places) to 13 bytes (12 decimal places)
- Casting always returns a precision of 6

```
SELECT CAST('2012-01-23-04.30.42.123456789‘  AS TIMESTAMP)
                AS TMS_CAST
FROM SYSIBM.SYSDUMMY1

Returns:  '2012-01-23-04.30.42.123456'
```

Current Timestamp Special Register

```
SELECT CURRENT TIMESTAMP                          AS CUR_TMS1,
       CURRENT TIMESTAMP(12)                      AS CUR_TMS12,
       CURRENT TIMEZONE                           AS CUR_TMZ,
       CURRENT TIMESTAMP WITH TIMEZONE            AS CUR_TMS3
FROM SYSIBM.SYSDUMMY1
```

CUR_TMS1	CUR_TMS12
2019-01-25-14.34.51.387362	2019-01-25-14.34.51.387362453234

CUR_TMZ	CUR_TMS3
-50000.	2019-01-25-14.34.51.387362-05:00

V10 Temporal Tables

Temporal: Pertaining to date and time

New table design options

Support of current and history data

New SQL query syntax

V10 Temporal Tables – System Time

```
CREATE TABLE EMP_TEMPORAL_CUR

(EMPNO     CHAR(6)                NOT NULL,
 FIRSTNME  VARCHAR(20)     NOT NULL,
 MIDINIT   CHAR(01)              NOT NULL,
 LASTNAME  VARCHAR(20)   NOT NULL,
 DEPTNO    CHAR(3)              NOT NULL,
 SALARY    DEC(9,2)              NOT NULL,
 BONUS     DEC(9,2)              NOT NULL,
 COMM      DEC(9,2)              NOT NULL,
 FROM_TS   TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
 TO_TS     TIMESTAMP(12)     NOT NULL GENERATED ALWAYS AS ROW END,
 TRANS_TS  TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS
                                                    TRANSACTION START ID,
 PERIOD   SYSTEM_TIME (FROM_TS, TO_TS)
) ;
```

V10 Temporal Tables – System Time

```
CREATE TABLE EMP_TEMPORAL_HIS

(EMPNO     CHAR(6)              NOT NULL,
 FIRSTNME  VARCHAR(20)   NOT NULL,
 MIDINIT   CHAR(01)             NOT NULL,
 LASTNAME  VARCHAR(20) NOT NULL,
 DEPTNO    CHAR(3)              NOT NULL,
 SALARY    DEC(9,2)             NOT NULL,
 BONUS     DEC(9,2)             NOT NULL,
 COMM      DEC(9,2)             NOT NULL,
 FROM_TS   TIMESTAMP(12) NOT NULL,
 TO_TS     TIMESTAMP(12) NOT NULL,
 TRANS_TS  TIMESTAMP(12) NOT NULL
) ;
```

```
ALTER TABLE MP_TEMPORAL_CUR
  ADD VERSIONING
  USE HISTORY TABLE
EMP_TEMPORAL_HIS
```

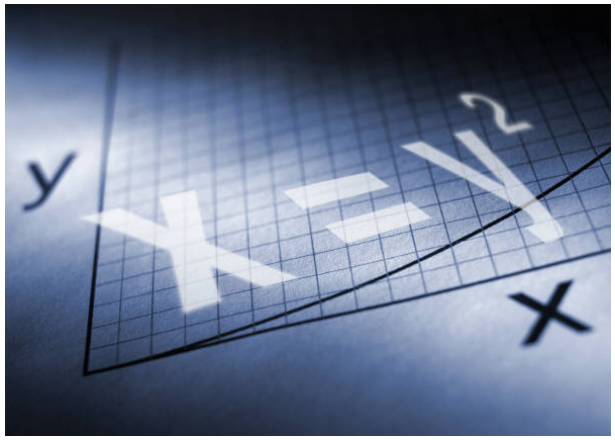
V10 Temporal Tables – Queries

```
SELECT * FROM EMP_TEMPORAL_CUR
 FOR SYSTEM_TIME AS OF '2018-01-30-13.46.12.107358109000'
WHERE EMPNO = '000010'
;
```

```
SELECT * FROM EMP_TEMPORAL_CUR
 FOR SYSTEM_TIME BETWEEN '2016-01-30-13.44.12.107358109000'
                     AND '2016-03-30-15.46.12.107358109000'
WHERE EMPNO = '000010'
;
```

V11 Global Variables

- Named memory variables in Db2 that may be accessed and modified by SQL statements
- Enable sharing of data between different SQL statements without an application facilitating the data transfer
- Values are unique to an application scope



Global Variable Creation

Choose a naming convention that identifies this as a global variable

```
CREATE VARIABLE    GV_TSP    TIMESTAMP
   DEFAULT CURRENT TIMESTAMP;

GRANT ALL PRIVILEGES ON VARIABLE GV_TSP
   TO PUBLIC;
```

Global Variable Usage

```
EXEC SQL
    SET GV_TSP = '2014-12-31.00.00.00.000000'
END-EXEC.
...
EXEC SQL
    DECLARE C1 CURSOR FOR
    SELECT CUSTNO       FROM CUSTOMER
      WHERE LAST_CALL < GV_TSP
END-EXEC.
...
EXEC SQL
    DECLARE C2 CURSOR FOR
    SELECT ORDERNO       FROM ORDER
       WHERE  ORDER_TSP < GV_TSP
END-EXEC.
```

Transparent Archiving

- Db2 10 introduced “Temporal Tables”
- Db2 11 adds the ability to automatically archive deleted data as an alternative to “System Temporal” Tables
- Like System Temporal Tables an archive table is used to capture deleted rows

V11 Archive Table

```
CREATE TABLE CUST
        (CUSTNO          INTEGER          NOT NULL,
         FIRSTNME        VARCHAR(12)      NOT NULL,
         MIDINIT                  CHAR(1)          NOT NULL,
         LASTNAME        VARCHAR(15)      NOT NULL,
         PHONENO         CHAR(12)         NOT NULL,
         LAST_CONTACT_DATE   DATE         NOT NULL,
         GENDER    CHAR(1),
         BIRTHDATE DATE,
         PRIMARY KEY(CUSTNO)   )
    IN THEMISDB.TSCUST
```

No timestamps needed

V11 Archive Table Usage

```
CREATE TABLE CUST_ARCHIVE
    LIKE CUST
    IN THEMISDB.TSCUSTA;

ALTER TABLE CUST ENABLE ARCHIVE
    USE CUST_ARCHIVE;
```

Connects the two tables in an archive relationship

V11 Global Variables for Archiving

- SYSIBMADM.MOVE_TO_ARCHIVE
- SYSIBMADM.GET_ARCHIVE

Note: Both were initially defaulted to 'N'. Not so in V12 and retrofitted with an APAR in V11

zPARM MOVE_TO_ARCHIVE_DEFAULT

Some current ways to delete data

Program single deletes with commits (cursor processing)

Program multi-row with commits (cursor processing)

Program multi-rows fetch and delete (cursor processing)

Utility Unloads

Utility REORG with discard logic

Mass Delete: Delete from table

Truncate table

Partition Rotation (drops a partition)

V12 Piece-wise Delete

When you have a need to delete a large amount of data from a table
Concerns: Locking, Concurrency, Restarts

Options: Batch program with frequent commits
(Single deletes, Multi Row deletes)
Db2 REORG Utility with discard option

But what if you needed to do this with a single SQL statement,
or without cursor processing ?

```
DELETE FROM ORDER_TBL
WHERE ORDER_DATE < ?
```

V12 Piece-wise Delete

Concerns: Possibility of millions of rows ➔ Too much locking. Huge Rollback if a failure occurs

V12: Allows for the FETCH to be coded within a delete statement. No need to declare a cursor.

```
DELETE FROM ORDER_TBL
 WHERE ORDER_DATE < ?
  FETCH FIRST 5000 ROWS  ONLY;
  COMMIT;
```

V12 Piece-wise Delete

```
CREATE PROCEDURE SPDELETE
      (IN  P_TABLE      CHAR(128)
      ,IN P_LOGIC      VARCHAR(500)
      ,IN  P_FETCH_CNT SMALLINT
      ,OUT P_SQL_TEXT  VARCHAR(1000)
      ,OUT P_TOT_COUNT INTEGER
      ,OUT P_MESSAGE_TEXT VARCHAR(1000)
      ,OUT P_RETCODE   INTEGER
      )
-- SP Options
  CALLED ON NULL INPUT
  RESULT SETS 0
-- COMMIT ON RETURN NO  ==>  Commit on Return cannot be
                           --    coded with AUTONOMOUS
                           --   You get a -628 on deploy
-- Bind Parameters
  DEGREE ANY
  ISOLATION LEVEL CS
  VALIDATE BIND
  QUALIFIER ODYTA
  PACKAGE OWNER ODYTA
  ASUTIME LIMIT 50000
```

V12 Piece-wise SQL PL Code
Dynamic SQL Delete Logic

Themis
Leaders in IT Education

V12 Piece-wise SQL PL Code Dynamic SQL Delete Logic

```
--
-- Build the SQL Delete using input parameters
--

 set v_sql_delete =
    'DELETE FROM '|| P_table  ||
                     P_logic      ||
                ' FETCH FIRST ‘  ||
                P_fetch_cnt       ||
               ' ROWS ONLY'
  ;

    set p_sql_text = v_sql_delete;
```

```
fetch_loop:  loop

   execute immediate v_sql_delete;
    begin
      get diagnostics
       v_delete_cnt = row_count;
      set p_tot_count = v_delete_cnt;
   end;

  if v_sqlcode = +100 then
    leave fetch_loop;
   else
    set p_tot_count = p_tot_count  +
                                      v_delete_cnt;
     commit;
   end if;
 end loop;
```

V12 Piece-wise delete COBOL

```
PERFORM UNTIL NO-MORE-ROWS

   EXEC SQL
     DELETE FROM EMPPROJACT
     WHERE PROJNO = 'AD3112'
     FETCH FIRST :WS-NUMBER ROWS ONLY
   END-EXEC
   EVALUATE TRUE
     WHEN SQLCODE = ZERO
        DISPLAY 'SQLCODE = ZERO ON DELETE'
        ADD SQLERRD(3) TO WS-TOTAL-COUNT
     WHEN SQLCODE = 100
        ADD SQLERRD(3) TO WS-TOTAL-COUNT
        DISPLAY 'SQLCODE = 100 END OF DELETES'
        DISPLAY 'TOTAL NUMBER OF DELETES = ' WS-TOTAL-
COUNT
        SET NO-MORE-ROWS TO TRUE
      ……..
      ……..
```

Thank you for allowing me and Themis to share some of our experience and knowledge today!

Tony Andrews
tandrews@themisinc.com

I hope that you learned something new today !!!!

Themis
Leaders in IT Education

The material in this presentation is further developed in the following Themis courses:

DB1041 – Advanced SQL FOR Db2
SQ1010 – Dealing With Complex Queries
Cross Platform SQL
DB1037 – Advanced Query Tuning With IBM Data Studio on z/OS
DB1032 – Db2 for z/OS Optimization Performance and Tuning
DB2111 – New Application Features V11

Links to these courses may be found at: www.themisinc.com

Tony's Email: tandrews@themisinc.com
Twitter: @ThemisTraining

Themis
Leaders in IT Education

Education. Check Out www.themisinc.com

- On-site and Public
- Instructor -led
- Hands-on
- Customization
- Experience
- Over 30 DB2 courses
- Over 400 IT courses

US 1-800-756-3000
Intl. 1-908-233-8900

Themis
Leaders in IT Education