Leveraging New SQL Features in DB2 10 for z/OS

Guogen (Gene) Zhang
IBM

August 10, 2011
Session 9840

Agenda

- SQL PL extensions
  - Scalar UDF, Table UDF, and XML support
- Bi-temporal for historical data
  - Time travel query
- Fine granularity access control
  - Row permission & column mask
- New OLAP functions
- Timestamp with more precision, timestamp with time zone
- Extended implicit cast
- XML features

DB2 10 for z/OS

- CPU reductions for most workloads
- Five to 10 times more concurrent users
- Greater concurrency for data definition and access
- More online changes for definitions and utilities
- Improved security with improved granularity
- Temporal or versioned data
- pureXML and SQL enhancements to improve portability
- Productivity improved

SQL PL Extensions for Scalar UDF, Table UDF, and XML support

SQL PL: SQL procedural language background

- Native SQL procedures (V9)
- **Simplifies** the task of writing database applications
- DB2 9 for z/OS
  - Scalar function support limited to single RETURN statement
  - No support for SQL table functions; only external table functions are supported

§ Extended in V10 to allow for use for:
  - SQL scalar functions
  - SQL table functions (minimal subset)
  - XML type

SQL Scalar Function

```
CREATE   FUNCTION   REVERSE  (INSTR
   VARCHAR(4000))
     RETURNS VARCHAR (4000)
     DETERMINISTIC
     NO EXTERNAL ACTION
     CONTAINS SQL
BEGIN
  DECLARE REVSTR, RESTSTR VARCHAR(4000)
   DEFAULT '';
  DECLARE LEN INT;
  IF INSTR IS NULL THEN
     RETURN NULL;
  END IF;
  SET RESTSTR = INSTR;
  SET LEN = LENGTH(INSTR);
  WHILE LEN > 0 DO
     SET REVSTR = SUBSTR(RESTSTR, 1, 1)
           CONCAT REVSTR;
     SET RESTSTR = SUBSTR(RESTSTR, 2, LEN - 1);
     SET LEN = LEN - 1;
  END WHILE;
  RETURN   REVSTR;
END
```

§ Function body contains control statements.

§ If the input data is null, the function simply returns null.

§ Otherwise, the function reverses the order of the characters in the input string and returns the modified string to the invoking statement.

SQL Table UDF

```
CREATE  FUNCTION  JTABLE (COLD_VALUE CHAR(9), T2_FLAG CHAR(1))
RETURNS TABLE (COLA INT, COLB INT, COLC INT)
LANGUAGE SQL
SPECIFIC DEPTINFO
NOT DETERMINISTIC
READS SQL DATA
RETURN
SELECT A.COLA, B.COLB, B.COLC
   FROM TABLE1 AS A
     LEFT OUTER JOIN
              TABLE2 AS B
        ON A.COL1 = B.COL1 AND  T2_FLAG = 'Y'
WHERE A.COLD = COLD_VALUE;
```

- function body specifies an SQL query that returns a result table
- result table is returned to the invoking statement

XML type in SQL PL proc

Decomposition into multiple tables

```
CREATE PROCEDURE DECOMP1(IN XDOC XML)  /* or IN DOC BLOB */
LANGUAGE SQL
 BEGIN
      /* DECLARE XDOC XML;
      SET XDOC = XMLPARSE(document DOC); */
      INSERT INTO tab1 SELECT *
       FROM XMLTABLE('/doc/head/row' PASSING XDOC
           COLUMNS C1 INT PATH 'C1',
                C2 VARCHAR(10) PATH 'C2') AS X;

      INSERT INTO tab2 SELECT *
       FROM XMLTABLE('/doc/body/row' PASSING XDOC
           COLUMNS C3 INT PATH 'C3',
                C4 VARCHAR(10) PATH 'C4') AS X;
 END
```

```
Tables: TAB1( C1, C2)
        TAB2(C3, C4)
Document:
<doc>
  <head>
    <row>
      <C1>1</C1>
      <C2>AAA</C2>
    </row>
  </head>
  <body>
  <row>
    <C3>10</C3>
    <C4>XXXX</C4>
  </row>
  <row>
    <C3>20</C3>
    <C4>YYYYY</C4>
  </row>
 </body>
</doc>
```

Parse once and decompose into multiple tables
If using Java caller, document could be parsed into binary XML in the client

Bitemporal Support - Time travel query

Bitemporal Support

- New concept of System_time and Business_time period
  - System_time captures DB2's creation and deletion of rows and automatically keeps historical versions of rows.
  - Business_time allows users to create their own validity period for a given row.
- Value to customers
  - meet compliance requirements: automatic propagation of old rows to a history table.
  - performs better than the home-grown solution.
  - easier to manage

Bitemporal Support – Example

```
CREATE TABLE  policy
(client      CHAR(4)       NOT NULL,
type         CHAR(4)       NOT NULL,
copay        SMALLINT      NOT NULL,
eff_beg      DATE          NOT NULL,
eff_end      DATE          NOT NULL,
sys_start    TIMESTAMP(12) NOT NULL IMPLICITLY HIDDEN
               GENERATED ALWAYS AS ROW BEGIN,
sys_end      TIMESTAMP(12) NOT NULL IMPLICITLY HIDDEN
               GENERATED ALWAYS AS ROW END,
trans_id     TIMESTAMP(12) IMPLICITLY HIDDEN
               GENERATED  ALWAYS AS TRANSACTION START ID,
PERIOD BUSINESS_TIME(eff_beg, eff_end),
PERIOD SYSTEM_TIME(sys_start, sys_end));
```

Bitemporal Support – Example (cont)

```
CREATE TABLE  policy_hist LIKE policy;

ALTER TABLE policy
ADD VERSIONING USE HISTORY TABLE  policy_hist;

CREATE UNIQUE INDEX ix_policy
ON policy (client, BUSINESS_TIME WITHOUT OVERLAPS);
```

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
1	01/01/2004	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004 (future event).
2	09/01/2004	Customer called and changed to HMO as of today (present event)
3	03/01/2006	Copay increase to $15 starting 01/01/2007 (future event)
4	06/01/2008	Cancel policy as of today (present event)
5	09/01/2008	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007 (past event)

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
1	01/01/2004	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004 (future event).

```
INSERT INTO policy VALUES
('C882', 'PPO',10,'02/01/2004','12/31/9999');
```

Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	12/31/9999	2004-01-01...	9999-12-31…

Table: policy

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
2	09/01/2004	Customer called and changed to HMO as of today (present event)

```
UPDATE policy FOR PORTION OF BUSINESS_TIME
  FROM '09/01/2004' TO '12/31/9999'
  SET type = 'HMO'
    WHERE client = 'C882';
```

Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01…	9999-12-31…
C882	HMO	10	09/01/2004	12/31/9999	2004-09-01…	9999-12-31…

Table: policy

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
3	03/01/2006	Copay increase to $15 starting 01/01/2007 (future event)

```
UPDATE policy FOR PORTION OF BUSINESS_TIME
  FROM '01/01/2007' TO '12/31/9999'
  SET copay = 15
    WHERE client = 'C882';
```

Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01…	9999-12-31…
C882	HMO	10	09/01/2004	01/01/2007	2006-03-01…	9999-12-31…
C882	HMO	15	01/01/2007	12/31/9999	2006-03-01…	9999-12-31…

Table: policy

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
4	06/01/2008	Cancel policy as of today (present event)

```
UPDATE policy
  SET eff_end = '06/01/2008'
    WHERE client = 'C882'
      AND eff_end = '12/31/9999';
```

Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01…	9999-12-31…
C882	HMO	10	09/01/2004	01/01/2007	2006-03-01…	9999-12-31…
C882	HMO	15	01/01/2007	06/01/2008	2008-06-01…	9999-12-31…

Table: policy

Bitemporal Support – Example (cont)

Step	Actual Date	Activity
5	09/01/2008	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007 (past event)

```
UPDATE policy FOR PORTION OF BUSINESS_TIME
  FROM '05/01/2006' TO '10/01/2007'
  SET type = 'POS'
    WHERE client = 'C882';
```

Bitemporal Support – Example (cont)

Step	Date	Activity
1	01/01/2004 (Future)	Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004
2	09/01/2004 (Present)	Customer called and changed to HMO as of today
3	03/01/2006 (Future)	Copay increase to $15 starting 01/01/2007
4	06/01/2008 (Present)	Cancel Policy as of today
5	09/01/2008 (Past)	Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01…	9999-12-31…
C882	HMO	10	09/01/2004	05/01/2006	2008-09-01…	9999-12-31…
C882	POS	10	05/01/2006	01/01/2007	2008-09-01…	9999-12-31…
C882	POS	15	01/01/2007	10/01/2007	2008-09-01…	9999-12-31…
C882	HMO	15	10/01/2007	06/01/2008	2008-09-01…	9999-12-31…

Table: policy

Bitemporal Support – Example (cont)

Question: On 09/15/2008, client calls and complains. Client saw an out-of-network specialist on 07/01/2007.
Claims dept. denied client's claim due to HMO coverage for this visit on 07/15/2007. Client demands reimbursement.

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01…	9999-12-31…
C882	HMO	10	09/01/2004	05/01/2006	2008-09-01…	9999-12-31…
C882	POS	10	05/01/2006	01/01/2007	2008-09-01…	9999-12-31…
C882	POS	15	01/01/2007	10/01/2007	2008-09-01…	9999-12-31…
C882	HMO	15	10/01/2007	06/01/2008	2008-09-01…	9999-12-31…

Table: policy

Bitemporal Support – Example (cont)

SELECT * FROM POLICY
FOR BUSINESS_TIME AS OF '2007-07-01'
WHERE CLIENT='C882';
Answer: Customer has "POS", so should be reimbursed.

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	09/01/2004	2004-09-01…	9999-12-31…
C882	HMO	10	09/01/2004	05/01/2006	2008-09-01…	9999-12-31…
C882	POS	10	05/01/2006	01/01/2007	2008-09-01…	9999-12-31…
C882	POS	15	01/01/2007	10/01/2007	2008-09-01…	9999-12-31…
C882	HMO	15	10/01/2007	06/01/2008	2008-09-01…	9999-12-31…

Table: policy

Bitemporal Support – Example (cont)

Question: Did our claims department make an error denying the client's claim on 07/15/2007?

To answer this: Need historical data to see what claims department saw on 07/15/2007. Thus, the need for a bitemporal solution.

Bitemporal Support – Example (cont)

```
SELECT * FROM policy
  FOR BUSINESS_TIME AS OF '2007-07-01'
  FOR SYSTEM_TIME AS OF '2007-07-15'
   WHERE client='C882';
```

Answer: At 07/15/2007, claims saw 'HMO'.

client	type	copay	eff_beg	eff_end	sys_start	sys_end
C882	PPO	10	02/01/2004	12/31/9999	2004-01-01…	2004-09-01…
C882	HMO	10	09/01/2004	12/31/9999	2004-09-01…	2006-03-01…
C882	HMO	15	01/01/2007	12/31/9999	2006-03-01…	2008-06-01…
C882	HMO	10	09/01/2004	01/01/2007	2008-06-01…	2008-09-01…
C882	HMO	15	01/01/2007	06/01/2008	2008-06-01…	2008-09-01…

Table: policy_hist

Fine granularity access control: Row permission and Column Mask

Concerns about Database Security

- Separation of duties
  - Database administrators such as DBADM can access sensitive data
  - No designated authority such as SECADM to manage security policies
- Granularity of privilege model
  - Privileges are granted at database object level
  - Difficult to protect personal and sensitive information within the object
  - Cannot easily comply with data protection laws such as HIPPA, GLBA
- Overloading applications with security logic
  - Can be bypassed by malicious users
  - Hampers the ability to use ad-hoc query tools, report generation tools
- Alternative views for each group of users
  - Can be bypassed by malicious users
  - View's updatability may not correctly reflect security policies
- Evolution of security policies
  - Difficult to manage and maintain

Solution : Row and Column Access Control

- Tighter security
  - Data-centric within database
  - No backdoor to bypass views or applications
  - More granularity via row permissions and column masks
  - Separation of duties
  - Designated SECADM authority
  - No authority including DBADM is exempted from the control
  - Relief for the evolution of security policies
- Easy to implement
  - More flexibility via SQL
  - Separation of security logic and application logic

Row and Column Access Control – new terminology

- Row Permission
  - a database object that expresses a row access control rule for a table
  - contains a rule in the form of an SQL search condition that describes to which rows the users have access
  - applied by DB2 after the checking of table privileges (e.g. SELECT, INSERT privilege, etc.)

Row and Column Access Control – new terminology (cont'd)

- Column Mask
  - a database object that expresses a column access control rule for a specific column in a table
  - contains a rule in the form of an SQL CASE expression that describes to what masked value returned for a column value the users have access
  - applied by DB2 after the checking of table privileges (e.g. SELECT, UPDATE privilege, etc.)

Row and Column Access Control – Concept

Think a Decomposed View

```
CREATE VIEW EMPOLYEE_VIEW AS
   SELECT (CASE … END) SSN, (CASE … END) SALARY
      FROM EMPLOYEE
      WHERE STATE = 'CA' AND
            LASTNAME = 'SMITH' AND
            BDATE > '1970-01-01'
```

- Row Permission
  - The WHERE clause of the EMPLOYEE_VIEW
- Column Mask
  - The outermost SELECT clause in the EMPLOYEE_VIEW definition

Row and Column Access Control – Examples

- **Row Permission**

```
CREATE PERMISSION EMPLOYEE_PERMISSION ON EMPLOYEE
   FOR ROWS WHERE STATE = 'CA' AND
                  LASTNAME = 'SMITH' AND
                  BDATE > '1970-01-01'
ENFORCED FOR ALL ACCESS ENABLE ;
```

- **Column Mask**

```
CREATE MASK SSN_MASK ON EMPLOYEE
    FOR COLUMN SSN RETURN
      CASE WHEN SESSION_USER = 'SMITH'
              THEN SSN
              ELSE CHAR('XXX-XX-') || SUBSTR(SSN,8,4)
       END
ENABLE;
```

  - **SELECT SSN FROM EMPLOYEE**;

Who can see what?

- New Built-in Functions
  - VERIFY_GROUP_FOR_USER
    - Verify primary and secondary authorization IDs

```
WHERE
   VERIFY_GROUP_FOR_USER (SESSION_USER, 'MGR', 'PAYROLL') = 1
```

  - VERIFY_TRUSTED_CONTEXT_ROLE_FOR_USER
    - Verify primary authorization ID's role

```
WHERE
   VERIFY_TRUSTED_CONTEXT_ROLE_FOR_USER (SESSION_USER,
                                         'MGR', 'PAYROLL') = 1
```

Activate Row and Column Access Control

- Activated by SECADM authority only
  - Job card … ,USER=SECADM, …
- Invalidate packages and cached statements
- Row permissions and column masks become effective in DML
  - All row permissions are merged to filter out rows
    - Multiple row permissions are connected with 'OR'
  - All column masks are applied to mask output columns
- Generate default row permission 1 = 0 if activated for row

```
ALTER  TABLE   table-name
   ACTIVATE ROW        ACCESS  CONTROL
   ACTIVATE COLUMN  ACCESS CONTROL;

ALTER  TABLE   table-name
   ACTIVATE ROW        ACCESS  CONTROL;
```

Deactivate Row and Column Access Control

- Deactivated by SECADM authority only
  - Job card … ,USER=SECADM, …
- Invalidate packages and cached statements
- Row permissions and column masks become ineffective in DML
  - Remove default row permission 1 = 0 if deactivated for row
  - Open all access to the table

```
ALTER  TABLE   table-name
   DEACTIVATE ROW       ACCESS  CONTROL
   DEACTIVATE COLUMN  ACCESS CONTROL;

ALTER  TABLE   table-name
   DEACTIVATE ROW       ACCESS  CONTROL;
```

New OLAP functions: Moving Average, Running Total, etc.

Overview

- DB2 9 for z/OS has already supported 2 classes of OLAP specifications
  - Ranking – RANK(), DENSE_RANK()
  - Numbering – ROW_NUMBER().
- DB2 10 for z/OS introduces the last class of OLAP specifications
  - Aggregation Specifications – SUM(), AVG() and other aggregate functions etc.

Moving Sums and Moving Averages

- compute a single value for the current row based on some or all of the rows in a defined group.
- support cumulative sums and moving averages by using a window.
- can be used in a select-list, or in the ORDER BY clause of a select-statement.
- Limitation: cannot use with XMLQUERY function or an XMLEXISTS predicate,

Example Data

```
CREATE TABLE EMP
(EMPNO CHAR(6) NOT NULL,
FIRSTNME VARCHAR(10) NOT NULL,
LASTNAME VARCHAR(10) NOT NULL,
WORKDEPT CHAR(3) ,
SALARY DECIMAL(7, 2)
);
```

Data in the table:

```
EMPNO  FIRSTNAME  LASTNAME   WORKDEPT SALARY
------ ---------- ---------- -------- ---------
000010 CHRISTINE  HASS       A00       52750.00
000030 SALLY      KWAN       C01       38250.00
000110 VINCENZO   LUCCHESSI  A00       46500.00
000140 KIM        NATZ       C01       47250.00
000150 HEATHER    NICHOLLS   C01       47250.00
200010 DIAN       HEMMINGER  A00       29250.00
200120 GREG       ORLANDO    A00       29250.00
200130 DOLORES    QUINTANA   C01       19350.00
```

EXAMPLE for RANK,DENSE_RANK,ROW_NUMBER from V9

Display the workdept, salary, firstname, lastname, rank in the dept based on salary, dense_rank in the dept base on salary, row_nUmber in the dept based on salary.

```
SELECT workdept, salary, firstname, lastname,
       RANK() OVER (PARTITION BY workdept order by salary desc) as
  dept_rank,
       DENSE_RANK() OVER (PARTITION BY workdept order by salary desc)
  as denserank,
       ROW_NUMBER() OVER (PARTITION BY workdept order by salary desc)
  as rownum
FROM EMP;
```

```
WORKDEPT SALARY    FIRSTNAME  LASTNAME   DEPT_RANK DENSERANK ROWNUM
-------- --------- ---------- ---------- --------- --------- ------
A00       52750.00 CHRISTINE  HASS               1         1      1
A00       46500.00 VINCENZO   LUCCHESSI          2         2      2
A00       29250.00 DIAN       HEMMINGER          3         3      3
A00       29250.00 GREG       ORLANDO            3         3      4
C01       47250.00 KIM        NATZ               1         1      1
C01       47250.00 HEATHER    NICHOLLS           1         1      2
C01       38250.00 SALLY      KWAN               3         2      3
C01       19350.00 DOLORES    QUINTANA           4         3      4
```

Example Data

```
create table Sales_history (
  Territory    VARCHAR(10), -- Business Territory
  Month        INTEGER,     -- Six-digit in YYYYMM format
  Sales        INTEGER      -- Total sales for Territory/Month
   )
```

Data in the table:

```
Territory  Month        Sales
---------- ---------- -----------
East       199810              10
East       199811               4
East       199812              10
East       199901               7
East       199902              10
West       199810               8
West       199811              12
West       199812               7
West       199901              11
West       199902               6
```

EXAMPLE for Moving SUM, Moving AVG, etc

Display the business territory, month, total sales for each territory/month and the sale in the territory averaged over the current month and the preceding two months.

```
SELECT Sh.Territory, Sh.Month, Sh.Sales,
       AVG(Sh.Sales) OVER (PARTITION BY Sh.Territory
                           ORDER BY Sh.Month
                          ROWS 2 PRECEDING) as Moving _average
  FROM Sales_history as Sh;
```

```
Territory  Month        Sales       Moving_average
---------- ---------- ----------- ----------------
East       199810              10               10
East       199811               4                7
East       199812              10                8
East       199901               7                7
East       199902              10                9
West       199810               8                8
West       199811              12               10
West       199812               7                9
West       199901              11               10
West       199902               6                8
```

Example Data

```
create table stock (date date,symbol char(3),close_price
  dec(9,3));
insert into stock values ('2007-04-23','XYZ',110.125);
insert into stock values ('2007-04-24','XYZ',109.500);
insert into stock values ('2007-04-25','XYZ',110.000);
insert into stock values ('2007-04-26','XYZ',119.750);
insert into stock values ('2007-04-27','XYZ',110.625);
insert into stock values ('2007-04-30','XYZ',111.125);
insert into stock values ('2007-05-01','XYZ',113.750);
insert into stock values ('2007-05-02','XYZ',114.000);
insert into stock values ('2007-05-03','XYZ',113.750);
insert into stock values ('2007-05-04','XYZ',112.125);
insert into stock values ('2007-05-07','XYZ',109.750);
insert into stock values ('2007-05-08','XYZ',111.000);
insert into stock values ('2007-05-09','XYZ',110.750);
```

EXAMPLE for Moving SUM, Moving AVG, etc -- ROWS

Find the seven day centered moving average of XYZ stock for each day the stock traded. The window is specified by the rows clause.

```
SELECT date,symbol, close_price, decimal(avg(close_price) over (order by
  date rows between 3 preceding and 3 following),6,3)
   as smooth_cp
  FROM stock
```

DATE	SYMBOL	CLOSE_PRICE	SMOOTH_CP
04/23/2007	XYZ	110.125	112.343
04/24/2007	XYZ	109.500	112.000
04/25/2007	XYZ	110.000	111.854
04/26/2007	XYZ	119.750	112.125
04/27/2007	XYZ	110.625	112.678
04/30/2007	XYZ	111.125	113.285
05/01/2007	XYZ	113.750	113.589
05/02/2007	XYZ	114.000	112.160
05/03/2007	XYZ	113.750	112.214
05/04/2007	XYZ	112.125	112.160
05/07/2007	XYZ	109.750	111.339
05/08/2007	XYZ	111.000	110.642
05/09/2007	XYZ	110.750	110.125
05/10/2007	XYZ	108.000	109.725
05/11/2007	XYZ	109.125	109.718

EXAMPLE for Moving SUM, Moving AVG, etc -- RANGE

For the stock XYZ, find the 7 day historical average for each day the stock traded.
The window is specified by the range clause.

```
SELECT date,substr(DSN8.dayname(date),1,9) as day, close_price,
      decimal(avg(close_price) over (order by date range
        00000006. preceding),7,2) as avg_7_range,
      count(close_price) over (order by date range
        00000006. preceding) as count_7_range
  FROM stock
```

DATE	DAY	CLOSE_PRICE	AVG_7_RANGE	COUNT_7_RANGE
04/23/2007	Monday	110.125	110.12	1
04/24/2007	Tuesday	109.500	109.81	2
04/25/2007	Wednesday	110.000	109.87	3
04/26/2007	Thursday	119.750	112.34	4
04/27/2007	Friday	110.625	112.00	5
04/30/2007	Monday	111.125	112.20	5
05/01/2007	Tuesday	113.750	113.05	5
05/02/2007	Wednesday	114.000	113.85	5
05/03/2007	Thursday	113.750	112.65	5
05/04/2007	Friday	112.125	112.95	5
05/07/2007	Monday	109.750	112.67	5
05/08/2007	Tuesday	111.000	112.12	5
05/09/2007	Wednesday	110.750	111.47	5
05/10/2007	Thursday	108.000	110.32	5
05/11/2007	Friday	109.125	109.72	5

Greater Timestamp Precision, Timestamp WITH TIME ZONE

Background

Problem:

- The existing TIMESTAMP data type does not capture an associated time zone, the timestamp is ambiguous
- Some customers store time zone in another column

Solution:

- New data type

  TIMESTAMP WITH TIME ZONE

New Terminology

- **fractional second** - A portion of a second that is greater than 0 but less than 1.
- **timestamp precision** - The maximum number of digits that can be included in a fractional second.

Timestamp

- A timestamp is a **six or** seven-part value (year, month, day, hour, minute, second, and **optional fractional second**) **with an optional time zone specification** that represents a date and time. The time could include specification of a fraction of a second.
- The number of digits in the fractional second is specified using an attribute in the range from 0 to 12 with a default of 6.
- Example: TIMESTAMP(12)
  '2000-01-15-08.30.00.123456789012'

- TIMESTAMP WITH TIME ZONE format:
  <timestamp> TZH:TZM
  TZH (time zone hour) – '*xhh*' -24 to 24
  TZM (time zone minute) – 'mm' 00 to 59 (has to be 00 if TZH is 24)
- Example:
  New York is 4 hours behind UTC, so New York time "11:42" on 2009-06-09 can be represented as '2009-06-09-11.42.00.000000-04:00'.
  Same UTC representation:
  '2009-06-09-11.42.00.000000-04:00'
  '2009-06-09-08.42.00.000000-07:00'
  '2009-06-09-15.42.00.000000-00:00'

More Examples:

```
CREATE TABLE LAB15_Table
  (ID    INTEGER      NOT NULL WITH DEFAULT,
TSTZ12  TIMESTAMP(12) WITH TIME ZONE WITH DEFAULT
         '9999-12-31-23.59.59.123456789012 +14:00');

INSERT INTO LAB15_Table(ID, TSTZ12)
 VALUES (4,
   TIMESTAMP '2012-12-31 02:02:02.123456789012+08:00' AT TIMEZONE '-
   08:00'     );

INSERT INTO LAB15_Table (ID, TSTZ12)
 VALUES (5,
TIMESTAMP '2012-12-31 02:02:02.123456789012+08:00'
         AT TIMEZONE SESSION TIME ZONE     )
```

Extended Implicit CAST Support

Overview

- Extended support for implicit casts: implicit cast between string and numeric data types
- Indexable/sargable predicates
- new semantics of assignments & comparisons & unions & expressions & function resolution

Examples

```
CREATE TABLE employee (
      empno  INTEGER,
      level  CHAR(3),
      salary DECIMAL(15,2));

INSERT INTO EMPLOYEE VALUES( '1001', 3, 89000.39);

UPDATE employee
    SET level = level + 1,
    salary = salary * '1.1'
    WHERE empno = '1001';
```

From Numeric to String

Numbers ⟷ Strings: character string (no CLOB, no FOR BIT DATA subtype) or graphic string (no DBCLOB, UNICODE encoding scheme)

Source Data Type	Target Data Type
SMALLINT	VARCHAR(6)
INTEGER	VARCHAR(11)
BIGINT	VARCHAR(20)
NUMERIC/DECIMAL	VARCHAR(precision+2)
FLOAT (REAL, DOUBLE)	VARCHAR(24)
DECFLOAT	VARCHAR(42)

From String to Numeric

Source Data Type	Target Data Type
CHAR	DECFLOAT(34)
VARCHAR	DECFLOAT(34)
GRAPHIC	DECFLOAT(34)
VARGRAPHIC	DECFLOAT(34)
CHAR/VARCHAR **FOR BIT DATA**	N/A
BINARY/VARBINARY	N/A
CLOB/BLOB/DBCLOB	N/A

XML features

- XML Schema enhancements
  - Automatically enforce schema conformance
  - Validation inside engine
- Native XML date and time support and index support
- XML update support
- XML multi-versioning
- Binary XML
- CHECK XML

Summary

- DB2 10 for z/OS offers great improvements in performance, simplicity and productivity
- We've learned some important SQL features that will improve app development productivity