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**

```sql
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

```sql
CREATE FUNCTION JTABLE (COLD_VALUE CHAR(9), T2_FLAG CHAR(1))
RETURNS TABLE (COLA INT, COLE 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;
```

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

### XML type in SQL PL proc

**Decomposition into multiple tables**

```sql
CREATE PROCEDURE DECOMPT1(IN DOC XML) /
  OR IN DOC BLOB /
LANGUAGE SQL
BEGIN
  /* DECLARE DDOC XML;
   SET DDOC = XMLPARSE(document DOC); /*
   INSERT INTO tab1 SELECT *
     FROM XMLTABLE(DDOC/row) PASSING XMLDOC
       COLUMNS C1 INT PATH 'C1',
       C2 VARCHAR(10) PATH 'C2' AS X;

   INSERT INTO tab2 SELECT *
     FROM XMLTABLE(DDOC/row) PASSING XMLDOC
       COLUMNS C3 INT PATH 'C3',
       C4 VARCHAR(10) PATH 'C4' AS X;

END
```

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

---

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>YYYY</C4>
    </row>
  </body>
</doc>
```
Bitemporal Support
- Time travel query

- 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)

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

**INSERT INTO policy VALUES**

`('C882', 'PPO',10,'02/01/2004','12/31/9999');`
### Bitemporal Support – Example (cont)

**Table: policy**

<table>
<thead>
<tr>
<th>client</th>
<th>type</th>
<th>copay</th>
<th>eff_beg</th>
<th>eff_end</th>
<th>sys_start</th>
<th>sys_end</th>
</tr>
</thead>
<tbody>
<tr>
<td>C882</td>
<td>PPO</td>
<td>10</td>
<td>02/01/2004</td>
<td>12/31/9999</td>
<td>2004-01-01</td>
<td>9999-12-31</td>
</tr>
</tbody>
</table>

### Bitemporal Support – Example (cont)

**Activity**

**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

**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)

<table>
<thead>
<tr>
<th>Step</th>
<th>Date</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01/01/2004</td>
<td>Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004</td>
</tr>
<tr>
<td>2</td>
<td>09/01/2004</td>
<td>Customer called and changed to HMO as of today</td>
</tr>
<tr>
<td>3</td>
<td>03/01/2006</td>
<td>Copay increase to $15 starting 01/01/2007</td>
</tr>
<tr>
<td>4</td>
<td>06/01/2008</td>
<td>Cancel Policy as of today</td>
</tr>
<tr>
<td>5</td>
<td>09/01/2008</td>
<td>Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007</td>
</tr>
</tbody>
</table>

#### Table: policy

<table>
<thead>
<tr>
<th>client</th>
<th>type</th>
<th>copay</th>
<th>eff_beg</th>
<th>eff_end</th>
<th>sys_start</th>
<th>sys_end</th>
</tr>
</thead>
<tbody>
<tr>
<td>C882</td>
<td>PPO</td>
<td>10</td>
<td>02/01/2004</td>
<td>09/01/2004</td>
<td>2004-09-01…</td>
<td>9999-12-31…</td>
</tr>
<tr>
<td>C882</td>
<td>HMO</td>
<td>10</td>
<td>09/01/2004</td>
<td>12/31/9999</td>
<td>2004-09-01…</td>
<td>9999-12-31…</td>
</tr>
</tbody>
</table>

### Bitemporal Support – Example (cont)

<table>
<thead>
<tr>
<th>Step</th>
<th>Actual Date</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>03/01/2006</td>
<td>Copay increase to $15 starting 01/01/2007 (future event)</td>
</tr>
</tbody>
</table>

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)

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

```
UPDATE policy
SET eff_end = '06/01/2008'
WHERE client = 'C882'
AND eff_end = '12/31/9999';
```
### Bitemporal Support – Example (cont)

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

#### Table: policy

<table>
<thead>
<tr>
<th>client</th>
<th>type</th>
<th>copay</th>
<th>eff_beg</th>
<th>eff_end</th>
<th>sys_start</th>
<th>sys_end</th>
</tr>
</thead>
<tbody>
<tr>
<td>C882</td>
<td>PPO</td>
<td>10</td>
<td>02/01/2004</td>
<td>09/01/2004</td>
<td>2004-09-01…</td>
<td>9999-12-31…</td>
</tr>
<tr>
<td>C882</td>
<td>HMO</td>
<td>10</td>
<td>09/01/2004</td>
<td>01/01/2007</td>
<td>2006-03-01…</td>
<td>9999-12-31…</td>
</tr>
<tr>
<td>C882</td>
<td>HMO</td>
<td>15</td>
<td>01/01/2007</td>
<td>06/01/2008</td>
<td>2008-06-01…</td>
<td>9999-12-31…</td>
</tr>
</tbody>
</table>

### Bitemporal Support – Example (cont)

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)**

<table>
<thead>
<tr>
<th>Step</th>
<th>Date</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01/01/2004</td>
<td>Issue PPO Policy to Customer C882 with copay amount $10 starting from 02/01/2004</td>
</tr>
<tr>
<td>2</td>
<td>09/01/2004</td>
<td>Customer called and changed to HMO as of today</td>
</tr>
<tr>
<td>3</td>
<td>03/01/2006</td>
<td>Copay increase to $15 starting 01/01/2007</td>
</tr>
<tr>
<td>4</td>
<td>06/01/2008</td>
<td>Cancel Policy as of today</td>
</tr>
<tr>
<td>5</td>
<td>09/01/2008</td>
<td>Correct error by retroactively updating policy to POS from 05/01/2006 to 10/01/2007</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date</th>
<th>sys_start</th>
<th>sys_end</th>
<th>type</th>
<th>copay</th>
<th>eff_beg</th>
<th>eff_end</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/01/2004</td>
<td>02/01/2004</td>
<td>09/01/2004</td>
<td>POS/10</td>
<td>10</td>
<td>02/01/2004</td>
<td>09/01/2004</td>
</tr>
<tr>
<td>09/01/2004</td>
<td>05/01/2006</td>
<td>10/01/2007</td>
<td>POS/15</td>
<td>15</td>
<td>01/01/2007</td>
<td>10/01/2007</td>
</tr>
<tr>
<td>06/01/2008</td>
<td>06/01/2008</td>
<td></td>
<td>HMO/10</td>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


**Bitemporal Support – Example (cont)**

<table>
<thead>
<tr>
<th>Date</th>
<th>sys_start</th>
<th>sys_end</th>
<th>type</th>
<th>copay</th>
<th>eff_beg</th>
<th>eff_end</th>
</tr>
</thead>
<tbody>
<tr>
<td>02/01/2004</td>
<td>09/01/2004</td>
<td>05/01/2006</td>
<td>POS/10</td>
<td>10</td>
<td>09/01/2004</td>
<td>05/01/2006</td>
</tr>
<tr>
<td>05/01/2006</td>
<td>01/01/2007</td>
<td>10/01/2007</td>
<td>POS/15</td>
<td>15</td>
<td>01/01/2007</td>
<td>10/01/2007</td>
</tr>
<tr>
<td>06/01/2008</td>
<td></td>
<td></td>
<td>HMO/10</td>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>client</th>
<th>type</th>
<th>copay</th>
<th>eff_beg</th>
<th>eff_end</th>
<th>sys_start</th>
<th>sys_end</th>
</tr>
</thead>
<tbody>
<tr>
<td>C882</td>
<td>PPO</td>
<td>10</td>
<td>02/01/2004</td>
<td>09/01/2004</td>
<td>2004-09-01…</td>
<td>9999-12-31…</td>
</tr>
<tr>
<td>C882</td>
<td>HMO</td>
<td>10</td>
<td>09/01/2004</td>
<td>05/01/2006</td>
<td>2008-09-01…</td>
<td>9999-12-31…</td>
</tr>
<tr>
<td>C882</td>
<td>POS</td>
<td>10</td>
<td>05/01/2006</td>
<td>01/01/2007</td>
<td>2008-09-01…</td>
<td>9999-12-31…</td>
</tr>
<tr>
<td>C882</td>
<td>POS</td>
<td>15</td>
<td>01/01/2007</td>
<td>10/01/2007</td>
<td>2008-09-01…</td>
<td>9999-12-31…</td>
</tr>
<tr>
<td>C882</td>
<td>HMO</td>
<td>15</td>
<td>10/01/2007</td>
<td>06/01/2008</td>
<td>2008-09-01…</td>
<td>9999-12-31…</td>
</tr>
</tbody>
</table>

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’.

<table>
<thead>
<tr>
<th>client</th>
<th>type</th>
<th>copay</th>
<th>eff_beg</th>
<th>eff_end</th>
<th>sys_start</th>
<th>sys_end</th>
</tr>
</thead>
<tbody>
<tr>
<td>C882</td>
<td>PPO</td>
<td>10</td>
<td>02/01/2004</td>
<td>12/31/9999</td>
<td>2004-01-01...</td>
<td>2004-09-01...</td>
</tr>
<tr>
<td>C882</td>
<td>HMO</td>
<td>10</td>
<td>09/01/2004</td>
<td>12/31/9999</td>
<td>2004-09-01...</td>
<td>2006-03-01...</td>
</tr>
<tr>
<td>C882</td>
<td>HMO</td>
<td>15</td>
<td>01/01/2007</td>
<td>12/31/9999</td>
<td>2006-03-01...</td>
<td>2008-06-01...</td>
</tr>
<tr>
<td>C882</td>
<td>HMO</td>
<td>10</td>
<td>09/01/2004</td>
<td>01/01/2007</td>
<td>2008-06-01...</td>
<td>2008-09-01...</td>
</tr>
<tr>
<td>C882</td>
<td>HMO</td>
<td>15</td>
<td>01/01/2007</td>
<td>06/01/2008</td>
<td>2008-06-01...</td>
<td>2008-09-01...</td>
</tr>
</tbody>
</table>

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 EMPLOYEE_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
  END ENABLE;
  ```

  • SELECT SSN FROM EMPLOYEE;
Who can see what?

- New Built-in Functions
  - VERIFY_GROUP_FOR_USER
    - Verify primary and secondary authorization IDs
  - VERIFY_TRUSTED_CONTEXT_ROLE_FOR_USER
    - Verify primary authorization ID’s role

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

```
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:

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>FIRSTNAME</th>
<th>LASTNAME</th>
<th>WORKDEPT</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>000010</td>
<td>CHRISTINE</td>
<td>HASS</td>
<td>A00</td>
<td>52750.00</td>
</tr>
<tr>
<td>000030</td>
<td>SALLY</td>
<td>KWAN</td>
<td>C01</td>
<td>38250.00</td>
</tr>
<tr>
<td>000110</td>
<td>VINCENZO</td>
<td>LUCHESSI</td>
<td>A00</td>
<td>46500.00</td>
</tr>
<tr>
<td>000140</td>
<td>KIM</td>
<td>NATZ</td>
<td>C01</td>
<td>47250.00</td>
</tr>
<tr>
<td>000150</td>
<td>HEATHER</td>
<td>NICHOLLS</td>
<td>C01</td>
<td>47250.00</td>
</tr>
<tr>
<td>200010</td>
<td>DIAN</td>
<td>HEMMINGER</td>
<td>A00</td>
<td>29250.00</td>
</tr>
<tr>
<td>200120</td>
<td>GREG</td>
<td>ORLANDO</td>
<td>A00</td>
<td>29250.00</td>
</tr>
<tr>
<td>200130</td>
<td>DOLORES</td>
<td>QUINTANA</td>
<td>C01</td>
<td>19350.00</td>
</tr>
</tbody>
</table>

**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;

<table>
<thead>
<tr>
<th>WORKDEPT</th>
<th>SALARY</th>
<th>FIRSTNAME</th>
<th>LASTNAME</th>
<th>DEPT_RANK</th>
<th>DENSERANK</th>
<th>ROWNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>A00</td>
<td>52750.00</td>
<td>CHRISTINE</td>
<td>HASS</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A00</td>
<td>46500.00</td>
<td>VINCENZO</td>
<td>LUCHESSI</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A00</td>
<td>29250.00</td>
<td>DIAN</td>
<td>HEMMINGER</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>A00</td>
<td>29250.00</td>
<td>GREG</td>
<td>ORLANDO</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>C01</td>
<td>47250.00</td>
<td>KIM</td>
<td>NATZ</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C01</td>
<td>47250.00</td>
<td>HEATHER</td>
<td>NICHOLLS</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>C01</td>
<td>38250.00</td>
<td>SALLY</td>
<td>KWN</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C01</td>
<td>19350.00</td>
<td>DOLORES</td>
<td>QUINTANA</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Territory</th>
<th>Month</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>East</td>
<td>199810</td>
<td>10</td>
</tr>
<tr>
<td>East</td>
<td>199811</td>
<td>4</td>
</tr>
<tr>
<td>East</td>
<td>199812</td>
<td>10</td>
</tr>
<tr>
<td>East</td>
<td>199901</td>
<td>7</td>
</tr>
<tr>
<td>East</td>
<td>199902</td>
<td>10</td>
</tr>
<tr>
<td>West</td>
<td>199810</td>
<td>8</td>
</tr>
<tr>
<td>West</td>
<td>199811</td>
<td>12</td>
</tr>
<tr>
<td>West</td>
<td>199812</td>
<td>7</td>
</tr>
<tr>
<td>West</td>
<td>199901</td>
<td>11</td>
</tr>
<tr>
<td>West</td>
<td>199902</td>
<td>6</td>
</tr>
</tbody>
</table>

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;
```

<table>
<thead>
<tr>
<th>Territory</th>
<th>Month</th>
<th>Sales</th>
<th>Moving_average</th>
</tr>
</thead>
<tbody>
<tr>
<td>East</td>
<td>199810</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>East</td>
<td>199811</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>East</td>
<td>199812</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>East</td>
<td>199901</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>East</td>
<td>199902</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>West</td>
<td>199810</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>West</td>
<td>199811</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>West</td>
<td>199812</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>West</td>
<td>199901</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>West</td>
<td>199902</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>
### Example Data

```sql
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.

```sql
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;
```

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

<table>
<thead>
<tr>
<th>DATE</th>
<th>DAY</th>
<th>CLOSE_PRICE</th>
<th>AVG_7_RANGE</th>
<th>COUNT_7_RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>04/23/2007</td>
<td>Monday</td>
<td>110.125</td>
<td>110.12</td>
<td>1</td>
</tr>
<tr>
<td>04/24/2007</td>
<td>Tuesday</td>
<td>109.500</td>
<td>109.81</td>
<td>2</td>
</tr>
<tr>
<td>04/25/2007</td>
<td>Wednesday</td>
<td>110.000</td>
<td>109.87</td>
<td>3</td>
</tr>
<tr>
<td>04/26/2007</td>
<td>Thursday</td>
<td>119.750</td>
<td>112.34</td>
<td>4</td>
</tr>
<tr>
<td>04/27/2007</td>
<td>Friday</td>
<td>110.625</td>
<td>112.00</td>
<td>5</td>
</tr>
<tr>
<td>04/30/2007</td>
<td>Monday</td>
<td>111.125</td>
<td>112.20</td>
<td>5</td>
</tr>
<tr>
<td>05/01/2007</td>
<td>Tuesday</td>
<td>113.750</td>
<td>113.05</td>
<td>5</td>
</tr>
<tr>
<td>05/02/2007</td>
<td>Wednesday</td>
<td>114.000</td>
<td>113.85</td>
<td>5</td>
</tr>
<tr>
<td>05/03/2007</td>
<td>Thursday</td>
<td>113.750</td>
<td>112.65</td>
<td>5</td>
</tr>
<tr>
<td>05/04/2007</td>
<td>Friday</td>
<td>112.125</td>
<td>112.95</td>
<td>5</td>
</tr>
<tr>
<td>05/07/2007</td>
<td>Monday</td>
<td>119.750</td>
<td>112.47</td>
<td>5</td>
</tr>
<tr>
<td>05/08/2007</td>
<td>Tuesday</td>
<td>111.000</td>
<td>112.12</td>
<td>5</td>
</tr>
<tr>
<td>05/09/2007</td>
<td>Wednesday</td>
<td>110.750</td>
<td>111.47</td>
<td>5</td>
</tr>
<tr>
<td>05/10/2007</td>
<td>Thursday</td>
<td>108.000</td>
<td>110.32</td>
<td>5</td>
</tr>
<tr>
<td>05/11/2007</td>
<td>Friday</td>
<td>109.125</td>
<td>109.72</td>
<td>5</td>
</tr>
</tbody>
</table>

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.
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: \texttt{TIMESTAMP(12)}

\texttt{'2000-01-15-08.30.00.123456789012'}

TIMESTAMP WITH TIME ZONE format:

\texttt{<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:

\texttt{2009-06-09-11.42.00.000000-04:00}’
\texttt{2009-06-09-08.42.00.000000-07:00}’
\texttt{2009-06-09-15.42.00.000000-00:00}’
More Examples:

```sql
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

```sql
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)

<table>
<thead>
<tr>
<th>Source Data Type</th>
<th>Target Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLINT</td>
<td>VARCHAR(6)</td>
</tr>
<tr>
<td>INTEGER</td>
<td>VARCHAR(11)</td>
</tr>
<tr>
<td>BIGINT</td>
<td>VARCHAR(20)</td>
</tr>
<tr>
<td>NUMERIC/DECIMAL</td>
<td>VARCHAR(precision+2)</td>
</tr>
<tr>
<td>FLOAT (REAL, DOUBLE)</td>
<td>VARCHAR(24)</td>
</tr>
<tr>
<td>DECFLOAT</td>
<td>VARCHAR(42)</td>
</tr>
</tbody>
</table>

### From String to Numeric

<table>
<thead>
<tr>
<th>Source Data Type</th>
<th>Target Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>DECFLOAT(34)</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>DECFLOAT(34)</td>
</tr>
<tr>
<td>GRAPHIC</td>
<td>DECFLOAT(34)</td>
</tr>
<tr>
<td>VARGRAPHIC</td>
<td>DECFLOAT(34)</td>
</tr>
<tr>
<td>CHAR/VARCHAR FOR BIT DATA</td>
<td>N/A</td>
</tr>
<tr>
<td>BINARY/VARBINARY</td>
<td>N/A</td>
</tr>
<tr>
<td>CLOB/BLOB/DBCLOB</td>
<td>N/A</td>
</tr>
</tbody>
</table>
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