Introducing DB2 10 Temporal Data Features

Peter Vanroose

GSE DB2 Belgium
Joint User Group Meeting
IBM, Brussels, 6 June 2013
“30th anniversary of DB2”
DB2 10 temporal data features

Objectives:

- Relational databases and historic (or versioned) data
- New SELECT query syntax for “temporal” requests
- Table setup for “system time” versioning
- Business time: data validity time period
- Bi-temporal tables
- ...
Relational databases and historic (or versioned) data

DB2 ==> very efficient transactional data server:
- ACID: atomic (transactions) ==> commit / rollback
  consistent ==> each visible DB state makes sense
  isolated ==> through locking (& isolation levels)
  durable ==> permanent changes
  BUT no notion of “keeping track of history”

Data warehouse & business intelligence:
- often needs / wants historic data
  (“how did the data look on 1 February?”)
  (trend analysis: “predict future sales from past trends”)
- not typically a task for a transactional DB server
  but can be integrated

What we miss: “what was the (ACID) state of my data on <time instant>”?
Reasons for “temporal” data queries in a relational DB

1. Not really meant for BI or DW
2. Tracability of data changes for auditing purposes:
   - “What data was used in last month’s investment assessment?”
   - “Please re-run the tax computation of last 31 December”
   - “Since when are you giving a 5% price reduction to that client?”
   - “Please trace back <certain business data> over the last year.”
3. Tracability of data changes for business tracing purposes:
   - “Where did we send that order to last week?”
     => What was the customer address on May 30 at 15:43 ?
4. Storing data validity information:
   - Customer: “My address as of 1 September will be ...”
   - Insurance record(s): “covered time interval: 1 January -- 30 June”
   - Promotional action: “Price will be 20% off between ... and ...”
   - Product availability period(s) (possibly with retroactive effect)
New SELECT query syntax for "temporal" requests

Example table: customers

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
<th>telephone</th>
<th>amount_sold</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>016/123456</td>
<td>1043.50</td>
</tr>
<tr>
<td>2</td>
<td>Dupont</td>
<td>A.Max 3</td>
<td>02/9876543</td>
<td>745.00</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/1234567</td>
<td>6100.00</td>
</tr>
<tr>
<td>8</td>
<td>Van Dijk</td>
<td>Dijk 8</td>
<td>0476/54321</td>
<td>75.25</td>
</tr>
<tr>
<td>9</td>
<td>Berends</td>
<td>Dorp 17</td>
<td>09/8765432</td>
<td>3201.43</td>
</tr>
<tr>
<td>10</td>
<td>Zander</td>
<td>Centre 4</td>
<td>-</td>
<td>123.45</td>
</tr>
</tbody>
</table>

SELECT * FROM customers WHERE id = 3 ;

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
<th>telephone</th>
<th>amount_sold</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/1234567</td>
<td>6100.00</td>
</tr>
</tbody>
</table>

SELECT * FROM customers AS OF SYSTEM TIME '2013-05-30-15.45.00' WHERE id = 3 ; (*)

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
<th>telephone</th>
<th>amount_sold</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Zand 98</td>
<td>03/1234567</td>
<td>6100.00</td>
</tr>
</tbody>
</table>
New SELECT query syntax for “temporal” requests

- New ANSI / ISO SQL:2011 Standard syntax:
  ... FROM <table> AS OF SYSTEM_TIME <timestamp> ...

- DB2 syntax (2 alternative forms):
  ... FROM <table> FOR SYSTEM_TIME AS OF <timestamp> ...
  ... FROM <table> AS OF TIMESTAMP <timestamp> ...

- DB2 for LUW extension:
  ... FROM <table> FOR SYSTEM_TIME AS OF <date> ...

- Oracle syntax:
  ... FROM <table> AS OF TIMESTAMP <timestamp> ...

- Examples:
  SELECT * FROM customers FOR SYSTEM_TIME AS OF current timestamp ;
  SELECT * FROM customers FOR SYSTEM_TIME AS OF current date - 3 days ;
  SELECT * FROM customers AS OF TIMESTAMP current timestamp - 1 min ;
  SELECT * FROM customers FOR SYSTEM_TIME AS OF :hv ;
Table setup for “system time” versioning

But ... tables are (still) not “versioned” by default!

Think about how you would implement “versioned data” manually:

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
<th>telephone</th>
<th>amount_sold</th>
<th>valid_from</th>
<th>valid_until</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>016/123456</td>
<td>1043.50</td>
<td>2013-02-02-14.02.02</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Dupont</td>
<td>A.Max 3</td>
<td>02/9876543</td>
<td>745.00</td>
<td>2004-08-20-11.11.11</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/1234567</td>
<td>6100.00</td>
<td>2013-06-04-15.13.32</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>Van Dijk</td>
<td>Dijk 8</td>
<td>0476/54321</td>
<td>75.25</td>
<td>2012-01-04-12.00.00</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>Berends</td>
<td>Dorp 17</td>
<td>09/8765432</td>
<td>3201.43</td>
<td>2012-04-12-18.00.00</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>Zander</td>
<td>Centre 4</td>
<td></td>
<td>123.45</td>
<td>2012-11-15-09.00.00</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>016/123456</td>
<td>943.50</td>
<td>2011-03-12-09.13.42</td>
<td>2013-02-02-14.02.02</td>
</tr>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td></td>
<td>943.50</td>
<td>2004-03-30-15.13.42</td>
<td>2011-03-12-09.13.42</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Zand 98</td>
<td>03/1234567</td>
<td>6100.00</td>
<td>2010-01-01-00.00.00</td>
<td>2013-06-04-15.13.32</td>
</tr>
<tr>
<td>4</td>
<td>Pieters</td>
<td>Rand 7A</td>
<td></td>
<td>100.00</td>
<td>2010-08-31-12.21.53</td>
<td>2012-07-21-16.24.13</td>
</tr>
<tr>
<td>4</td>
<td>Pieters</td>
<td>Berg 71</td>
<td></td>
<td>100.00</td>
<td>2012-07-21-16.24.13</td>
<td>2012-12-31-23.59.59</td>
</tr>
</tbody>
</table>

Technical challenges:

store delta’s? duplicate PK values; query performance&complexity;
triggers for update & delete; default values for hidden cols; ...
Table setup for “system time” versioning: how DB2 wants it

CREATE TABLE customers
(id int NOT NULL,
 , name varchar(64)
 , address varchar(128)
 , telephone varchar(32)
 , amount_sold dec(9,2)
 , valid_from timestamp(12) GENERATED ALWAYS AS ROW BEGIN NOT NULL
 , valid_until timestamp(12) GENERATED ALWAYS AS ROW END NOT NULL
 , trans_id timestamp(12) GENERATED ALWAYS AS TRANSACTION START ID
 , PRIMARY KEY (id)
 , PERIOD SYSTEM_TIME (valid_from, valid_until)
);

CREATE TABLE customers_history LIKE customers ;

ALTER TABLE customers
 ADD VERSIONING USE HISTORY TABLE customers_history ;

• may also ALTER customers: ADD three columns & PERIOD spec
• the three columns could be declared as IMPLICITLY HIDDEN
Table setup for “system time” versioning: configuration issues

- base and history table(space) must have byte-compatible rows:
  - same column names, same data types, same order & NOT NULL
  - exactly what “CREATE .. LIKE ..” provides
- no further similarities needed
  - may have different indexes
  - may have different check constraints and FKs
    (typically, the history table should have none)
  - may have different partitioning, buffer pool, page size, compress
  - history table should have all direct DML blocked
    (since it should be completely transparent to applications)
- ALTER TABLE (column alterations or additions) on base table
  automatically updates the history table definition
### Table setup for “system time” versioning: sample data

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
<th>telephone</th>
<th>amount_sold</th>
<th>valid_from</th>
<th>valid_until</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel</td>
<td>016/123456</td>
<td>1043.50</td>
<td>2013-02-02-14.02.02</td>
<td>9999-12-30-00.00.00</td>
</tr>
<tr>
<td>2</td>
<td>Dupont</td>
<td>A.Max</td>
<td>02/9876543</td>
<td>745.00</td>
<td>2004-08-20-11.11.11</td>
<td>9999-12-30-00.00.00</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square</td>
<td>03/1234567</td>
<td>6100.00</td>
<td>2013-06-04-15.13.32</td>
<td>9999-12-30-00.00.00</td>
</tr>
<tr>
<td>8</td>
<td>Van Dijk</td>
<td>Dijk</td>
<td>0476/54321</td>
<td>75.25</td>
<td>2012-01-04-12.00.00</td>
<td>9999-12-30-00.00.00</td>
</tr>
<tr>
<td>9</td>
<td>Berends</td>
<td>Dorp</td>
<td>09/8765432</td>
<td>3201.43</td>
<td>2012-04-12-18.00.00</td>
<td>9999-12-30-00.00.00</td>
</tr>
<tr>
<td>10</td>
<td>Zander</td>
<td>Centre</td>
<td>-</td>
<td>123.45</td>
<td>2012-11-15-09.00.00</td>
<td>9999-12-30-00.00.00</td>
</tr>
</tbody>
</table>

### customers_history table:

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
<th>telephone</th>
<th>amount_sold</th>
<th>valid_from</th>
<th>valid_until</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel</td>
<td>016/123456</td>
<td>943.50</td>
<td>2011-03-12-09.13.42</td>
<td>2013-02-02-14.02.02</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Zand</td>
<td>03/1234567</td>
<td>6100.00</td>
<td>2010-01-01-00.00.00</td>
<td>2013-06-04-15.13.32</td>
</tr>
<tr>
<td>4</td>
<td>Pieters</td>
<td>Rand</td>
<td>-</td>
<td>100.00</td>
<td>2010-08-31-12.21.53</td>
<td>2012-07-21-16.24.13</td>
</tr>
<tr>
<td>4</td>
<td>Pieters</td>
<td>Berg</td>
<td>-</td>
<td>100.00</td>
<td>2012-07-21-16.24.13</td>
<td>2012-12-31-23.59.59</td>
</tr>
</tbody>
</table>

(note: precision of timestamp columns: contain 12 additional fractional digits!)
(transaction start id column: will be NULL, or equal to (oldest) valid_from)
**Table setup for “system time” versioning: effects on DML**

- **customer_history rows**: *never* inserted/updated/deleted manually!

- **on INSERT in customer:**
  - the three additional columns are auto-filled by DB2:
    - valid_from: with *current timestamp*(12)
    - valid_until: with ‘9999-12-30-00.00.00.0000000000000000’
      (this makes sure that the value cannot become invalid after time zone conversions / mappings !)
    - trans_id: with NULL (if nullable), otherwise with current timestamp

- **on UPDATE of row(s) in customer:**
  - original (unchanged) row is “moved” to customer_history
    - where valid_until is changed to *current timestamp*(12)
  - modified row: valid_from is modified to *current timestamp*(12)

- **on DELETE of row(s) in customer:**
  - original (old) row is “moved” to customer_history
    - where valid_until is changed to *current timestamp*(12)
Interpretation of system time validity intervals

<table>
<thead>
<tr>
<th>Start Time</th>
<th>End Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-01-01-00.00.00</td>
<td>2012-01-04-12.00.00</td>
</tr>
<tr>
<td>2012-01-04-12.00.00</td>
<td>2012-12-31-23.59.59</td>
</tr>
<tr>
<td>2010-08-31-12.21.53</td>
<td>2012-04-12-18.00.00</td>
</tr>
<tr>
<td>2012-04-12-18.00.00</td>
<td>2013-02-02-14.02.02</td>
</tr>
<tr>
<td>2011-03-12-09.13.42</td>
<td>2012-07-21-16.24.13</td>
</tr>
</tbody>
</table>

1. Singel 9 [943.50]
2. A.Max 3
3. Zand 98
4. Rand 7A
5. Berg 71
6. Dijk 8
7. Dorp 17

**PK temporal uniqueness:** for every time instant, there is at most one data row per PK value.

**Since e.g. 2011-03-12-09.13.42 is the commit timestamp of the update,** it belongs to the middle time interval, *not* the left one.

**In general, the “valid_from” (start) time is an inclusive boundary,** while the “valid_until” (end) time is an *exclusive* boundary.
System time: guarantees

4.1

- “Ordinary” SELECT queries never need to access the history table
  ==> base table looks exactly as before (except for additional columns)
  ==> no need to revisit existing applications identical access paths
- “Ordinary” INSERT/UPDATE/DELETE notice additional overhead similar to classical triggers
- History can only be forged by modifying the history table directly
  - base table ROW BEGIN & END columns are not updatable
  - BUT history table ROW BEGIN & END columns are updatable
  ==> it is possible to create inconsistent data (viz. violate temporal uniqueness on PK)
System time: additional DML possibilities

SELECT ... FROM customers AS OF TIMESTAMP current timestamp(12)
is equivalent to
SELECT ... FROM customers
and should not need to access the history table (but it does!)

SELECT ... FROM customers AS OF TIMESTAMP current date
is NOT equivalent to the above!

==&gt; it’s equivalent to “last midnight”

SELECT ... FROM customers AS OF TIMESTAMP current date + 1 day
is INVALID (as is any future date)

==&gt; but is accepted!! (both on LUW and on z/OS)

SELECT ... FROM customers FOR SYSTEM_TIME FROM <ts1> TO <ts2>
- the time range is <ts1> inclusive but <ts2> exclusive
- might return multiple rows for the same PK
- makes sense to include (one of) the “valid_from” or “valid_until” columns in selection
- if <ts1> is larger than or equal to <ts2>, the result set is empty

SELECT ... FROM customers FOR SYSTEM_TIME BETWEEN <ts1> AND <ts2>
- the time range is <ts1> inclusive and also <ts2> inclusive
System time: additional DML possibilities

- Use of the CURRENT TEMPORAL SYSTEM_TIME special register:

  ```sql
  SELECT address FROM customers WHERE id=3
  ADDRESS
  -----------------------------------------------
  Square 1
  1 record(s) selected.
  
  SELECT address FROM customers AS OF TIMESTAMP '2012-01-01' WHERE id=3
  ADDRESS
  -----------------------------------------------
  Zand 89
  1 record(s) selected.
  
  SET current temporal system_time='2012-01-01';
  SELECT address FROM customers WHERE id=3;
  ADDRESS
  -----------------------------------------------
  Zand 89
  1 record(s) selected.
  
  SELECT address FROM customers AS OF TIMESTAMP '2012-01-01' WHERE id=3
  SQL20524N The statement failed because of an invalid period specification or period clause for period "SYSTEM_TIME". Reason code "6". SQLSTATE=428HY
  
  UPDATE customers SET address = 'Avenue Louise 9' WHERE id=3
  SQL20535N The data change operation "UPDATE" is not supported for the target object "CUSTOMERS" because of an implicit or explicit period specification involving "SYSTEM_TIME". Reason code: "1". SQLSTATE=51046
  
  SET current temporal system_time=NULL;
  ```
The query

SELECT ... FROM customers AS OF TIMESTAMP <ts> WHERE <cond>

is implemented as follows (as can be seen from EXPLAIN):

SELECT ... FROM customers
   WHERE <cond>
   AND valid_from <= <ts>
UNION ALL
   SELECT ... FROM customers_history
   WHERE <cond>
   AND valid_from <= <ts>
   AND valid_until > <ts>

Could be important to create index(es)

on columns valid_from and/or valid_until,
possibly composite with other columns
System time: some use cases

1. Compliance & auditing:
   - never need to use “AS OF” queries
   - history table functions as a “change log”
   - to re-run an application on the data of last month:
     · use the CURRENT TEMPORAL SYSTEM_TIME register
     · make sure that all tables are “temporal”!

2. Business Intelligence related to time evolution of data:
   - “who was our best customer at the end of last month?”
   - application could directly query the base + history tables
   - or: take summary snapshots at several time instants: eg:

```sql
WITH dates(t) AS (
    SELECT date('2012-01-01') FROM sysibm.sysdummy1
    UNION ALL
    SELECT t + 1 month FROM dates WHERE t + 1 month < current date
)
SELECT SUM(amount_sold) FROM dates d, customers AS OF TIMESTAMP d.t
GROUP BY d.t -- (although this won’t work syntactically: need host variable)
```
System time: some use cases

3. Compare data at two times in the past (or current)
   - detailed:
     
     ```sql
     SELECT a.id, a.address AS old, b.address AS new
     FROM customers AS OF TIMESTAMP :date1  a
     FULL OUTER JOIN
     customers AS OF TIMESTAMP :date2  b
     ON a.id = b.id
     WHERE a.address is distinct from b.address
     ```
   
   - summaries:
     
     ```sql
     SELECT SUM(amount_sold), :date1
     FROM customers AS OF TIMESTAMP :date1
     UNION ALL
     SELECT SUM(amount_sold), :date2
     FROM customers AS OF TIMESTAMP :date2
     ```

   Resembles use case of versioning systems (git, subversion, CVS)

4. Point in time recovery
   
   ```sql
   UPDATE customers c
   SET address = (SELECT address FROM customers AS OF TIMESTAMP :x
     WHERE id = c.id)
   ```
Business time: data validity time period

Want more control over the “valid_from” and “valid_until” values
- time instant of UPDATE is not necessarily
time instant of when this new fact becomes valid
- example: address change should become active on 1 September

No longer about “transaction time” but about “effective” timespans.

Application should be able to insert into or update the validity dates
But still want “temporal uniqueness” guarantees from DB2

- DB2 syntax for querying a “business temporal” table:
  ... FROM <table> FOR BUSINESS_TIME AS OF <timestamp or date> ... 
- no SQL ANSI/ISO standard (yet)

Careful: without an “AS OF”, returns the full history (all versions):
  ... FROM <table> ...
CREATE TABLE customers
    (id int NOT NULL,
     name varchar(64),
     address varchar(128),
     telephone varchar(32),
     amount_sold dec(9,2),
     valid_from timestamp(6) NOT NULL,
     valid_until timestamp(6) NOT NULL,
     PERIOD BUSINESS_TIME (valid_from, valid_until),
     PRIMARY KEY (id, BUSINESS_TIME WITHOUT OVERLAPS));

• No history table!
• “id” could now have duplicates
  ==> need a composite primary key
• New uniqueness concept: *temporal uniqueness*
• Enforced by a new type of unique index:
  
  CREATE UNIQUE INDEX <name>
  ON <table> (<cols>, BUSINESS_TIME WITHOUT OVERLAPS);
No defaults for “valid_from” and “valid_until”

==> application *must* explicitly state the validity period
(since these columns are NOT NULL)

==> “valid_until” could still be set to e.g. 9999-12-30 or 9999-12-31
Business time: updating data

- Update statements without “temporal” specification will update ALL rows, not just the ones “as of current timestamp”:

UPDATE customers SET telephone = '03/7654321' WHERE id = 3

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
<th>telephone</th>
<th>amount_sold</th>
<th>valid_from</th>
<th>valid_until</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>016/123456</td>
<td>1043.50</td>
<td>2013-02-02-14.02.02</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>2</td>
<td>Dupont</td>
<td>A.Max 3</td>
<td>02/9876543</td>
<td>745.00</td>
<td>2004-08-20-11.11.11</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/7654321</td>
<td>6100.00</td>
<td>2013-06-04-15.13.32</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>8</td>
<td>Van Dijk</td>
<td>Dijk 8</td>
<td>0476/54321</td>
<td>75.25</td>
<td>2012-01-04-12.00.00</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>9</td>
<td>Berends</td>
<td>Dorp 17</td>
<td>09/8765432</td>
<td>3201.43</td>
<td>2012-04-12-18.00.00</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>10</td>
<td>Zander</td>
<td>Centre 4</td>
<td>-</td>
<td>123.45</td>
<td>2012-11-15-09.00.00</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>016/123456</td>
<td>943.50</td>
<td>2011-03-12-09.13.42</td>
<td>2013-02-02-14.02.02</td>
</tr>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>-</td>
<td>943.50</td>
<td>2004-03-30-15.13.42</td>
<td>2011-03-12-09.13.42</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Zand 98</td>
<td>03/7654321</td>
<td>6100.00</td>
<td>2010-01-01-00.00.00</td>
<td>2013-06-04-15.13.32</td>
</tr>
<tr>
<td>4</td>
<td>Pieters</td>
<td>Rand 7A</td>
<td>-</td>
<td>100.00</td>
<td>2010-08-31-12.21.53</td>
<td>2012-07-21-16.24.13</td>
</tr>
<tr>
<td>4</td>
<td>Pieters</td>
<td>Berg 71</td>
<td>-</td>
<td>100.00</td>
<td>2012-07-21-16.24.13</td>
<td>2012-12-31-23.59.59</td>
</tr>
</tbody>
</table>
Business time: updating data

- Update statements with “temporal” specification:

\[
\text{UPDATE customers FOR PORTION OF BUSINESS\_TIME}
\]
\[
\text{FROM '2013-09-01-00.00.00' TO '9999-12-31-23.59.59'}
\]
\[
\text{SET telephone = '03/7654321' WHERE id = 3}
\]

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
<th>telephone</th>
<th>amount_sold</th>
<th>valid_from</th>
<th>valid_until</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>016/123456</td>
<td>1043.50</td>
<td>2013-02-02-14.02.02</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>2</td>
<td>Dupont</td>
<td>A.Max 3</td>
<td>02/9876543</td>
<td>745.00</td>
<td>2004-08-20-11.11.11</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/7654321</td>
<td>6100.00</td>
<td>2013-09-01-00.00.00</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/1234567</td>
<td>6100.00</td>
<td>2013-06-04-15.13.32</td>
<td>2013-09-01-00.00.00</td>
</tr>
<tr>
<td>8</td>
<td>Van Dijk</td>
<td>Dijk 8</td>
<td>0476/54321</td>
<td>75.25</td>
<td>2012-01-04-12.00.00</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>9</td>
<td>Berends</td>
<td>Dorp 17</td>
<td>09/8765432</td>
<td>3201.43</td>
<td>2012-04-12-18.00.00</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>10</td>
<td>Zander</td>
<td>Centre 4</td>
<td>-</td>
<td>123.45</td>
<td>2012-11-15-09.00.00</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>016/123456</td>
<td>943.50</td>
<td>2011-03-12-09.13.42</td>
<td>2013-02-02-14.02.02</td>
</tr>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>-</td>
<td>943.50</td>
<td>2004-03-30-15.13.42</td>
<td>2011-03-12-09.13.42</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Zand 98</td>
<td>03/1234567</td>
<td>6100.00</td>
<td>2010-01-01-00.00.00</td>
<td>2013-06-04-15.13.32</td>
</tr>
<tr>
<td>4</td>
<td>Pieters</td>
<td>Rand 7A</td>
<td>-</td>
<td>100.00</td>
<td>2010-08-31-12.21.53</td>
<td>2012-07-21-16.24.13</td>
</tr>
<tr>
<td>4</td>
<td>Pieters</td>
<td>Berg 71</td>
<td>-</td>
<td>100.00</td>
<td>2012-07-21-16.24.13</td>
<td>2012-12-31-23.59.59</td>
</tr>
</tbody>
</table>

\[=>\text{automatic row split when necessary!}\]
Business time: deleting data

Delete statements with “temporal” specification:

DELETE FROM customers FOR PORTION OF BUSINESS_TIME
  FROM '2014-01-01-00.00.00' TO '9999-12-31-23.59.59'
  WHERE id = 3

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
<th>telephone</th>
<th>amount_sold</th>
<th>valid_from</th>
<th>valid_until</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>016/123456</td>
<td>1043.50</td>
<td>2013-02-02-14.02.02</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>2</td>
<td>Dupont</td>
<td>A.Max 3</td>
<td>02/9876543</td>
<td>745.00</td>
<td>2004-08-20-11.11.11</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/7654321</td>
<td>6100.00</td>
<td>2013-09-01-00.00.00</td>
<td>2014-01-01-00.00.00</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/1234567</td>
<td>6100.00</td>
<td>2013-06-04-15.13.32</td>
<td>2013-09-01-00.00.00</td>
</tr>
<tr>
<td>8</td>
<td>Van Dijk</td>
<td>Dijk 8</td>
<td>0476/54321</td>
<td>75.25</td>
<td>2012-01-04-12.00.00</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>9</td>
<td>Berends</td>
<td>Dorp 17</td>
<td>09/8765432</td>
<td>3201.43</td>
<td>2012-04-12-18.00.00</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>10</td>
<td>Zander</td>
<td>Centre 4</td>
<td>-</td>
<td>123.45</td>
<td>2012-11-15-09.00.00</td>
<td>9999-12-31-23.59.59</td>
</tr>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>016/123456</td>
<td>943.50</td>
<td>2011-03-12-09.13.42</td>
<td>2013-02-02-14.02.02</td>
</tr>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>-</td>
<td>943.50</td>
<td>2004-03-30-15.13.42</td>
<td>2011-03-12-09.13.42</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Zand 98</td>
<td>03/1234567</td>
<td>6100.00</td>
<td>2010-01-01-00.00.00</td>
<td>2013-06-04-15.13.32</td>
</tr>
<tr>
<td>4</td>
<td>Pieters</td>
<td>Rand 7A</td>
<td>-</td>
<td>100.00</td>
<td>2010-08-31-12.21.53</td>
<td>2012-07-21-16.24.13</td>
</tr>
<tr>
<td>4</td>
<td>Pieters</td>
<td>Berg 71</td>
<td>-</td>
<td>100.00</td>
<td>2012-07-21-16.24.13</td>
<td>2012-12-31-23.59.59</td>
</tr>
</tbody>
</table>

==> automatic row split when necessary!
Business time: guarantees & caveats

- “Ordinary” SELECT always accesses the full table
  ==> including history!
- “Ordinary” INSERT/UPDATE/DELETE update all versions
  ==> unless WHERE on valid_from or valid_until
- INSERTs and “temporal” UPDATEs sometimes refused:
  ==> “duplicate” error from unique index
  when time intervals would overlap
  ==> temporal uniqueness is guaranteed by DB2
SELECT ... FROM customers FOR BUSINESS_TIME AS OF current timestamp is \textit{NOT} equivalent to
SELECT ... FROM customers

SELECT ... FROM customers FOR BUSINESS_TIME AS OF current date + 1 day is totally \textit{VALID} (as is any future date)

SELECT ... FROM customers FOR BUSINESS_TIME FROM \textlangle ts1\textrangle TO \textlangle ts2\textrangle
- the time range is \textlangle ts1\textrangle \textit{inclusive} but \textlangle ts2\textrangle \textit{exclusive}
- might return multiple rows for the same id
- if \textlangle ts1\textrangle is larger than or equal to \textlangle ts2\textrangle, or one is NULL, the result set is empty

SELECT ... FROM customers FOR BUSINESS_TIME BETWEEN \textlangle ts1\textrangle AND \textlangle ts2\textrangle
- the time range is \textlangle ts1\textrangle \textit{inclusive} and also \textlangle ts2\textrangle \textit{inclusive}
Business time: additional DML possibilities

- Use of the CURRENT TEMPORAL BUSINESS_TIME special register:

  SELECT address FROM customers WHERE id=3
  ADDRESS
  Square 1
  1 record(s) selected.

  SELECT address FROM customers FOR BUSINESS_TIME AS OF '2012-01-01'
  WHERE id=3
  ADDRESS
  Zand 89
  1 record(s) selected.

  SET current temporal business_time='2012-01-01';
  SELECT address FROM customers WHERE id=3;
  ADDRESS
  Zand 89
  1 record(s) selected.

  SELECT address FROM customers FOR BUSINESS_TIME AS OF '2012-01-01'
  WHERE id=3
  SQL20524N  The statement failed because of an invalid period specification or
  period clause for period "BUSINESS_TIME". Reason code "7".  SQLSTATE=428HY
  SET current temporal business_time=NULL;
Business time: use case

Product price & availability

```
CREATE TABLE products
( prid INTEGER NOT NULL,
  price DEC(9,2),
  valid_from date NOT NULL,
  valid_until date NOT NULL,
  PERIOD BUSINESS_TIME (valid_from, valid_until),
  PRIMARY KEY (prid, BUSINESS_TIME WITHOUT OVERLAPS)
);
```

```
CREATE UNIQUE INDEX prid -- not necessary: is automatically created!
  ON products (prid, BUSINESS_TIME WITHOUT OVERLAPS);
```

<table>
<thead>
<tr>
<th>prid</th>
<th>price</th>
<th>valid_from</th>
<th>valid_until</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>250.00</td>
<td>2004-01-01</td>
<td>9999-12-30</td>
</tr>
<tr>
<td>102</td>
<td>750.00</td>
<td>2012-01-01</td>
<td>9999-12-30</td>
</tr>
<tr>
<td>103</td>
<td>150.00</td>
<td>2012-01-01</td>
<td>2013-07-01</td>
</tr>
<tr>
<td>103</td>
<td>120.00</td>
<td>2013-07-01</td>
<td>2013-09-01</td>
</tr>
<tr>
<td>103</td>
<td>3201.43</td>
<td>2014-01-01</td>
<td>9999-12-30</td>
</tr>
</tbody>
</table>
## Business time: use case

```sql
select * from products where prid=103;
```

<table>
<thead>
<tr>
<th>PRID</th>
<th>PRICE</th>
<th>VALID_FROM</th>
<th>VALID_UNTIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>103</td>
<td>150.00</td>
<td>01/01/2012</td>
<td>07/01/2013</td>
</tr>
<tr>
<td>103</td>
<td>120.00</td>
<td>07/01/2013</td>
<td>09/01/2013</td>
</tr>
<tr>
<td>103</td>
<td>3201.43</td>
<td>01/01/2014</td>
<td>12/30/9999</td>
</tr>
</tbody>
</table>

3 record(s) selected.

```sql
select * from products for business_time as of current date where prid=103;
```

<table>
<thead>
<tr>
<th>PRID</th>
<th>PRICE</th>
<th>VALID_FROM</th>
<th>VALID_UNTIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>103</td>
<td>150.00</td>
<td>01/01/2012</td>
<td>07/01/2013</td>
</tr>
</tbody>
</table>

1 record(s) selected.

```sql
select * from products for business_time from '01.01.2013' to '01.01.2014' where prid=103;
```

<table>
<thead>
<tr>
<th>PRID</th>
<th>PRICE</th>
<th>VALID_FROM</th>
<th>VALID_UNTIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>103</td>
<td>150.00</td>
<td>01/01/2012</td>
<td>07/01/2013</td>
</tr>
<tr>
<td>103</td>
<td>120.00</td>
<td>07/01/2013</td>
<td>09/01/2013</td>
</tr>
</tbody>
</table>

2 record(s) selected.

```sql
select * from products for business_time between '01.01.2013' and '01.07.2013' where prid=103;
```

<table>
<thead>
<tr>
<th>PRID</th>
<th>PRICE</th>
<th>VALID_FROM</th>
<th>VALID_UNTIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>103</td>
<td>150.00</td>
<td>01/01/2012</td>
<td>07/01/2013</td>
</tr>
<tr>
<td>103</td>
<td>120.00</td>
<td>07/01/2013</td>
<td>09/01/2013</td>
</tr>
</tbody>
</table>

Further reading:
1. Relational databases and historic (or versioned) data
2. New SELECT query syntax for “temporal” requests
3. Table setup for “system time” versioning
4. Interpretation of system time validity intervals
5. System time: some use cases
6. Business time: data validity time period
7. Bi-temporal tables
Bi-temporal tables

Contain both a system time indication (system maintained) and a business time indication (application maintained)

“What is valid at time instant X, and when did we know that?”

Necessary to answer questions like:

- When did we decide on the 20% off promotional price?
- What prices did our customers see last week?

ALTER TABLE products
ADD start GENERATED ALWAYS AS ROW BEGIN NOT NULL implicitly hidden
ADD end GENERATED ALWAYS AS ROW END NOT NULL implicitly hidden
ADD trans_id GENERATED ALWAYS AS TRANSACTION START ID implicitly hidden
;
ALTER TABLE products ADD PERIOD SYSTEM_TIME(start,end);
CREATE TABLE products_history LIKE products;
ALTER TABLE products ADD VERSIONING USE HISTORY TABLE products_history;
Bi-temporal tables: use cases

What was the 20% reduction timespan, as seen last week?

```
SELECT prid, valid_from, valid_until
FROM products AS OF TIMESTAMP current timestamp - 7 days p
WHERE price = ( SELECT 0.8*price FROM products WHERE prid = p.prid) ;
```

What price(s) did we announce last week for the summer months?

```
SELECT prid, price
FROM products AS OF TIMESTAMP current timestamp - 7 days
    FOR BUSINESS_TIME FROM '2013-07-01' TO '2013-09-01' ;
```
DB2 10 temporal data features

1. Relational databases and historic (or versioned) data
2. New SELECT query syntax for “temporal” requests
3. Table setup for “system time” versioning
4. Interpretation of system time validity intervals
5. System time: some use cases
6. Business time: data validity time period
7. Bi-temporal tables
8. Further reading

Further reading

Very good summary in Chapter 4 of the eFlashBook
“DB 10 for Linux, UNIX, and Windows New Features”
(Paul Zikopoulos et al.), see

Syntax details: to be found in the SQL reference manuals:
(search for “period-specification” and “row-transaction” in the syntax diagrams)

LUW: documents SC27-3885 and SC27-3886,
and DB2SQLRefVol2-db2s2e1011.pdf

Questions, remarks, feedback, ... ?

Thank you!

Peter Vanroose
ABIS Training & Consulting
pvanroose@abis.be

Introducing DB2 10 temporal data features -- GSE DB2 6 June 2013

DB2 10 temporal data features
1. Relational databases and historic (or versioned) data
2. New SELECT query syntax for “temporal” requests
3. Table setup for “system time” versioning
4. Interpretation of system time validity intervals
5. System time: some use cases
6. Business time: data validity time period
7. Bi-temporal tables
8. Further reading