Temporal Data & Time Travel in PostgreSQL

Peter Vanroose

FOSDEM 2015 - PGDay
30 January 2015
Marriott Hotel, Brussels

Temporal tables in PostgreSQL

Outline:
- Relational databases and historic (or versioned) data: what & why
- ISO SQL:2011 standards description
- new SELECT query syntax for “temporal” requests
- System-time (“versioning”) vs. Business-time (“validity”)
- implementation choices: details & argumentations
- use cases & performance aspects
PostgreSQL ==> 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’s address on January 22 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)
Temporal tables and the ISO SQL:2011 standard

official name: ISO/IEC 9075:2011

Most important “new SQL feature” in this standard’s version (viz. in Part 2: SQL/Foundation):

“Temporal Data Support”
as an optional (non-mandatory) feature

What?

• tables may have additional time-period (interval) information
  ("ACID validity" periods; temporal versions; ...)

• UPDATE / DELETE / INSERT must automatically maintain this info
  (TRUNCATE not)

• SELECT should add syntax to interrogate non-current time-period:
  SELECT ... FROM t AS OF SYSTEM TIME <time indication>

• not meant to be used as "application time"
Transaction time versus business time

“Transaction time”: what was the (ACID) state of my data on \( <\text{time}> \)?
- also called “system time”
- should be system maintained: “system-versioned” tables
- useful for questions on auditing & tracing
- only supports history: time instants must never be future
- default is “now”; history should not burden “normal” activity

“Business time”: validity time, application time:
- should be application (user) maintained, not automatic
- useful for questions on “covered time interval” of business reality
e.g.: insurance contract, address, promotional action, membership
- future dates could make sense
- time resolution decided by application (day / microsecond / year ...)

Temporal tables and PostgreSQL
1. Relational databases and historic (or versioned) data
2. Temporal tables and the ISO SQL:2011 standard
3. Transaction time versus business time
4. Table setup for “system time” versioning
5. System time in PostgreSQL
6. System time: some use cases
7. Business time: data validity time period
8. Business time in PostgreSQL
9. Business time: use case
10. Bi-temporal tables
11. Further reading
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 '2015-01-22 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 since version 10 (2010) (2 alternative forms):**
  
  ... FROM <table> FOR SYSTEM_TIME AS OF <timestamp> ...
  
  ... FROM <table> AS OF TIMESTAMP <timestamp> ...

- **Oracle syntax since version 10g (2005) (“Flashback Query”):**
  
  ... FROM <table> AS OF TIMESTAMP <timestamp> ...

- **Examples:**
  
  SELECT * FROM customers AS OF SYSTEM TIME current_timestamp ;
  SELECT * FROM customers AS OF SYSTEM TIME current_date - 3 days ;
  SELECT * FROM customers AS OF SYSTEM TIME current_timestamp - 1 min ;
  SELECT * FROM customers AS OF SYSTEM TIME TIMESTAMP '2015-01-22' ;
  SELECT * FROM customers AS OF SYSTEM TIME :hv ;

Note: Oracle & DB2 also have similar syntax for “business time”;

the standard does not (yet) propose a syntax for this

**DB2:** ... FROM <table> FOR BUSINESS_TIME AS OF <timestamp>

**Oracle:** ... FROM <table> AS OF PERIOD FOR <business_time> <timestamp>
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>2015-01-28 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>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>2015-01-28 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” columns; ...

Temporal tables and PostgreSQL - FOSDEM 2015

Temporal tables in PostgreSQL
1. Relational databases and historic (or versioned) data
2. Temporal tables and the ISO SQL:2011 standard
3. Transaction time versus business time
4. Table setup for “system time” versioning
5. System time in PostgreSQL
6. System time: some use cases
7. Business time: data validity time period
8. Business time in PostgreSQL
9. Business time: use case
10. Bi-temporal tables
11. Further reading
Table setup for “system time” versioning: SQL:2011

CREATE TABLE customers
    (id integer NOT NULL,
     name varchar(64),
     address varchar(128),
     telephone varchar(32),
     amount_sold dec(9,2),
     valid_from timestamp GENERATED ALWAYS AS SYSTEM VERSION START,
     valid_until timestamp GENERATED ALWAYS AS SYSTEM VERSION END,
     PRIMARY KEY (id)
    )
WITH SYSTEM VERSIONING;

- may also ALTER existing table: ADD two columns & versioning:
  ALTER TABLE customers
      ADD COLUMN s1 timestamp GENERATED ALWAYS AS SYSTEM VERSION START;
  ALTER TABLE customers
      ADD COLUMN s2 timestamp GENERATED ALWAYS AS SYSTEM VERSION END;
  ALTER TABLE customers
      ADD SYSTEM VERSIONING;

- may later drop the versioning again:
  ALTER TABLE customers DROP SYSTEM VERSIONING;
Table setup for “system time” versioning: how DB2 does it

Available since 2010 (DB2 version 10)

```
CREATE TABLE customers
(id integer 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
```
“Oracle Flashback Query” since Oracle 10g (2005):

- **Available for all tables (when Flashback is enabled)**

- **Pseudo-columns:**
  
  - `versions_starttime`
  - `versions_endtime`
  - `versions_xid`

- **integrated with “ordinary” rollback mechanisms**
Table setup for “system time” versioning: configuration issues

- base and history table *must* have byte-compatible rows:
  - same column names, same data types, same order & NOT NULLs
  - 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 physical implementation
    (like, e.g., partitioning, compression, tablespace&buffer choices)
  - 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
- No necessity of having a PK !!
Temporal tables in PostgreSQL

1. Relational databases and historic (or versioned) data
2. Temporal tables and the ISO SQL:2011 standard
3. Transaction time versus business time
4. Table setup for “system time” versioning
5. System time in PostgreSQL
6. System time: some use cases
7. Business time: data validity time period
8. Business time in PostgreSQL
9. Business time: use case
10. Bi-temporal tables
11. Further reading

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 9</td>
<td>016/123456</td>
<td>1043.50</td>
<td>2013-02-02 14:02:02</td>
<td>infinity</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>infinity</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/1234567</td>
<td>6100.00</td>
<td>2015-01-28 15:13:32</td>
<td>infinity</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>infinity</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>infinity</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>infinity</td>
</tr>
</tbody>
</table>

customers table:

id | name   | address  | telephone | amount_sold | valid_from     | valid_until |
---|--------|----------|-----------|-------------|---------------|-------------|
1  | Janssen| Singel 9 | 016/123456| 943.50      | 2013-02-02 14:02:02 | infinity    |
1  | Janssen| Singel 9 | 016/123456| 943.50      | 2004-03-30 15:13:42 | 2013-02-02 14:02:02 |
3  | Thiery | Zand 98  | 03/1234567| 6100.00     | 2010-01-01 00:00:00 | 2015-01-28 15:13:32 |
4  | Pieters| Rand 7A  | 010.00    | 100.00      | 2010-08-31 12:21:53 | 2012-07-21 16:24:13 |
4  | Pieters| Berg 71  | 010.00    | 100.00      | 2012-07-21 16:24:13 | 2012-12-31 23:59:59 |

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>
</table>
1  | Janssen| Singel 9 | 016/123456| 943.50      | 2011-03-12 09:13:42 | 2013-02-02 14:02:02 |
3  | Thiery | Zand 98  | 03/1234567| 6100.00     | 2010-01-01 00:00:00 | 2015-01-28 15:13:32 |
4  | Pieters| Rand 7A  | 010.00    | 100.00      | 2010-08-31 12:21:53 | 2012-07-21 16:24:13 |
4  | Pieters| Berg 71  | 010.00    | 100.00      | 2012-07-21 16:24:13 | 2012-12-31 23:59:59 |

(note: precision of timestamp columns: may contain fractional digits and/or time zone)
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 the RDBMS:
    - `valid_from`: with `current_timestamp`
    - `valid_until`: with ‘infinity’
- **on UPDATE of row(s) in customer:**
  - original (unchanged) row is “moved” to `customer_history`
    - where `valid_until` is changed to `current_timestamp`
  - modified row: `valid_from` is modified to `current_timestamp`
- **on DELETE of row(s) in customer:**
  - original (old) row is “moved” to `customer_history`
    - where `valid_until` is changed to `current_timestamp`
Interpretation of system time validity intervals

<table>
<thead>
<tr>
<th>Date</th>
<th>Date</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-01-01-00.00.00</td>
<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>
<td>2013-02-02-14.02.02</td>
</tr>
</tbody>
</table>

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

- “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
  (even with 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 START & END columns should be non-updatable
  - history table should be non-updatable
  ==> otherwise it would be possible to create inconsistent data
     (viz. violate temporal uniqueness on PK)
     & to forge the real history
     (DB2 allows this! => carefully set history table authorisations)
System time: additional DML possibilities

SELECT ... FROM customers AS OF SYSTEM TIME current_timestamp is equivalent to
SELECT ... FROM customers
and should not need to access the history table (but it does in DB2!)

SELECT ... FROM customers AS OF SYSTEM TIME current_date is NOT equivalent to the above!
==> it’s equivalent to “last midnight”

SELECT ... FROM customers AS OF SYSTEM TIME current_date + INTERVAL '1 day'
is essentially INVALID (as is any future date), but will be accepted (by DB2, not Oracle)

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

SELECT ... FROM customers AS OF SYSTEM TIME current_timestamp is equivalent to
SELECT ... FROM customers
and should not need to access the history table (but it does in DB2!)

SELECT ... FROM customers AS OF SYSTEM TIME current_date is NOT equivalent to the above!
==> it’s equivalent to “last midnight”

SELECT ... FROM customers AS OF SYSTEM TIME current_date + INTERVAL '1 day'
is essentially INVALID (as is any future date), but will be accepted (by DB2, not Oracle)

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
The query

SELECT ... FROM customers AS OF SYSTEM TIME <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>

So it could be important to create index(es)

on columns valid_from and/or valid_until,

possibly composite with other columns
There are some partial implementations & proposals:

   (see pgxn.org/dist/temporal_tables/1.0.1/)
   ==> working implementation for the “auto-archive” part (triggers)
   ==> does not modify the SELECT or CREATE TABLE syntax

2. Miroslav Šimulcik
   (see wiki.postgresql.org/wiki/SQL2011Temporal)
   ==> is only a design proposal: implementation not (yet) public

Technical choices -- important aspects to consider:

- preferably use the TSTZRANGE type ? (see next page)
- is there need for a TRANS_ID ?
- performance considerations: the triggers; GiST & GIN indexes; ...
- visibility/accessibility of the history table ?
Temporal tables in PostgreSQL

1. Relational databases and historic (or versioned) data
2. Temporal tables and the ISO SQL:2011 standard
3. Transaction time versus business time
4. Table setup for “system time” versioning
5. System time in PostgreSQL
6. System time: some use cases
7. Business time: data validity time period
8. Business time in PostgreSQL
9. Business time: use case
10. Bi-temporal tables
11. Further reading

Intermezzo - range data types in PostgreSQL

Since Pg 9.2; non-standard

Range value = a pair of values of a certain (ordinal) base type

Interpretation: is the “range” of all values between those end points

CREATE TYPE <name> AS RANGE ( SUBTYPE = <type> )

e.g.: CREATE TYPE tsrange AS RANGE ( SUBTYPE = timestamp )
   ==> is a predefined type, together with tstzrange, int4range, daterange, ...

Notation for range constants: (mix of inclusive / exclusive the end points)
' [ 3, 7 ]'       '[ 2015-01-01, 2016-01-01 )'       '( 3.1415 , 3.1416 )'       '[2015-01-01,)

New predicates for range columns: (beware: need explicit casts ...)
   “contains”:   '[ 3, 7 ]' @>  4
   isempty( '(' 4, 5 ')

New operators:
   '[3,7] * '[5,9]'  (returns '[5,8]')
   upper('[3,8]')   (returns 8 )
   '[3,7] + '[5,9]'  (returns '3.10')
   lower('[3,8]')   (returns '3' )
   '[3,7] - '[5,9]'  (returns '3.5')

===> is a predefined type, together with tstzrange, int4range, daterange, ...

Notation for range constants: (mix of inclusive / exclusive the end points)
' [ 3, 7 ]'       '[ 2015-01-01, 2016-01-01 )'       '( 3.1415 , 3.1416 )'       '[2015-01-01,)

New predicates for range columns: (beware: need explicit casts ...)
   “contains”:   '[ 3, 7 ]' @>  4
   isempty( '(' 4, 5 ')

New operators:
   '[3,7] * '[5,9]'  (returns '[5,8]')
   upper('[3,8]')   (returns 8 )
   '[3,7] + '[5,9]'  (returns '3.10')
   lower('[3,8]')   (returns '3' )
   '[3,7] - '[5,9]'  (returns '3.5')

===> is a predefined type, together with tstzrange, int4range, daterange, ...

Notation for range constants: (mix of inclusive / exclusive the end points)
' [ 3, 7 ]'       '[ 2015-01-01, 2016-01-01 )'       '( 3.1415 , 3.1416 )'       '[2015-01-01,)

New predicates for range columns: (beware: need explicit casts ...)
   “contains”:   '[ 3, 7 ]' @>  4
   isempty( '(' 4, 5 ')

New operators:
   '[3,7] * '[5,9]'  (returns '[5,8]')
   upper('[3,8]')   (returns 8 )
   '[3,7] + '[5,9]'  (returns '3.10')
   lower('[3,8]')   (returns '3' )
   '[3,7] - '[5,9]'  (returns '3.5')
System time: some use cases

1. Compliance & auditing:
   - never need to use “AS OF” queries
   - history table functions as a “change log”

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 RECURSIVE dates(t) AS (
     SELECT cast('2014-01-01' AS timestamp)
     UNION ALL
     SELECT t + INTERVAL '1 month' FROM dates WHERE t < current_date
   )
   SELECT SUM(amount_sold) FROM dates d, customers AS OF SYSTEM TIME d.t
   GROUP BY d.t
   -- (although this won’t work syntactically: “AS OF” needs host variable or constant)
   ```

Temporal tables and PostgreSQL - FOSDEM 2015

Temporal tables in PostgreSQL

1. Relational databases and historic (or versioned) data
2. Temporal tables and the ISO SQL:2011 standard
3. Transaction time versus business time
4. Table setup for “system time” versioning
5. System time in PostgreSQL
6. System time: some use cases
7. Business time: data validity time period
8. Business time in PostgreSQL
9. Business time: use case
10. Bi-temporal tables
11. Further reading
System time: some use cases

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

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

4. Point in time recovery
   UPDATE customers c
   SET address = (SELECT address FROM customers AS OF SYSTEM TIME :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 the RDBMS

Careful: without an “AS OF”: returns the full history (all versions):
... FROM <table> ...

Temporal tables and PostgreSQL - FOSDEM 2015

Temporal tables in PostgreSQL
1. Relational databases and historic (or versioned) data
2. Temporal tables and the ISO SQL:2011 standard
3. Transaction time versus business time
4. Table setup for “system time” versioning
5. System time in PostgreSQL
6. System time: some use cases
7. Business time: data validity time period
8. Business time in PostgreSQL
9. Business time: use case
10. Bi-temporal tables
11. Further reading
Table setup for “business time” versioning: how DB2 does it

CREATE TABLE customers

(id integer NOT NULL
, name varchar(64)
, address varchar(128)
, telephone varchar(32)
, amount_sold decimal(9,2)
, valid_from timestamp NOT NULL
, valid_until timestamp NOT NULL
, PERIOD BUSINESS_TIME (valid_from, valid_until) -- no “FOR” ...
, 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 (standards compliant?):

  CREATE UNIQUE INDEX <name>
  ON <table> (<cols>, BUSINESS_TIME WITHOUT OVERLAPS) ;

Temporal tables and PostgreSQL - FOSDEM 2015

1. Relational databases and historic (or versioned) data
2. Temporal tables and the ISO SQL:2011 standard
3. Transaction time versus business time
4. Table setup for “system time” versioning
5. System time in PostgreSQL
6. System time: some use cases
7. Business time: data validity time period
8. Business time in PostgreSQL
9. Business time: use case
10. Bi-temporal tables
11. Further reading
Since Oracle 12c (2013):

```sql
CREATE TABLE customers
(id integer NOT NULL,
 , name varchar(64),
 , address varchar(128),
 , telephone varchar(32),
 , amount_sold decimal(9,2),
 , valid_from timestamp DEFAULT current_timestamp,
 , valid_until timestamp DEFAULT to_timestamp('31.12.9999'),
 , PERIOD FOR business_time (valid_from, valid_until)));
```

- “No overlaps” must be manually guaranteed, it seems ...
- should not declare “id” as primary key, of course ...
Business time: inserting data

- No defaults for “valid_from” and “valid_until”
  
  ==> application *must* explicitly state the validity period
  (if these columns are NOT NULL, which they shouldn’t be)

  ==> “valid_until” could still be set to e.g. 2999-12-31 or ‘infinity’

<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>infinity</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>infinity</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/1234567</td>
<td>6100.00</td>
<td>2015-01-28 15:13:32</td>
<td>infinity</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>infinity</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>infinity</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>infinity</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>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>2015-01-28 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>
Temporal tables and PostgreSQL

Business time: updating data

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

```
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>infinity</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>infinity</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/7654321</td>
<td>6100.00</td>
<td>2015-01-28 15:13:32</td>
<td>infinity</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>infinity</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>infinity</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>infinity</td>
</tr>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>016/123456</td>
<td>943.50</td>
<td>2004-03-30 15:13:42</td>
<td>2013-02-02 14:02:02</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>2015-01-28 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:

```
UPDATE customers FOR PORTION OF BUSINESS_TIME
FROM TIMESTAMP '2015-09-01 00:00:00' TO TIMESTAMP 'infinity'
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>infinity</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>infinity</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/7654321</td>
<td>6100.00</td>
<td>2015-09-01 00:00:00</td>
<td>infinity</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/1234567</td>
<td>6100.00</td>
<td>2015-01-28 15:13:32</td>
<td>2015-09-01 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>infinity</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>infinity</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>infinity</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>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>2015-01-28 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!```
Temporal tables in PostgreSQL

1. Relational databases and historic (or versioned) data
2. Temporal tables and the ISO SQL:2011 standard
3. Transaction time versus business time
4. Table setup for “system time” versioning
5. System time in PostgreSQL
6. System time: some use cases
7. Business time: data validity time period
8. Business time in PostgreSQL
9. Business time: use cases
10. Bi-temporal tables
11. Further reading

Business time: deleting data

- **Delete statements with “temporal” specification:**

```sql
DELETE FROM customers FOR PORTION OF BUSINESS_TIME
FROM TIMESTAMP '2016-01-01 00:00:00' TO TIMESTAMP 'infinity'
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>infinity</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>infinity</td>
</tr>
<tr>
<td>3</td>
<td>Thiery</td>
<td>Square 1</td>
<td>03/7654321</td>
<td>6100.00</td>
<td>2015-09-01 00:00:00</td>
<td>2016-01-01 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>2015-01-28 15:13:32</td>
<td>2015-09-01 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>infinity</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>infinity</td>
</tr>
<tr>
<td>10</td>
<td>Zander</td>
<td>Centre 4</td>
<td></td>
<td>123.45</td>
<td>2011-03-15 09:00:00</td>
<td>infinity</td>
</tr>
<tr>
<td>1</td>
<td>Janssen</td>
<td>Singel 9</td>
<td>016/123456</td>
<td>943.50</td>
<td>2013-02-02 14:02:02</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>2015-01-28 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!
Temporal tables and PostgreSQL

Business time: guarantees & caveats

- “Ordinary” SELECT always accesses the full table
  ==> including history!
  ==> unless including the necessary WHERE predicates on columns valid_from & valid_until ...

  OR by using the “new” AS OF syntax (similar to system time):

- DB2 syntax for querying a “business temporal” table:
  ... FROM <table> FOR BUSINESS_TIME AS OF <timestamp or date> ...

- Oracle syntax: ... FROM <table> AS OF PERIOD FOR <business_time> <timestamp>

- no SQL ANSI/ISO standard (yet) ?

- “Ordinary” INSERT/UPDATE/DELETE: all versions (incl. history) !
  ==> unless with 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 should be guaranteed by the RDBMS
Business time: additional DML possibilities in DB2

- SELECT ... FROM customers FOR BUSINESS_TIME AS OF current_timestamp is *NOT* equivalent to
- SELECT ... FROM customers

- SELECT ...
  FROM customers FOR BUSINESS_TIME AS OF current_date + INTERVAL '1 day'
is totally *VALID* (as is any future date)

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

- SELECT ... FROM customers FOR BUSINESS_TIME BETWEEN <ts1> AND <ts2>
  - the time range is <ts1> *inclusive* and also <ts2> *inclusive*
Business time in PostgreSQL

No additional infrastructure needed, except for (ideally):

- new “WITHOUT OVERLAP” indexes / UNIQUE constraints
- new “temporal” table expression syntax to be added to SELECT:
  SELECT ... FROM <t> AS OF <time_period_spec> <timestamp>

  SELECT ... FROM <t> FOR <time_period_spec> BETWEEN <ts1> AND <ts2>

- possibly support mixed syntax for the “time range” column pair:
  - separate columns, ordinary timestamp predicates (< >= etc)
  - tsrange syntax:
    - “contains”: e.g. '[ 2015-01-01, 2015-07-01 )' @> current_timestamp
    - “intersects”: e.g. '[ 2015-01-01, 2015-07-01 )' && '[ 2014-11-01, 2015-02-01 ]'
    - “intersection”: e.g. '[ 2015-01-01, 2015-07-01 )' * '[ 2014-11-01, 2015-02-01 ]'
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 FOR 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>infinity</td>
</tr>
<tr>
<td>102</td>
<td>750.00</td>
<td>2012-01-01</td>
<td>infinity</td>
</tr>
<tr>
<td>103</td>
<td>150.00</td>
<td>2012-01-01</td>
<td>2015-07-01</td>
</tr>
<tr>
<td>103</td>
<td>120.00</td>
<td>2015-07-01</td>
<td>2015-09-01</td>
</tr>
<tr>
<td>103</td>
<td>3201.43</td>
<td>2016-01-01</td>
<td>infinity</td>
</tr>
</tbody>
</table>
Business time: use case

select * from products where prid=103;
PRID        PRICE       VALID_FROM VALID_UNTIL
----------- ----------- ---------- -----------
103         150.00 01/01/2012 07/01/2015
103         120.00 07/01/2015 09/01/2015
103         3201.43 01/01/2016 12/30/9999

3 record(s) selected.

select * from products for business_time as of current_date where prid=103;
PRID        PRICE       VALID_FROM VALID_UNTIL
----------- ----------- ---------- -----------
103         150.00 01/01/2012 07/01/2015

1 record(s) selected.

select * from products for business_time from '01.01.2015' to '01.01.2016'
where prid=103;
PRID        PRICE       VALID_FROM VALID_UNTIL
----------- ----------- ---------- -----------
103         150.00 01/01/2012 07/01/2015
103         120.00 07/01/2015 09/01/2015

2 record(s) selected.

select * from products for business_time between '01.01.2015' and '01.07.2015'
where prid=103;
PRID        PRICE       VALID_FROM VALID_UNTIL
----------- ----------- ---------- -----------
103         150.00 01/01/2012 07/01/2015
103         120.00 07/01/2015 09/01/2015

2 record(s) selected.
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?

```sql
SELECT prid, valid_from, valid_until
FROM products AS OF SYSTEM TIME current_timestamp - 7 days
WHERE price = ( SELECT 0.8*price FROM products WHERE prid = p.prid) ;
```

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

```sql
SELECT prid, price
FROM products AS OF SYSTEM TIME current_timestamp - 7 days
FOR BUSINESS_TIME FROM '2015-07-01' TO '2015-09-01' ;
```
### Further reading

**Wikipedia:**

**SIGMOD 2012 paper (41:3, pp. 34-43) by Kulkarni & Michels (IBM)**

**Presentation by K. Kulkarni (IBM) on temporal features in SQL:2011**

**ISO SQL:2011 final draft:**
Questions, remarks, feedback, ... ?

Thank you!

Peter Vanroose
ABIS Training & Consulting
pvanroose@abis.be