Bitemporal Data
Making it happened in Postgres

Henrietta Dombrovskaya
Braviant Holding
Chad Slaughter
Scale Genius, Inc.

Including time into data models

Why we may need to include time into our data model?

- When was a change made? What exactly the change was? Was it an actual update or a correction of a mistake previously made?
- How did the data look at some moment in the past? How the data will look at some moment in the future?
- When did attribute X has value ‘a’? What was the value of attribute Y, when the value of attribute X was ‘a’?

References

- C.J. Date, Hugh Darwen, and Nikos Lorentzos. 2014. *Time and Relational Theory, Second Edition: Temporal Databases in the Relational Model and SQL (2nd ed.)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Tom Johnston. 2014. *Bitemporal Data: Theory and Practice (1st ed.)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Tom Johnston and Randall Weis. 2010. *Managing Time in Relational Databases: How to Design, Update and Query Temporal Data*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Krishna Kulkarni and Jan-Eike Michels. 2012. Temporal features in SQL: 2011. *SIGMOD Rec*. 41, 3 (October 2012), 34-43.
- Richard Thomas Snodgrass. 1999. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- ISO/IEC 9075-2:2011, Information technology — Database languages — SQL , 2011

The Linguistic Problem

Too many common terms with overlapping meaning.

- Valid Time
- Transaction Time
- System Time
- State Time
- Inscription Time
- Application Time
- Speech Act Time
- Assertive Time
- Effective Time

Why Bitemporal?

- Why do we need a second time dimension? Why isn't a timestamp good enough? Why isn't my history table or data changes table sufficient?
- How do you capture changes to the past, present, and future? Second time dimension is needed to capture changes for the past, present, and future while still allowing for corrections to the past and present.
- How do you correct bad data that already has time associated with it? How do you do it with a full history of corrects and normal updates? How do you audit the corrections?
- How do you insert data about the past and present but that will only be visible in the future?
- How can you merge active, Terabyte-sized databases with a instance switch over?
- How do you handle all this change data over time without changing your query?
- How do you handle a quarterly financial report with corrections and no query charges?

Regular, Temporal and Bitemporal tables

Regular Table

		Customer Number	Customer Name	Customer Type

Unitemporal Table

	Time Interval	Customer Number	Customer Name	Customer Type

Bitemporal Table

Time Interval	Time Interval	Customer Number	Customer Name	Customer Type

Bitemporal insert

When this happened

`now = 2015-05-01`

#	Effective Interval	Assertive Interval	Customer Number	Name	Type
1	[2015-06-01, ∞)	[2015-05-01 , ∞)	C100	John Doe	Silver

Bitemporal update

When this happened

`now = 2015-09-15`

#	Effective Interval	Assertive Interval	Customer No.	Name	Type
1	[2015-06-01, ∞)	[2015-05-01, 2015-09-15)	C100	John Doe	Silver
2	[2015-06-01, 2015-09-15)	[2015-09-15 , ∞)	C100	John Doe	Silver
3	[2015-09-15 , ∞)	[2015-09-15 , ∞)	C100	John Doe	Gold

Bitemporal correction

When this happened

`now = 2015-09-22`

#	Effective Interval	Assertive Interval	Customer No.	Name	Type
1	[2015-06-01, ∞)	[2015-05-01, 2015-09-15)	C100	John Doe	Silver
2	[2015-06-01, 2015-09-15)	[2015-09-15, ∞)	C100	John Doe	Silver
3	[2015-09-15, ∞)	[2015-09-15, 2015-09-22)	C100	John Doe	Gold
4	[2015-09-15, ∞)	[2015-09-22 , ∞)	C100	John Doe	Platinum

Bitemporal inactivation

When this happened

`now = 2015-11-05`

#	Effective Interval	Assertive Interval	Customer No.	Name	Type
1	[2015-06-01, ∞)	[2015-05-01, 2015-09-15)	C100	John Doe	Silver
2	[2015-06-01, 2015-09-15)	[2015-09-15, ∞)	C100	John Doe	Silver
3	[2015-09-15, ∞)	[2015-09-15, 2015-09-22)	C100	John Doe	Gold
4	[2015-09-15, ∞)	[2015-09-22, 2015-11-05)	C100	John Doe	Platinum
5	[2015-09-15, *2015-12-31*)	[2015-11-05 , ∞)	C100	John Doe	Platinum

Bitemporal deletion

When this happened

`now = 2015-11-17`

#	Effective Interval	Assertive Interval	Customer No.	Name	Type
1	[2015-06-01, ∞)	[2015-05-01, 2015-09-15)	C100	John Doe	Silver
2	[2015-06-01, 2015-09-15)	[2015-09-15, ∞)	C100	John Doe	Silver
3	[2015-09-15, ∞)	[2015-09-15, 2015-09-22)	C100	John Doe	Gold
4	[2015-09-15, ∞)	[2015-09-22, 2015-11-05)	C100	John Doe	Platinum
5	[2015-09-15, 2015-12-31)	[2015-11-05, 2015-11-17)	C100	John Doe	Platinum

Bitemporal constraints

- Temporal Entity Integrity (TEI) – the primary key and unique constraints for temporal tables are different – how?
- Temporal Referential Integrity (TRI) – the foreign keys should be defined differently – how?

How to support bitemporal data in PG 9.4 and up

*Ranges support and gist indexes makes it **so** easy!*

```
CREATE TABE bi_temporal.customers(
cust-nbr INTEGER,
cust-nm TEXT,
cust-type TEXT,
effective_range TSRANGE,
asserted_range TSRANGE,
row_created_at timestampz,
EXCLUDE USING gist (cust-nbr  WITH =, effective_range WITH &&, asserted_range WITH &&))
```

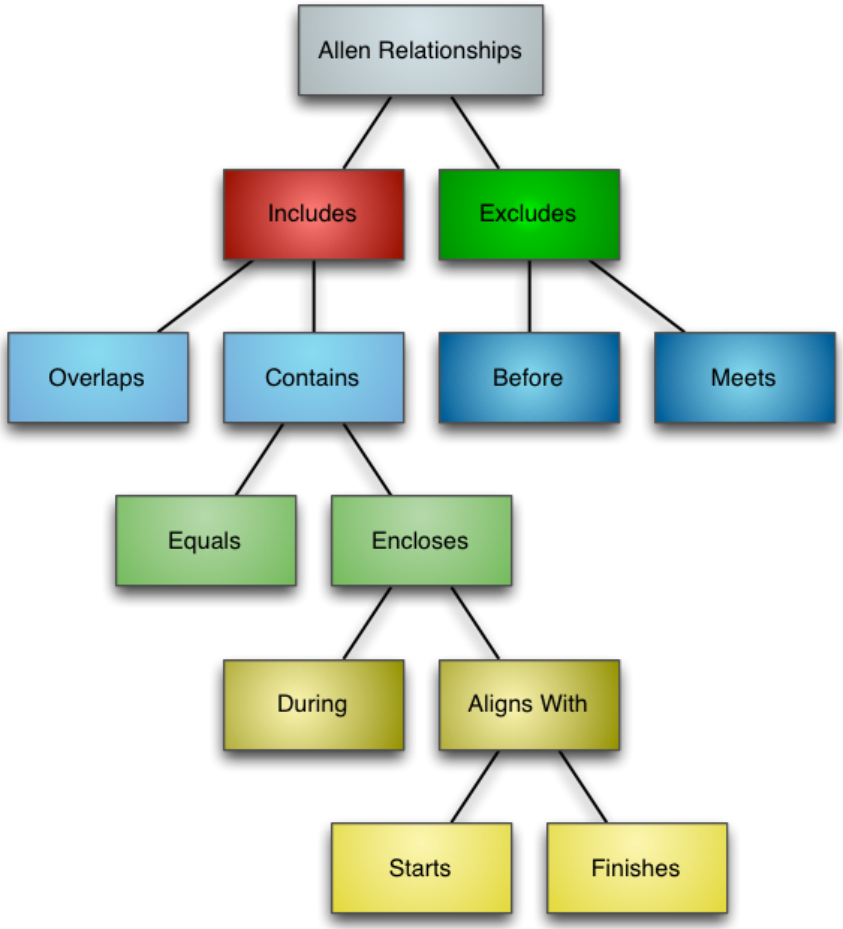
Table example: database_version

```
CREATE TABLE database_versions(
release_version_key serial NOT NULL,
release_version_id integer,
release_version numeric(3,1),
effective temporal_relationships.timeperiod,
asserted temporal_relationships.timeperiod,
row_created_at timestamp with time zone NOT NULL
    DEFAULT now(),
CONSTRAINT release_version_id_asserted_effective_excl
EXCLUDE  USING gist
(release_version_id WITH =, asserted WITH &&, effective
WITH &&)
)
```

Table example: postgres_clusters

```
CREATE TABLE bi_temp_tables.postgres_clusters(
 postgres_cluster_key serial NOT NULL
,postgres_cluster_id integer NOT NULL
,port integer
,name character varying(16)
,postgres_version integer
,archive boolean NOT NULL DEFAULT false
,preferred_auth_method text
,effective temporal_relationships.timeperiod
,asserted temporal_relationships.timeperiod
,row_created_at timestamp with time zone NOT NULL DEFAULT now(),
CONSTRAINT postgres_clusters_preferred_auth_method_fkey FOREIGN
KEY (preferred_auth_method) REFERENCES
bi_temp_tables.postgres_auth_methods (auth_method)
,CONSTRAINT postgres_cluster_id_asserted_effective_excl EXCLUDE
USING gist (postgres_cluster_id WITH =, asserted WITH &&,
effective WITH &&))
```

Allen Relationships



Functions representing Allen relationships

```
has_starts
has_finishes
equals
is_during
is_contained_in
has_during
is_overlaps
has_overlaps
is_before
```

```
is_after
has_before
is_meets
has_meets
has_includes
has_contains
has_aligns_with
has_encloses
has_excludes
```

Defining domains and ranges

```
create domain timeperiod as tstzrange;
create domain time_endpoint as timestamptz;


create or replace function timeperiod(
p_range_start time_endpoint,
p_range_end time_endpoint)
RETURNS timeperiod
language sql IMMUTABLE
as $func$
select tstzrange(p_range_start,
p_range_end,'[)')::timeperiod;
$func$;
```

Create bitemporal table

```
bitemporal_internal.ll_create_bitemporal_
table(
p_table text,
p_table_definition text,
p_business_key text)

RETURNS BOOLEAN
```

Examples

```
select bitemporal_internal.ll_create_bitemporal_table(
'bi_temp_tables.database_versions'
,'release_version_key serial
 ,release_version_id    integer  --business key
 ,release_version   numeric(3,1)'  -- temporal unique
,'release_version_id');

select bitemporal_internal.ll_create_bitemporal_table(
'bi_temp_tables.postgres_clusters'
,'postgres_cluster_key serial
 ,postgres_cluster_id integer NOT NULL
 ,port                integer -- bitemporal unique
 ,name                varchar(16)
 ,postgres_version    integer  -- bitemporal fk
 ,archive             boolean   DEFAULT false NOT NULL
 ,preferred_auth_method text  references
bi_temp_tables.postgres_auth_methods ( auth_method )'
,'postgres_cluster_id')
```

Check, whether the table in question is bitemporal

```
ll_is_bitemporal_table(
p_table text)
RETURNS boolean
```

Bitemporal insert

When this happened

```
now = 2015-05-01
```

Function call

```
select ll_bitemporal_insert(
    'customers',
    $$ customer_number, customer_name, customer_type $$,
    $$ 'C100','John Doe', 'Silver' $$,
    timeperiod('2015-06-01','infinity'),
    timeperiod('2015-05-01','infinity')
);
```

#	Effective Interval	Assertive Interval	Customer Number	Name	Type
1	[2015-06-01, ∞)	[2015-05-01 , ∞)	C100	John Doe	Silver

Bitemporal correction

When this happened

```
now = 2015-09-22
```

Function call

```
select ll_bitemporal_correction(
    'customers',
    $$ customer_type $$,
    $$ 'Platinum' $$,
    $$ customer_number $$,
    $$ 'C100' $$,
    timeperiod('2015-09-15','infinity'),
);
```

#	Effective Interval	Assertive Interval	Customer No.	Name	Type
1	[2015-06-01, ∞)	[2015-05-01, 2015-09-15)	C100	John Doe	Silver
2	[2015-06-01, 2015-09-15)	[2015-09-15, ∞)	C100	John Doe	Silver
3	[2015-09-15, ∞)	[2015-09-15, 2015-09-22)	C100	John Doe	Gold
4	[2015-09-15, ∞)	[2015-09-22 , ∞)	C100	John Doe	Platinum

Bitemporal update

When this happened

```
now = 2015-09-15
```

Function call

```
select ll_bitemporal_update(
    'customers',
    $$ customer_type $$,
    $$ 'Gold' $$,
    $$ customer_number $$,
    $$ 'C100' $$,
    timeperiod('2015-09-15','infinity'),
    timeperiod('2015-09-15','infinity')
);
```

#	Effective Interval	Assertive Interval	Customer No.	Name	Type
1	[2015-06-01, ∞)	[2015-05-01, 2015-09-15)	C100	John Doe	Silver
2	[2015-06-01, 2015-09-15)	[2015-09-15, ∞)	C100	John Doe	Silver
3	[2015-09-15, ∞)	[2015-09-15, ∞)	C100	John Doe	Gold

Bitemporal inactivate

When this happened

```
now = 2015-11-05
```

Function call

```
select ll_bitemporal_inactivate(
    'customers',
    $$ customer_number $$,
    $$ 'C100' $$,
    timeperiod('2015-12-31','infinity'),
    timeperiod('2015-11-05','infinity'),
);
```

#	Effective Interval	Assertive Interval	Customer No.	Name	Type
1	[2015-06-01, ∞)	[2015-05-01, 2015-09-15)	C100	John Doe	Silver
2	[2015-06-01, 2015-09-15)	[2015-09-15, ∞)	C100	John Doe	Silver
3	[2015-09-15, ∞)	[2015-09-15, 2015-09-22)	C100	John Doe	Gold
4	[2015-09-15, ∞)	[2015-09-22, 2015-11-05)	C100	John Doe	Platinum
5	[2015-09-15, *2015-12-31*)	[2015-11-05, ∞)	C100	John Doe	Platinum

Bitemporal delete

```
now = 2015-11-17
```

Function call

```
select ll_bitemporal_delete(
    'customers',
    $$ customer_number $$,
    $$ 'C100' $$,
    timeperiod('2015-11-17','infinity'),
);
```

#	Effective Interval	Assertive Interval	Customer No.	Name	Type
1	[2015-06-01, ∞)	[2015-05-01, 2015-09-15)	C100	John Doe	Silver
2	[2015-06-01, 2015-09-15)	[2015-09-15, ∞)	C100	John Doe	Silver
3	[2015-09-15, ∞)	[2015-09-15, 2015-09-22)	C100	John Doe	Gold
4	[2015-09-15, ∞)	[2015-09-22, 2015-11-05)	C100	John Doe	Platinum
5	[2015-09-15, 2015-12-31)	[2015-11-05, 2015-11-17)	C100	John Doe	Platinum

How to define constraints

- We need to support the following constraint types:
  - Primary key
  - Unique
  - Check – no difference from regular tables
  - IS/IS NOT NULL – no difference from regular tables
  - Foreign key
- Metacode
  - How we record the presence of the bitemporal constraints?

Define PK

We define a bitemporal primary key when we create a bitemporal table, i.e. there can't be a temporal table without a bitemporal PK:

```
,CONSTRAINT postgres_cluster_id_asserted_effective_excl EXCLUDE
USING gist (postgres_cluster_id WITH =, asserted WITH &&, effective
WITH &&))
```

In addition we define a corresponding check constraint:

```
,CONSTRAINT "bitemporal pk postgres_cluster_id" check(true or 'pk' <>
'@postgres_cluster_id@')
```

Function:

```
select
bitemporal_internal.pk_constraint('postgres_cluster_id');
```

Define unique constraint

Defining unique constraint is no different from defining the PK, we just name them differently:

```
CONSTRAINT "postgres_port_asserted_effective_excl” EXCLUDE USING gist
(port WITH =, asserted WITH &&, effective WITH &&)
```

In addition we define a corresponding check constraint:

```
,CONSTRAINT "bitemporal pk postgres_cluster_id" check(true or
'unique' <> '@postgres_port@')
```

Function:

```
select bitemporal_internal.unique_constraint('port');
```

Define foreign key constraint (bi-temporal referential integrity)

The difficulty of verifying the bi-temporal FK is that the PK/UQ in the parent table should be effective and asserted all the time when a dependent record is effective/asserted

```
 CONSTRAINT "bitemporal fk
postgres_version_database_versionsrelease_version"      check
(true or 'fk' <> '@postgres_version ->
database_versions(release_version)@'));
```

Function:

```
select bitemporal_internal.fk_constraint(
'postgres_version'
,'database_versions'
,'release_version');
```

FK constraint validation internals

What happens, when a new FK is being created:

```
bitemporal_internal.ll_add_fk(
        p_schema_name text,
        p_table_name text,
        p_column_name text,
        p_source_schema_name text,
        p_source_table_name text,
        p_source_column_name text)
returns text
```

FK creation steps

- Check whether the referencing field is a PK/UQ
  - validate_bitemporal_pk_uq
- Create check constraint
  - fk_constraint
- Check whether the validation on the parent table field already exists
  - ll_lookup_validation_function
- If not, create it
  - ll_generate_fk_validate
- create trigger on insert/update and a trigger function

FK validation on insert/update

When a new record is inserted or a FK attriibute is updated:

- For each attribute which is a FK, run a matching validation function, and insert/ update record only if all are valid .

Repo:

https://github.com/scalegenius/pg_bitemporal

Future work

- Complete Bitemporal Foreign Keys Support
- Deploy in a Production Application
- Convert the Code from PL/PgSQL to C and package as an Extension
- Convert Constraints to use the Catalog
- Better Indexing Strategy