TEMPORAL DATABASES:

THEORY AND POSTGRES 2024

Paul A. Jungwirth

20 June 2024

pdxpug

THE PROBLEM

LOST INFORMATION

THE PROBLEM

LOST INFORMATION

- questionnaires: changing questions, options

THE PROBLEM

LOST INFORMATION

- questionnaires: changing questions, options
- e-commerce: product price, other attributes

THE PROBLEM

LOST INFORMATION

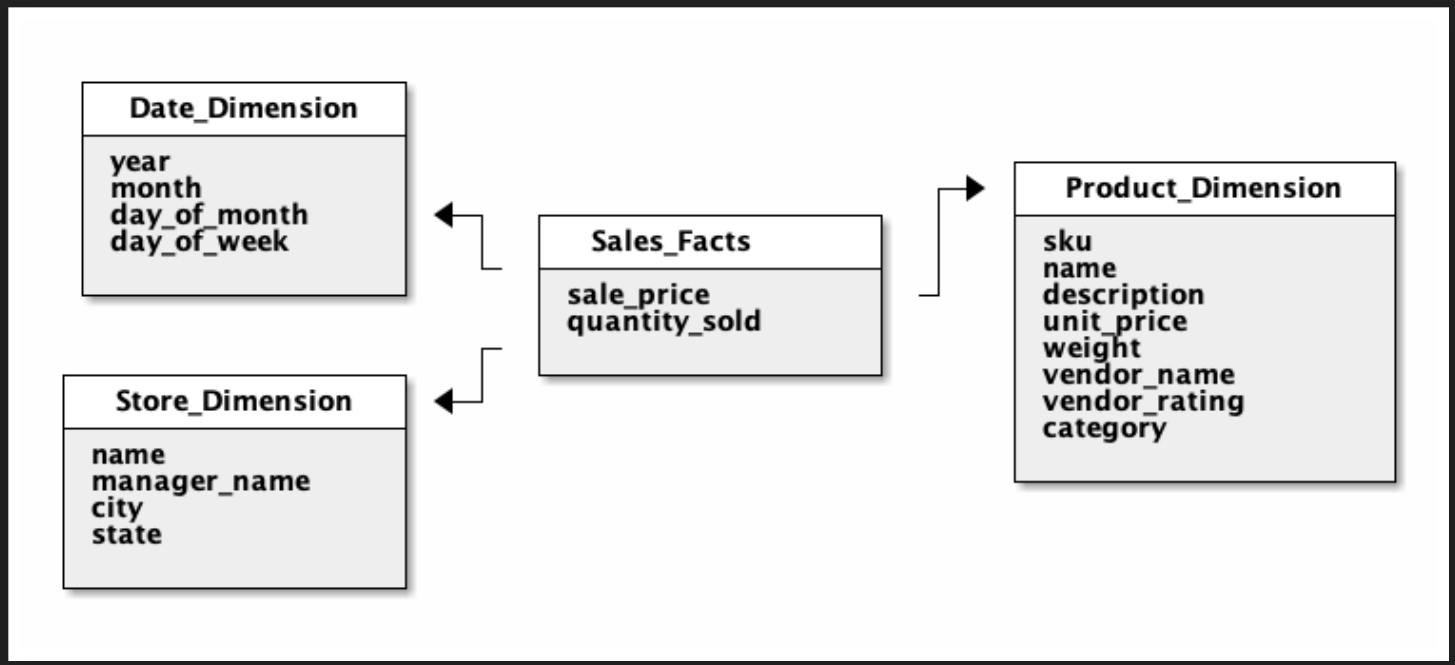
- questionnaires: changing questions, options
- e-commerce: product price, other attributes
- real estate: house renovations

THE PROBLEM

LOST INFORMATION

- questionnaires: changing questions, options
- e-commerce: product price, other attributes
- real estate: house renovations
- employees: position, salary, employment period

OLAP PROBLEMS TOO



"SLOWLY-CHANGING DIMENSIONS"

- Type I: Overwrite it
- Type II: Add a Row
- Type III: Add a Column (good for one change)

"SLOWLY-CHANGING DIMENSIONS"

- Type I: Overwrite it
- Type II: Add a Row
- Type III: Add a Column (good for one change)

later:

- Type IV: Mini-dimensions
- Type V: Mini-dimensions with outriggers
- Type VI: Original vs Current
- Type VII: Type I + Type II

THE FIRST DBA?

ΑΡΙΣΤΟΤΕΛΟΥΣ ΚΑΤΗΓΟΡΙΑΙ·

ΠΕΡΙ ΟΜΩΝΥΜΩΝ·

ΟΜΩΝΥΜΑ λέγεται, ὧν ὄνομα μόνον κοινόν, ὁ δὲ κατὰ τοὔνομα λόγος τῆς οὐσίας ἕτερος, οἷον ζῶον, ὅ, τε ἄνθρωπος, καὶ τὸ γεγραμμένον· τούτων γὰρ ὄνομα μόνον κοινόν, ὁ δὲ κατὰ τοὔνομα λόγος τῆς οὐσίας ἕτερος· ἐὰν γὰρ ἀποδιδῶτις τί ἐστιν αὐτῶν ἑκατέρῳ, τὸ ζώῳ εἶναι, ἴδιον ἑκατέρου λόγον ἀποδώσει·

ΠΕΡΙ ΣΥΝΩΝΥΜΩΝ·

Συνώνυμα δὲ λέγεται, ὧν, τό τε ὄνομα κοινόν, καὶ ὁ κατὰ τοὔ

of access paths in the total model for the community of users of a data bank would eventually become excessively large.

1.3. A Relational View of Data

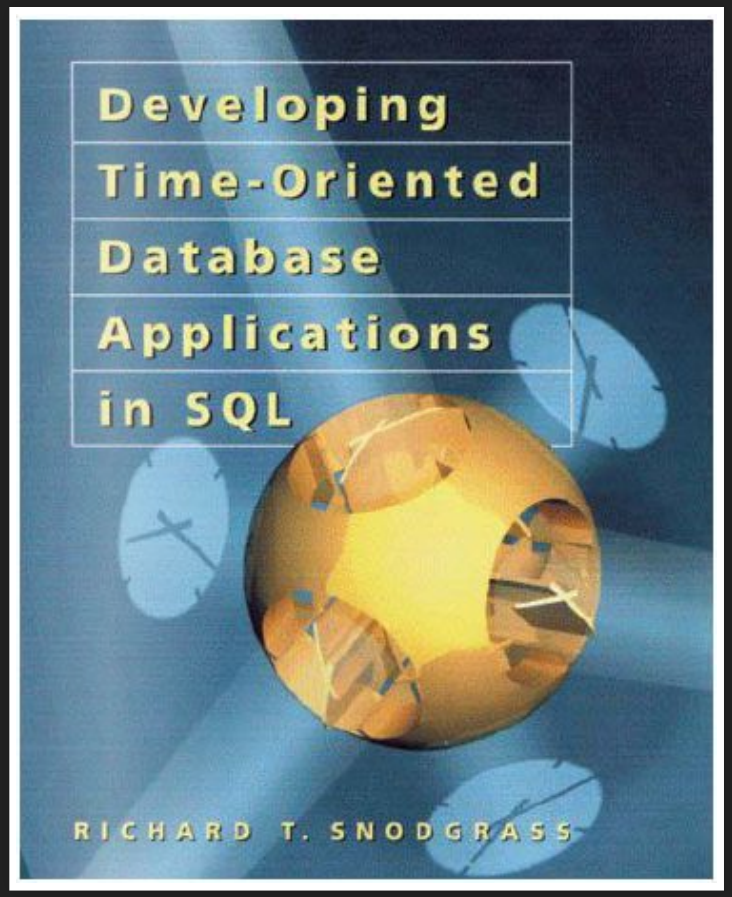
The term *relation* is used here in its accepted mathematical sense. Given sets $S_1, S_2, \cdots, S_n$ (not necessarily distinct), R is a relation on these n sets if it is a set of n-tuples each of which has its first element from S_1, its second element from S_2, and so on.[1] We shall refer to S_j as the jth *domain* of R. As defined above, R is said to have *degree n*. Relations of degree 1 are often called *unary*, degree 2 *binary*, degree 3 *ternary*, and degree n *n-ary*.

For expository reasons, we shall frequently make use of an array representation of relations, but it must be remembered that this particular representation is not an essential part of the relational view being expounded. An ar-

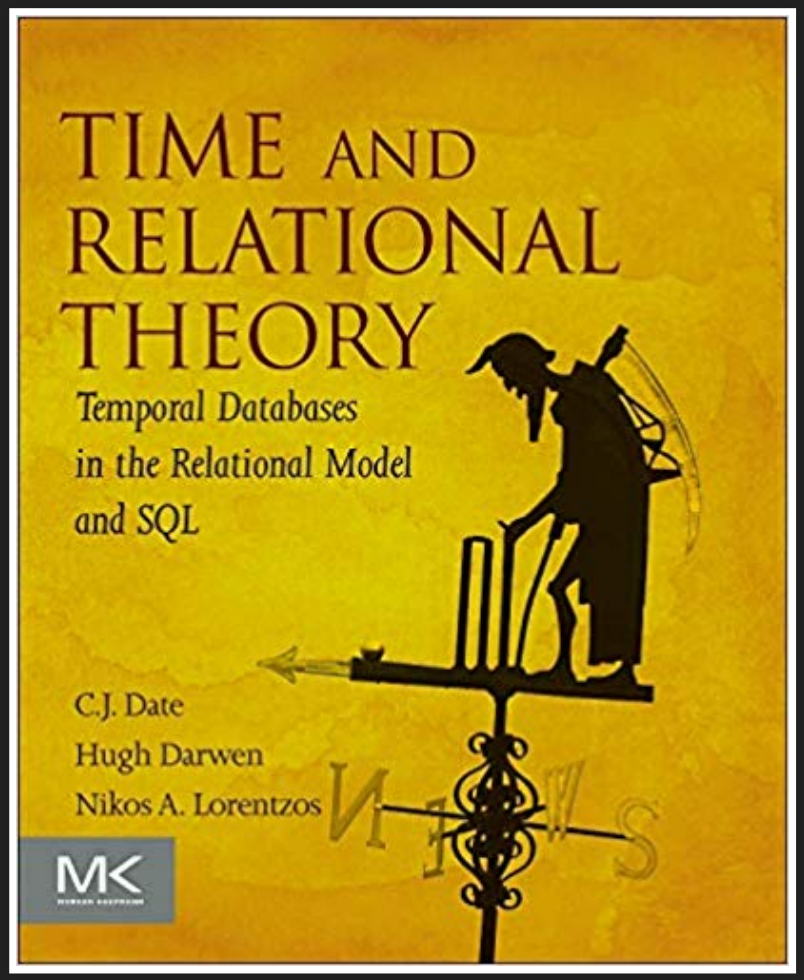
[1] More concisely, R is a subset of the Cartesian product $S_1 \times S_2 \times \cdots \times S_n$.

Volume 13 / Number 6 / June, 1970

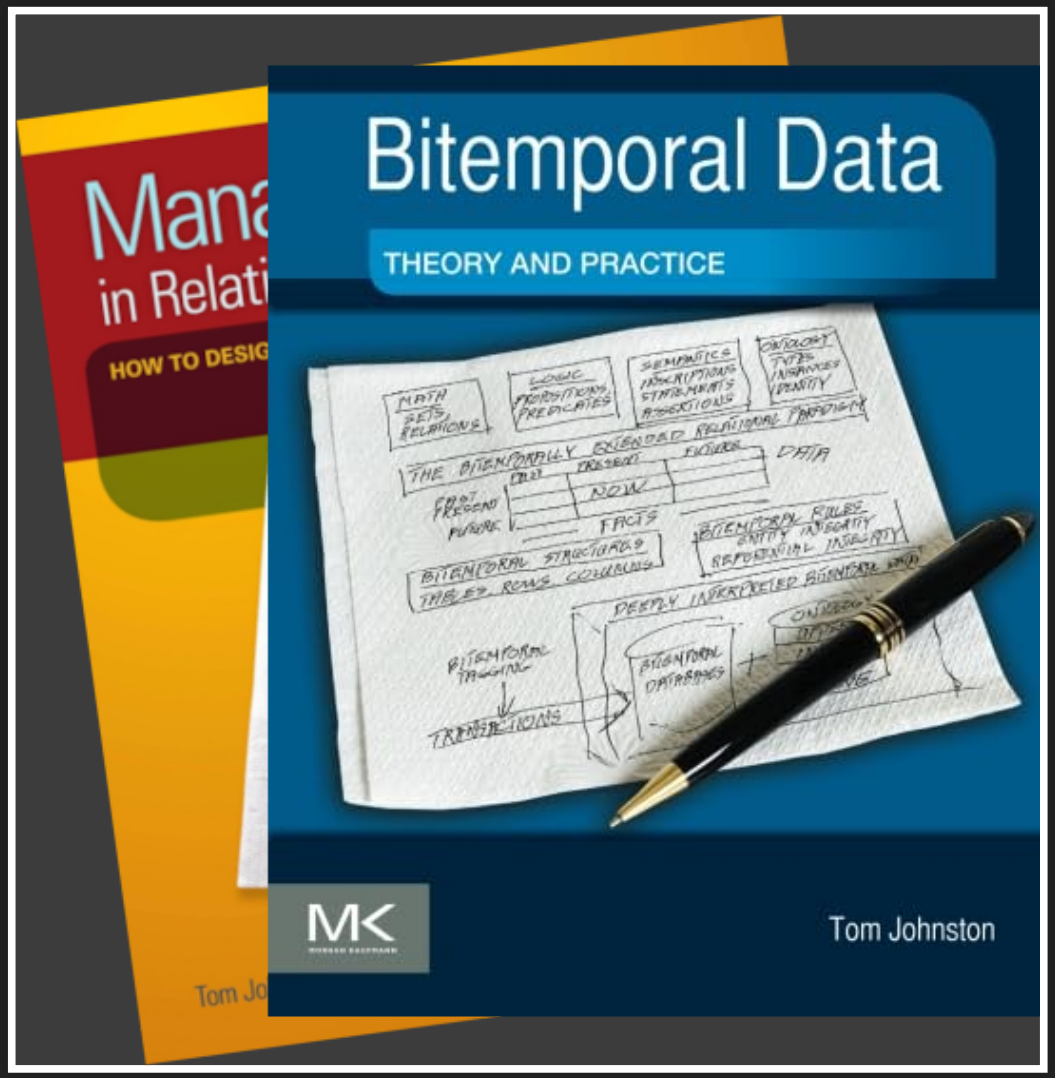
RESEARCH


Developing
Time-Oriented
Database
Applications
in SQL
RICHARD T. SNODGRASS

RESEARCH


TIME AND
RELATIONAL
THEORY
Temporal Databases
in the Relational Model
and SQL
C.J. Date
Hugh Darwen
Nikos A. Lorentzos
MK

RESEARCH



TEMPORAL IS DISTINCT FROM TIME-SERIES

time-series	temporal
single timestamp	two timestamps
records events	records things, "versions"
IoT sensors, finance	auditing, history
challenge is scale	challenge is complexity
partitioning	ranges, exclusion constraints
TimescaleDB	`periods`, `pg_bitemporal`

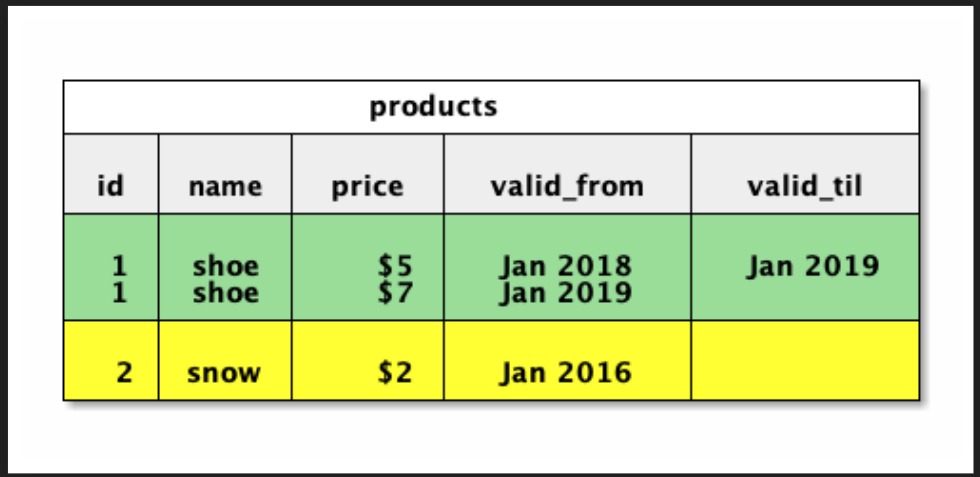
TWO DIMENSIONS

Application Time	System Time
history of the thing	history of the database
application features	auditing, compliance
user can edit	immutable
maintained by you	maintained by Postgres
constraints matter	look Ma, no hands!
`periods`, `pg_bitemporal`	`temporal_tables`, `pg_audit_log`
nothing	Rails: `papertrail`, `audited`, `chronomodel`, ...

TERMINOLOGY

Snodgrass	valid time	transaction time
Fowler	actual time	record time
Date/Darwen/Lorentzos	stated time	logged time
Johnston	effective time/ state time	assertion time
SQL:2011	application time	system time

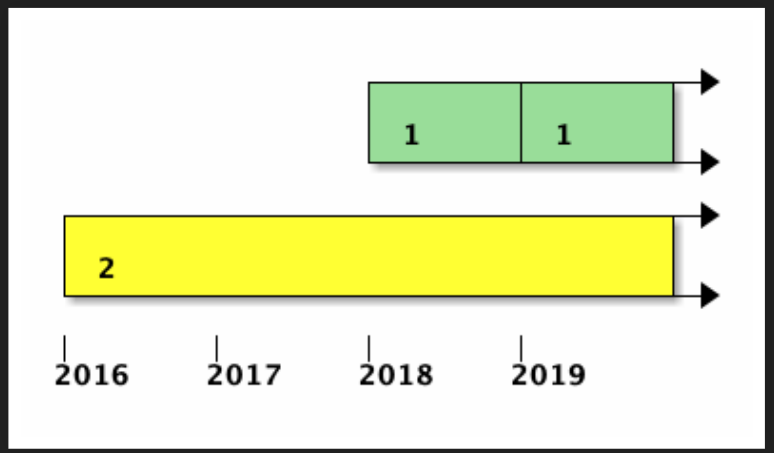
TEMPORAL EXAMPLE



products				
id	name	price	valid_from	valid_til
1 1	shoe shoe	$5 $7	Jan 2018 Jan 2019	Jan 2019
2	snow	$2	Jan 2016	

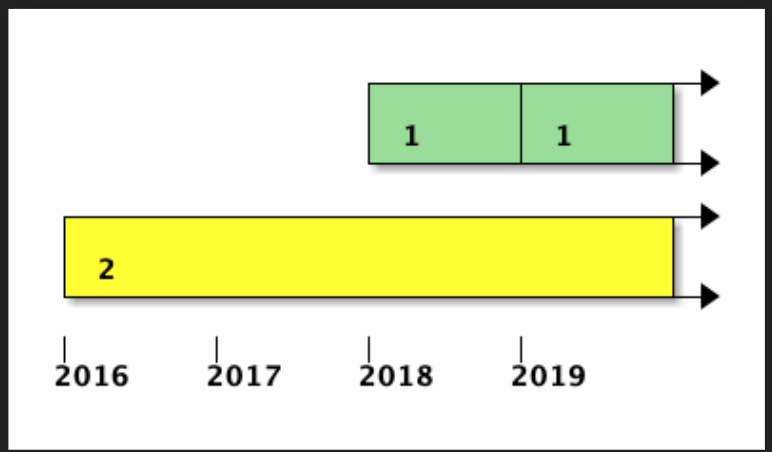
TEMPORAL EXAMPLE

products				
id	name	price	valid_from	valid_til
1 1	shoe shoe	$5 $7	Jan 2018 Jan 2019	Jan 2019
2	snow	$2	Jan 2016	



TEMPORAL EXAMPLE

products			
id	name	price	valid_at
1	shoe	$5	[Jan 2018, Jan 2019)
1	shoe	$7	[Jan 2019,)
2	snow	$2	[Jan 2016,)



RANGE OPERATORS

Operator	Description	Example	Result
=	equal	int4range(1,5) = '[1,4]'::int4range	t
<>	not equal	numrange(1.1,2.2) <> numrange(1.1,2.3)	t
<	less than	int4range(1,10) < int4range(2,3)	t
>	greater than	int4range(1,10) > int4range(1,5)	t
<=	less than or equal	numrange(1.1,2.2) <= numrange(1.1,2.2)	t
>=	greater than or equal	numrange(1.1,2.2) >= numrange(1.1,2.0)	t

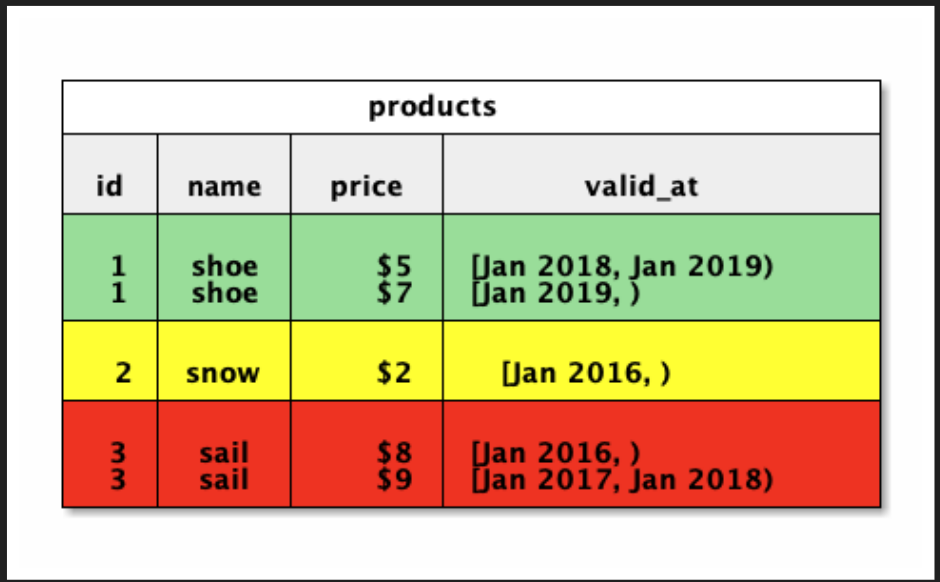
MORE OPERATORS

@>	contains range	int4range(2,4) @> int4range(2,3)	t
@>	contains element	'[2011-01-01,2011-03-01)'::tsrange @> '2011-01-10'::timestamp	t
<@	range is contained by	int4range(2,4) <@ int4range(1,7)	t
<@	element is contained by	42 <@ int4range(1,7)	f
&&	overlap (have points in common)	int8range(3,7) && int8range(4,12)	t
<<	strictly left of	int8range(1,10) << int8range(100,110)	t
>>	strictly right of	int8range(50,60) >> int8range(20,30)	t

AND MORE

&<	does not extend to the right of	int8range(1,20) &< int8range(18,20)	t
&>	does not extend to the left of	int8range(7,20) &> int8range(5,10)	t
-|-	is adjacent to	numrange(1.1,2.2) -|- numrange(2.2,3.3)	t
+	union	numrange(5,15) + numrange(10,20)	[5,20)
*	intersection	int8range(5,15) * int8range(10,20)	[10,15)
-	difference	int8range(5,15) - int8range(10,20)	[5,10)

NON-UNIQUE PKS



products

id	name	price	valid_at
1	shoe	$5	[Jan 2018, Jan 2019)
1	shoe	$7	[Jan 2019,)
2	snow	$2	[Jan 2016,)
3	sail	$8	[Jan 2016,)
3	sail	$9	[Jan 2017, Jan 2018)

EXCLUSION CONSTRAINTS

```
ALTER TABLE products
ADD CONSTRAINT products_pk
EXCLUDE USING gist
(id WITH =, valid_at WITH &&);
```

TEMPORAL PKS

```
ALTER TABLE products
ADD CONSTRAINT products_pk
PRIMARY KEY (id, valid_at WITHOUT OVERLAPS);
```

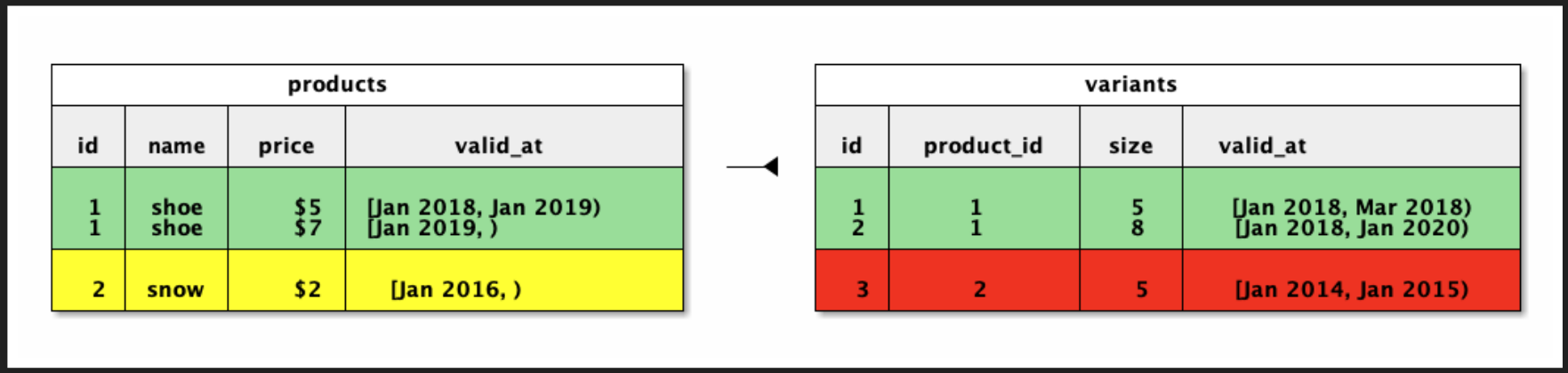
TEMPORAL PKS

```
ALTER TABLE products
ADD CONSTRAINT products_pk
PRIMARY KEY (id, valid_at WITHOUT OVERLAPS);
```

```
ALTER TABLE products
ADD CONSTRAINT products_uq
UNIQUE (id, valid_at WITHOUT OVERLAPS);
```

FOREIGN KEYS

TEMPORAL FOREIGN KEYS



TEMPORAL FOREIGN KEYS

```
CHECK (
  NOT EXISTS (
    SELECT 1
    FROM   variants AS v
    -- There was a p when v started:
    WHERE NOT EXISTS (
      SELECT 1
      FROM   products AS p
      WHERE  v.product_id = p.id
      AND    coalesce(lower(p.valid_at), '-infinity')
               <= coalesce(lower(v.valid_at), '-infinity')
      AND    coalesce(lower(v.valid_at), '-infinity')
               <= coalesce(upper(p.valid_at), 'infinity'))
    -- ...
```

TEMPORAL FOREIGN KEYS

```
-- ...
-- There was a p when v ended:
OR NOT EXISTS (
  SELECT 1
  FROM   products AS p
  WHERE  v.product_id = p.id
  AND    coalesce(lower(p.valid_at), '-infinity')
           < coalesce(upper(v.valid_at), 'infinity')
  AND    coalesce(upper(v.valid_at), 'infinity')
           <= coalesce(upper(p.valid_at), 'infinity'))
-- ...
```

TEMPORAL FOREIGN KEYS

```
-- ...
-- There are no gaps in p throughout v:
OR EXISTS (
  SELECT 1
  FROM   products AS p
  WHERE  v.product_id = p.id
  AND    coalesce(lower(v.valid_at), '-infinity')
           < coalesce(upper(p.valid_at), 'infinity')
  AND    coalesce(upper(p.valid_at), 'infinity')
           < coalesce(upper(v.valid_at), 'infinity')
-- ...
```

TEMPORAL FOREIGN KEYS

```
-- ...
  AND NOT EXISTS (
    SELECT 1
    FROM   products AS p2
    WHERE  p2.id = p.id
    AND    coalesce(lower(p2.valid_at), '-infinity')
             <= coalesce(upper(p.valid_at), 'infinity')
    AND    coalesce(upper(p.valid_at), 'infinity')
             < coalesce(upper(p2.valid_at), 'infinity'))))
```

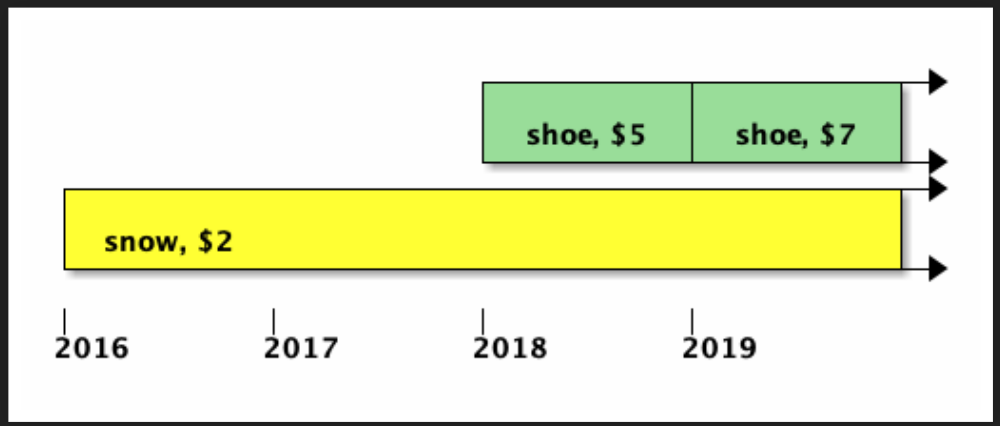
TEMPORAL FOREIGN KEYS

```
ALTER TABLE variants
ADD CONSTRAINT variants_product_id_fk
FOREIGN KEY (product_id, PERIOD valid_at)
REFERENCES (id, PERIOD valid_at);
```

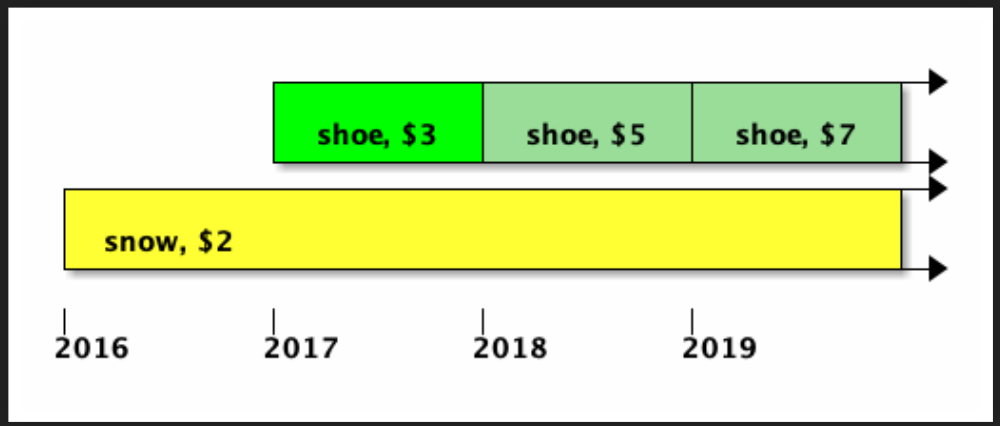
QUERIES

snapshot ("current")	at a given moment	returns a traditional table (removes `valid_at`)	`WHERE valid_at @> t`
sequenced	across time	returns a temporal table (preserves `valid_at`)	nothing, or `WHERE valid_at && r`
non-sequenced	time is just another column	returns ???	

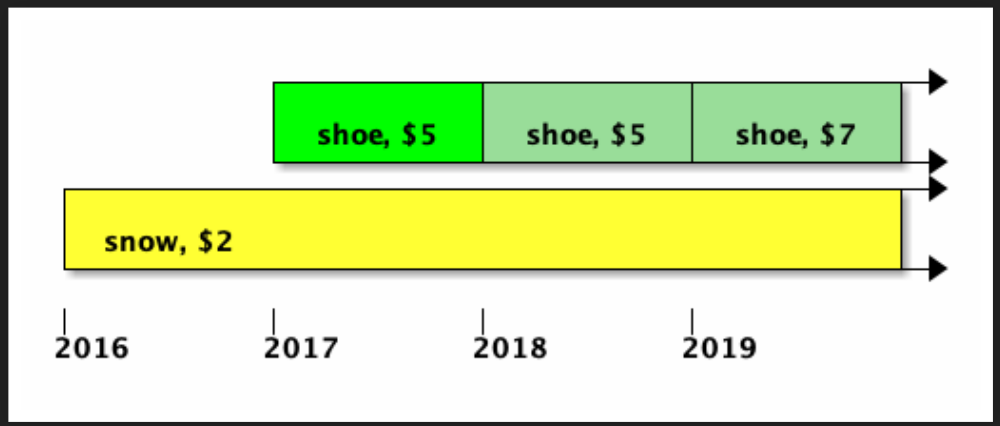
TEMPORAL INSERT



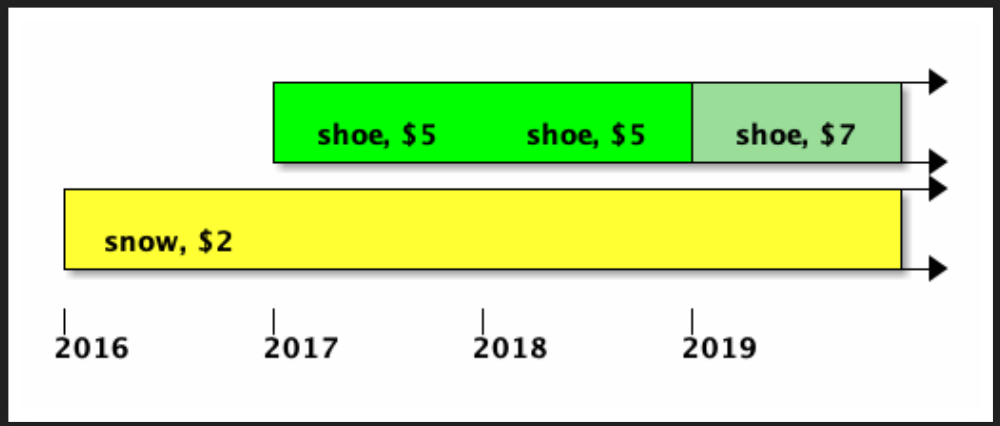
TEMPORAL INSERT



TEMPORAL INSERT

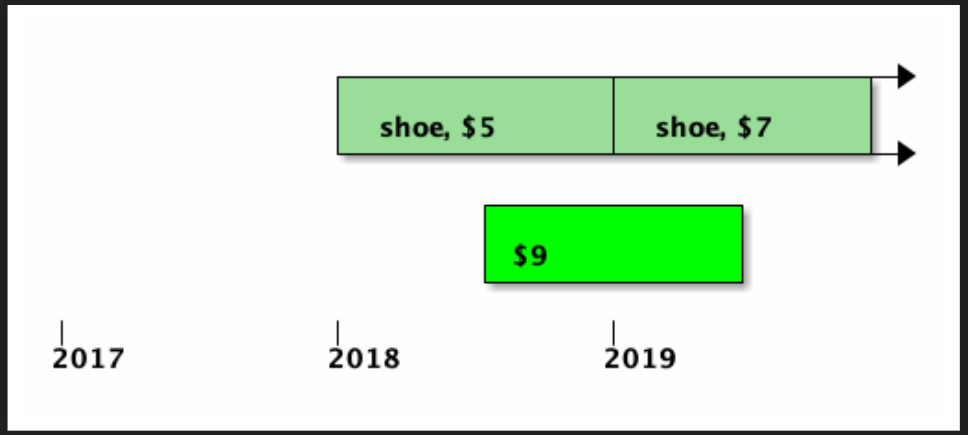


TEMPORAL INSERT

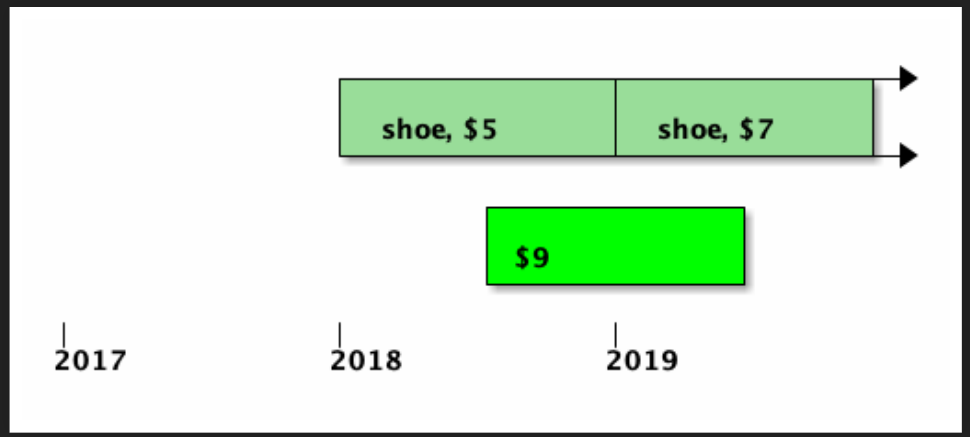


TEMPORAL UPDATE

TEMPORAL UPDATE

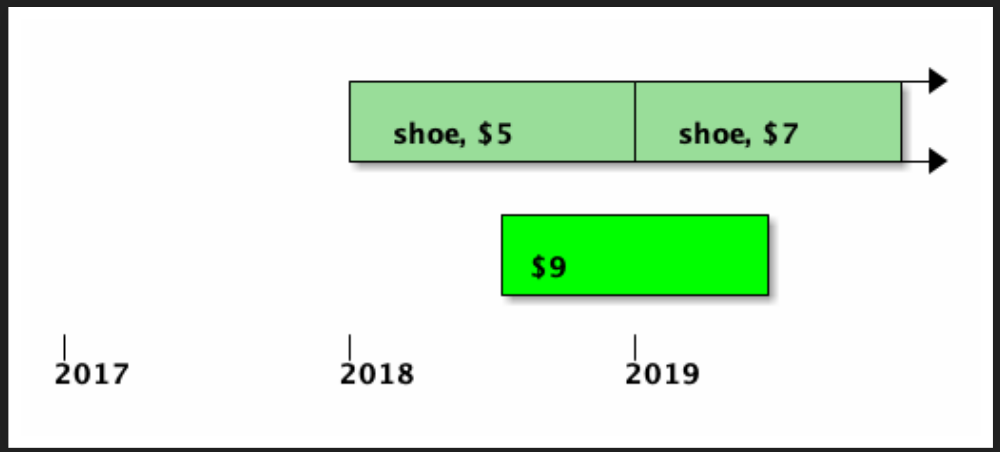


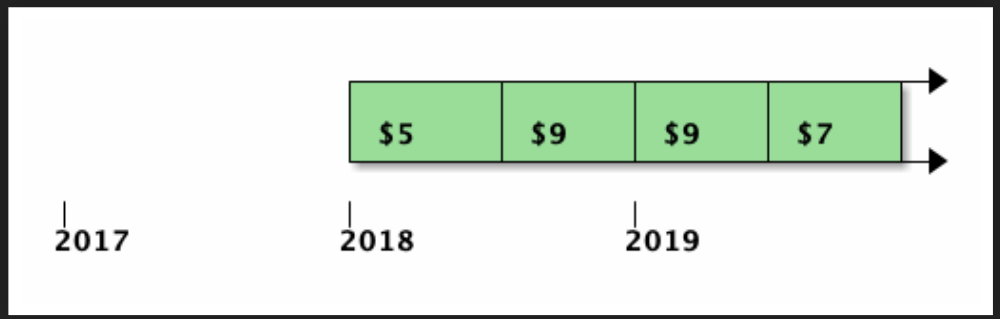
TEMPORAL UPDATE




```
UPDATE  products
FOR PORTION OF valid_at
        FROM '2018-07-01'
        TO '2019-07-01'
SET     price = 9
WHERE   id = 'shoe';
```

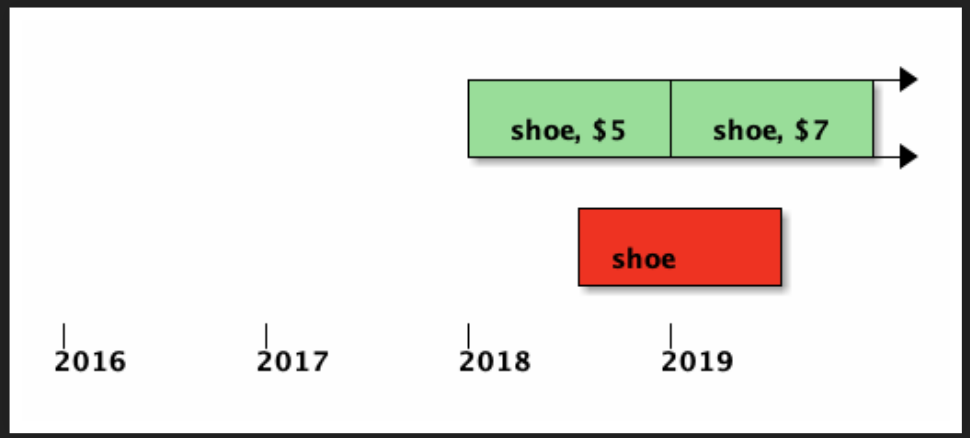
TEMPORAL UPDATE



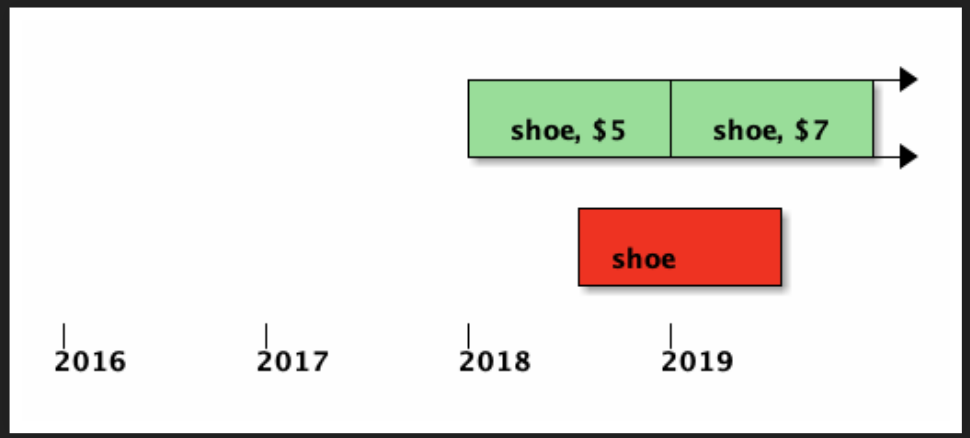


TEMPORAL DELETE

TEMPORAL DELETE

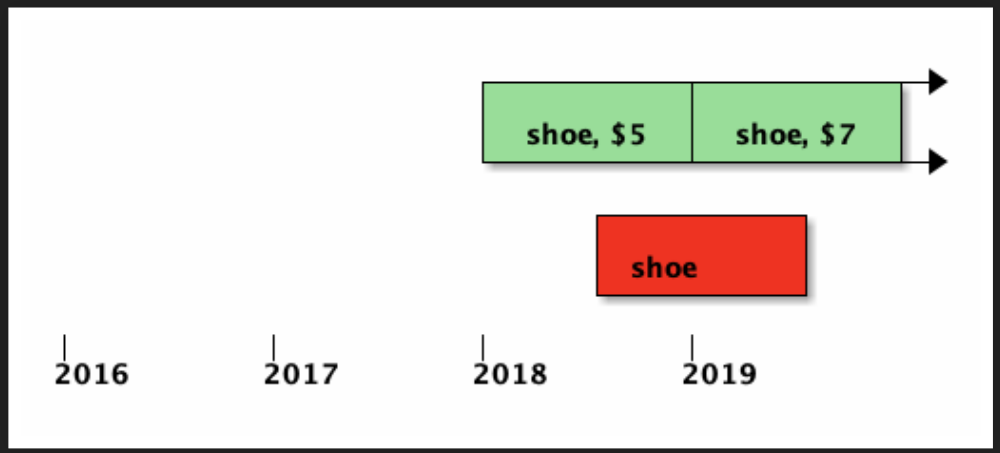


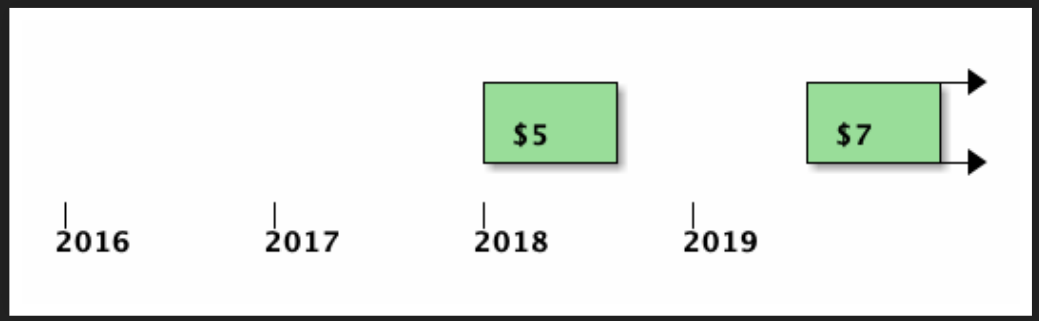
TEMPORAL DELETE



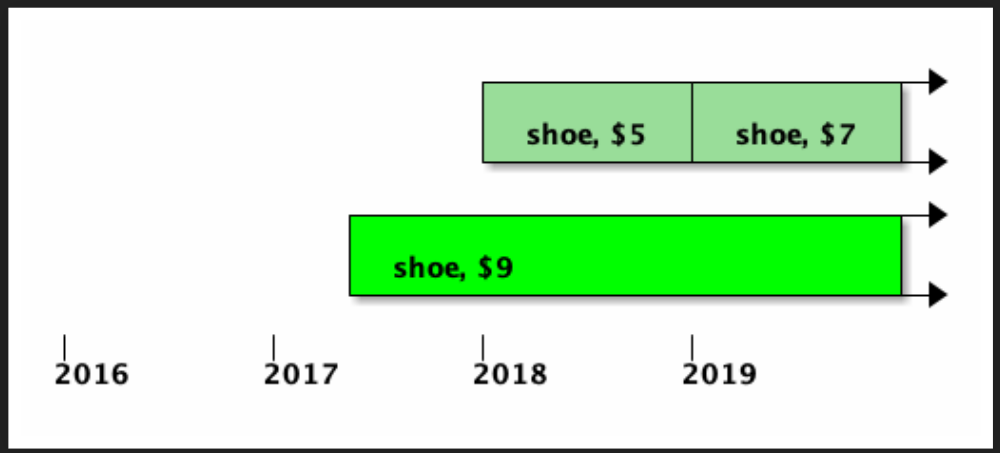

```
DELETE FROM products
FOR PORTION OF valid_at
        FROM '2018-07-01'
        TO '2019-07-01'
WHERE   id = 'shoe';
```

TEMPORAL DELETE

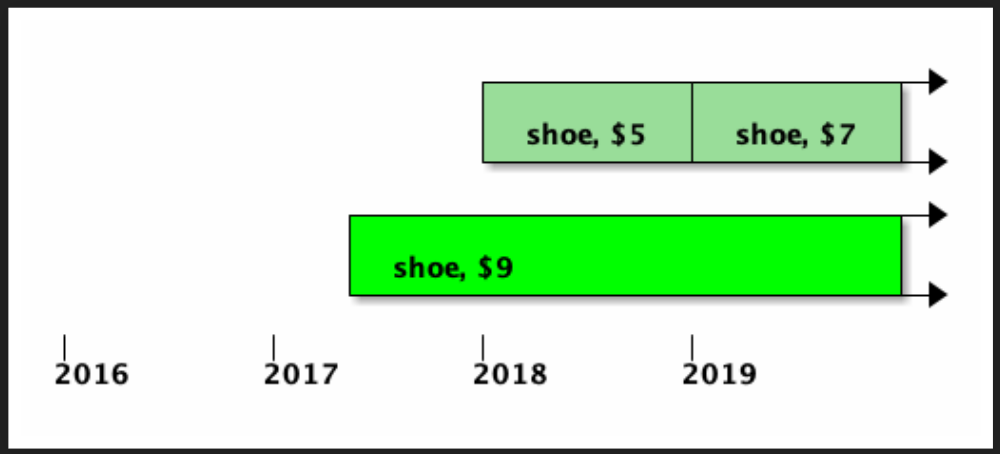


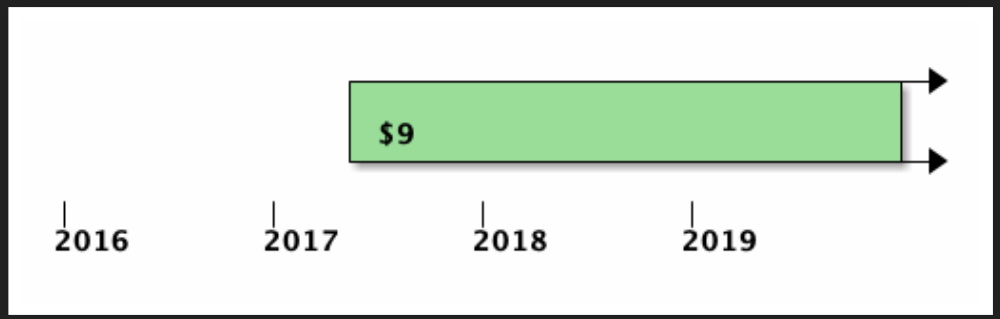


TEMPORAL UPSERT

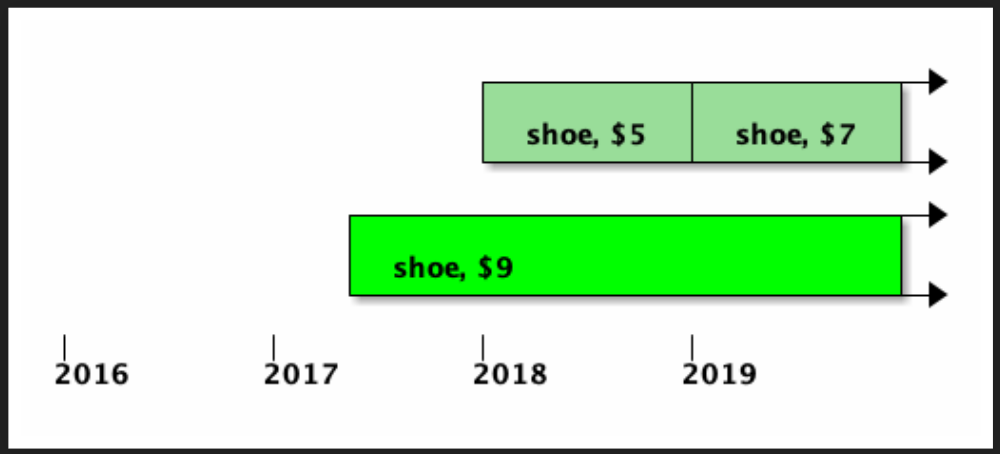


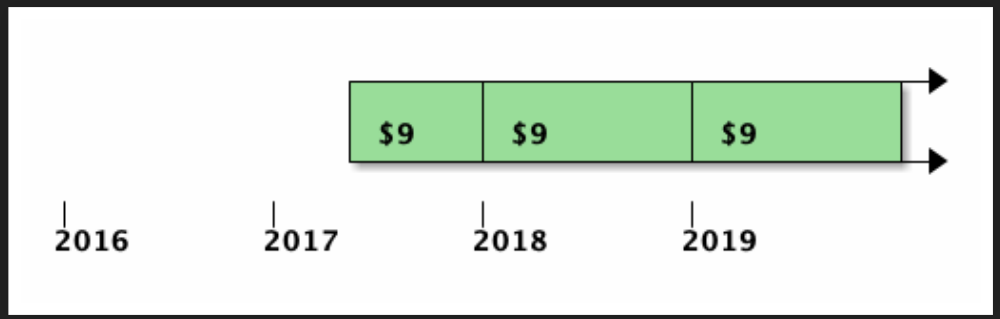
TEMPORAL UPSERT



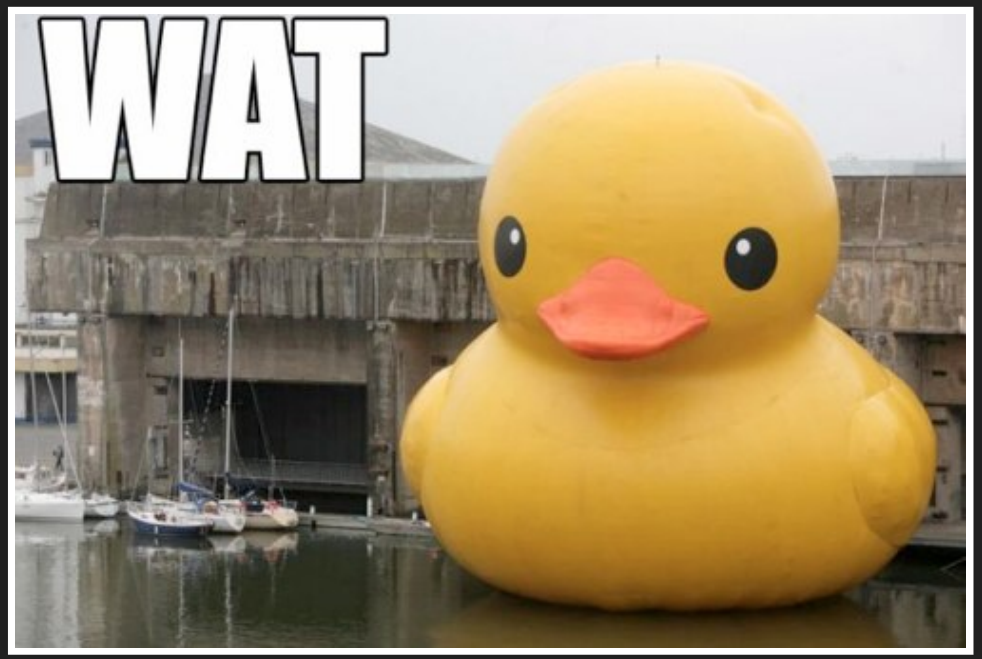


TEMPORAL UPSERT





SQL:2011



RANGE

```
CREATE TABLE products (
  id       integer,
  valid_at tstzrange,

  name     text,
  price    decimal(10,2),

  CONSTRAINT pk_products
    PRIMARY KEY
    (id, valid_at WITHOUT OVERLAPS)
);
```

RANGE

```
CREATE TABLE products (
  id       integer,
  valid_at tstzrange,

  name     text,
  price    decimal(10,2),

  CONSTRAINT pk_products
    PRIMARY KEY
    (id, valid_at WITHOUT OVERLAPS)
);
```

PERIOD

```
CREATE TABLE products (
  id          integer,
  valid_from timestamptz NOT NULL,
  valid_til  timestamptz NOT NULL,

  name        text,
  price       decimal(10,2),

  PERIOD FOR valid_at
    (valid_from, valid_til),
  CONSTRAINT pk_products
    PRIMARY KEY
    (id, valid_at WITHOUT OVERLAPS)
);
```

SYSTEM TIME

```
CREATE TABLE products (
  id        integer,
  sys_from timestamp GENERATED ALWAYS AS ROW START,
  sys_til  timestamp GENERATED ALWAYS AS ROW END,

  name        text,
  price       decimal(10,2),

  PERIOD FOR SYSTEM_TIME
    (sys_from, sys_til)
) WITH SYSTEM VERSIONING;
```

SYSTEM TIME

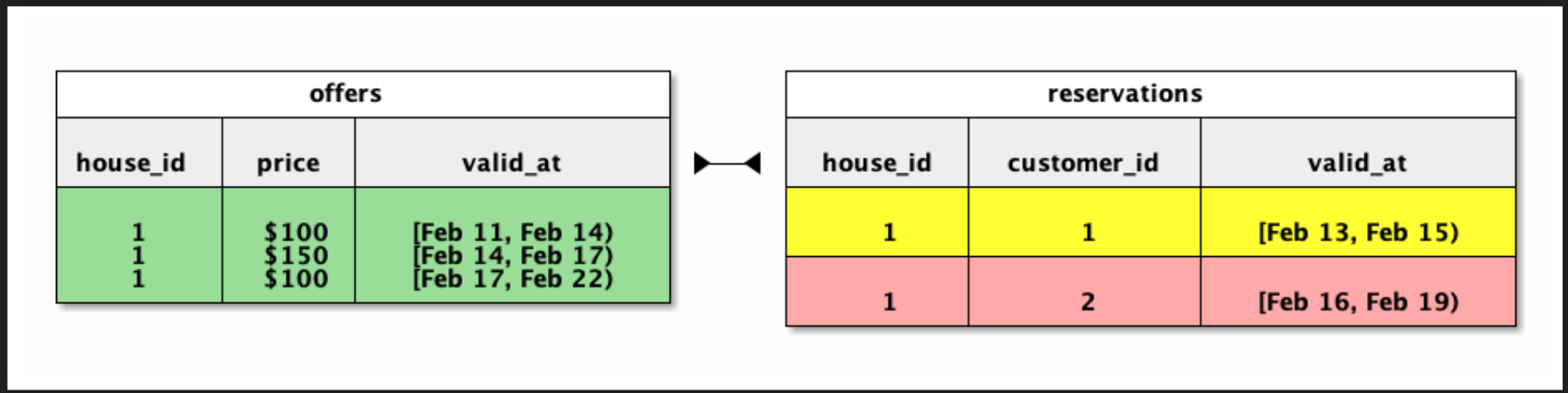
```
SELECT *
FROM   products
FOR SYSTEM_TIME AS OF t;

SELECT *
FROM   products
FOR SYSTEM_TIME FROM t1 TO t2;
```

INNER JOINS

```
SELECT  e.name, e.salary, p.name,
        e.valid_at * p.valid_at
FROM    employees AS e
JOIN    positions AS p
ON      p.employee_id = e.id
AND     p.valid_at && e.valid_at
```

OUTER JOINS


offers
house_id
price
valid_at
1
1
1
$100
$150
$100
[Feb 11, Feb 14)
[Feb 14, Feb 17)
[Feb 17, Feb 22)
reservations
house_id
customer_id
valid_at
1
1
[Feb 13, Feb 15)
1
2
[Feb 16, Feb 19)

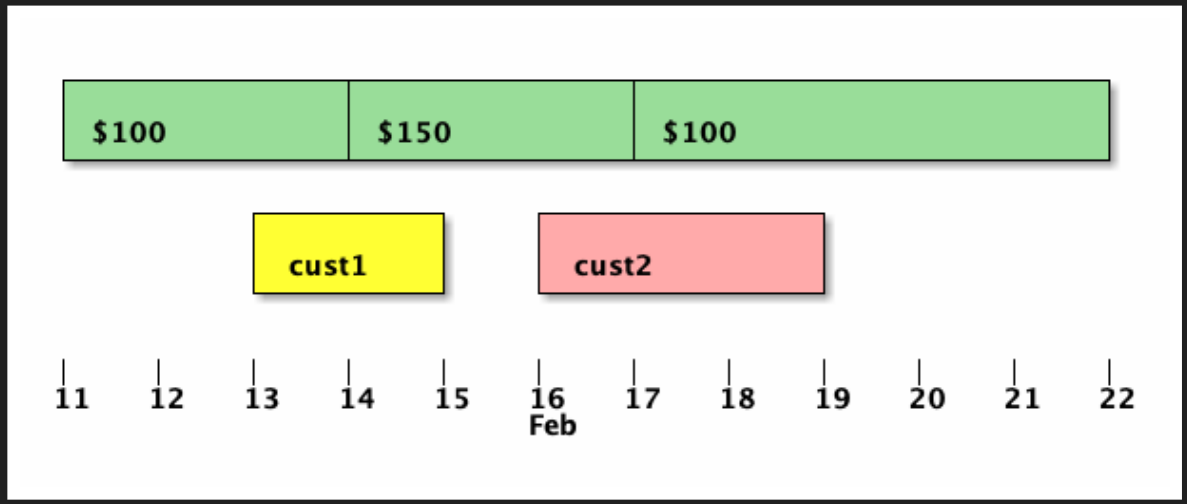
OUTER JOINS

offers		
house_id	price	valid_at
1 1 1	$100 $150 $100	[Feb 11, Feb 14) [Feb 14, Feb 17) [Feb 17, Feb 22)

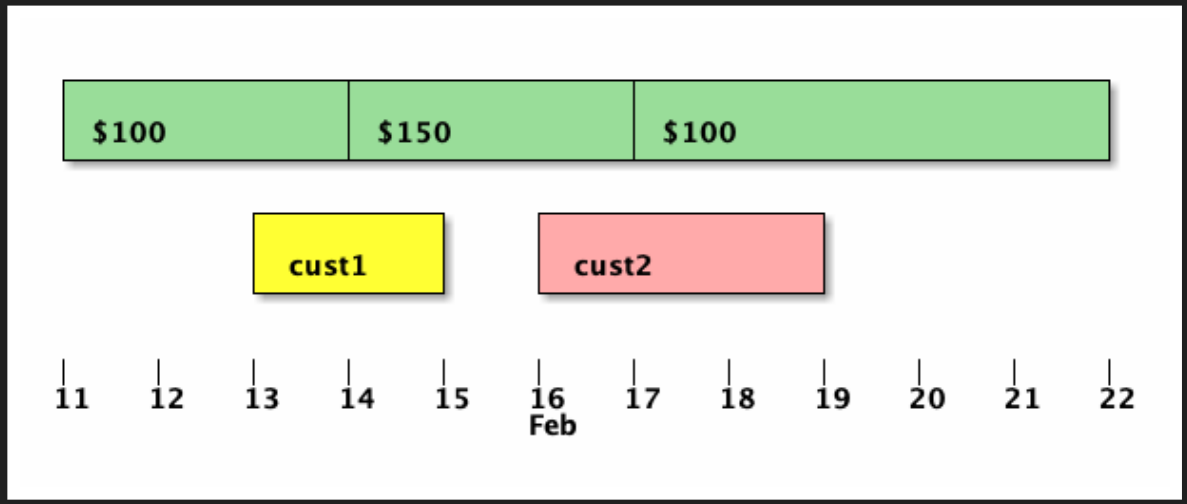
reservations		
house_id	customer_id	valid_at
1	1	[Feb 13, Feb 15)
1	2	[Feb 16, Feb 19)

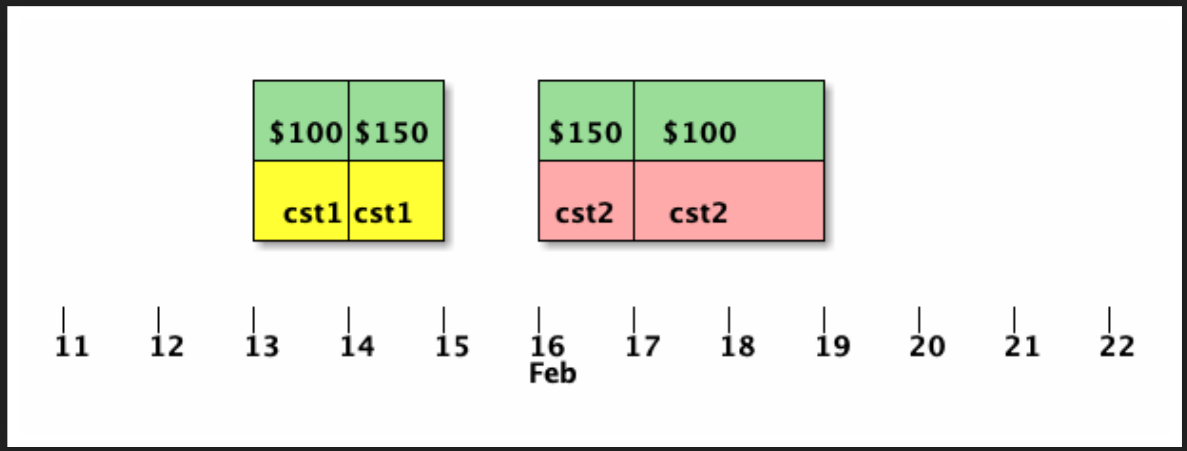
customer_id	price	valid_at
1 1	$100 $150	[Feb 13, Feb 14) [Feb 14, Feb 15)
2 2	$150 $100	[Feb 16, Feb 17) [Feb 17, Feb 19)

OUTER JOINS



OUTER JOINS


$100
$150
$100
cust1
cust2
11
12
13
14
15
16
Feb
17
18
19
20
21
22


$100
$150
cst1
cst1
$150
$100
cst2
cst2
11
12
13
14
15
16
Feb
17
18
19
20
21
22

SEMIJOINS

```
SELECT  a.id, a.valid_at
FROM    a
WHERE   EXISTS (
  SELECT  1
  FROM    b
  WHERE   a.id = b.id
  AND     a.valid_at && b.valid_at);
```

SEMIJOINS

```
SELECT  a.id, a.valid_at
FROM    a
WHERE   EXISTS (
  SELECT  1
  FROM    b
  WHERE   a.id = b.id
  AND     a.valid_at && b.valid_at);
```

SEMIJOINS

```
SELECT  a.id, j.id, j.valid_at
FROM    a
JOIN LATERAL (
  SELECT  b.id,
          UNNEST(
            multirange(a.valid_at) *
            range_agg(b.valid_at)
          ) AS valid_at
  FROM    b
  WHERE   a.id = b.id
  AND     a.valid_at && b.valid_at
  GROUP BY b.id
) AS j ON true;
```

SEMIJOINS

```
SELECT  a.id, j.id, j.valid_at
FROM    a
JOIN LATERAL (
  SELECT  b.id,
          UNNEST(
            multirange(a.valid_at) *
            range_agg(b.valid_at)
          ) AS valid_at
  FROM    b
  WHERE   a.id = b.id
  AND     a.valid_at && b.valid_at
  GROUP BY b.id
) AS j ON true;
```

SEMIJOINS

```
SELECT  a.id, j.id, j.valid_at
FROM    a
JOIN LATERAL (
  SELECT  b.id,
          UNNEST(
            multirange(a.valid_at) *
            range_agg(b.valid_at)
          ) AS valid_at
  FROM    b
  WHERE   a.id = b.id
  AND     a.valid_at && b.valid_at
  GROUP BY b.id
) AS j ON true;
```

ANTIJOINS

```
SELECT  a.id, j.id,
        UNNEST(
          CASE WHEN j.valid_at IS NULL
               THEN multirange(a.valid_at)
               ELSE multirange(a.valid_at) - j.valid_at END
        ) AS valid_at
FROM    a
LEFT JOIN LATERAL (
  SELECT  b.id, range_agg(b.valid_at) AS valid_at
  FROM    b
  WHERE   a.id = b.id
  AND     a.valid_at && b.valid_at
  GROUP BY b.id
) AS j ON true
WHERE   NOT isempty(a.valid_at);
```

ANTIJOINS

```
SELECT  a.id, j.id,
        UNNEST(
          CASE WHEN j.valid_at IS NULL
               THEN multirange(a.valid_at)
               ELSE multirange(a.valid_at) - j.valid_at END
        ) AS valid_at
FROM    a
LEFT JOIN LATERAL (
  SELECT  b.id, range_agg(b.valid_at) AS valid_at
  FROM    b
  WHERE   a.id = b.id
  AND     a.valid_at && b.valid_at
  GROUP BY b.id
) AS j ON true
WHERE   NOT isempty(a.valid_at);
```

ANTIJOINS

```
SELECT  a.id, j.id,
        UNNEST(
          CASE WHEN j.valid_at IS NULL
               THEN multirange(a.valid_at)
               ELSE multirange(a.valid_at) - j.valid_at END
        ) AS valid_at
FROM    a
LEFT JOIN LATERAL (
  SELECT  b.id, range_agg(b.valid_at) AS valid_at
  FROM    b
  WHERE   a.id = b.id
  AND     a.valid_at && b.valid_at
  GROUP BY b.id
) AS j ON true
WHERE   NOT isempty(a.valid_at);
```

ANTIJOINS

```
SELECT  a.id, j.id,
        UNNEST(
          CASE WHEN j.valid_at IS NULL
               THEN multirange(a.valid_at)
               ELSE multirange(a.valid_at) - j.valid_at END
        ) AS valid_at
FROM    a
LEFT JOIN LATERAL (
  SELECT  b.id, range_agg(b.valid_at) AS valid_at
  FROM    b
  WHERE   a.id = b.id
  AND     a.valid_at && b.valid_at
  GROUP BY b.id
) AS j ON true
WHERE   NOT isempty(a.valid_at);
```

ANTIJOINS

```
SELECT  a.id, j.id,
        UNNEST(
          CASE WHEN j.valid_at IS NULL
               THEN multirange(a.valid_at)
               ELSE multirange(a.valid_at) - j.valid_at END
        ) AS valid_at
FROM    a
LEFT JOIN LATERAL (
  SELECT  b.id, range_agg(b.valid_at) AS valid_at
  FROM    b
  WHERE   a.id = b.id
  AND     a.valid_at && b.valid_at
  GROUP BY b.id
) AS j ON true
WHERE   NOT isempty(a.valid_at);
```

AGGREGATES

TODO

UNION, INTERSECT, EXCEPT

TODO

POSTGRES V17 REVERT

- Primary keys
- Foreign keys

POSTGRES V17 REVERT

```
EXCLUDE (id WITH =, valid_at WITH &&)
...
INSERT INTO products (id, valid_at) VALUES (5, 'empty');
INSERT INTO products (id, valid_at) VALUES (5, 'empty');
```

MORE

- History UX
- CRUD: REST, GraphQL
- ORM

THANKS!

ME

- https://github.com/pjungwir/temporal-databases-postgres-talk
- https://illuminatedcomputing.com/posts/2017/12/temporal-databases-bibliography/

RESEARCH

- https://www2.cs.arizona.edu/~rts/publications.html
- http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=F78723B857463955C76E540DCAB8FDF5?doi=10.1.1.116.7598&rep=rep1&type=pdf
- https://files.ifi.uzh.ch/boehlen/Papers/modf174-dignoes.pdf
- http://www.zora.uzh.ch/id/eprint/130374/1/Extending_the_kernel.pdf
- https://github.com/pjungwir/temporal_ops

SQL:2011

- https://www.wiscorp.com/SQLStandards.html
- https://sigmodrecord.org/publications/sigmodRecord/1209/pdfs/07.industry.kulkarni.pdf

OTHER VENDORS

- https://illuminatedcomputing.com/posts/2019/08/sql2011-survey/
- https://mariadb.com/kb/en/library/system-versioned-tables/
- https://docs.oracle.com/database/121/ADFNS/adfns_flashback.htm#ADFNS610
- https://docs.oracle.com/database/121/ADFNS/adfns_design.htm#ADFNS967
- https://docs.microsoft.com/en-us/sql/relational-databases/tables/temporal-tables?view=sql-server-2017
- https://www.ibm.com/support/knowledgecenter/en/SSEPGG_10.1.0/com.ibm.db2.luw.admin.dbobj.doc/doc/t0058926.html

PATCHES

- https://commitfest.postgresql.org/48/4308/
- https://www.postgresql-archive.org/PROPOSAL-Temporal-query-processing-with-range-types-tt5913058.html
- https://www.postgresql-archive.org/SQL-2011-PERIODS-vs-Postgres-Ranges-tt6055264.html

TOOLS

- https://github.com/xocolatl/periods
- https://github.com/hettie-d/pg_bitemporal
- https://github.com/arkhipov/temporal_tables
- https://www.youtube.com/watch?v=TRgni5q0YM8
- https://github.com/ifad/chronomodel

ARISTOTLE

- Photo from University of Glasgow Library, https://www.flickr.com/photos/uofglibrary/18242587063/in/photostream/

THANKS!

https://github.com/pjungwir/temporal-databses-theory-and-postgres-2024