TEMPORAL DATABASES:

THEORY AND POSTGRES

Paul A. Jungwirth

31 May 2019

PGCon 2019

TEMPORAL IS DISTINCT FROM TIME-SERIES

time-series	temporal
single timestamp	two timestamps
records events	records things
IoT sensors, finance	auditing, history
challenge is scale	challenge is complexity
Partitioning	ranges, exclusion constraints
Citus, TimescaleDB	Teradata, `temporal_tables`

TEMPORAL = HISTORICAL

- e-commerce: product price

TEMPORAL = HISTORICAL

- e-commerce: product price
- real estate: house renovations

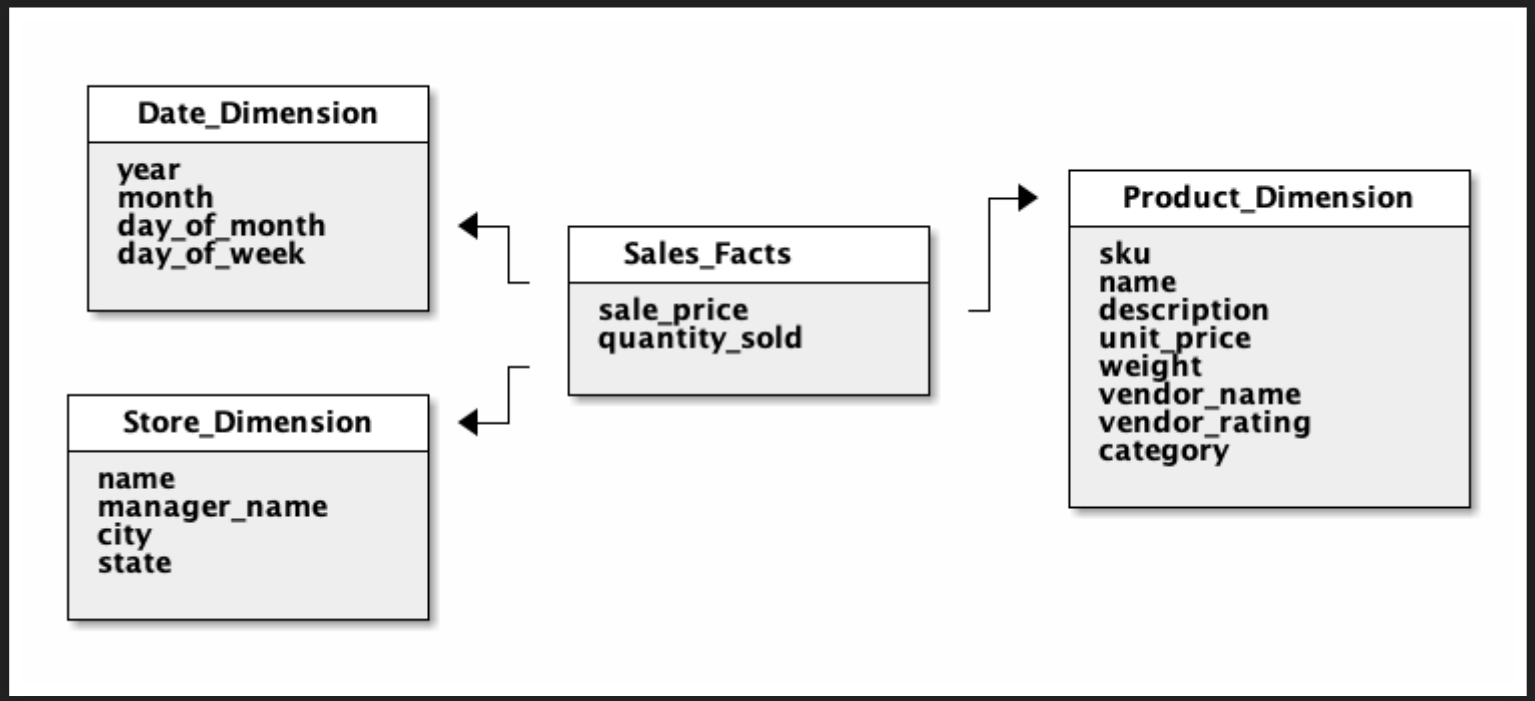
TEMPORAL = HISTORICAL

- e-commerce: product price
- real estate: house renovations
- employees: position, salary, employment period

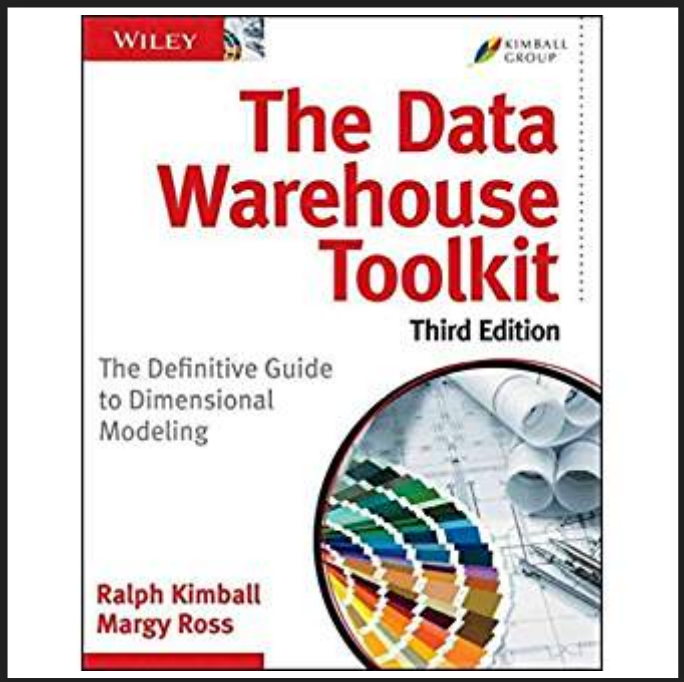
TEMPORAL = HISTORICAL

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

OLAP PROBLEMS TOO



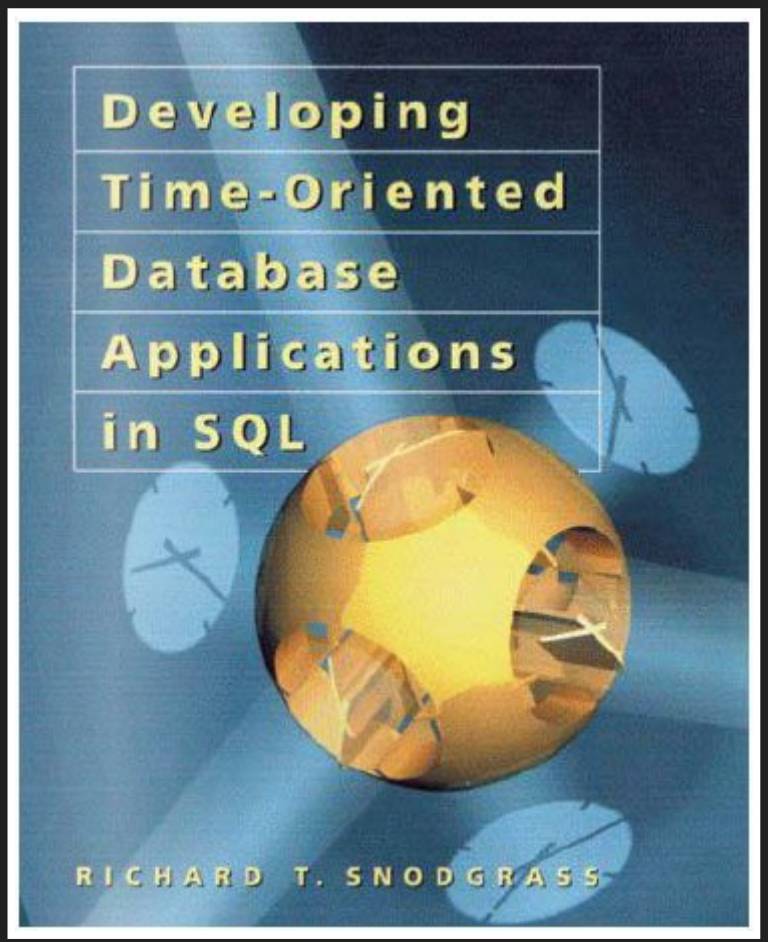
OLAP PROBLEMS TOO



"SLOWLY-CHANGING DIMENSIONS"

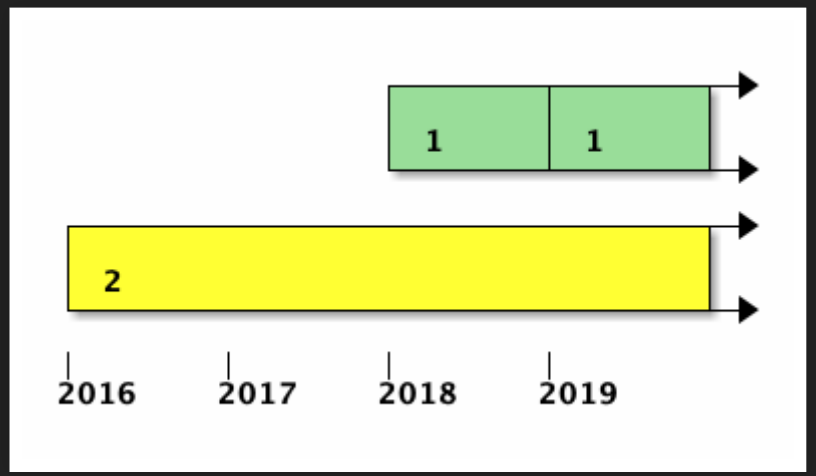
- Type I: Overwrite it
- Type II: Add a Row
- Type III: Add a Column

RESEARCH


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

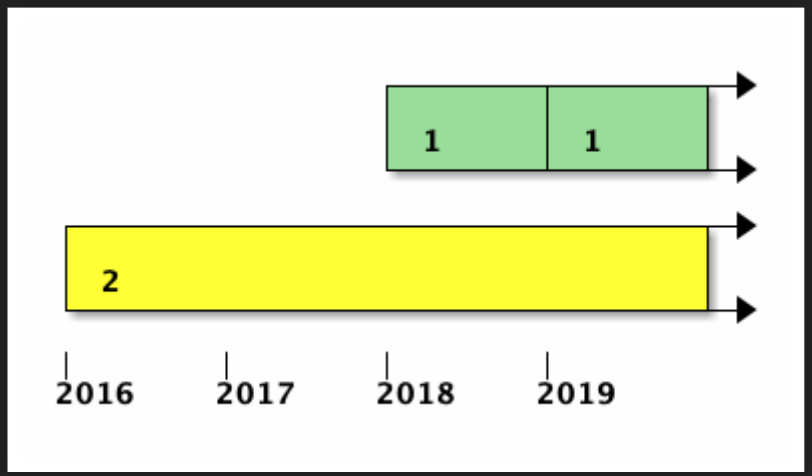
TEMPORAL EXAMPLE

products				
id	name	price	valid_from	valid_til
1	shoe	$5	Jan 2018	Jan 2019
1	shoe	$7	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)

LOTS EASIER

```
WHERE employed_during @@ [2018-01-01,2019-01-01)
```

VS

```
WHERE employed_from < '2019-01-01'
AND   '2018-01-01'  < employed_til
```

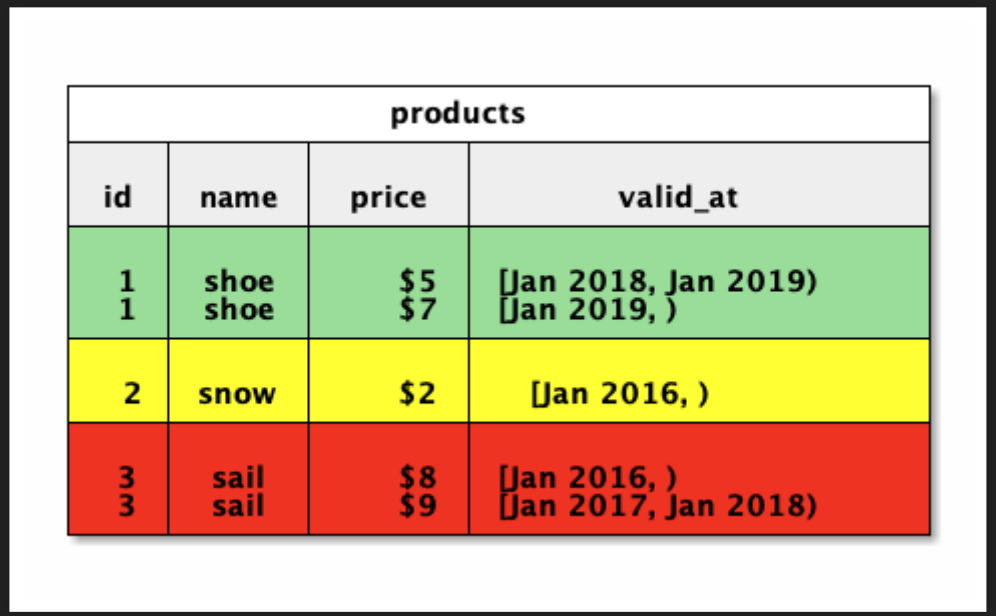
TWO DIMENSIONS

Valid Time	Transaction Time
history of the thing	history of the database
application features	auditing, compliance
user can edit	immutable
maintained by your app	maintained by triggers
constraints matter	look Ma, no hands!
nothing	pg: `temporal_tables`, "A Tardis for Your ORM", `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

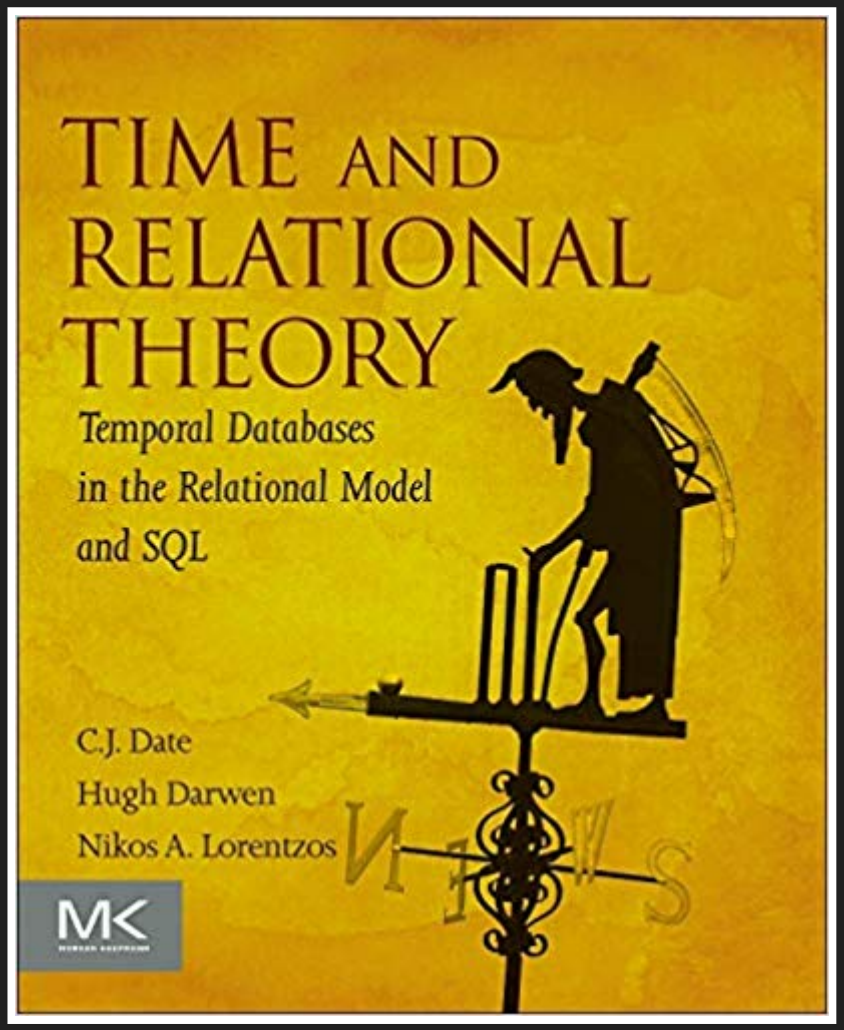
NON-UNIQUE PKS



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)

ANOTHER BOOK



EXCLUSION CONSTRAINTS

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

EXCLUSION CONSTRAINTS

```
ALTER TABLE products
ADD CONSTRAINT pk_products
EXCLUDE
(id WITH =);
```

EXCLUSION CONSTRAINTS

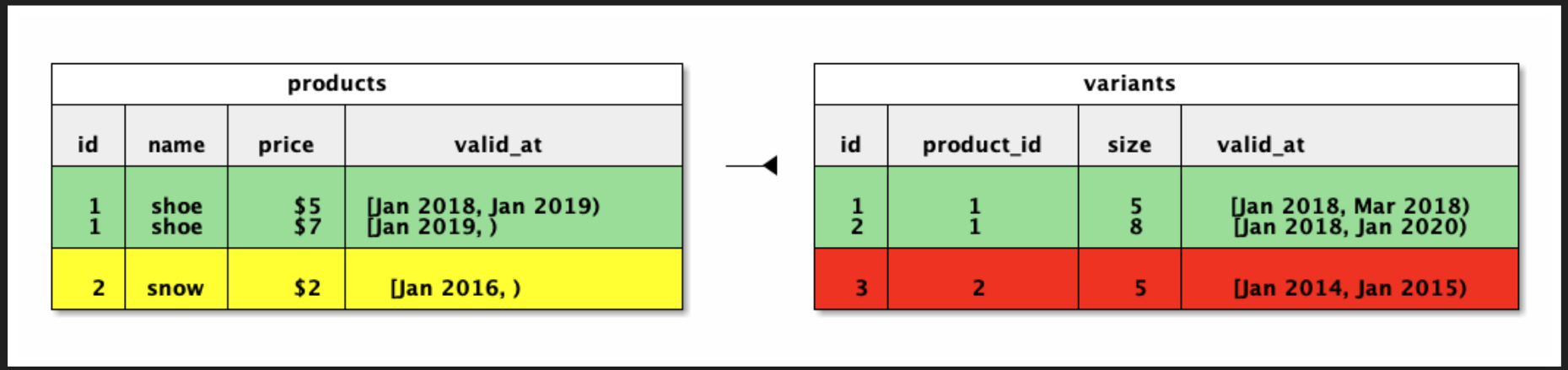
```
ALTER TABLE products
ADD CONSTRAINT pk_products
EXCLUDE USING gist
(id WITH =, valid_at WITH &&);
```

EXCLUSION CONSTRAINTS

```
CHECK (
  NOT EXISTS (
    SELECT 1
    FROM   products t1
    WHERE  1 < (
      SELECT COUNT(id)
      FROM   products t2
      WHERE  t1.id = t2.id
      AND    t1.valid_at && t2.valid_at))
  AND NOT EXISTS (
    SELECT 1
    FROM   products t3
    WHERE  p3.id IS NULL)
)
```

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')
-- ...
```

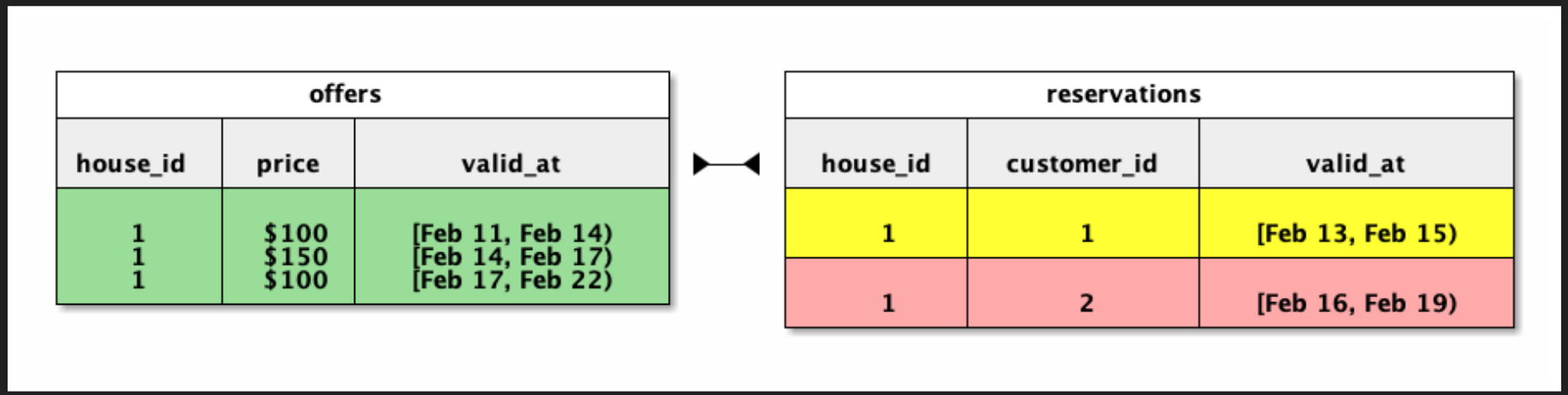
TEMPORAL FOREIGN KEYS

```
-- ...
  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'))))
```

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 ???	

JOINS



offers

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

reservations

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

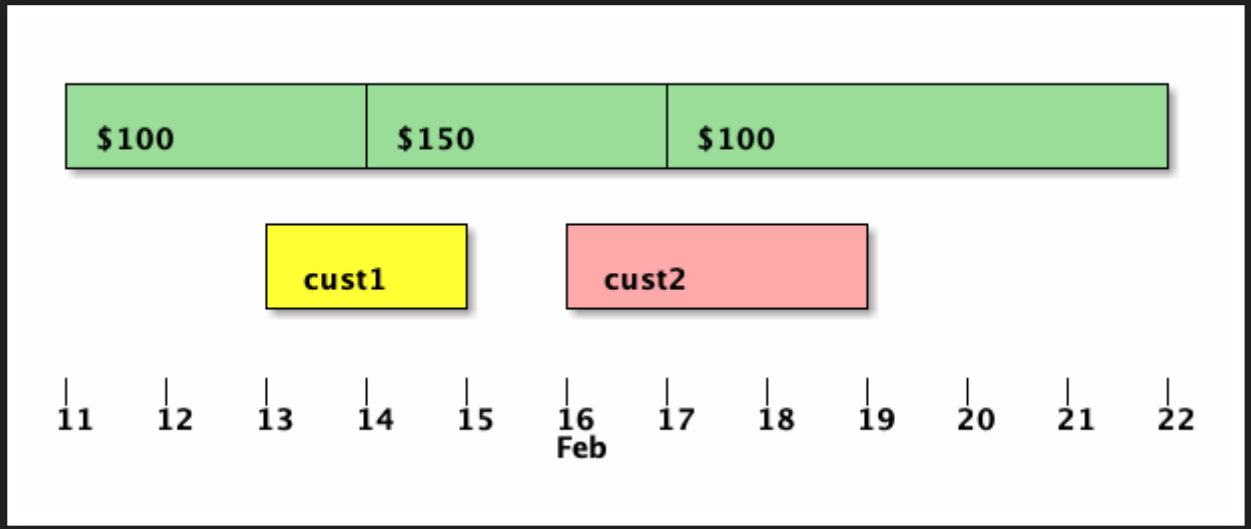
JOINS

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

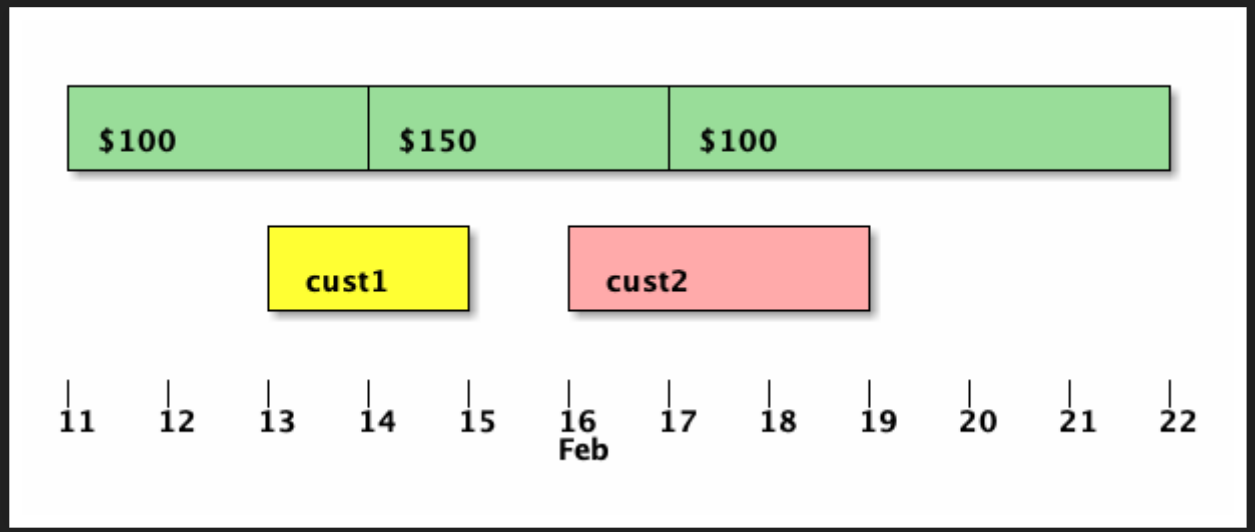
reservations		
house_id	customer_id	valid_at
1	1	[Feb 13, Feb 15)
1	2	[Feb 16, Feb 19)

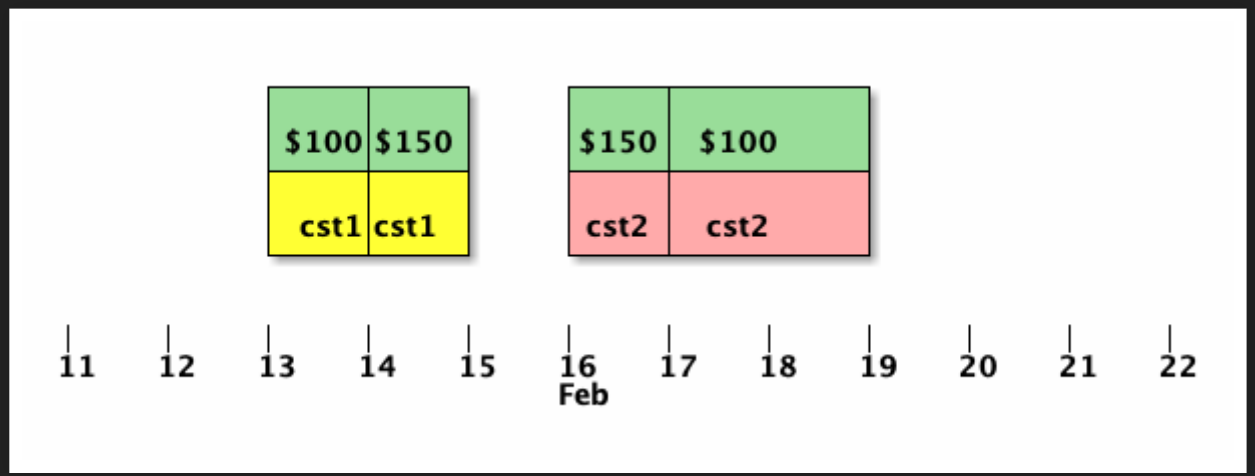
customer_id	price	valid_at
1 1	$100 $150	[Feb 13, Feb 14) [Feb 14, Feb 15)
2 2	$150 $100	[Feb 16, Feb 17) [Feb 17, Feb 19)

JOINS

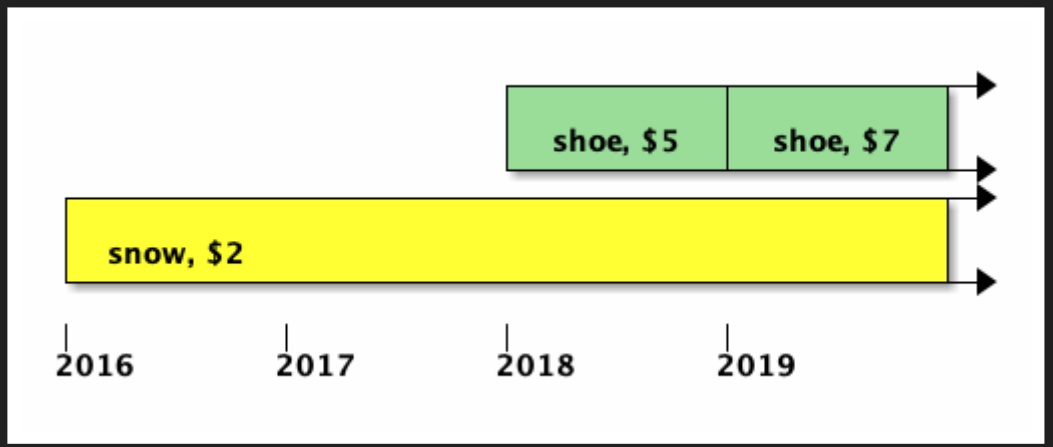


JOINS

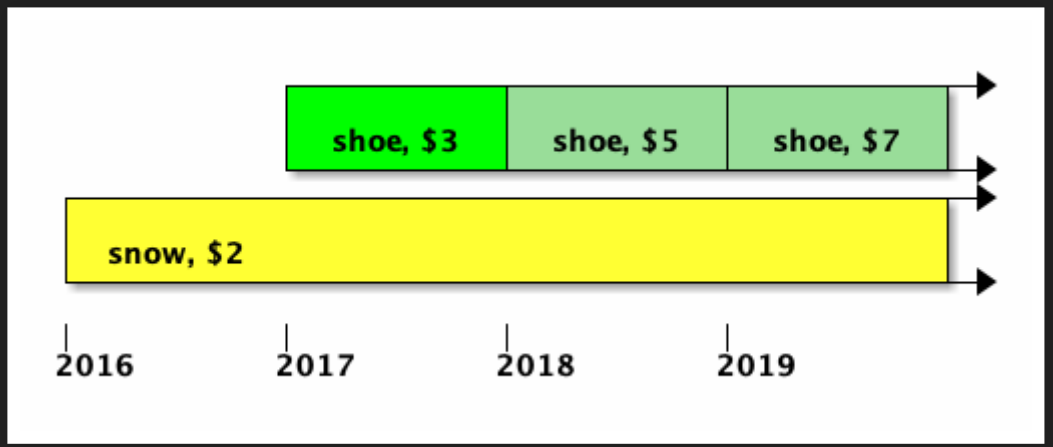

$100
$150
$100
cust1
cust2
11
12
13
14
15
16
17
18
19
20
21
22
Feb


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

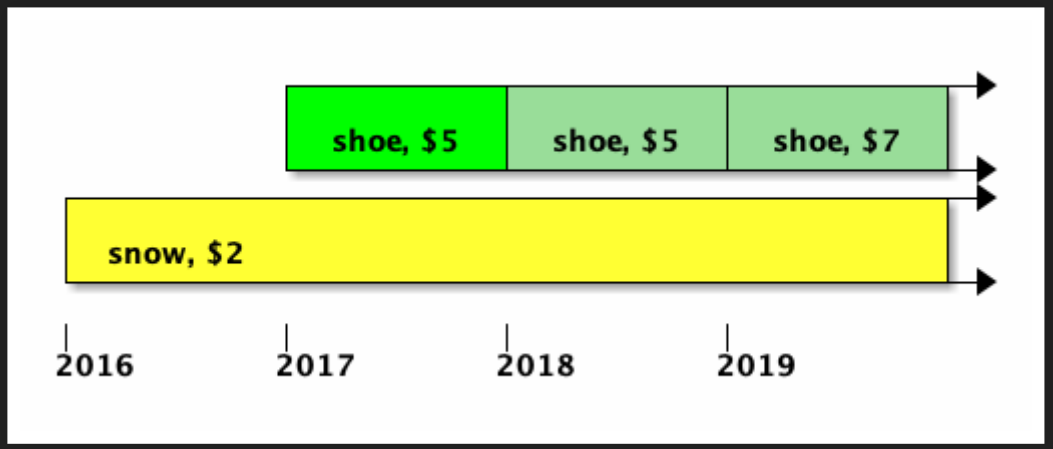
TEMPORAL INSERT



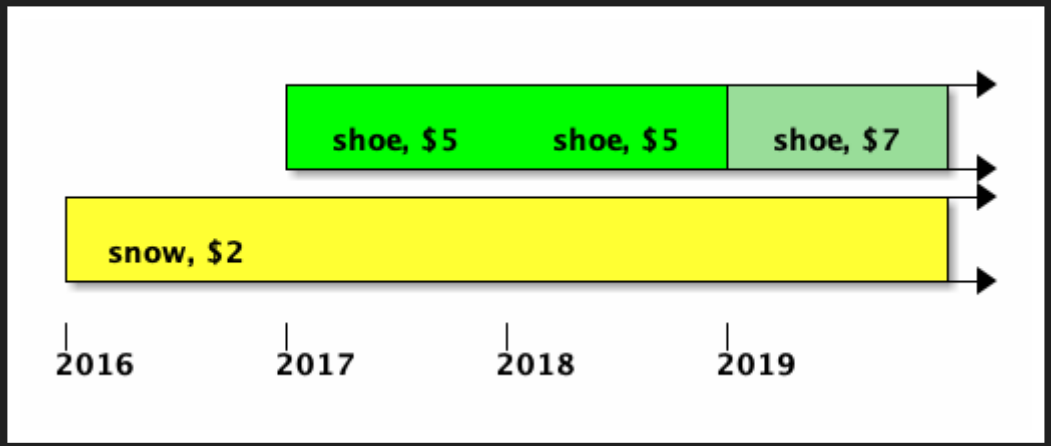
TEMPORAL INSERT



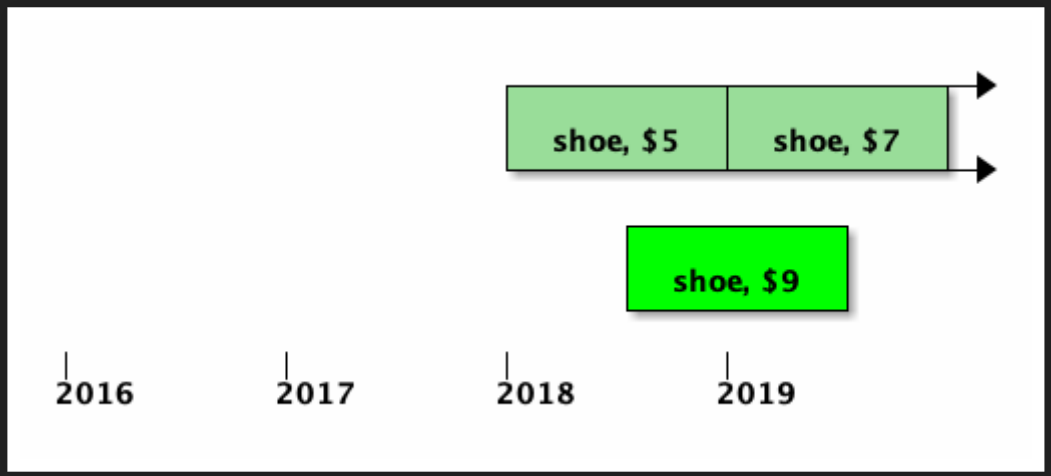
TEMPORAL INSERT



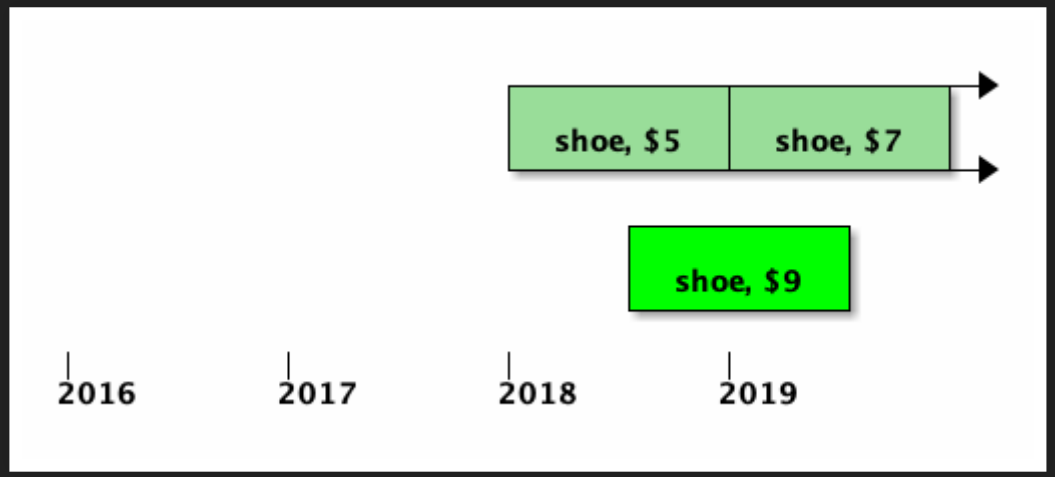
TEMPORAL INSERT

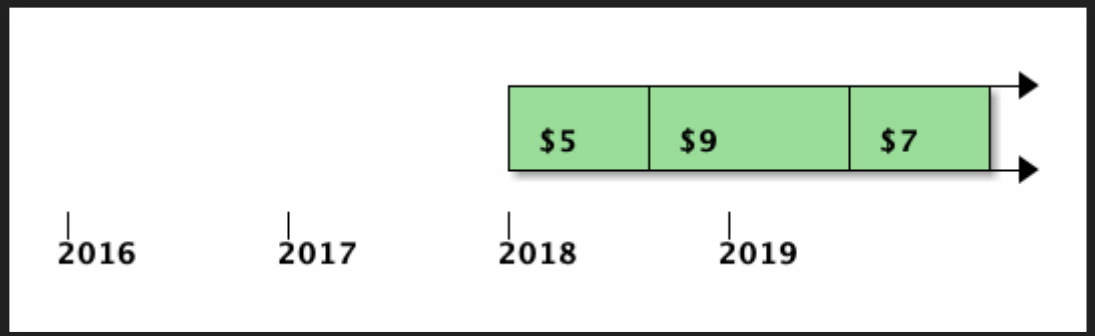


TEMPORAL UPDATE

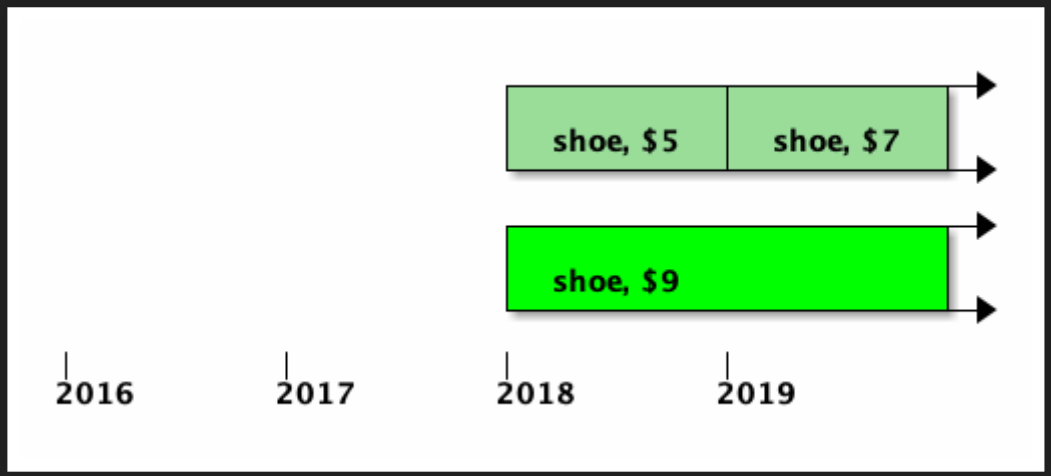


TEMPORAL UPDATE

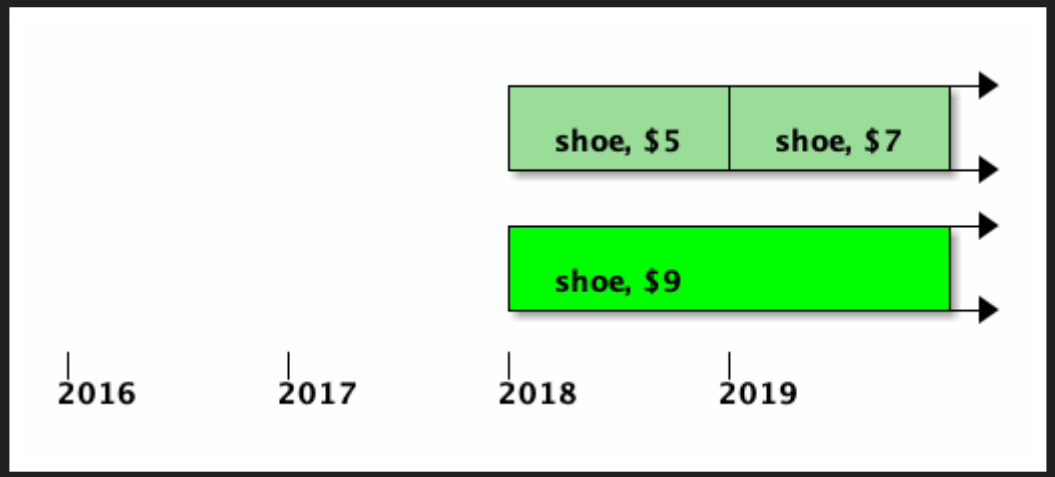


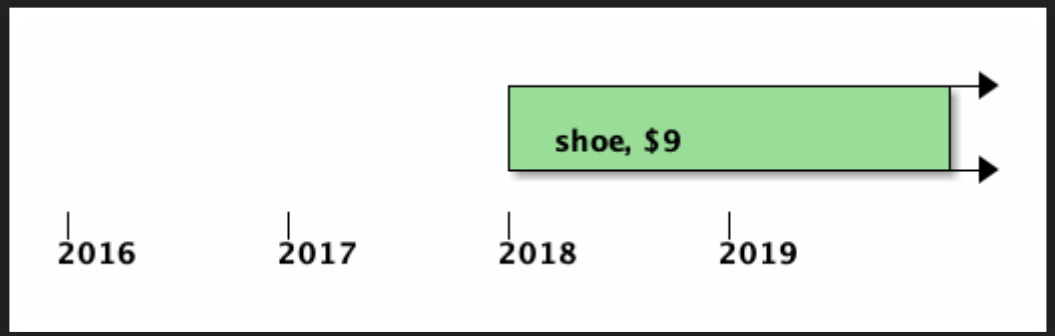


TEMPORAL UPDATE

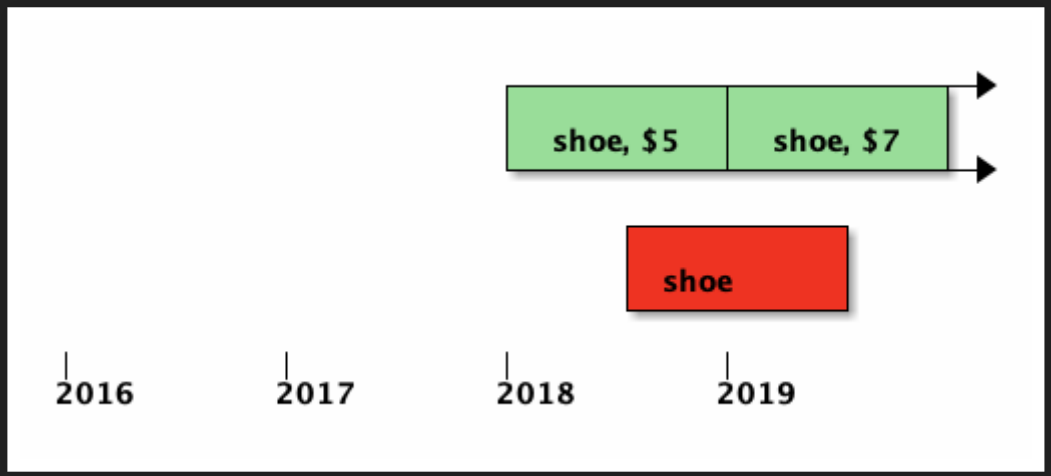


TEMPORAL UPDATE

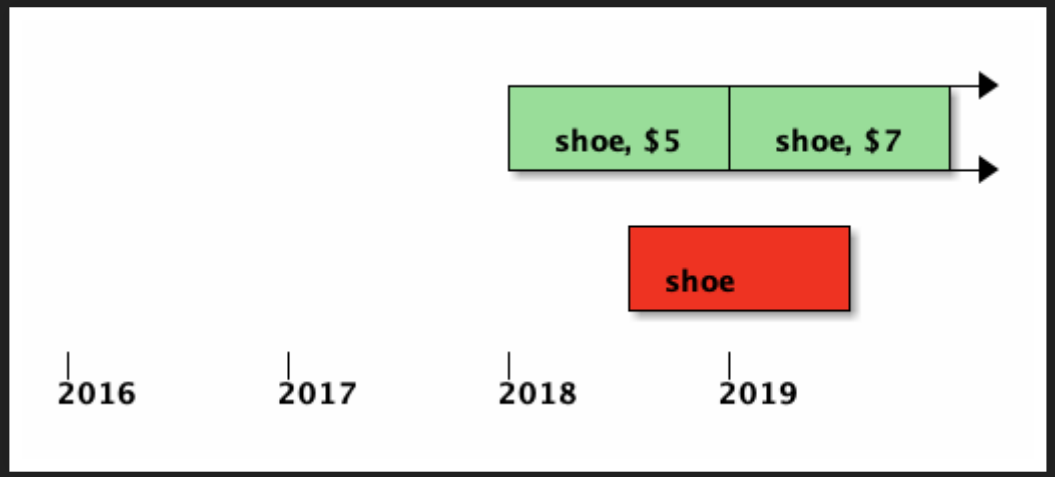


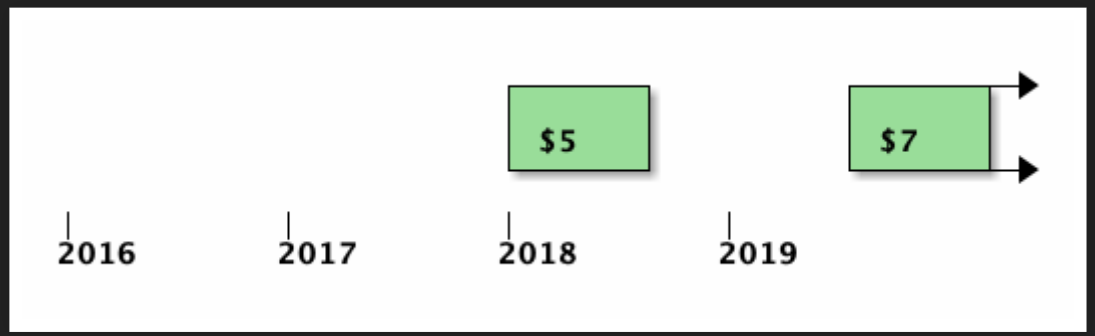


TEMPORAL DELETE

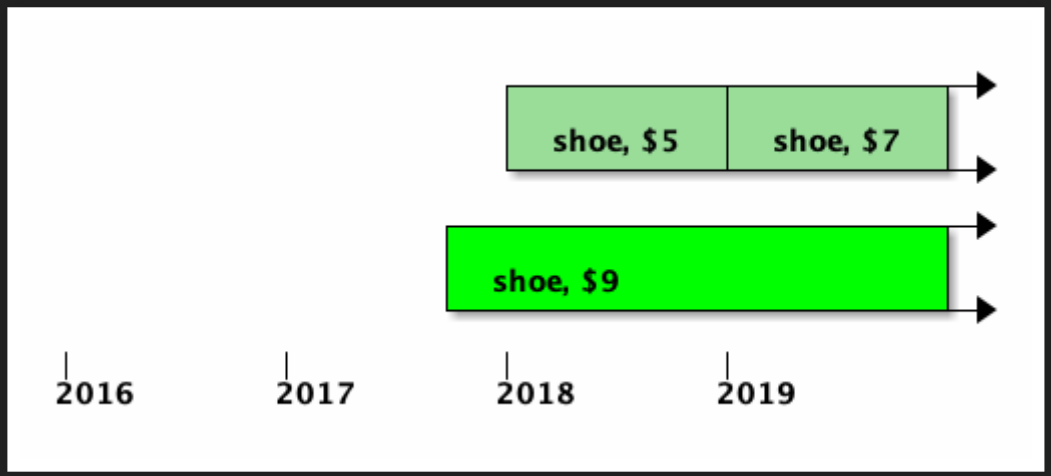


TEMPORAL DELETE

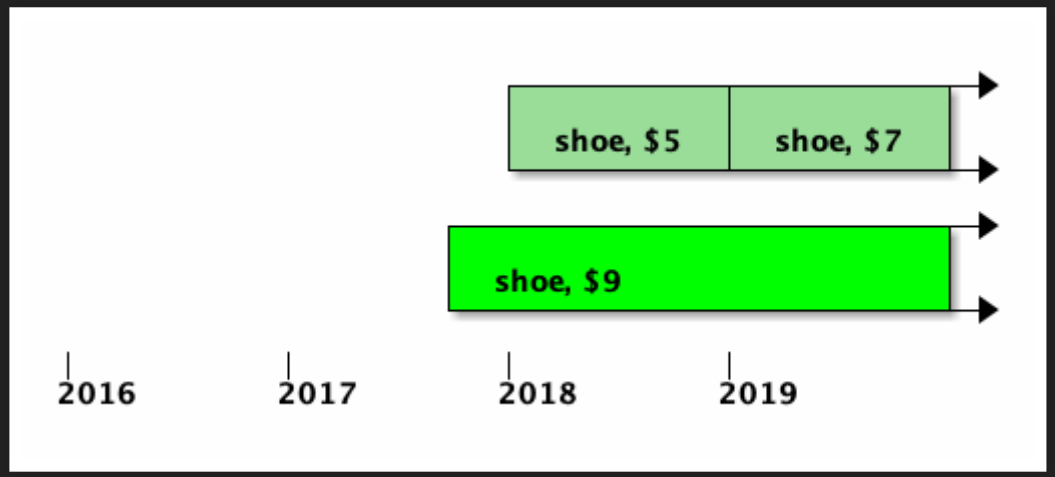


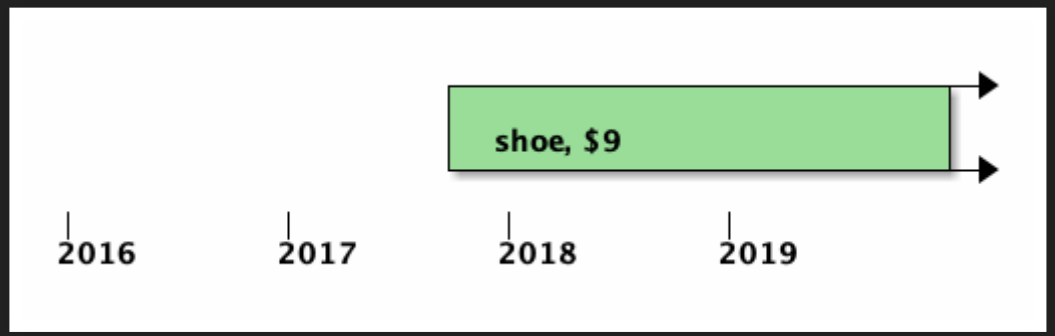


TEMPORAL UPSERT



TEMPORAL UPSERT





SQL:2011

RANGE

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

  name     text,
  price    decimal(10,2),

  CONSTRAINT pk_products
    EXCLUDE
    (id WITH =, valid_at WITH &&)
);
```

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)
);
```

PERIODS

```
SELECT * FROM t;
```

RANGES

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

  name     text,
  price    decimal(10,2),

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

DML

```
INSERT INTO products (id, price, valid_at)
VALUES (1, 5, tstzrange(.....);

UPDATE products
FOR PORTION OF valid_at FROM t1 TO t2
SET price = 4
WHERE id = 1;

DELETE FROM products
FOR PORTION OF valid_at FROM t1 TO t2
WHERE id = 1;
```

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;
```

MARIADB

```
CREATE TABLE products (
  id    integer,
  name  text,
  price decimal(10,2)
) WITH SYSTEM VERSIONING;

SELECT *, row_start, row_end FROM products;
```

ORACLE

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;

SELECT * FROM t AS OF TIMESTAMP t;
```

ORACLE

```
CREATE TABLE products (
  id         int,
  valid_from timestamp,
  valid_til  timestamp,

  name  text,
  price decimal(10,2),

  PERIOD FOR valid_at (valid_from, valid_til)
);

SELECT * FROM products
AS OF PERIOD FOR valid_at t;

SELECT * FROM products
VERSIONS PERIOD FOR valid_at BETWEEN t1 AND t2;
```

MS SQL SERVER

```
CREATE TABLE dbo.Products (
  Id       integer PRIMARY KEY CLUSTERED,
  SysFrom datetime2(2) GENERATED ALWAYS AS ROW START,
  SysTil  datetime2(2) GENERATED ALWAYS AS ROW END,
  Name     text,
  Price    decimal(10, 2),
  PERIOD FOR SYSTEM_TIME (SysFrom, SysTil)
) WITH (
  SYSTEM VERSIONING = ON
    (HISTORY TABLE = dbo.ProductsHistory)
);
```

IBM DB2

```
CREATE TABLE products (
  id        int,
  sys_from timestamp(12) GENERATED ALWAYS AS ROW BEGIN,
  sys_til  timestamp(12) GENERATED ALWAYS AS ROW END,
  tx_id    timestamp(12) GENERATED ALWAYS AS TRANSACTION START

  name  text,
  price decimal(10,2),

  PERIOD SYSTEM_TIME (sys_from, sys_til)
);
```

IBM DB2

```
CREATE TABLE products (
  id         int,
  valid_from timestamp(12) NOT NULL,
  valid_til  timestamp(12) NOT NULL,

  name  text,
  price decimal(10,2),

  PERIOD BUSINESS_TIME (valid_from, valid_til)
);
```

POSTGRES?

```
CREATE TABLE products (
  id         integer,
  claimed_at tstzrange GENERATED ALWAYS AS ROW RANGE,
  name       text,
  price      decimal(10,2)
) WITH SYSTEM VERSIONING (claimed_at);
```

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

SQL:2011

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

OTHER VENDORS

- 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://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/arkhipov/temporal_tables
- https://www.youtube.com/watch?v=TRgni5q0YM8
- https://github.com/ifad/chronomodel