LIVE WEBINAR

Bitemporality and the Art of Maintaining Accurate Databases

August 10 @ 12 noon ET (17:00 UK)

Jeremy Taylor

Head of Product,
XTDB @ JUXT

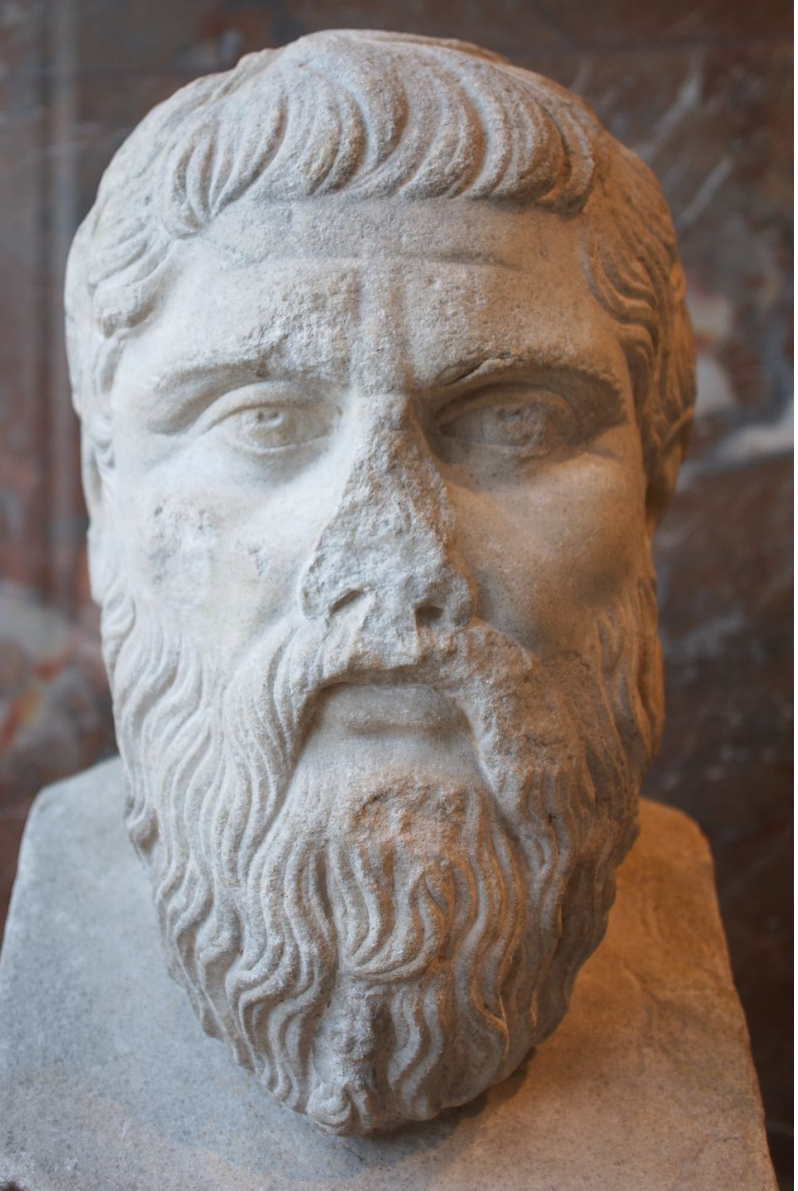
James Henderson

Engineering Lead,
XTDB @ JUXT

JUXT

Agenda

1 / The state of things – time is intrinsic to data
2 / What is bitemporality
3 / SQL:2011 – "temporal tables"
4 / Challenges for users, developers & vendors
5 / How we can embrace bitemporality



The moving image
of eternity.

Reality is messy.

- Business domains are understood and encoded gradually, therefore good design happens through iteration
- Successful systems evolve to cope with inevitable and unpredictable delays, mistakes and changes
- Software must capture complex business processes with “adjustments”, “corrections”, “exceptions”
- “we want what’s recorded in the system to match the real world” – Kent Beck, 2023
- ...and so do industry auditors & regulators!

BUSINESS ARCHITECTURE

Eventual Business Consistency

Executive Summary of Bi-temporality

KENT BECK
4 AUG 2023

I'm a geek speaking to you, a technology-savvy executive, about why we are doing things in a more complicated way than seems necessary. You may have heard the word "bi-temporal". What's that about?

In a nutshell, we want what's recorded in the system to match the real world. We know this is impossible (delays, mistakes, changes) but are getting as close as we can. The promise is that if what's in the system matches the real world as closely as possible, costs go down, customer satisfaction goes up, & we are able to scale further faster.

Here's how it works.

Scenarios

We'll take addresses as our example. Addresses are useful for sending correspondence, calculating taxes, determining regulations, & targeting marketing. Addresses, though,

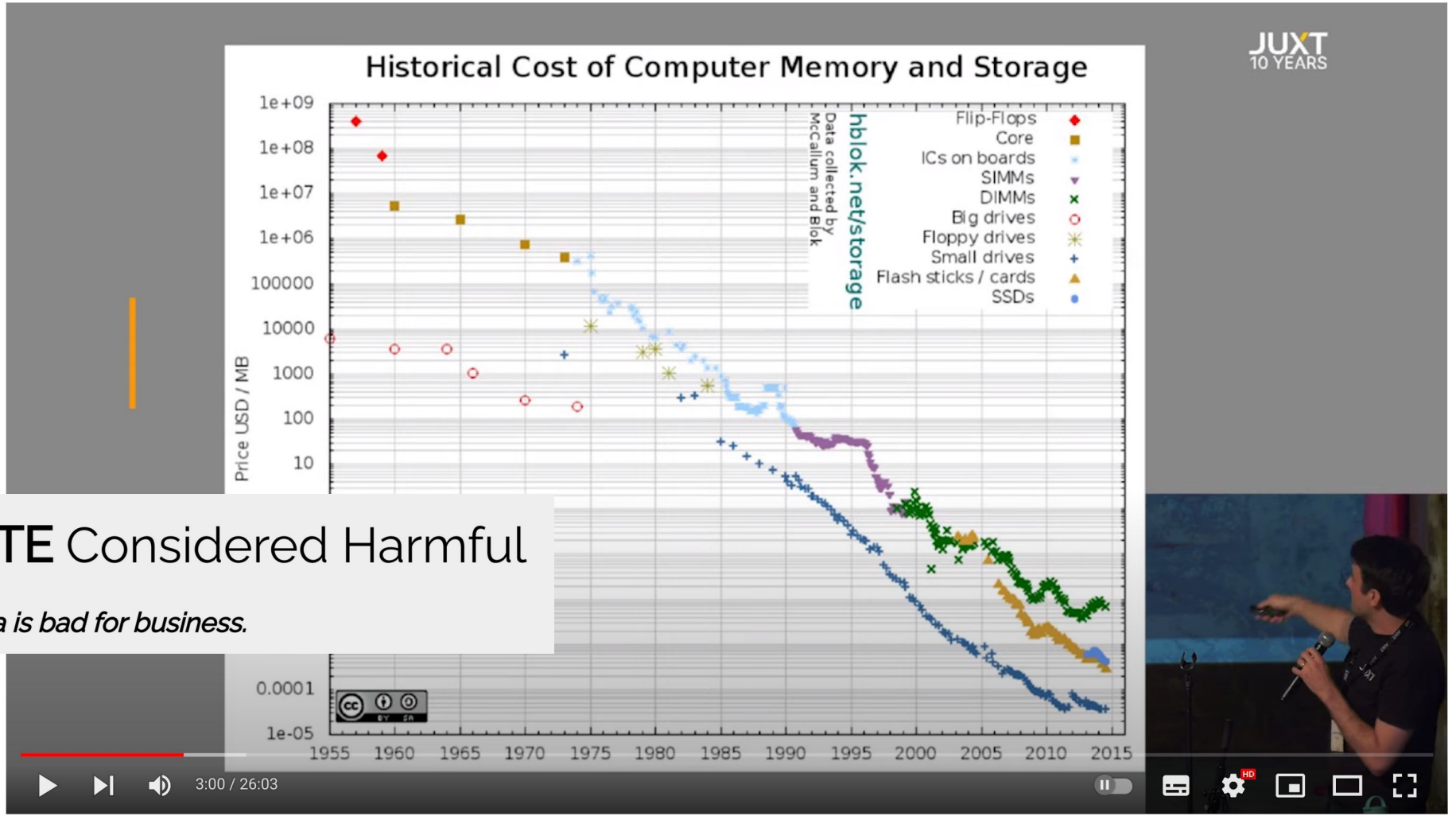
https://tidyfirst.substack.com/p/eventual-business-consistency

“if what’s in the system matches the real world as closely as possible, costs go down, customer satisfaction goes up, & we are able to scale further faster”

But databases are (still) optimized for ‘now’.

- Transactional applications are routinely built on databases that were not designed for the modern era of ~infinite storage
- Applications are carefully designed to “remember” and “forget” information in various ways – remembering things is harder than it should be!
- Stale data has to be migrated to keep acceptable performance

https://www.youtube.com/watch?v=JxMz-tyicgo



"UPDATE Considered Harmful" by Jeremy Taylor

JUXT
593 subscribers
Subscribe

15 Share Download Clip Save

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
UPDATE Customers
SET first_name = 'Johnny'
WHERE customer_id = 1;
```

customer_id	first_name	last_name	age	country
1	Johnny	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Harry	Potter	31	USA

Image credit: programiz.com

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
UPDATE Customers
SET first_name = 'Johnny'
WHERE customer_id = 1;
```

"John"

customer_id	first_name	last_name	age	country
1	Johnny	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Harry	Potter	31	USA

Image credit: programiz.com

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
UPDATE Customers
SET first_name = 'Johnny'
WHERE customer_id = 1;
```

customer_id	first_name	last_name	age	country
1	Johnny	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Harry	Potter	31	USA

Image credit: programiz.com

Living without *data* version control?!

How can we incorporate time & history into databases that operate with ‘now’?

Soft deletion probably isn't worth it (brandur.org)

654 points by lfittl 64 days ago | hide | past | favorite | 497 comments

▲ JohnBooty 64 days ago | next [–]

I've been a software dev since the 90s and at this point, I've learned to basically do things like audit trails and soft deletion by default, unless there's some reason *not* to.

Somebody *always* wants to undelete something, or examine it to see why it was deleted, or see who changed something, or blah blah blah. It helps the business, it helps *you* as developer by giving you debug information as well as helping you to cover your ass when you are blamed for some data loss bug that was really user error.

Soft deletion has obvious drawbacks but is usually far less work than implementing equivalent functionality out-of-stream, with verbose logging or some such.

Retrofitting your app and adding soft deletion and audit trails after the fact is usually an order of magnitude more work. Can always add it pre-launch and leave it turned off.

If performance is a concern, this is usually something that can be mitigated. You can e.g. have a reaper job that runs daily and hard-deletes everything that was soft-deleted more than *n* days ago, or whatever.

▲ gmiller123456 64 days ago | parent | next [–]

The author uses the "no one ever undeleted anything" as the primary justification. I think this is the part they miss. I've never undeleted a user either, but there have been many times I've gone back to look at something. Either a complaint finally gets around to me as to why the user wanted their account deleted (e.g. feature not working) and it helps to figure out why. Or they're returning and want things set up like they were. Or someone is taking over their roll and needs to be set up like the last person who's already gone.

Though you really shouldn't be relying on a database for an audit trail. It might help find some issues, but things actually used for security shouldn't be writable so easily.

▲ JohnBooty 63 days ago | root | parent | next [–]

▲ The trouble with soft delete (richarddingwall.name)

37 points by concretecode on Nov 21, 2009 | hide | past | favorite | 19 comments

▲ patio11 on Nov 21, 2009 | next [–]

A million inserts into a table a year is causing performance problems? Assuming 200 work days and all accesses being in a four hour period that is about one row inserted every three seconds. DBs are not my bag, baby, but that presumably should not be killing you.

At the day job, when we do a soft delete as defined here, we tend to create a view of the active rows in the table. Accessing through the view rather than the table prevents many of the "Whoopsie, missed a WHERE clause, now I'm summing over deleted records" errors. I'm told it also improves performance but take anything I say about DBs with a grain of salt.

▲ bmj on Nov 21, 2009 | parent | next [–]

We use soft deletes in our system (primarily because we *have* to maintain all data), and we use views to retrieve results. The views also flatten the data a bit, joining appropriate tables, which again simplifies queries.

▲ russell on Nov 21, 2009 | prev | next [–]

Dingwall lumps a number of separate issues under "soft delete": undo, audit trails, soft create, and performance in the presence of historical data. He presents several solutions, not all of which I would buy.

The is_deleted column is a pretty simple solution that we all use and there are a number of solutions to the problem of retrieving only the active columns, such as views.

Audit trails and performance are more interesting. A while back I worked for a web analytics company and we had the problem of the storage and performance costs of historical data. Only the 5% of the data was of any real interest, but the 95% historical data made writes slow because of the large number of indexes. They adopted the solution of historical tables with fewer indexes on cheaper drives.

I like the solution of serializing historical, deleted, and audit data and storing them in a NonSQL database of your choice. Then you can bring them back as individual undos, or into a data mining database for scenario playing.

I dont particularly like his suggestion of creating separate tables for each state of an element. I think that's needless complexity.

▲ The trouble with soft delete (richarddingwall.name)

37 points by concr... on Nov 21, 2009 | hide | past | favorite | 19 comments

Isn't this all overkill?

Probably. If you're already using soft delete and haven't had any problems then you don't need to worry — soft delete was a sensible trade-off for your application that hasn't caused any serious issues so far.

But if you're anticipating growth, or already encountering scalability problems as dead bodies pile up in your database, you might like to look at alternatives that better satisfy your application's requirements.

The truth is soft delete is a poor solution for most of the problems it promises to solve. Instead, focus on what you're *actually* trying to achieve. Keep everything simple and follow these guidelines:

- Primary transactional tables should only contain data that is valid and active right now.
- Do you really need to be able to undo deletes? If so, there are dedicated patterns to handle this.
- Audit logging at the row level sucks. Do it higher up where you know the full story.
- If a row doesn't apply yet, put it in a queue until it does.
- Physically separate items in different states based on their query usage.

Above all, make sure you're not gold plating tables with soft delete simply out of habit!

▲ patio11 on Nov

A million inse... a four hour period that is about ... ou.

At the day job... ng through the view rather than th... errors. I'm told it also improves perf...

▲ bmj on

We us... esults. The views also flatten...

▲ russell on Nov ...

Dingwall lump... sence of historical data. He pres...

The is_delete... rieving only the active columns, such...

Audit trails an... blem of the storage and performa... le writes slow because of th... es.

I like the solution of serializing historical, deleted, and audit data and storing them in a NonSQL database of your choice. Then you can bring them back as individual undos, or into a data mining database for scenario playing.

I dont particularly like his suggestion of creating separate tables for each state of an element. I think that's needless complexity.

Shubham Sonawane

Jun 30, 2021 · 7 min read · Listen

Soft deletes are tedious! Does an ideal deletion without loss even exist?

Factor	Details	Advantage
Ease of Setup	Soft Delete is easier to implement since it merely involves updating a column while hard delete would also involve copying the data to be deleted to an audit table.	Soft Delete
Debugging	Soft Delete makes it easy to debug data issues due to the deleted flag. But debugging via the Audit table is also easily possible. So its a tie.	N/A
Restoring data	It is extremely easy to restore data 'deleted' via soft delete since it just involved unsetting the deleted flag. However note that restoring data is an extremely rare occurance.	Soft Delete
Querying for active data	Extra deletion checks are required for Select and Update queries. Developers must exercise caution or else they risk retrieving results that have been deleted. (might not apply to ORMs)	Hard Delete
View Simplicity	Having all the data in the tables as active data relates to view simplicity. In Hard delete, all 'deleted' data will only be present in the audit table while the rest of the tables in the system will have 'active' data.	Hard Delete
Performance of operation	Update is a little faster than delete. So soft delete should technically be faster than hard delete (which also has the audit table insert to consider).	Soft Delete
Application Performance (Speed)	To support soft deletes, ALL select queries need to have a flagged condition In situations where JOINs are involved there will be multiple such conditions. Select queries with lesser conditions are faster than those with conditions.	Hard Delete
Application Performance (Size)	To support faster soft deletes, we need to have an index for every deleted flag in EVERY table Additionally the table size keeps increasing since the table has 'soft deleted' data + active data.	Hard Delete
Unique Index	Additionally we cannot update the old soft deleted entry of A1-B1 since it would mean rewriting some data which results in loss of recorded data.	Hard Delete
Cascading	For soft delete, we cannot make use of 'ON DELETE' cascading. The alternative is to create an 'UPDATE' trigger which keeps track of deleted_flag.	Hard Delete

https://medium.com/geekculture/soft-deletes-are-tedious-does-an-ideal-deletion-without-loss-even-exist-9cc5d78e9b10

A history of price mutations...

product_prices

id	price
apple	2.00
banana	3.00
carrot	1.00

product_prices

id	price
apple	2.00
banana	3.00
carrot	1.00

product_prices

id	price	created_at
apple	2.00	2023-01
banana	3.00	2023-01
carrot	1.00	2023-01

product_prices

id	price
apple	2.00
banana	3.00
carrot	1.00

product_prices

id	price	created_at
apple	2.00	2023-01
banana	3.00	2023-01
carrot	1.00	2023-01

product_prices

id	price	created_at	updated_at
apple	2.00	2023-01	2023-01
banana	3.00	2023-01	2023-08
carrot	1.00	2023-01	2023-01

product_prices

id	price
apple	2.00
banana	3.00
carrot	1.00

product_prices

id	price	created_at
apple	2.00	2023-01
banana	3.00	2023-01
carrot	1.00	2023-01

product_prices

id	price	created_at	updated_at
apple	2.00	2023-01	2023-01
banana	3.00	2023-01	2023-08
carrot	1.00	2023-01	2023-01

product_prices

id	version	price	created_at
apple	1	2.00	2023-01
banana	1	2.00	2023-01
banana	2	3.00	2023-08
carrot	1	1.00	2023-01

product_prices

id	price
apple	2.00
banana	3.00
carrot	1.00

product_prices

id	price	created_at
apple	2.00	2023-01
banana	3.00	2023-01
carrot	1.00	2023-01

product_prices

id	price	created_at	updated_at
apple	2.00	2023-01	2023-01
banana	3.00	2023-01	2023-08
carrot	1.00	2023-01	2023-01

product_prices

id	version	price	created_at
apple	1	2.00	2023-01
banana	1	2.00	2023-01
banana	2	3.00	2023-08
carrot	1	1.00	2023-01

current_product_prices

id	price	created_at	updated_at
apple	2.00	2023-01	2023-01
banana	3.00	2023-01	2023-08
carrot	1.00	2023-01	2023-01

historical_product_prices

id	updated_at	price	created_at
banana	2023-01	2.00	2023-01

Storing mutation history ***in*** the database affords us:

- Stable basis
  - Run reports consistently (e.g. calculate net margin as-of the start of the month)
  - Auditable answers to customer queries
  - Complex questions can be decomposed across multiple queries
  - Change detection (answer 'what changed' questions for integrating with 'derived data' systems)

Storing mutation history ***in*** the database affords us:

- Stable basis
  - Run reports consistently (e.g. calculate net margin as-of the start of the month)
  - Auditable answers to customer queries
  - Complex questions can be decomposed across multiple queries
  - Change detection (answer 'what changed' questions for integrating with 'derived data' systems)
- Developer safety
  - Retrieving & restoring old versions is Ops-free
  - Easier debugging (reproduce the conditions of a bug at the time it happened)

Is mutation history enough to capture your application complexities?

- Unlikely!
  - Schema changes must (somehow) not affect historical versions
  - Cannot import late-arriving, out-of-order data
  - No affordance to ‘correct’ earlier versions
  - Will your application ever need time travel?

Who invariably needs history & time travel?

Anyone who works with customers...

customer_address

id	posted_at	effective_from	address
kent	2018	2018	94417
kent	2021	2020	94414
kent	2022	2020	94415
kent	2023	2024	94419

“I forgot to tell you that I moved last year.”

“You got last year’s address change wrong.”

“I’m going to move next year.”

Anyone who works with employees...

“Carol’s salary needs to be increased by 10%,
backdated to the last payroll period.”

“Bob worked 2 days last week and his time needs
to be included in next month’s invoice.”

“Alice is going on holiday in 2 weeks’ time,
who is available to cover her?”

Anyone who works with calculations & reports...

“What was my customer’s credit rating last Monday as I knew it last Friday?”

“What is the financial risk exposure of my portfolio based on current market data compared with yesterday?”

“What did we think our customer’s credit rating was at the time Lehman defaulted when we told the SEC that all of our customers had high credit ratings?”

Anyone who sells things...

product_prices

id	version	price	created_at
apple	1	2.00	2023-01
banana	1	2.00	2023-01
banana	2	3.00	2023-08
carrot	1	1.00	2023-01

product_taxes

id	version	percentage	created_at
apple	1	10	2023-01
apple	2	15	2023-08
banana	1	20	2023-01
banana	2	18	2023-08
carrot	1	10	2023-01

product_discount

id	version	percentage	created_at
apple	1	10	2023-01
banana	1	10	2023-01
carrot	1	10	2023-01
carrot	2	20	2023-08

product_shipping

id	version	rate	created_at
apple	1	1.00	2023-01
apple	2	1.50	2023-08
banana	1	1.00	2023-01
carrot	1	1.00	2023-01
carrot	2	2.00	2023-08

...and so on...

How can we **rigorously** model historical versions of rows in the relational model?

Bitemporality

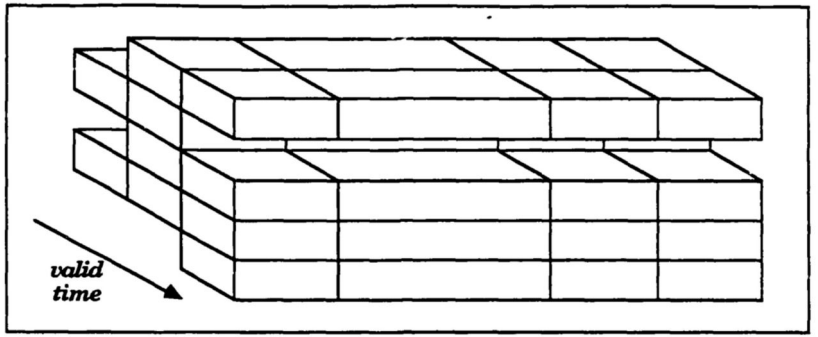


Figure 3: Historical Relation

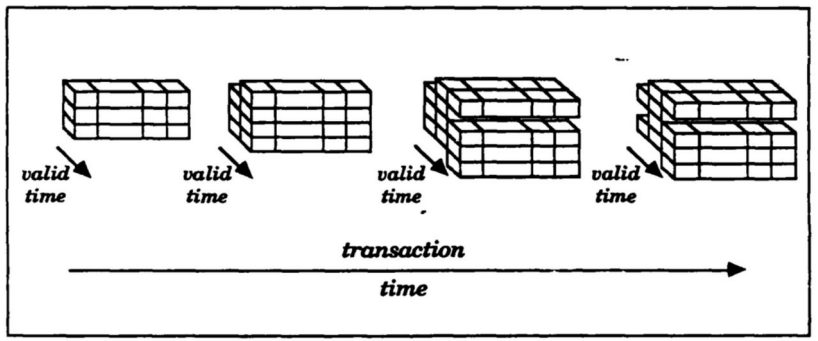
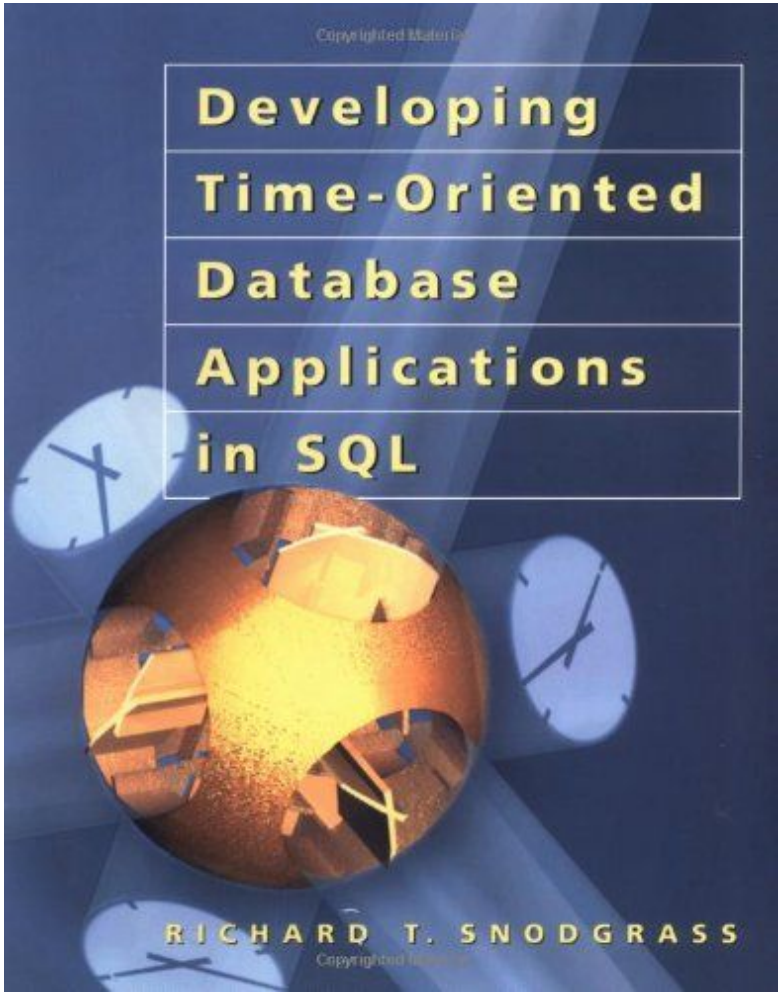


Figure 4: Temporal Relation

Scheme Evolution and the Relationship Algebra, May 1988
https://apps.dtic.mil/sti/tr/pdf/ADA201297.pdf



Released: 1st August 1999
Available freely online: https://www2.cs.arizona.edu/~rts/tdbbook.pdf


Nykredit

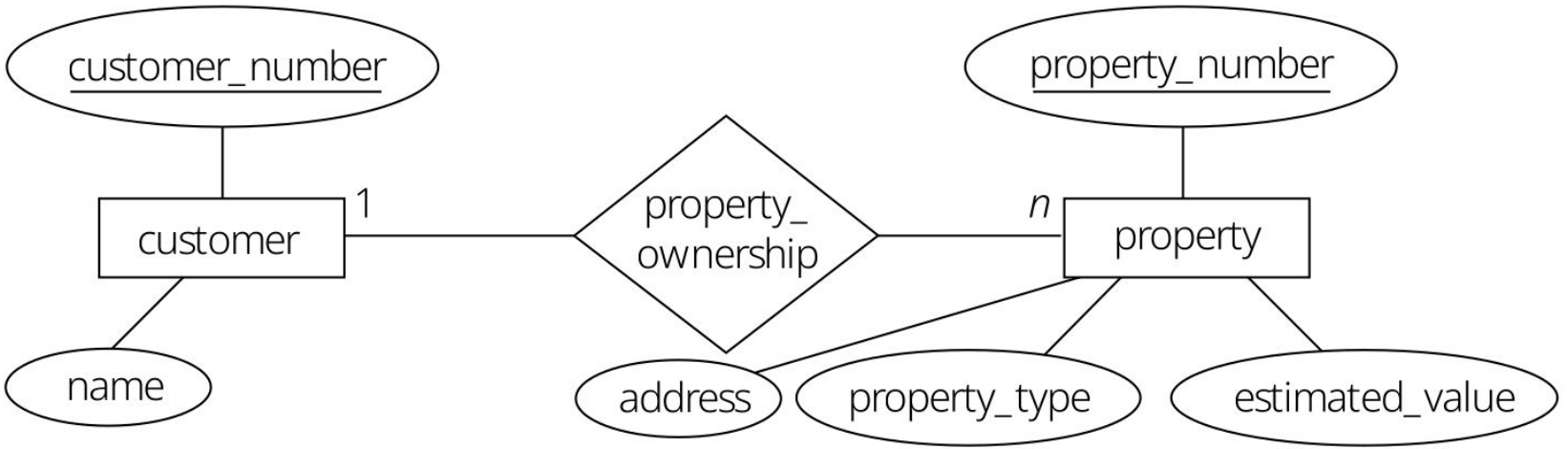
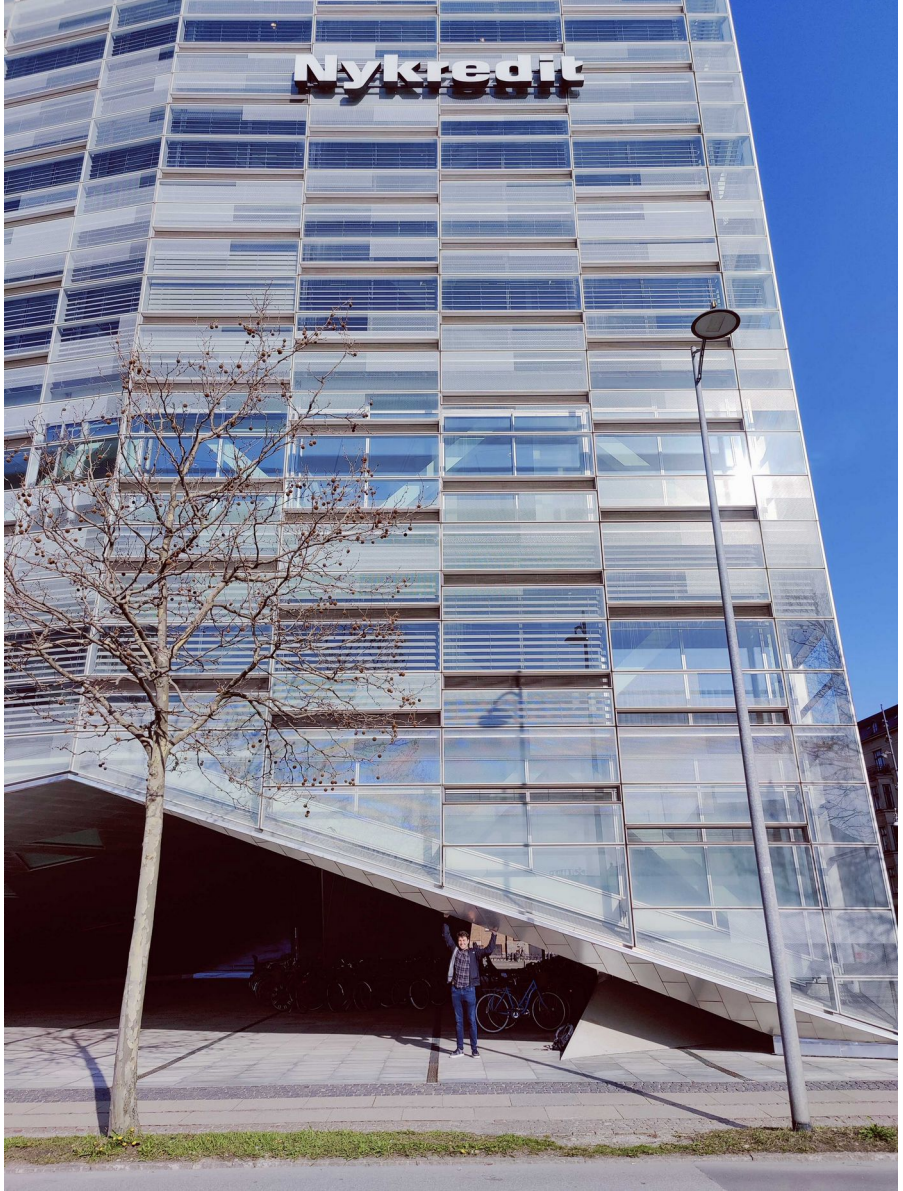
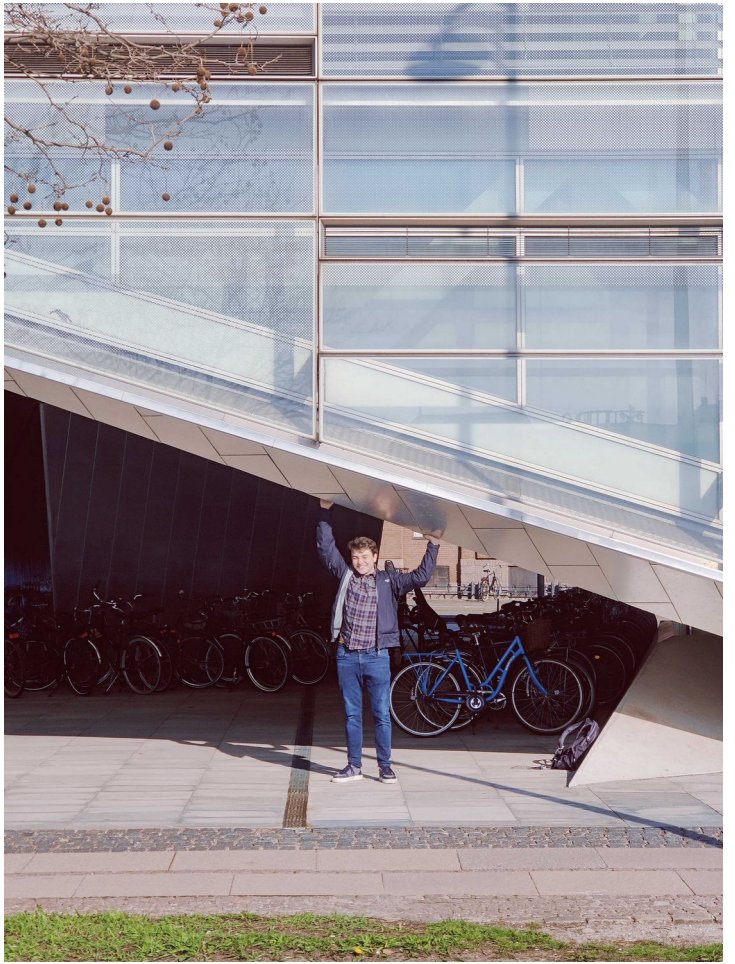


Figure 10.1 The property ownership relationship.

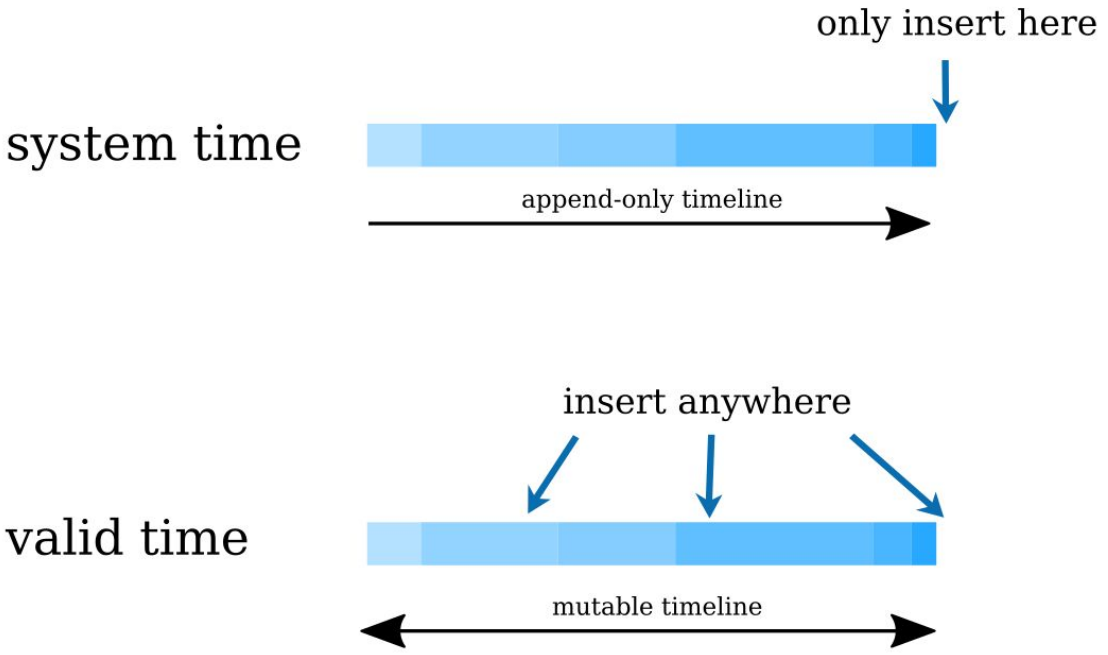

Nykredit



To maintain a ‘bitemporal’ version of a row:

- Always record two timestamps alongside your data
- One is the timeline of when things actually happened
- The other is the timeline of when information was inserted into the system
- “The purpose of the 2 dates is to make sure that our system is eventually consistent with reality”

To maintain a ‘bitemporal’ version of a row:



System Time

- Immutable!
- Auditable history of data *as we knew it*
- The system lifecycle of a record
- A stable **basis** for *decision making*

Valid Time

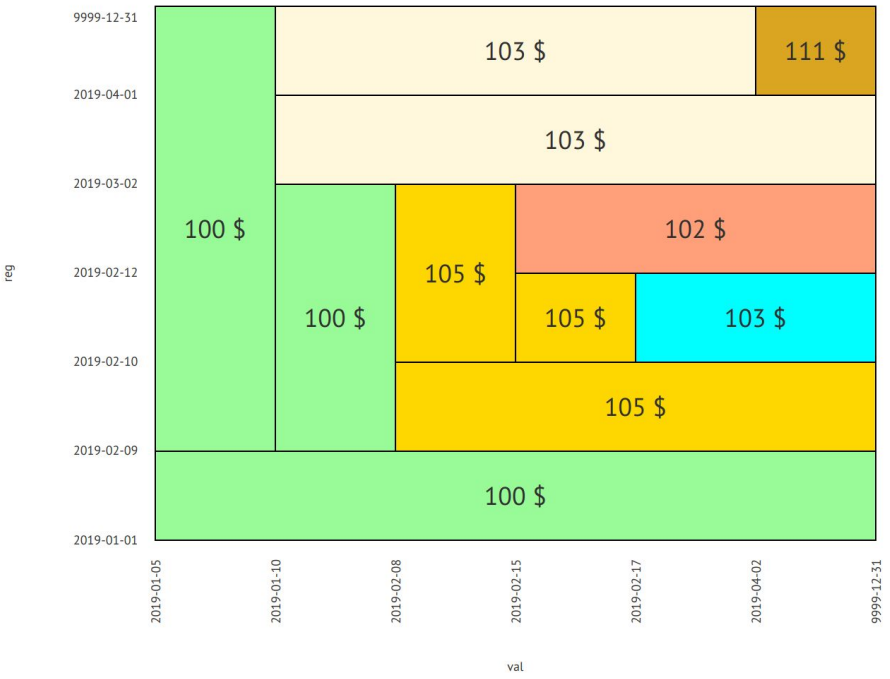
- Mutable!
- *Potentially* meaningful to the application
- The domain lifecycle of a record

Bitemporality

- Timestamps imply *periods* (e.g. from **now** until the **end of time**), modelled as intervals or start + end
- System time period = period during which a fact stored in the database is regarded as true (a fact is true until corrected)
- Valid time period = period during which a fact is true in the real world
- Bitemporal data simply combines both system and valid time dimensions

2, 3 or 4 timestamps? (and no overlaps allowed!)

product_id	registered_from	registered_to	valid_from	valid_to	price
1002	2019-01-01	2019-02-09	2019-01-05	9999-12-31	100
1002	2019-02-09	9999-12-31	2019-01-05	2019-01-10	100
1002	2019-02-09	2019-03-02	2019-01-10	2019-02-08	100
1002	2019-02-09	2019-02-10	2019-02-08	9999-12-31	105
1002	2019-02-10	2019-03-02	2019-02-08	2019-02-15	105
1002	2019-02-10	2019-02-12	2019-02-15	2019-02-17	105
1002	2019-02-10	2019-02-12	2019-02-17	9999-12-31	103
1002	2019-02-12	2019-03-02	2019-02-15	9999-12-31	102
1002	2019-03-02	2019-04-01	2019-01-10	9999-12-31	103
1002	2019-04-01	9999-12-31	2019-01-10	2019-04-02	103
1002	2019-04-01	9999-12-31	2019-04-02	9999-12-31	111



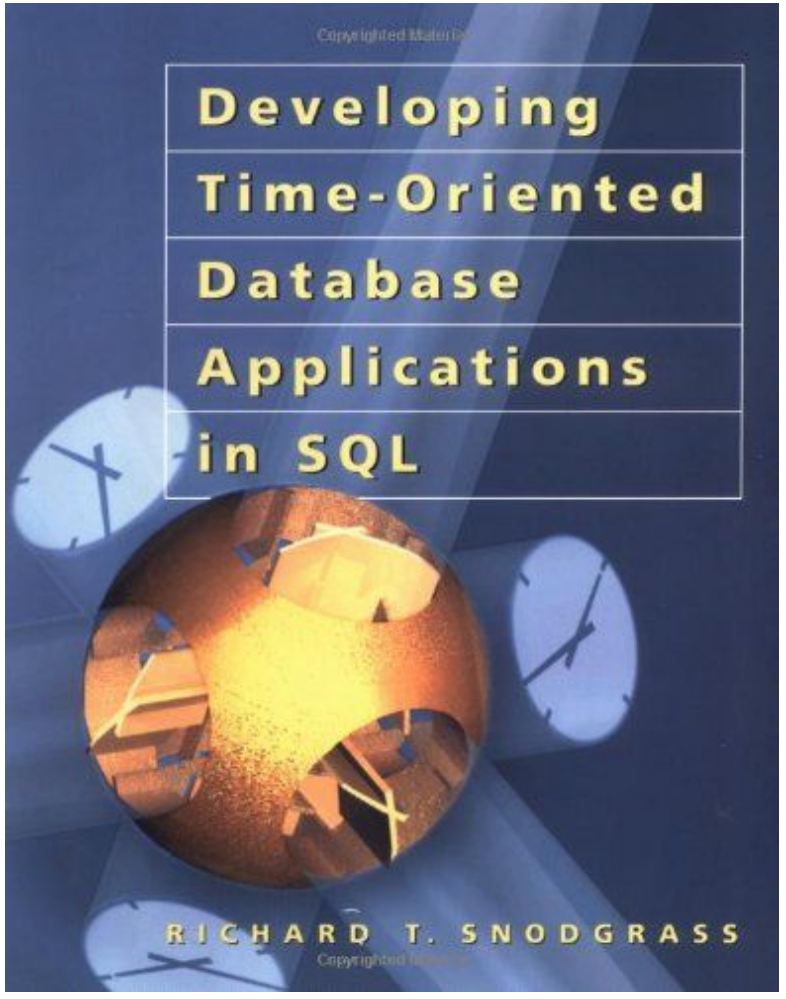
https://bitemporal.net/generate-bitemporal-intervals/

Terminology

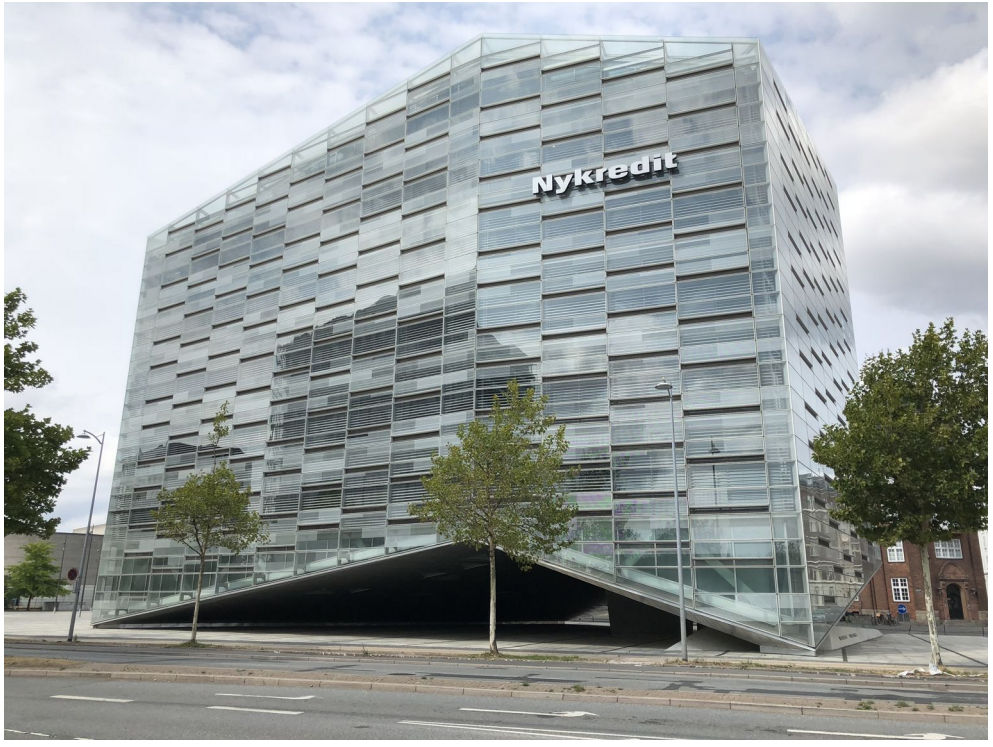
SQL:2011 vs.	Application Time Synonym	System Time Synonym
Snodgrass	Valid	Transaction
Beck	Effective	Posting
Fowler	Actual	Record
Date/Darwen	Stated	Logged
Johnston	Effective	Assertion
Kafka	Event	Ingestion

Implications

Valid Time	System Time
History of things being modelled	History of changes to the database
Underpinning user-facing features	Auditing and compliance
Modify the past, present and future	Immutable, append-only
Maintained by your app	Maintained by Triggers/ORM/DB



10.2 Modifications



Available freely online: https://www2.cs.arizona.edu/~rts/tdbbook.pdf

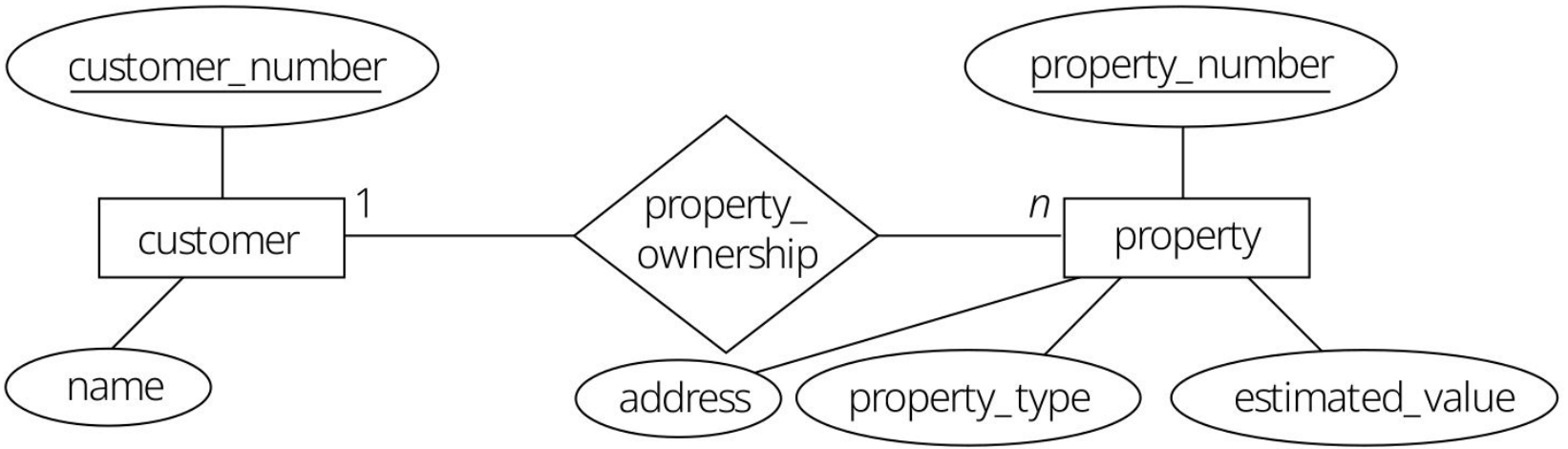


Figure 10.1 The property ownership relationship.

Code Fragment 10.1 **Create the** Prop_Owner **table.**

```
CREATE TABLE Prop_Owner (
  customer_number   INT,
  property_number   INT,
  VT_Begin   DATE,
  VT_End   DATE,
  TT_Start   TIMESTAMP,
  TT_Stop   TIMESTAMP)
```

```
CREATE TABLE Customer (
  name              CHAR,
  VT_Begin          DATE,
  VT_End            DATE,
  TT_Start          TIMESTAMP,
  TT_Stop           TIMESTAMP)

CREATE TABLE Property (
  property_number   INT,
  address           CHAR,
  property_type     INT,
  estimated_value   INT,
  VT_Begin          DATE,
  VT_End            DATE,
  TT_Start          TIMESTAMP,
  TT_Stop           TIMESTAMP)
```

Code Fragment 10.2 property_number **is a (valid-time sequenced, transaction-time sequenced) primary key for** Prop_Owner**.**

```
CREATE ASSERTION P_O_seq_primary_key
CHECK (NOT EXISTS (SELECT *
          FROM Prop_Owner AS P1
          WHERE property_number IS NULL
          OR 1 < (SELECT COUNT(customer_number)
                  FROM Prop_Owner AS P2
                  WHERE P1.property_number = P2.property_number
                    AND P1.VT_Begin < P2.VT_End
                    AND P2.VT_Begin < P1.VT_End
                    AND P1.TT_Stop = DATE '9999-12-31'
                    AND P2.TT_Stop = DATE '9999-12-31'))
)
```

Code Fragment 10.3 Prop_Owner.property_number **defines a contiguous valid-time history.**

```
CREATE ASSERTION P_O_Contiguous_History
CHECK (NOT EXISTS (SELECT *
       FROM Prop_Owner AS P, Prop_Owner AS P2
       WHERE P.VT_End < P2.VT_Begin
         AND P.property_number = P2.property_number
         AND P.TT_Stop = DATE '9999-12-31'
         AND P2.TT_Stop = DATE '9999-12-31'
         AND NOT EXISTS (
             SELECT *
             FROM Prop_Owner AS P3
             WHERE P3.property_number = P.property_number
               AND (((P3.VT_Begin <= P.VT_End)
                     AND (P.VT_End < P3.VT_End))
                   OR ((P3.VT_Begin < P2.VT_Begin)
                     AND (P2.VT_Begin <= P3.VT_End)))
               AND P3.TT_Stop = DATE '9999-12-31'))
)
```

Code Fragment 10.4 **Eva Nielsen buys the flat at Skovvej 30 in Aalborg on January 10, 1998.**

```
INSERT INTO Prop_Owner (customer_number, property_number, VT_Begin,
    VT_End, TT_Start, TT_Stop)
VALUES (145, 7797, CURRENT_DATE,
    DATE '9999-12-31', CURRENT_TIMESTAMP, DATE '9999-12-31')
```

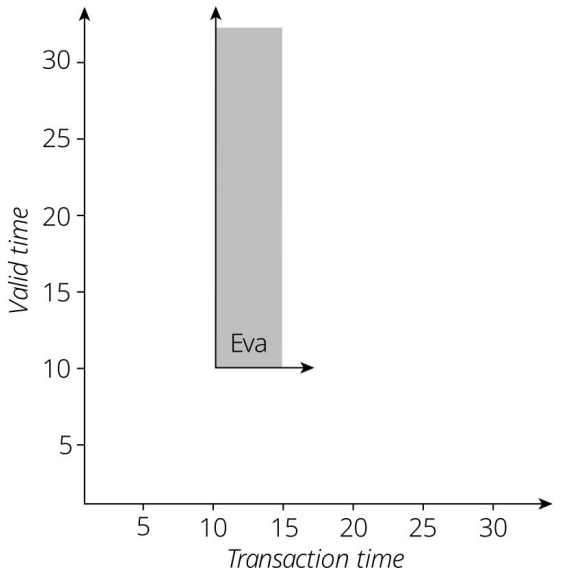


Figure 10.2 A bitemporal time diagram corresponding to Eva purchasing the flat, performed on January 10.

Code Fragment 10.5 **Peter Olsen buys the flat on January 15, 1998.**

```
UPDATE Prop_Owner
SET customer_number = 827
WHERE property_number = 7797
```

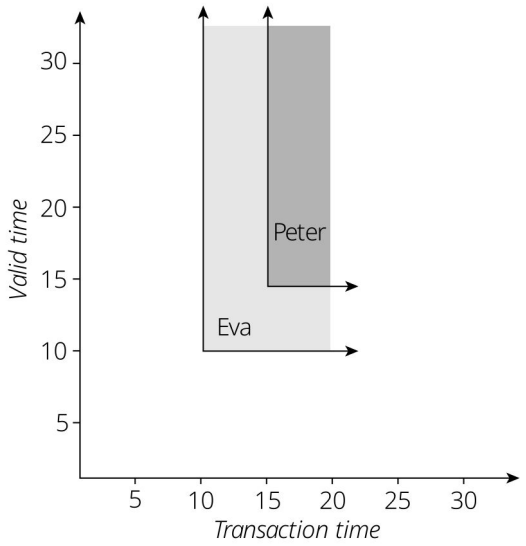


Figure 10.3 A current update: Peter buys the flat, performed on January 15.

Code Fragment 10.7 **Peter Olsen buys the flat on January 15, 1998, a current update.**

```
INSERT INTO Prop_Owner
SELECT 827, property_number, CURRENT_DATE, VT_End,
  CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner

WHERE property_number = 7797
  AND VT_Begin <= CURRENT_DATE
  AND VT_End > CURRENT_DATE
  AND TT_Stop = DATE '9999-12-31'

INSERT INTO Prop_Owner
SELECT customer_number, property_number, VT_Begin, CURRENT_DATE,
  CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner
WHERE property_number = 7797
  AND VT_Begin < CURRENT_DATE
  AND VT_End > CURRENT_DATE
  AND TT_Stop = DATE '9999-12-31'

UPDATE Prop_Owner
SET TT_Stop = CURRENT_TIMESTAMP
WHERE property_number = 7797
  AND VT_Begin < CURRENT_DATE
  AND VT_End > CURRENT_DATE
  AND TT_Stop = DATE '9999-12-31'

INSERT INTO Prop_Owner
SELECT 827, property_number, VT_Begin, VT_End,
  CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner
WHERE property_number = 7797
  AND VT_Begin > CURRENT_DATE
  AND TT_Stop = DATE '9999-12-31'

UPDATE Prop_Owner
SET TT_Stop = CURRENT_TIMESTAMP
WHERE property_number = 7797
  AND VT_Begin > CURRENT_DATE
  AND TT_Stop = DATE '9999-12-31'
```

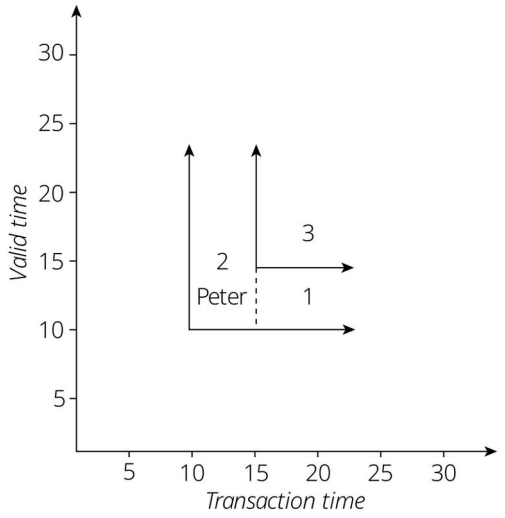


Figure 10.4 Splitting a polygonal region into rectangles.

Table 10.1 Result of the current insertion.

customer_ number	property_ number	VT_Begin	VT_End	TT_Start	TT_Stop
145	7797	1998-01-10	9999-12-31	1998-01-10	*1998-01-15*
145	*7797*	*1998-01-10*	*1998-01-15*	*1998-01-15*	*9999-12-31*
827	*7797*	*1998-01-15*	*9999-12-31*	*1998-01-15*	*9999-12-31*

Code Fragment 10.8 **Peter Olsen sells the flat on January 20, 1998.**

```
DELETE FROM Prop_Owner
WHERE property_number = 7797
```

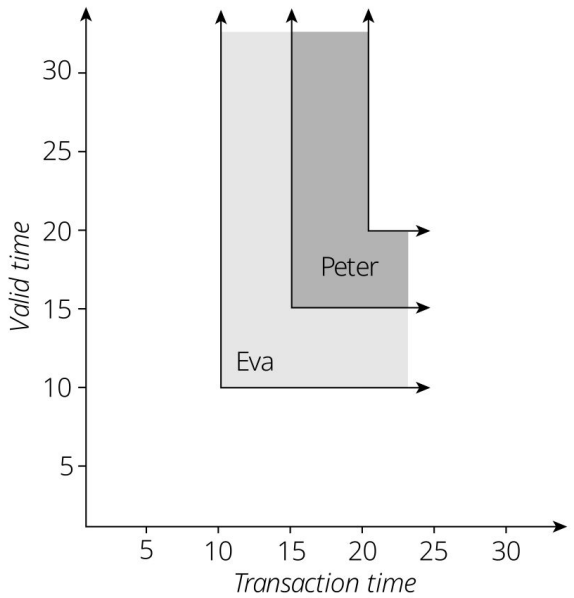


Figure 10.6 A current deletion: Peter sells the flat, performed on January 20.

Code Fragment 10.11 **Peter Olsen sells the flat on January 20, 1998, a current deletion, simplified version.**

```
INSERT INTO Prop_Owner
SELECT customer_number, property_number, VT_Begin, CURRENT_DATE,
    CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner

WHERE property_number = 7797
  AND VT_Begin < CURRENT_DATE
  AND VT_End > CURRENT_DATE
  AND TT_Stop = DATE '9999-12-31'

UPDATE Prop_Owner
SET TT_Stop = CURRENT_TIMESTAMP
WHERE property_number = 7797
  AND VT_End > CURRENT_DATE
  AND TT_Stop = DATE '9999-12-31'
```

Table 10.2 Result of the current deletion.

customer_ number	property_ number	VT_Begin	VT_End	TT_Start	TT_Stop
145	7797	1998-01-10	9999-12-31	1998-01-10	1998-01-15
145	7797	1998-01-10	1998-01-15	1998-01-15	9999-12-31
827	7797	1998-01-15	9999-12-31	1998-01-15	*1998-01-20*
827	*7797*	*1998-01-15*	*1998-01-20*	*1998-01-20*	*9999-12-31*

Code Fragment 10.12 **Eva actually purchased the flat on January 3, performed on January 23.**

```
INSERT INTO Prop_Owner (customer_number, property_number, VT_Begin,
    VT_End, TT_Start, TT_Stop)
VALUES (145, 7797, DATE '1998-01-03',
    DATE '1998-01-10', CURRENT_TIMESTAMP, DATE '9999-12-31')
```

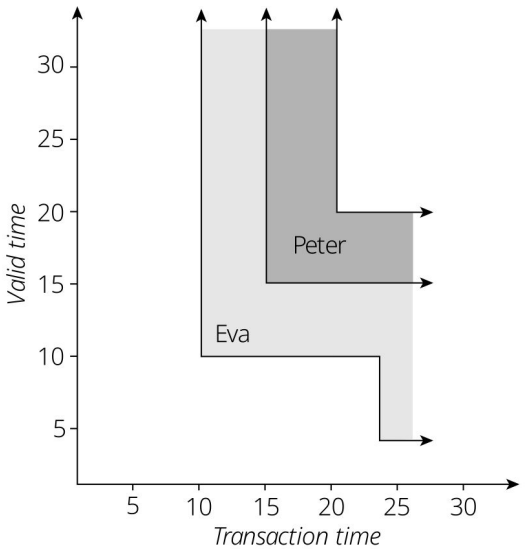


Figure 10.8 A sequenced insertion performed on January 23: Eva actually purchased the flat on January 3.

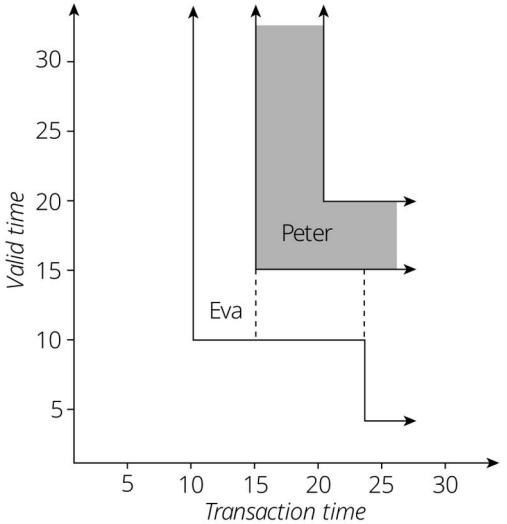


Figure 10.10 An alternate splitting into rectangles.

Table 10.3 Result of the sequenced insertion.

customer_ number	property_ number	VT_Begin	VT_End	TT_Start	TT_Stop
145	7797	1998-01-10	9999-12-31	1998-01-10	1998-01-15
145	7797	1998-01-10	1998-01-15	1998-01-15	9999-12-31
827	7797	1998-01-15	9999-12-31	1998-01-15	1998-01-20
827	7797	1998-01-15	1998-01-20	1998-01-20	9999-12-31
145	*7797*	*1998-01-03*	*1998-01-10*	*1998-01-23*	*9999-12-31*

Code Fragment 10.13 **Eva actually purchased the flat on January 3, with transaction-time splitting.**

```
-- Do normal insert if there are no overlapping rows that
-- do not contain the period of applicability
INSERT INTO Prop_Owner
SELECT 145, 7797, DATE '1998-01-03', DATE '1998-01-10',
    CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM DUAL
WHERE NOT EXISTS (SELECT *
                  FROM Prop_Owner
                      WHERE customer_number = 145
                        AND property_number = 7797
                        AND DATE '1998-01-03' < VT_End
                        AND VT_Begin < DATE '1998-01-10'
                        AND NOT (VT_Begin < DATE '1998-01-03'
                           AND DATE '1998-01-10' < VT_End)
                        AND TT_Stop = DATE '9999-12-31')
-- If there is an overlap, extend it, unless PA is contained in PV
INSERT INTO Prop_Owner
SELECT customer_number, property_number,
  CASE WHEN DATE '1998-01-03 < VT_Begin
    THEN DATE '1998-01-03'
    ELSE VT_Begin END,
  CASE WHEN DATE '1998-01-10 < VT_End
    THEN VT_End
    ELSE DATE '1998-01-10' END,
  CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner
WHERE customer_number = 145
  AND property_number = 7797
  AND DATE '1998-01-03' < VT_End
  AND VT_Begin < DATE '1998-01-10'
  AND NOT (VT_Begin < DATE '1998-01-03'
    AND DATE '1998-01-10' < VT_End)
  AND TT_Stop = DATE '9999-12-31'

UPDATE Prop_Owner
SET TT_Stop = CURRENT_TIMESTAMP
WHERE customer_number = 145
  AND property_number = 7797
  AND DATE '1998-01-03' < VT_End
  AND VT_Begin < DATE '1998-01-10'
  AND NOT (VT_Begin < DATE '1998-01-03'
    AND DATE '1998-01-10' < VT_End)
  AND TT_Stop = DATE '9999-12-31'
```

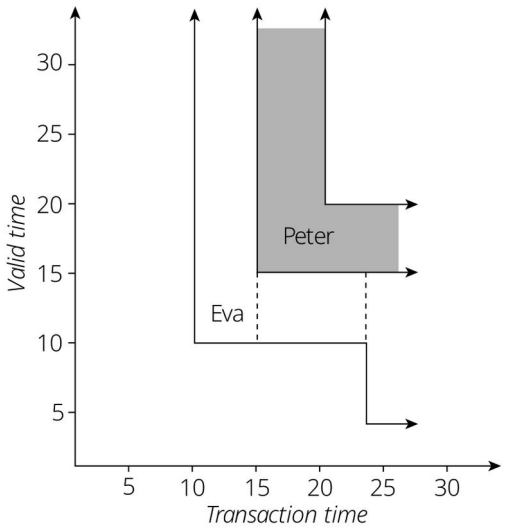


Figure 10.10 An alternate splitting into rectangles.

Table 10.4 Result of a second approach to the sequenced insertion.

customer_ number	property_ number	VT_Begin	VT_End	TT_Start	TT_Stop
145	7797	1998-01-10	9999-12-31	1998-01-10	1998-01-15
145	7797	1998-01-10	1998-01-15	1998-01-15	*1998-01-23*
827	7797	1998-01-15	9999-12-31	1998-01-15	1998-01-20
827	7797	1998-01-15	1998-01-20	1998-01-20	9999-12-31
145	*7797*	*1998-01-03*	*1998-01-15*	*1998-01-23*	*9999-12-31*

Code Fragment 10.14 **Eva actually purchased the flat on January 5 (nontemporal version).**

```
DELETE FROM Prop_Owner
WHERE property_number = 7977
```

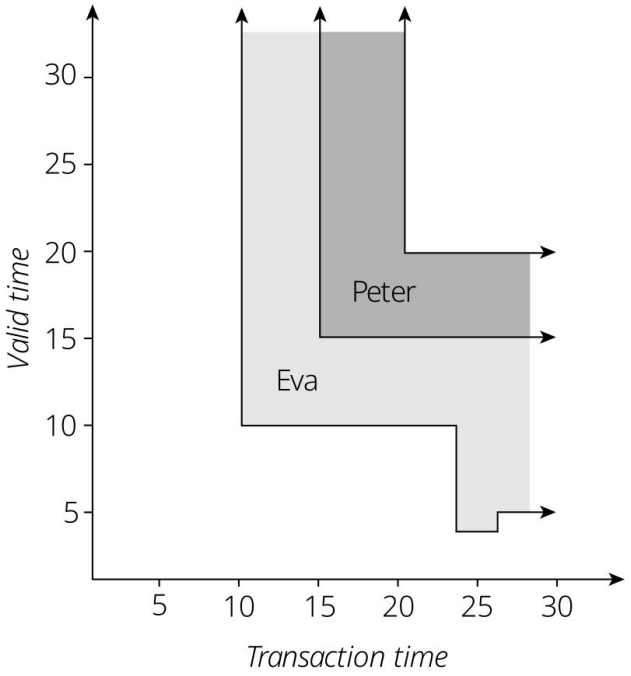


Figure 10.12 A sequenced deletion performed on January 26: Eva actually purchased the flat on January 5.

Code Fragment 10.15 **Eva actually purchased the flat on January 5.**

```
INSERT INTO Prop_Owner
SELECT customer_number, property_number, DATE '1998-01-05', VT_End,
    CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner
WHERE property_number = 7797
  AND VT_Begin < DATE '1998-01-02'
  AND VT_End > DATE '1998-01-05'
  AND TT_Stop = DATE '9999-12-31'

INSERT INTO Prop_Owner
SELECT customer_number, property_number, VT_Begin, DATE '1998-01-02',
    CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner
WHERE property_number = 7797
  AND VT_Begin < DATE '1998-01-02'
  AND VT_End > DATE '1998-01-02'
  AND TT_Stop = DATE '9999-12-31'

UPDATE Prop_Owner
SET TT_Stop = CURRENT_TIMESTAMP
WHERE property_number = 7797
  AND VT_Begin < DATE '1998-01-02'
  AND VT_End > DATE '1998-01-02'
  AND TT_Stop = DATE '9999-12-31'

INSERT INTO Prop_Owner
SELECT customer_number, property_number, DATE '1998-01-05', VT_End,
    CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner
WHERE property_number = 7797
  AND VT_Begin < DATE '1998-01-05'
  AND VT_End >= DATE '1998-01-05'
  AND TT_Stop = DATE '9999-12-31'

UPDATE Prop_Owner
SET TT_Stop = CURRENT_TIMESTAMP
WHERE property_number = 7797
  AND VT_Begin < DATE '1998-01-05'
  AND VT_End >= DATE '1998-01-05'
  AND TT_Stop = DATE '9999-12-31'

UPDATE Prop_Owner
SET TT_Stop = CURRENT_TIMESTAMP
WHERE property_number = 7797
  AND VT_Begin >= DATE '1998-01-02'
  AND VT_End <= DATE '1998-01-05'
  AND TT_Stop = DATE '9999-12-31'
```

Table 10.5 Result of the sequenced deletion.

customer_ number	property_ number	VT_Begin	VT_End	TT_Start	TT_Stop
145	7797	1998-01-10	9999-12-31	1998-01-10	1998-01-15
145	7797	1998-01-10	1998-01-15	1998-01-15	9999-12-31
827	7797	1998-01-15	9999-12-31	1998-01-15	1998-01-20
827	7797	1998-01-15	1998-01-20	1998-01-20	9999-12-31
145	7797	1998-01-03	1998-01-10	1998-01-23	*1998-01-26*
145	*7797*	*1998-01-05*	*1998-01-10*	*1998-01-26*	*9999-12-31*

Code Fragment 10.16 **Peter actually purchased the flat on January 12 (nontemporal version).**

```
UPDATE Prop_Owner
SET customer_number = 145
WHERE  property_number = 7797
  AND customer_number <> 145
```

Table 10.6 Result of the sequenced update.

customer_ number	property_ number	VT_Begin	VT_End	TT_Start	TT_Stop
145	7797	1998-01-10	9999-12-31	1998-01-10	1998-01-15
145	7797	1998-01-10	1998-01-15	1998-01-15	*1998-01-28*
827	7797	1998-01-15	9999-12-31	1998-01-15	1998-01-20
827	7797	1998-01-15	1998-01-20	1998-01-20	*1998-01-28*
145	7797	1998-01-03	1998-01-10	1998-01-23	1998-01-26
145	7797	1998-01-05	1998-01-10	1998-01-26	*1998-01-28*
145	*7797*	*1998-01-05*	*1998-01-12*	*1998-01-28*	*9999-12-31*
827	*7797*	*1998-01-12*	*1998-01-20*	*1998-01-28*	*9999-12-31*

Code Fragment 10.17 **Peter actually purchased the flat on January 12.**

```
INSERT INTO Prop_Owner
SELECT customer_number, property_number, VT_Begin, DATE '1998-01-12',
    CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner
WHERE property_number = 7797 AND customer_number <> 145
  AND VT_Begin < DATE '1998-01-12'
  AND VT_End > DATE '1998-01-12'
  AND TT_Stop = DATE '9999-12-31'

INSERT INTO Prop_Owner
SELECT customer_number, property_number, DATE '1998-01-15', VT_End,
    CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner
WHERE property_number = 7797 AND customer_number <> 145
  AND VT_Begin < DATE '1998-01-15'
  AND VT_End > DATE '1998-01-15'
  AND TT_Stop = DATE '9999-12-31'

INSERT INTO Prop_Owner
SELECT 145, property_number, VT_Begin, VT_End,
    CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner
WHERE property_number = 7797 AND customer_number <> 145
  AND VT_Begin < DATE '1998-01-15'
  AND VT_End > DATE '1998-01-12'
  AND TT_Stop = DATE '9999-12-31'

UPDATE Prop_Owner
SET TT_Stop = CURRENT_TIMESTAMP
WHERE property_number = 7797 AND customer_number <> 145
  AND VT_Begin < DATE '1998'01-15'
  AND VT_End > DATE '1998-01-12'
  AND TT_Stop = DATE '9999-12-31'
```

```
INSERT INTO Prop_Owner
SELECT customer_number, property_number, DATE '1998-01-12', VT_End,
  CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner
WHERE property_number = 7797 AND customer_number <> 145
  AND VT_Begin < DATE '1998-01-12'
  AND VT_End > DATE '1998-01-12'
  AND TT_Stop = DATE '9999-12-31'

UPDATE Prop_Owner
SET TT_Stop = CURRENT_TIMESTAMP
WHERE property_number = 7797 AND customer_number <> 145
  AND VT_Begin < DATE '1998-01-12'
  AND VT_End > DATE '1998-01-12'
  AND TT_Stop = DATE '9999-12-31'

INSERT INTO Prop_Owner
SELECT customer_number, property_number, VT_Begin, DATE '1998-01-15',
  CURRENT_TIMESTAMP, DATE '9999-12-31'
FROM Prop_Owner
WHERE property_number = 7797 AND customer_number <> 145
  AND VT_Begin < DATE '1998-01-15'
  AND VT_End > DATE '1998-01-15'
  AND TT_Stop = DATE '9999-12-31'

UPDATE Prop_Owner
SET TT_Stop = CURRENT_TIMESTAMP
WHERE property_number = 7797 AND customer_number <> 145
  AND VT_Begin < DATE '1998-01-15'
  AND VT_End > DATE '1998-01-15'
  AND TT_Stop = DATE '9999-12-31'
```

Code Fragment 10.18 **Delete all records with a valid-time duration of exactly one week.**

```
UPDATE Prop_Owner
SET TT_Stop = CURRENT_TIMESTAMP
WHERE (VT_End - VT_Begin DAY) = INTERVAL '7' DAY
  AND TT_Stop = DATE '9999-12-31'
```

Table 10.7 After a nonsequenced deletion.

customer_ number	property_ number	VT_Begin	VT_End	TT_Start	TT_Stop
145	7797	1998-01-10	9999-12-31	1998-01-10	1998-01-15
145	7797	1998-01-10	1998-01-15	1998-01-15	1998-01-28
827	7797	1998-01-15	9999-12-31	1998-01-15	1998-01-20
827	7797	1998-01-15	1998-01-20	1998-01-20	1998-01-28
145	7797	1998-01-03	1998-01-10	1998-01-23	1998-01-26
145	7797	1998-01-05	1998-01-10	1998-01-26	1998-01-28
145	7797	1998-01-05	1998-01-12	1998-01-28	*1998-01-30*
827	*7797*	*1998-01-12*	*1998-01-20*	*1998-01-28*	*9999-12-31*

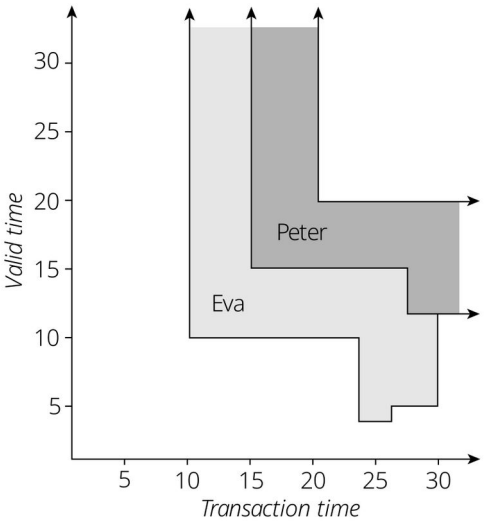
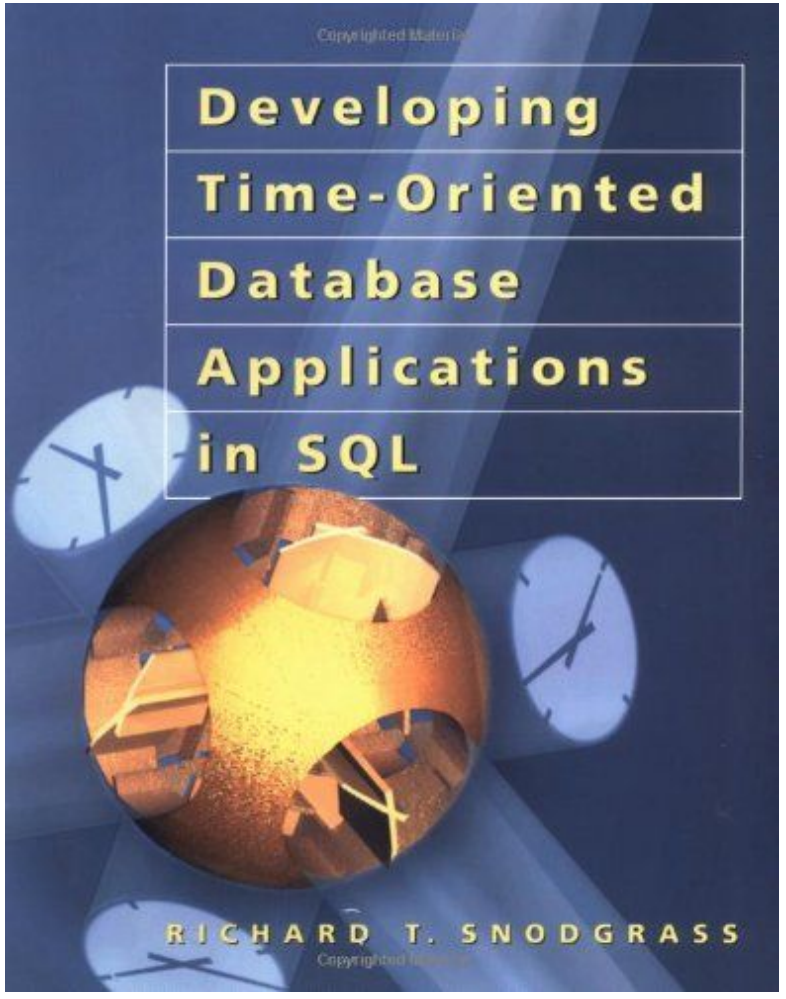
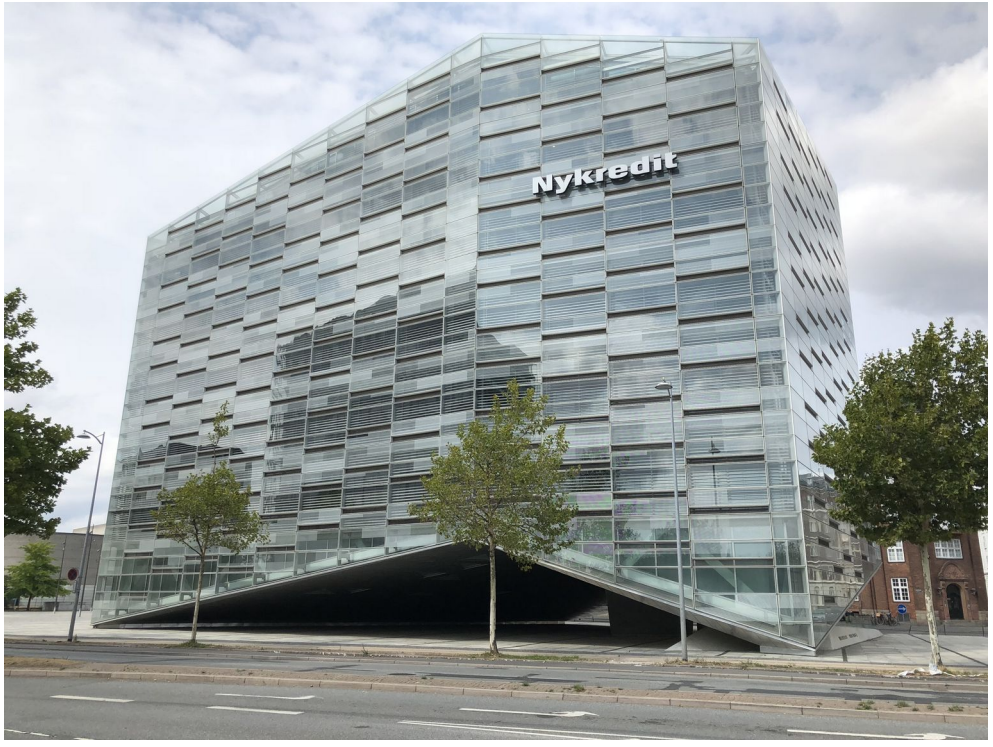


Figure 10.14 A nonsequenced deletion performed on January 30: Delete all records of exactly one-week duration.



10.3 Queries



Available freely online: https://www2.cs.arizona.edu/~rts/tdbbook.pdf

Code Fragment 10.19 **Give the history of owners of the flat at Skovvej 30 in Aalborg as of January 1, 1998.**

```
SELECT customer_number, VT_Begin, VT_End
FROM Prop_Owner
WHERE property_number = 7797
  AND TT_Start <= DATE '1998-01-01'
  AND DATE '1998-01-01' < TT_Stop
```

Taking a transaction time-slice as of January 14 results in a history with one entry:

customer_number	VT_Begin	VT_End
145	1998-01-10	9999-12-31

The time-slice as of January 18 tells a different story:

customer_number	VT_Begin	VT_End
145	1998-01-10	1998-01-15
827	1998-01-15	9999-12-31

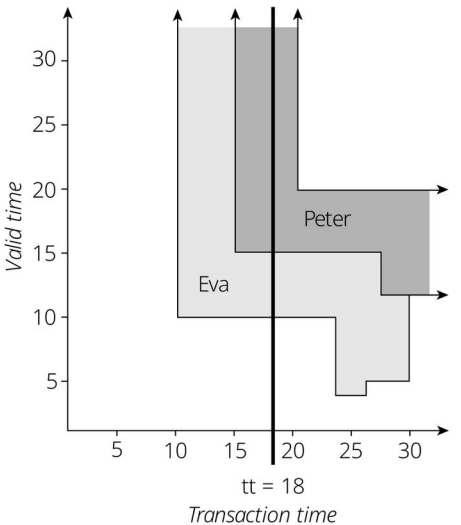


Figure 10.15 A transaction time-slice as of January 18.

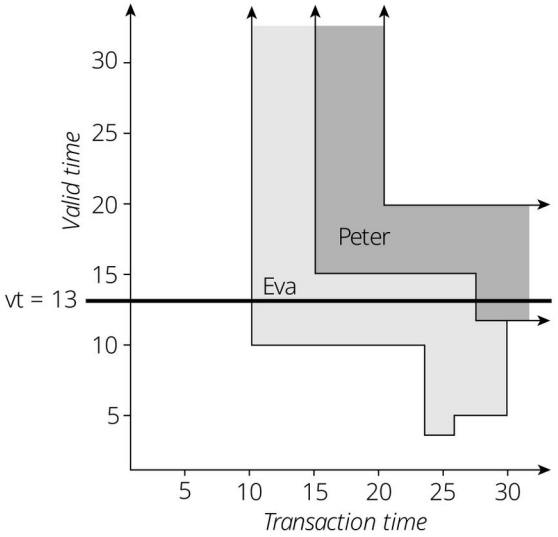


Figure 10.16 A valid time-slice on January 13.

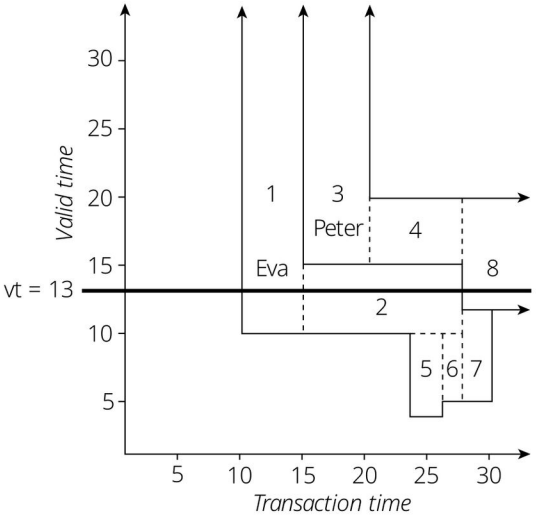


Figure 10.17 The underlying rectangles encoding the bitemporal regions.

Table 10.8 The valid time-slice on January 13.

customer_number	TT_Start	TT_Stop
145	1998-01-10	1998-01-15
145	1998-01-15	1998-01-28
827	1998-01-28	9999-12-31

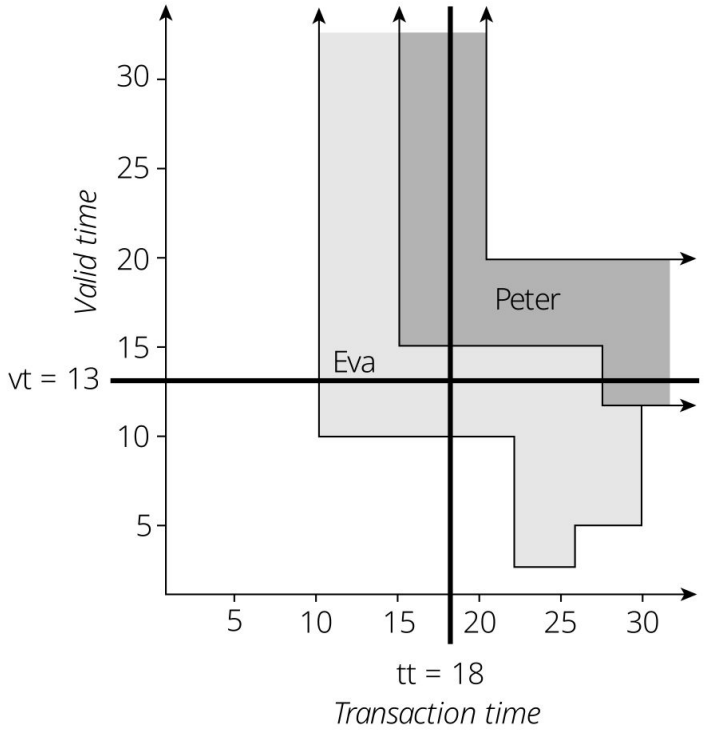


Figure 10.18 A bitemporal time-slice on a valid time of January 13 and as of a transaction time of January 18.

Summary: storing bitemporal data 101

- 2, 3 or 4 timestamps
- “No overlaps allowed” primary key constraint
- Performance, complexity and functionality considerations
- 4 timestamps is best (e.g. SQL:2011), complexity of updates/maintenance is harder, but read performance is good
- 2 timestamps requires lots of `MAX` and `<` ...whereas non-overlapping rectangles are easy to query

Technical challenges with the ad hoc approach:

- Do I need to store deltas?
- How to handle duplicate Primary Key values?
- How can I control the query performance?
- What does this mean for triggers?
- Code & query complexity!
- ...

Key tradeoff dimensions:

- Write latency & read latency
- Cost of storage and indexing
- What granularity of history is required, e.g.
  - Need profitability only as of end of month, only need to what it was daily
  - Do you need a complete audit trail of changes
- Duration of history (retention periods and policies)
- Easy of query development
- Ability to support new queries quickly or effectively
- Over-engineering or under-engineering can be expensive (cost and business opportunities)

SQL:2011

- A culmination of decades-long academic & industry reflection
- Extends the ISO standard with temporal operators
- Demanded by industry
- Partially implemented by a few DBMS vendors
- *Still* demanded by industry

Temporal features in SQL:2011

Krishna Kulkarni, Jan-Eike Michels (IBM Corporation)
{krishnak, janeike}@us.ibm.com

ABSTRACT

SQL:2011 was published in December of 2011, replacing SQL:2008 as the most recent revision of the SQL standard. This paper covers the most important new functionality that is part of SQL:2011: the ability to create and manipulate temporal tables.

1. Introduction

SQL is the predominant database query language standard published jointly by ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission). In December 2011, ISO/IEC published the latest edition of the SQL standard, SQL:2011. A recent article in *SIGMOD Record* provides a brief survey of the new features in SQL:2011 [1]. Because of space constraints, it did not cover the most important new feature in SQL:2011: the ability to create and manipulate temporal tables, i.e., tables whose rows are associated with one or more temporal periods. This is the subject of the current article.

2. Temporal data support

implement temporal support as part of the application logic, which often resulted in expensive development cycles and complex, hard-to-maintain code.

In 1995, the ISO SQL committee initiated a project to create a new part of SQL standard devoted to the language extensions for the temporal data support. A set of language extensions based on (but not identical to) TSQL2 [8] were submitted for standardization at that time. Unfortunately, these proposals generated considerable controversy (see [9] for more details), and failed to get adequate support from the ISO SQL committee's membership. In addition, there was no indication that key DBMS vendors were planning to implement these extensions in their products. Eventually, the work on this new part was cancelled in 2001.

Recently, a new set of language extensions for temporal data support were submitted to and accepted by the ISO SQL committee. These language extensions are now part of SQL:2011 Part 2, SQL/Foundation [10], instead of appearing as a new part. There is currently at least one commercial implementation [5] based on these extensions that the authors are aware of.

2.1 Periods

The cornerstone of temporal data support in SQL:2011

Bitemporal Tables

```
CREATE TABLE Emp(
  ENo INTEGER,
  EStart DATE,
  EEnd DATE,
  EDept INTEGER,
  PERIOD FOR EPeriod (EStart, EEnd),
  Sys_start TIMESTAMP(12) GENERATED
      ALWAYS AS ROW START,
  Sys_end TIMESTAMP(12) GENERATED
      ALWAYS AS ROW END,
  EName VARCHAR(30),
  PERIOD FOR SYSTEM_TIME
      (Sys_start, Sys_end),
  PRIMARY KEY (ENo,
      EPeriod WITHOUT OVERLAPS),
  FOREIGN KEY
      (Edept, PERIOD EPeriod)
   REFERENCES Dept
      (DNo, PERIOD DPeriod)
) WITH SYSTEM VERSIONING
```

Automatic, Auditable, Mutable History of Data

Eno	EStart	EEnd	EDept
22217	2010-01-01	2011-11-12	3

```
UPDATE Emp
 FOR PORTION OF EPeriod
   FROM DATE '2011-02-03'
   TO DATE '2011-09-10'
SET EDept = 4
WHERE ENo = 22217
```

ENo	EStart	EEnd	EDept
22217	2010-01-01	2011-02-03	3
22217	2011-02-03	2011-09-10	4
22217	2011-09-10	2011-11-12	3

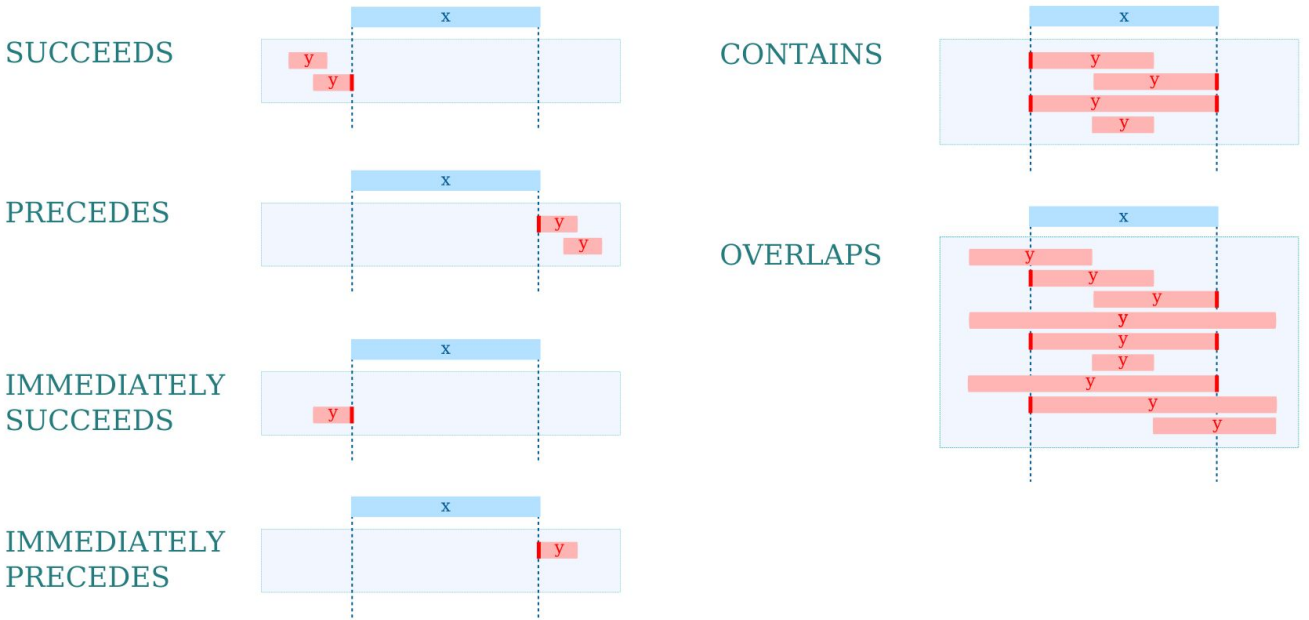
Expressive Querying

What was the department where the employee 22217 worked as of December 1, 2010, recorded in the database as of July 1, 2011?

```
SELECT ENo, EDept
FROM Emp FOR SYSTEM_TIME AS OF
   TIMESTAMP '2011-07-01 00:00:00'
WHERE ENo = 22217 AND
 EPeriod CONTAINS DATE '2010-12-01'
```

Period Predicates

Express conditions involving periods...



Give the owner of the flat at Skovvej 30 in Aalborg on January 13 as stored in the Prop_Owner table on January 18.

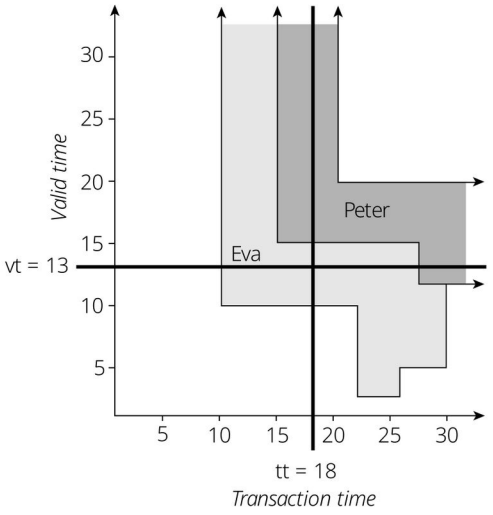


Figure 10.18 A bitemporal time-slice on a valid time of January 13 and as of a transaction time of January 18.

```
SELECT customer_number
FROM Prop_Owner
WHERE property_number = 7797 AND
  VT_Begin <= DATE 1998-01-13 AND
  DATE 1998-01-13 < VT_End AND
  TT_Start <= DATE 1998-01-18 AND
  DATE 1998-01-18 < TT_Stop
```

```
SELECT customer_number
FROM Prop_Owner FOR SYSTEM_TIME AS OF
   DATE '1999-01-18'
WHERE property_number = 7797 AND
   VT_period CONTAINS DATE '1999-01-13';
```

Bitemp 2.0 - The hype, the barriers and the actions to take

1,430 views • 9 Jul 2011

3 DISLIKE SHARE DOWNLOAD CLIP SAVE ...

https://www.youtube.com/watch?v=R7kQO5Z2JKk

STAC
SECURITIES TECHNOLOGY ANALYSIS CENTER

About STAC Research Research Domains

Home » Working groups

STAC Bi-Temporal Data SIG

Bi-temporal data (or BTD) are data for which changes are recorded over two independent dimensions of time. These dimensions are referred to as "valid time" and "transaction time," where valid time denotes the period during which a fact is true with respect to the real world and transaction time denotes the period during which a fact is stored in a database. Trading firms use BTD in applications ranging from risk management and back-testing of trading strategies to P&L Explain and regulatory reporting.

For more background on BTD, see the slides and video from presentations given by Bank of America Merrill Lynch and Bitemporaldata.com at the June STAC Analytics Technology Conference on current issues in bi-temporal data.

Background on the STAC BTD SIG

The STAC Bi-Temporal Data Special Interest Group (BTD SIG) is a group of end-user organizations and vendors that will meet periodically to discuss key issues related to BTD.

See this FAQ for eligibility requirements and other details.

The initial kickoff telecon was held on October 7, 2011. Slides and replay of the call are here.

The first in-person meetings were held in New York (Nov 2, 2011) and London (Nov 9, 20011). These were end-user only meetings.

https://www.stacresearch.com/btd

SQL:2011 has yet to become commonplace - why?

- For database vendors:
  - Complex to retrofit
  - Architecture mismatch
- For developers:
  - Lack of awareness & foresight
  - Complex performance

Temporal Databases Annotated Bibliography

2017-12-05

I've been reading about temporal databases for a few years now, so I think it's time I share my bibliography and notes. This is presented in "narrative order", so that you can get a sense of how the research has developed. This article somewhat overlaps a mini literature review I wrote on the Postgres hackers mailing list, but this article is more complete and in a place where I can keep it updated.

Temporal databases let you track the history of things over time: both the history of changes to the database (e.g. for auditing) and the history of the thing itself. They are not the same thing as time-series databases: whereas a time-series database has time-stamped *events*, a temporal database stores the history of *things*, typically by adding a start/end time to each row (so two timestamps, not one). With time-series the challenge is typically scale; with temporal the challenge is with complexity and correctness.

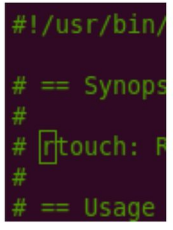
Research

Snodgrass, Richard T. *Developing Time-Oriented Database Applications in SQL*. 1999. The seminal work on temporal databases and still the most useful introduction I know. Covers the "combinatorial explosion" of non-temporal/state-temporal/system-temporal/bi-temporal tables, current/sequenced/non-sequenced queries, `SELECT` / `INSERT` / `UPDATE` / `DELETE` , different RDBMS

Paul A. Jungwirth

Blog
Email
Portfolio
Resume
Github
Stack Overflow

Code



Skill Spy
Alien Words
ElectNext
db_leftovers gem
Tech Notes
. . . more

Writing

Survey of SQL:2011 Temporal Features
Drawing Redux Form FieldArrays with Pug
Validating FieldArrays in Redux Form

extra pseudo-column metadata. I hope implementers will take their advice seriously and not build temporal features on such a distorting idea.

Something I disagreed with was their suggestion to use tables in sixth-normal form—basically every column gets its own table—since attributes can have different lifespans. I can see how that is purer, but it seems like just too much complexity. They probably suspected the same because they always show how to do things with either that approach or tables in a more traditional third-normal form (or BCNF if you prefer). Even that is slightly distorted in order to avoid `NULL`s, but you can easily look past that.

Finally, I appreciated that on page 282 they mention DDL on temporal databases. Like everyone they say it's beyond the scope of their current discussion, but it's a penetrating insight to say, "the database catalog might itself need to be treated as a temporal database."

SQL:2011 Draft standard. (pdf) Personally I find the standard pretty disappointing. It uses separate start/end columns instead of built-in range types, although range types offer benefits like exclusion constraints and convenient operators for things like "overlaps" that are verbose to code correctly by hand. It only mentions inner joins, not the various outer joins, semi-joins (`EXISTS`), anti-joins (`NOT EXISTS`), or aggregates. Many of its features apply only to system-time, not application-time, even though applicaion-time is the more interesting and less-available feature. (There are lots of auditing add-ons, but almost nothing for tracking the history of things.) The syntax seems too specific, lacking appropriate generality. A lot of these drawbacks seem motivated by a goal that goes back to

Survey of SQL:2011 Temporal Features

2019-09-04

Introduction

This blog post is a survey of SQL:2011 Temporal features from MariaDB, IBM DB2, Oracle, and MS SQL Server. I'm working on adding temporal features to Postgres, so I wanted to see how other systems interpret the standard.

If you're new to temporal databases, you also might enjoy this talk I gave at PGCon 2019.

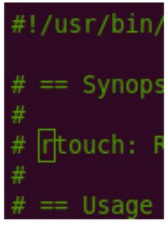
In this post I cover both application-time (aka valid-time) and system-time, but I focus more on valid-time. Valid-time tracks the history of the thing "out there", e.g. when a house was remodeled, when an employee got a raise, etc. System-time tracks the history of when you changed the database. In general system-time is more widely available, both as native SQL:2011 features and as extensions/plugins/etc., but is less interesting. It is great for compliance/auditing, but you're unlikely to build application-level features on it. Also since it's generated automatically you don't need special DML commands for it, and it is less important to protect yourself with temporal primary and foreign keys.

At this point all the major systems I survey have *some* temporal support, although none of them support it completely. On top of that the standard itself is quite modest, although in some ways it

Paul A. Jungwirth

Blog
Email
Portfolio
Resume
Github
Stack Overflow

Code



Skill Spy
Alien Words
ElectNext
db_leftovers gem
Tech Notes
. . . more

Writing

Survey of SQL:2011 Temporal Features
Drawing Redux Form FieldArrays with Pug
Validating FieldArrays in Redux Form
Testing Your

MS SQL Server

I tested an evaluation copy of MS SQL Server 2017 (version `14.0.1000.169, RTM`).

SQL Server doesn't support application-time periods at all, just system-time.

MariaDB

MySQL doesn't support any temporal features, but recent versions of MariaDB have started to add support. Version 10.3.4 (released Jan 2018) included system-time support; Version 10.4.3 (Feb 2019), valid-time.

Oracle

For my tests I used Oracle 19c (version 19.3) for Linux and ran it on CentOS 7.

System time

Oracle has its own way of tracking table history, so it doesn't bother with SQL:2011 system-time.

Application time

Oracle lets you declare a `PERIOD`, but like MariaDb you can't define a temporal primary key:

IBM DB2

DB2 has the fullest temporal support of all the databases I examined. My tests used version 11.5.0.0 on Linux.

https://mariadb.org/wp-content/uploads/2020/09/Temporal-Tables_serverfest2020.pdf
https://www.youtube.com/watch?v=uBoUlTsU1Tk

MarkLogic

BETTER WITH BITEMPORAL

MARKLOGIC WHITE PAPER • JUNE 2015

In our age of billion-dollar regulatory fines and time-consuming, costly litigation, a database must hold up as the main system of record. Unfortunately, traditional databases do not keep a complete history of the past. Only with a bitemporal database can you truly maintain a complete and accurate picture of the past, understanding exactly "what you knew" and "when you knew it."

ASSESSMENT: DO YOU NEED BITEMPORAL?

Before you go any further, it is probably helpful to first ask whether you might need bitemporal data management in your organization. If you answer "yes" to any of the following questions, then bitemporal is a solution that you should consider.

	YES	NO
1. Is tracking when events or transactions occur critical to your business?	✓	
2. Are there ever cases when historical data needs to be updated?	✓	
3. Do you run into circumstances in which there is a lag between when something happened in the real world, and when it was recorded in the database?	✓	
4. Do you get frequent requests from regulators to review historical data?	✓	
5. Do you work in an industry in which the sequence of when you learn about certain information is significant, such as in law and intelligence?	✓	
6. Is the cost and complexity of storing and accessing historical data in your organization overwhelming?	✓	
7. Does managing and accessing historical data cost significant developer resources, or carry increasing risk over time?	✓	

https://www.marklogic.com/wp-content/uploads/2015/07/MarkLogic-White-Paper-Better-With-Bitemporal.pdf

A Case Study of Temporal Data

By Richard T. Snodgrass,
Professor of Computer Science,
University of Arizona

TERADATA
Raising Intelligence

Executive Overview

Information is the key asset of many companies – and for most, this asset contains time-referenced data. It can be frustrating that standard SQL has so few language facilities for such data. Fortunately, Teradata® Database 13.10 has many new features in its SQL designed specifically for temporal support.

This case study will examine how the new SQL language features in Teradata Database naturally reflect the expression in English of modifications and how these features greatly reduce the length and complexity of such modifications in conventional SQL. The result is that developers can now much more easily convert their applications to support time-varying data and create new applications that exploit stored data about the past for deeper insight into the future.

TERADATA
Raising Intelligence

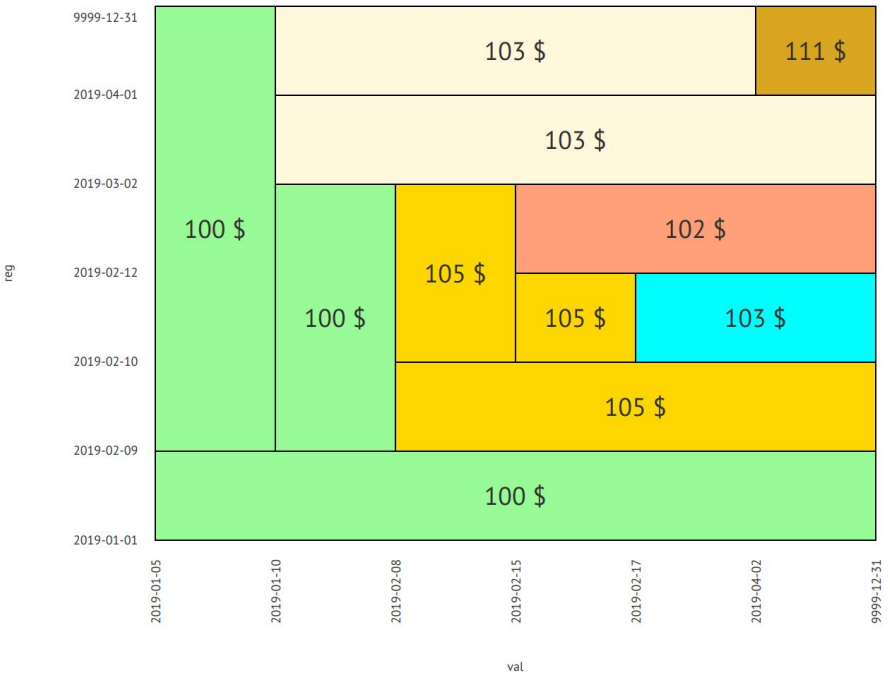
http://rts.cs.arizona.edu/pubs/Teradatacasestudy.pdf

Postgres...?

- Library of bitemporal functions using GiST indexes:
  - https://github.com/hettie-d/pg_bitemporal/
- system_time only extensions:
  - https://github.com/nearform/temporal_tables
  - https://github.com/arkhipov/temporal_tables

“How can I generate diagrams like this quickly?”

product_id	registered_from	registered_to	valid_from	valid_to	price
1002	2019-01-01	2019-02-09	2019-01-05	9999-12-31	100
1002	2019-02-09	9999-12-31	2019-01-05	2019-01-10	100
1002	2019-02-09	2019-03-02	2019-01-10	2019-02-08	100
1002	2019-02-09	2019-02-10	2019-02-08	9999-12-31	105
1002	2019-02-10	2019-03-02	2019-02-08	2019-02-15	105
1002	2019-02-10	2019-02-12	2019-02-15	2019-02-17	105
1002	2019-02-10	2019-02-12	2019-02-17	9999-12-31	103
1002	2019-02-12	2019-03-02	2019-02-15	9999-12-31	102
1002	2019-03-02	2019-04-01	2019-01-10	9999-12-31	103
1002	2019-04-01	9999-12-31	2019-01-10	2019-04-02	103
1002	2019-04-01	9999-12-31	2019-04-02	9999-12-31	111



https://bitemporal.net/generate-bitemporal-intervals/

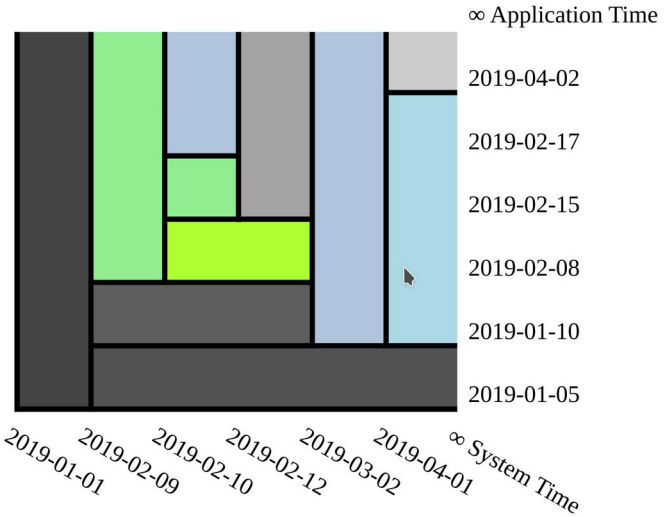
Bitemporal Visualizer

Show Textarea | Full timestamps | Hide About

Price (and tax) scheduling data example
Source: https://bitemporal.net/generate-bitemporal-intervals/

id	price	sys_start	sys_end	app_start	app_end
1002	100	2019-01-01	2019-02-09	2019-01-05	9999-12-31
1002	100	2019-02-09	9999-12-31	2019-01-05	2019-01-10
1002	100	2019-02-09	2019-03-02	2019-01-10	2019-02-08
1002	105	2019-02-09	2019-02-10	2019-02-08	9999-12-31
1002	105	2019-02-10	2019-03-02	2019-02-08	2019-02-15
1002	105	2019-02-10	2019-02-12	2019-02-15	2019-02-17
1002	103	2019-02-10	2019-02-12	2019-02-17	9999-12-31
1002	102	2019-02-12	2019-03-02	2019-02-15	9999-12-31
1002	103	2019-03-02	2019-04-01	2019-01-10	9999-12-31
1002	103	2019-04-01	9999-12-31	2019-01-10	2019-04-02
1002	111	2019-04-01	9999-12-31	2019-04-02	9999-12-31

11 Rows



Created for XTDB using Scittle

Note that the tool doesn't detect or prevent overlapping (or otherwise invalid) regions

This tool was originally created by the XTDB team but is open to ideas and feature requests (and contributions!) that might be useful with other databases also. Please feel free to open issues.

https://bitemporal-visualizer.github.io/

jdt@juxt.pro
@refset

jms@juxt.pro
@jarohen

Thank you 🙏

Any questions?

We hope you have enjoyed this presentation!
To learn more, see https://xtdb.com/v2 to discover JUXT's open source bitemporal database.
XTDB makes the SQL:2011 bitemporal table experience *easy*.

JUXT