Accelerated As-Of Reporting, *Without The Hassle*

OSinFinance Webinar Series, December 2024

FINOS

Jeremy Taylor | JUXT | @refset

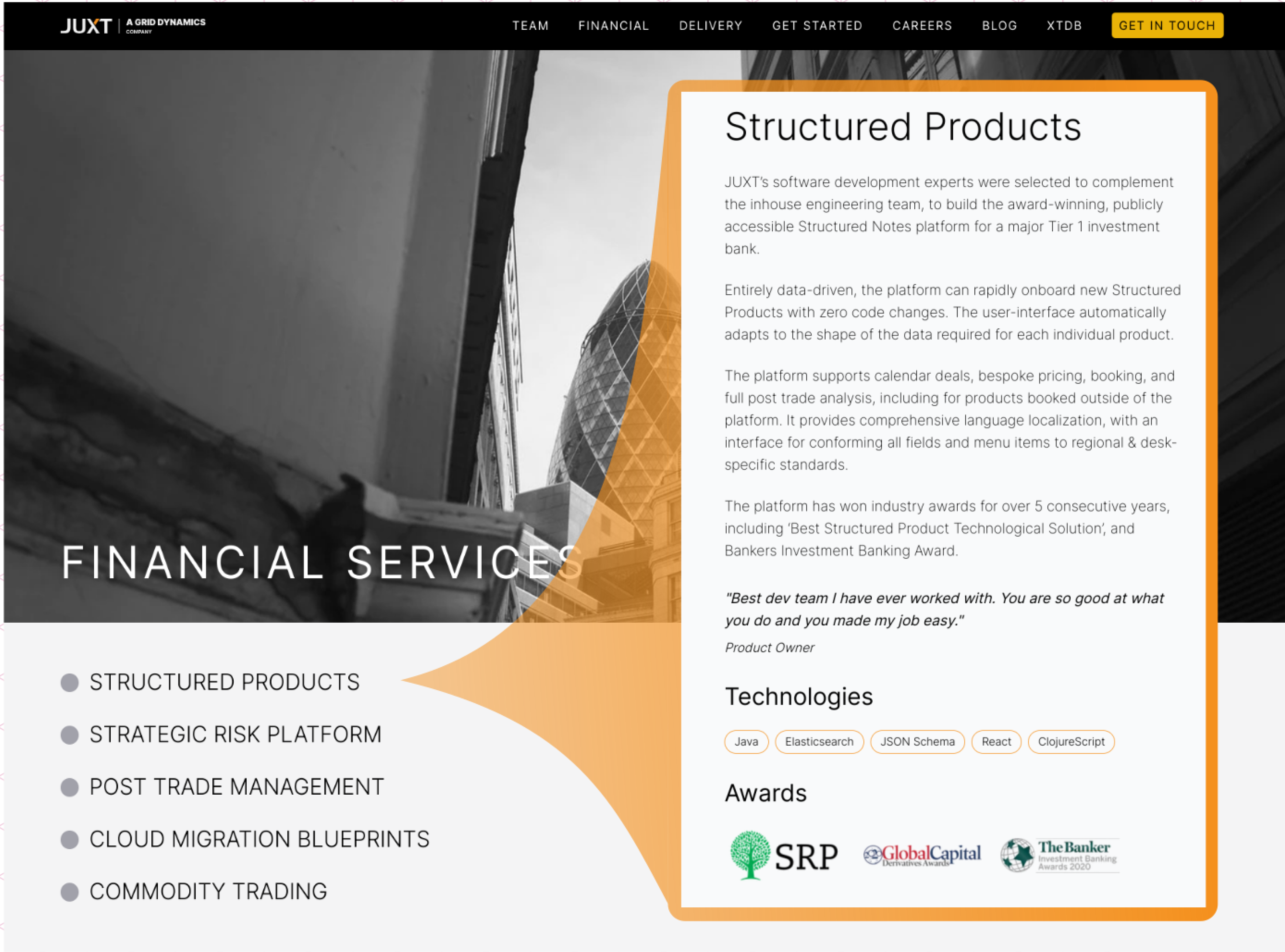
JUXT

A GRID DYNAMICS COMPANY

JUXT

A GRID DYNAMICS COMPANY

Delivering software engineering consultancy for finanical services and beyond, since 2013.



FINANCIAL SERVICES

- STRUCTURED PRODUCTS
- STRATEGIC RISK PLATFORM
- POST TRADE MANAGEMENT
- CLOUD MIGRATION BLUEPRINTS
- COMMODITY TRADING

Structured Products

JUXT's software development experts were selected to complement the inhouse engineering team, to build the award-winning, publicly accessible Structured Notes platform for a major Tier 1 investment bank.

Entirely data-driven, the platform can rapidly onboard new Structured Products with zero code changes. The user-interface automatically adapts to the shape of the data required for each individual product.

The platform supports calendar deals, bespoke pricing, booking, and full post trade analysis, including for products booked outside of the platform. It provides comprehensive language localization, with an interface for conforming all fields and menu items to regional & desk-specific standards.

The platform has won industry awards for over 5 consecutive years, including 'Best Structured Product Technological Solution', and Bankers Investment Banking Award.

"Best dev team I have ever worked with. You are so good at what you do and you made my job easy."

Product Owner

Technologies

Java · Elasticsearch · JSON Schema · React · ClojureScript

Awards

SRP
GlobalCapital Derivatives Awards
The Banker Investment Banking Awards 2020

https://juxt.pro

Common problems observed

- ✗ High costs to deliver timely auditability and reproducibility across bespoke and off-the-shelf software
- ✗ Non-trivial ETL/ELT work required to provide basic reporting and analytics over line-of-business applications
- ✗ Inflexible SQL databases struggle to accommodate complex domain models with evolving schemas

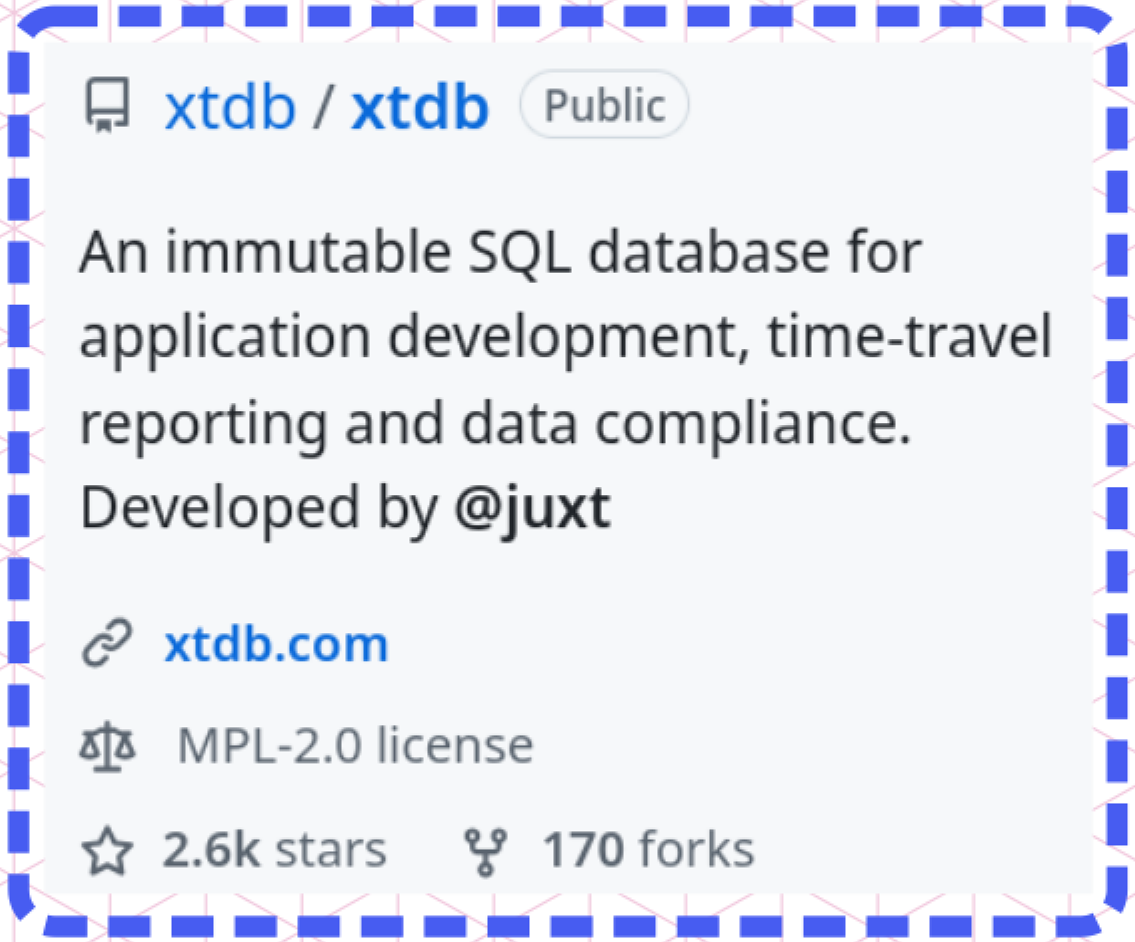
Common workarounds for these problems

- ✗ Excessive logging - not easy to retrieve or analyze
- ✗ Database snapshots - costly, infrequent and delayed
- ✗ Change Data Capture - excessive duplication of schema and queries in a dedicated analytics system
- ✗ Custom in-database versioning - complex schema designs risk silent data loss and reduce agility

Common solutions delivered, using XTDB

(fintech and beyond!)

BACHELIER TECHNOLOGY



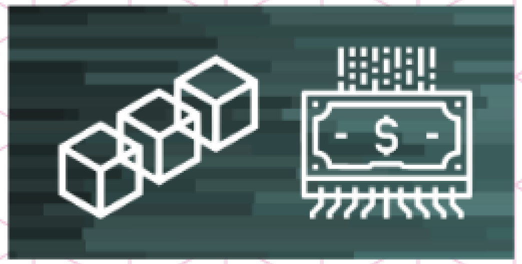
xtdb / **xtdb** Public

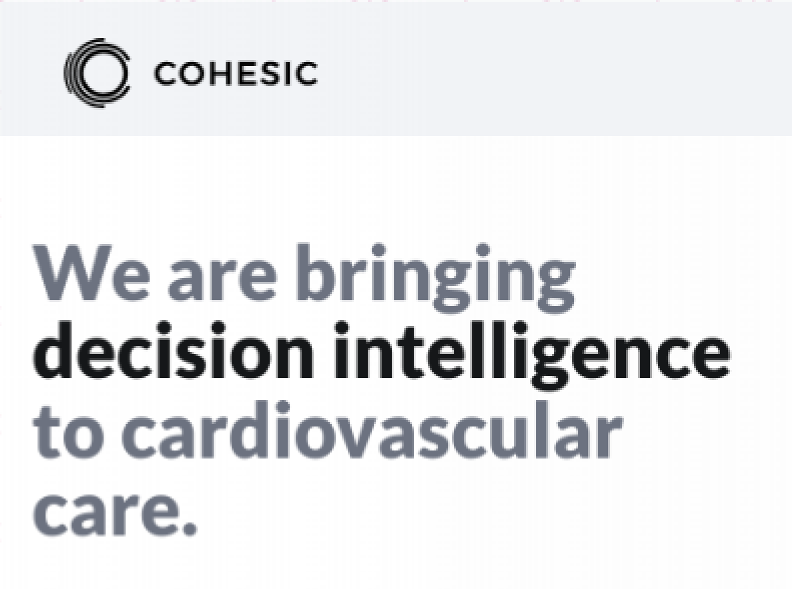
An immutable SQL database for application development, time-travel reporting and data compliance. Developed by **@juxt**

xtdb.com

MPL-2.0 license

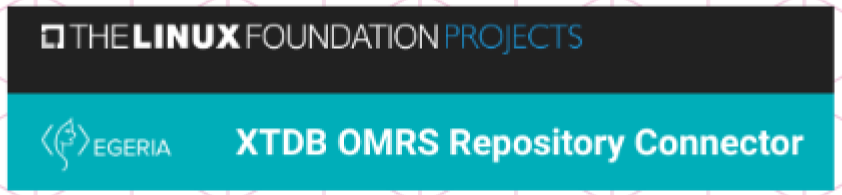
2.6k stars **170** forks





COHESIC

We are bringing **decision intelligence** to cardiovascular care.



THE **LINUX** FOUNDATION PROJECTS

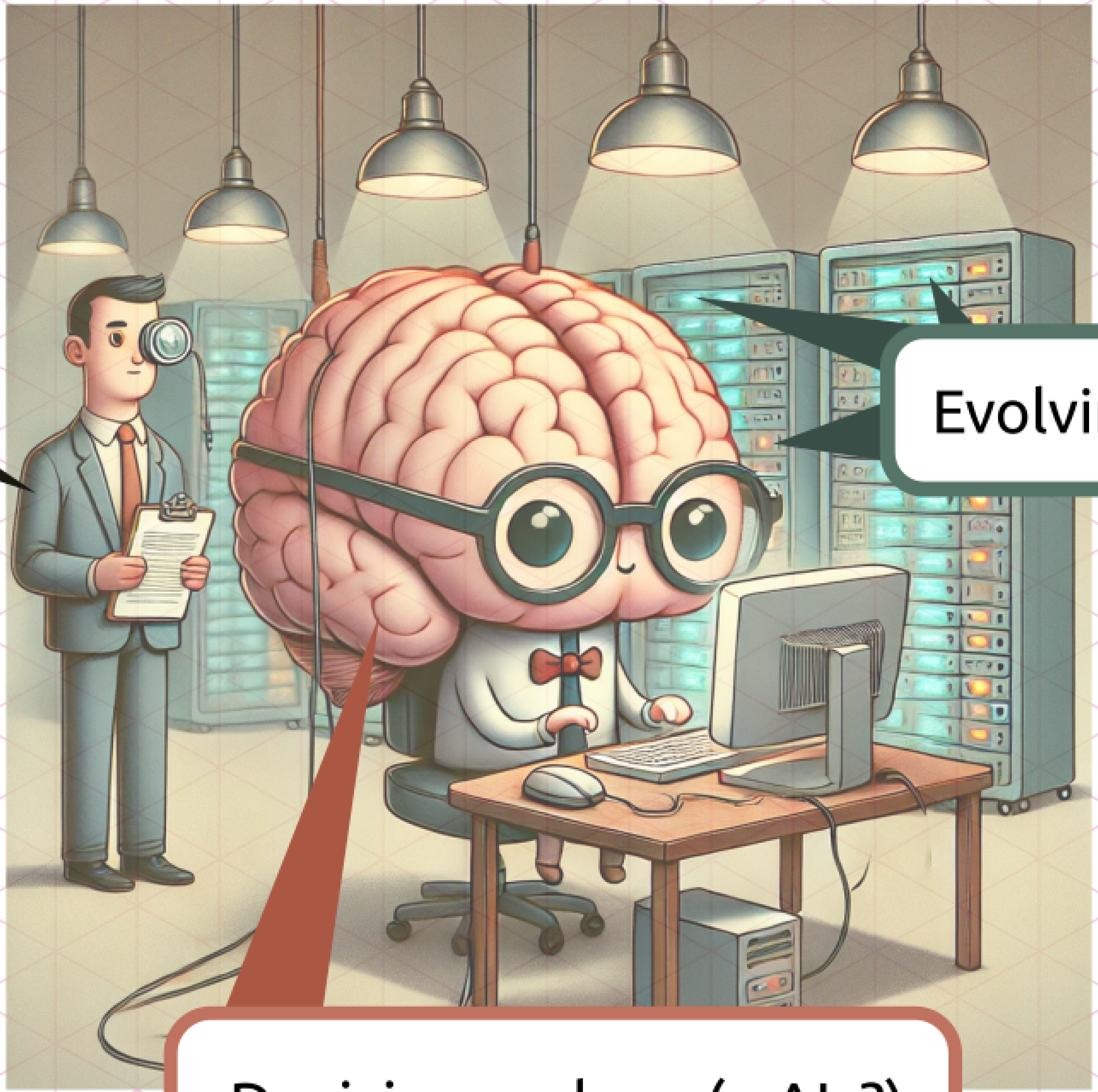
EGERIA **XTDB OMRS Repository Connector**

Rijksoverheid

OpenKat - Vulnerability Analysis Tool

2019 **Now**

Why does As-Of Reporting matter?



Managers, auditors, regulators

Evolving data from multiple sources

Decision makers (...AIs?)

- ✓ Auditability (i.e. tracing decisions to data)
- ✓ Proving value (e.g. best execution)
- ✓ Accurate insights into business performance

JUXT | A GRID DYNAMICS COMPANY

TEAM FINANCIAL DELIVERY GET STARTED CAREERS BLOG XTDB GET IN TOUCH

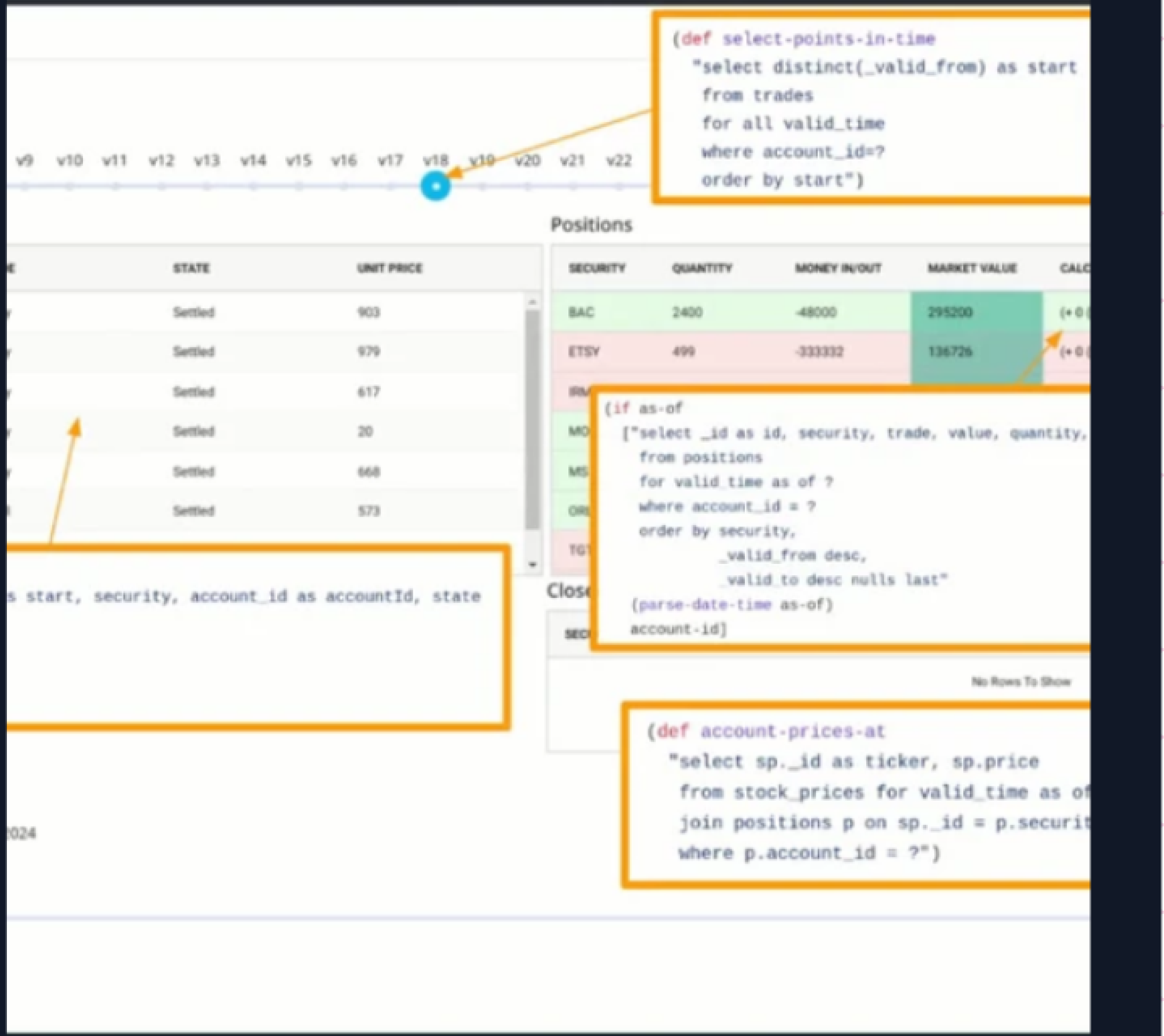
DATABASE

DEC 09, 2024

Bitemporal TraderX - augmenting a sample trading system

Adding XTDB and Clojure into a sample microservices trading application for the FINOS 2024 Tech Sprint

Michal Pisanko
Software Engineer



```
(def select-points-in-time
  "select distinct(_valid_from) as start
   from trades
   for all valid_time
   where account_id=?
   order by start")
```

STATE	UNIT PRICE
Settled	903
Settled	979
Settled	617
Settled	20
Settled	668
Settled	573

SECURITY	QUANTITY	MONEY IN/OUT	MARKET VALUE	CALC
BAC	2400	-48000	295200	(+ 0 (
ETSY	499	-333332	136726	(+ 0 (

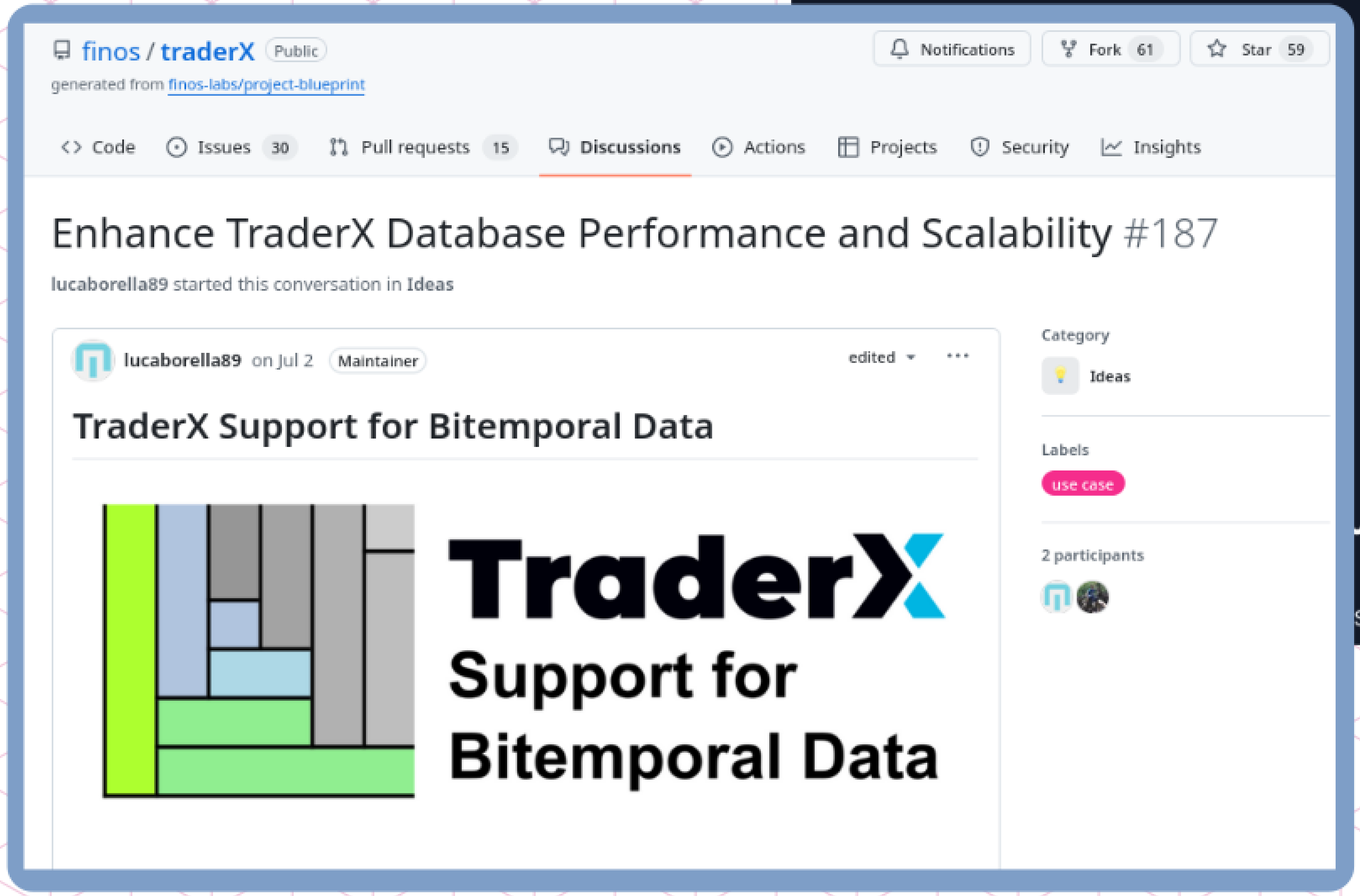
```
s start, security, account_id as accountId, state
```

```
(if as-of
  ["select _id as id, security, trade, value, quantity,
    from positions
    for valid_time as of ?
    where account_id = ?
    order by security,
             _valid_from desc,
             _valid_to desc nulls last"
   (parse-date-time as-of)
   account-id]
```

```
(def account-prices-at
  "select sp._id as ticker, sp.price
   from stock_prices for valid_time as of
   join positions p on sp._id = p.securit
   where p.account_id = ?")
```

Joining FINOS: The Fintech Open Source Foundation

se you missed the news, JUXT is now a member of FINOS! The Fintech Open Source Foundation



Enhance TraderX Database Performance and Scalability #187

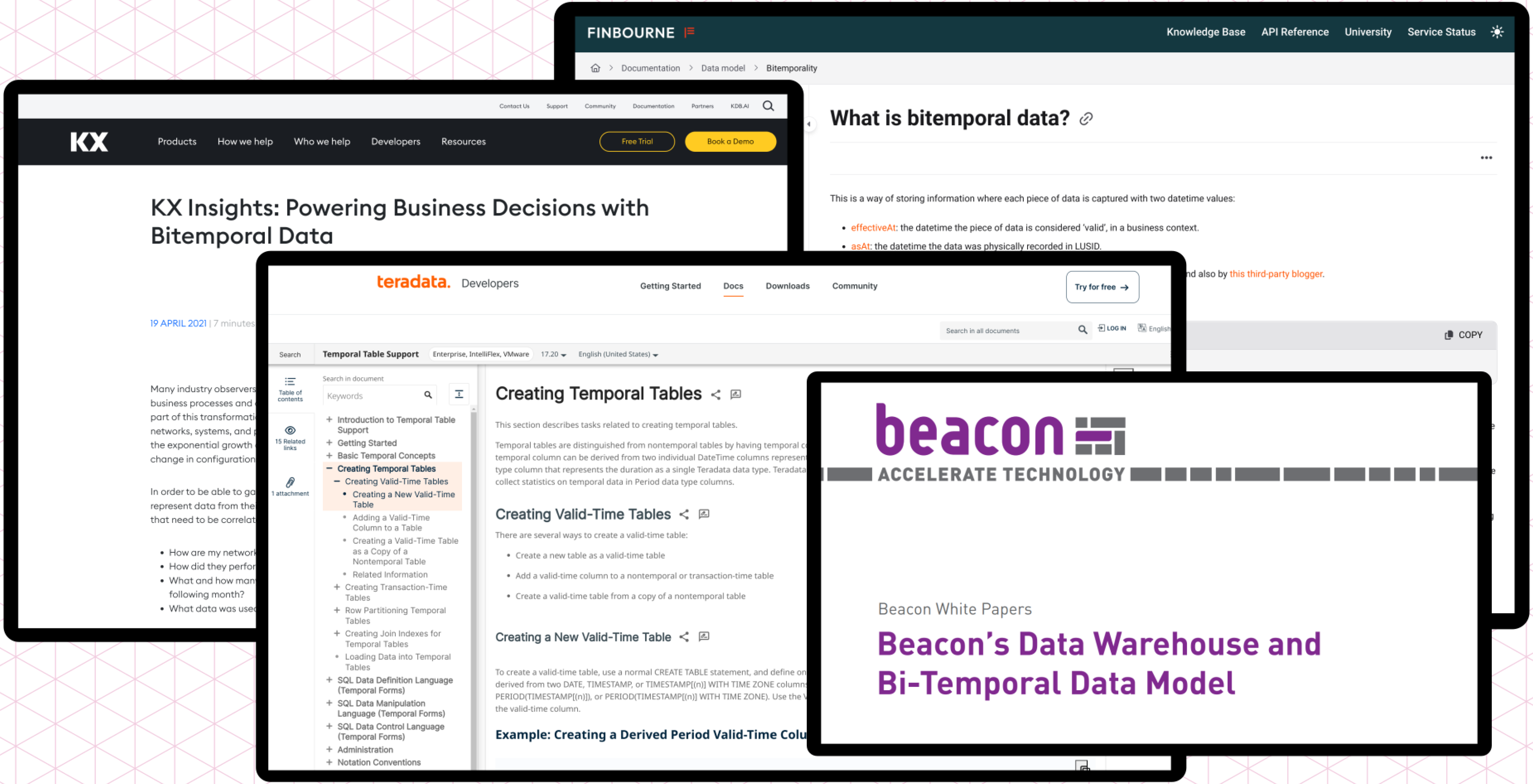
lucaborella89 started this conversation in Ideas

lucaborella89 on Jul 2 Maintainer

TraderX Support for Bitemporal Data

https://www.juxt.pro/blog/bitemporal-traderx/

There are many approaches to meeting As-Of requirements



FINBOURNE

What is bitemporal data?

This is a way of storing information where each piece of data is captured with two datetime values:

- effectiveAt: the datetime the piece of data is considered 'valid', in a business context.
- asAt: the datetime the data was physically recorded in LUSID.

nd also by this third-party blogger.

KX Insights: Powering Business Decisions with Bitemporal Data

19 APRIL 2021 | 7 minutes

Many industry observers
business processes and
part of this transformati
networks, systems, and
the exponential growth
change in configuration

In order to be able to ga
represent data from the
that need to be correlat

- How are my network
- How did they perfor
- What and how man following month?
- What data was used

- Introduction to Temporal Table Support
- Getting Started
- Basic Temporal Concepts
- Creating Temporal Tables
  - Creating Valid-Time Tables
    - Creating a New Valid-Time Table
    - Adding a Valid-Time Column to a Table
    - Creating a Valid-Time Table as a Copy of a Nontemporal Table
    - Related Information
  - Creating Transaction-Time Tables
  - Row Partitioning Temporal Tables
  - Creating Join Indexes for Temporal Tables
  - Loading Data into Temporal Tables
- SQL Data Definition Language (Temporal Forms)
- SQL Data Manipulation Language (Temporal Forms)
- SQL Data Control Language (Temporal Forms)
- Administration
- Notation Conventions

Creating Temporal Tables

This section describes tasks related to creating temporal tables.

Temporal tables are distinguished from nontemporal tables by having temporal c
temporal column can be derived from two individual DateTime columns represent
type column that represents the duration as a single Teradata data type. Teradata
collect statistics on temporal data in Period data type columns.

Creating Valid-Time Tables

There are several ways to create a valid-time table:

- Create a new table as a valid-time table
- Add a valid-time column to a nontemporal or transaction-time table
- Create a valid-time table from a copy of a nontemporal table

Creating a New Valid-Time Table

To create a valid-time table, use a normal CREATE TABLE statement, and define on
derived from two DATE, TIMESTAMP, or TIMESTAMP[(n)] WITH TIME ZONE columns
PERIOD(TIMESTAMP[(n)]), or PERIOD(TIMESTAMP[(n)] WITH TIME ZONE). Use the V
the valid-time column.

Example: Creating a Derived Period Valid-Time Colu

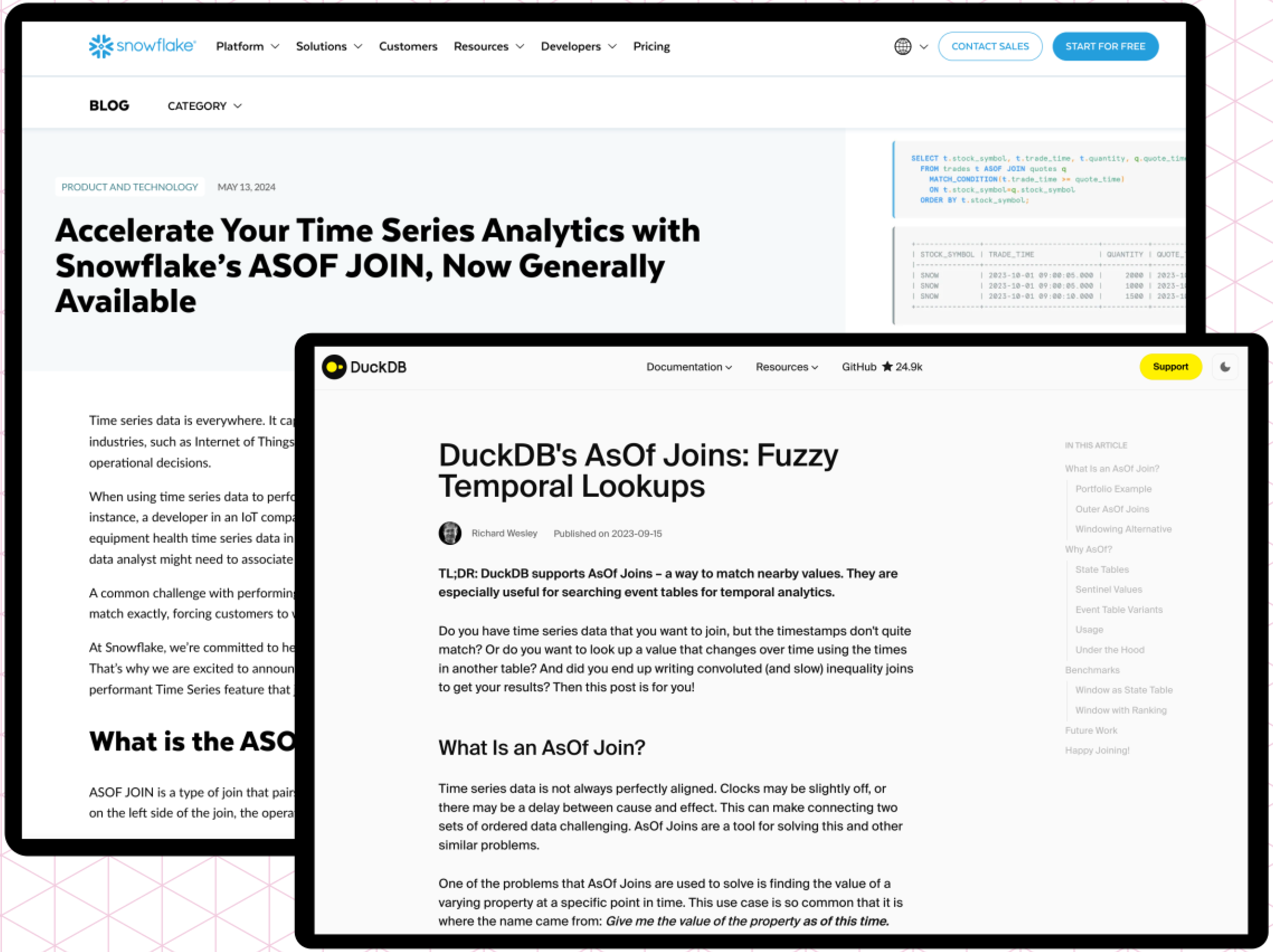
beacon

ACCELERATE TECHNOLOGY

Beacon White Papers

Beacon's Data Warehouse and Bi-Temporal Data Model

As-Of Joins in "Time Series" Data



PRODUCT AND TECHNOLOGY MAY 13, 2024

Accelerate Your Time Series Analytics with Snowflake's ASOF JOIN, Now Generally Available

```
SELECT t.stock_symbol, t.trade_time, t.quantity, q.quote_tim
  FROM trades t ASOF JOIN quotes q
    MATCH_CONDITION(t.trade_time >= quote_time)
    ON t.stock_symbol=q.stock_symbol
  ORDER BY t.stock_symbol;
```

```
+--------------+--------------------------+----------+--------
| STOCK_SYMBOL | TRADE_TIME               | QUANTITY | QUOTE_
|--------------+--------------------------+----------+--------
| SNOW         | 2023-10-01 09:00:05.000 |     2000 | 2023-1
| SNOW         | 2023-10-01 09:00:05.000 |     1000 | 2023-1
| SNOW         | 2023-10-01 09:00:10.000 |     1500 | 2023-1
+--------------+--------------------------+----------+--------
```

Time series data is everywhere. It ca
industries, such as Internet of Things
operational decisions.

When using time series data to perfo
instance, a developer in an IoT compa
equipment health time series data in
data analyst might need to associate

A common challenge with performin
match exactly, forcing customers to

At Snowflake, we're committed to he
That's why we are excited to announ
performant Time Series feature that

What is the ASO

ASOF JOIN is a type of join that pai
on the left side of the join, the opera

DuckDB's AsOf Joins: Fuzzy Temporal Lookups

Richard Wesley Published on 2023-09-15

TL;DR: DuckDB supports AsOf Joins – a way to match nearby values. They are especially useful for searching event tables for temporal analytics.

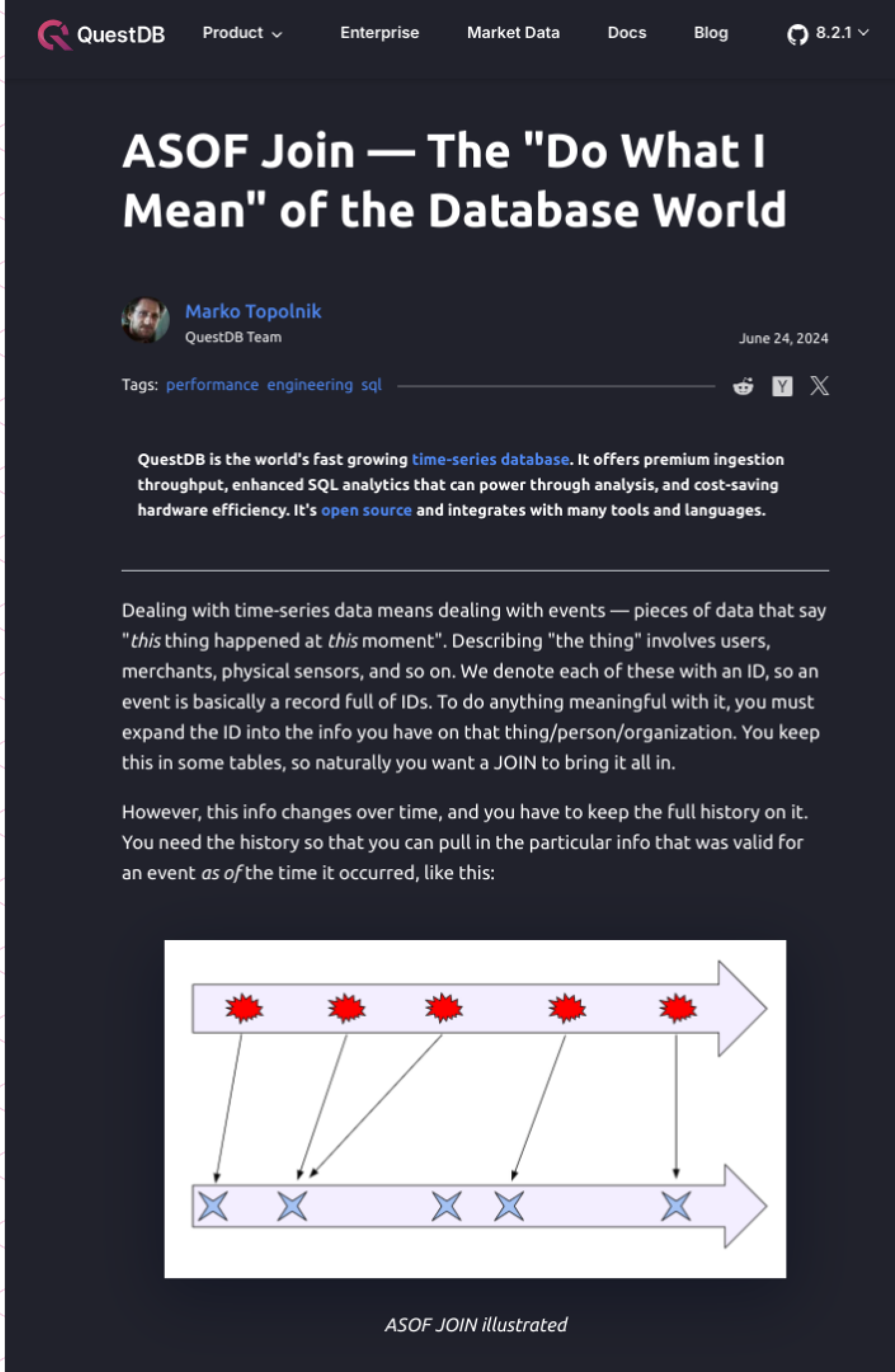
Do you have time series data that you want to join, but the timestamps don't quite match? Or do you want to look up a value that changes over time using the times in another table? And did you end up writing convoluted (and slow) inequality joins to get your results? Then this post is for you!

What Is an AsOf Join?

Time series data is not always perfectly aligned. Clocks may be slightly off, or there may be a delay between cause and effect. This can make connecting two sets of ordered data challenging. AsOf Joins are a tool for solving this and other similar problems.

One of the problems that AsOf Joins are used to solve is finding the value of a varying property at a specific point in time. This use case is so common that it is where the name came from: *Give me the value of the property as of this time.*

IN THIS ARTICLE



ASOF Join — The "Do What I Mean" of the Database World

Marko Topolnik
QuestDB Team
June 24, 2024

Tags: performance engineering sql

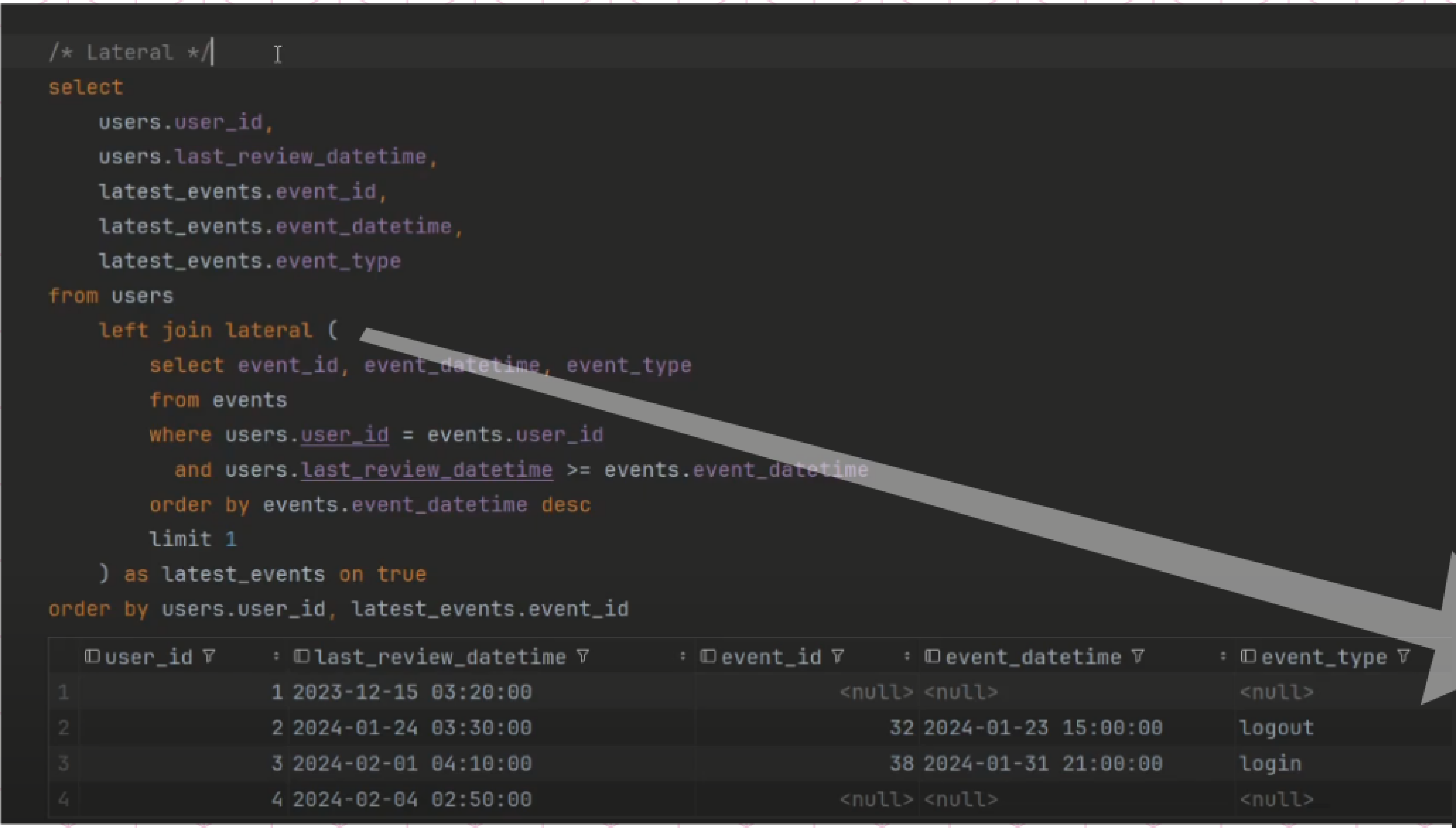
QuestDB is the world's fast growing time-series database. It offers premium ingestion throughput, enhanced SQL analytics that can power through analysis, and cost-saving hardware efficiency. It's open source and integrates with many tools and languages.

Dealing with time-series data means dealing with events — pieces of data that say "*this* thing happened at *this* moment". Describing "the thing" involves users, merchants, physical sensors, and so on. We denote each of these with an ID, so an event is basically a record full of IDs. To do anything meaningful with it, you must expand the ID into the info you have on that thing/person/organization. You keep this in some tables, so naturally you want a JOIN to bring it all in.

However, this info changes over time, and you have to keep the full history on it. You need the history so that you can pull in the particular info that was valid for an event *as of* the time it occurred, like this:

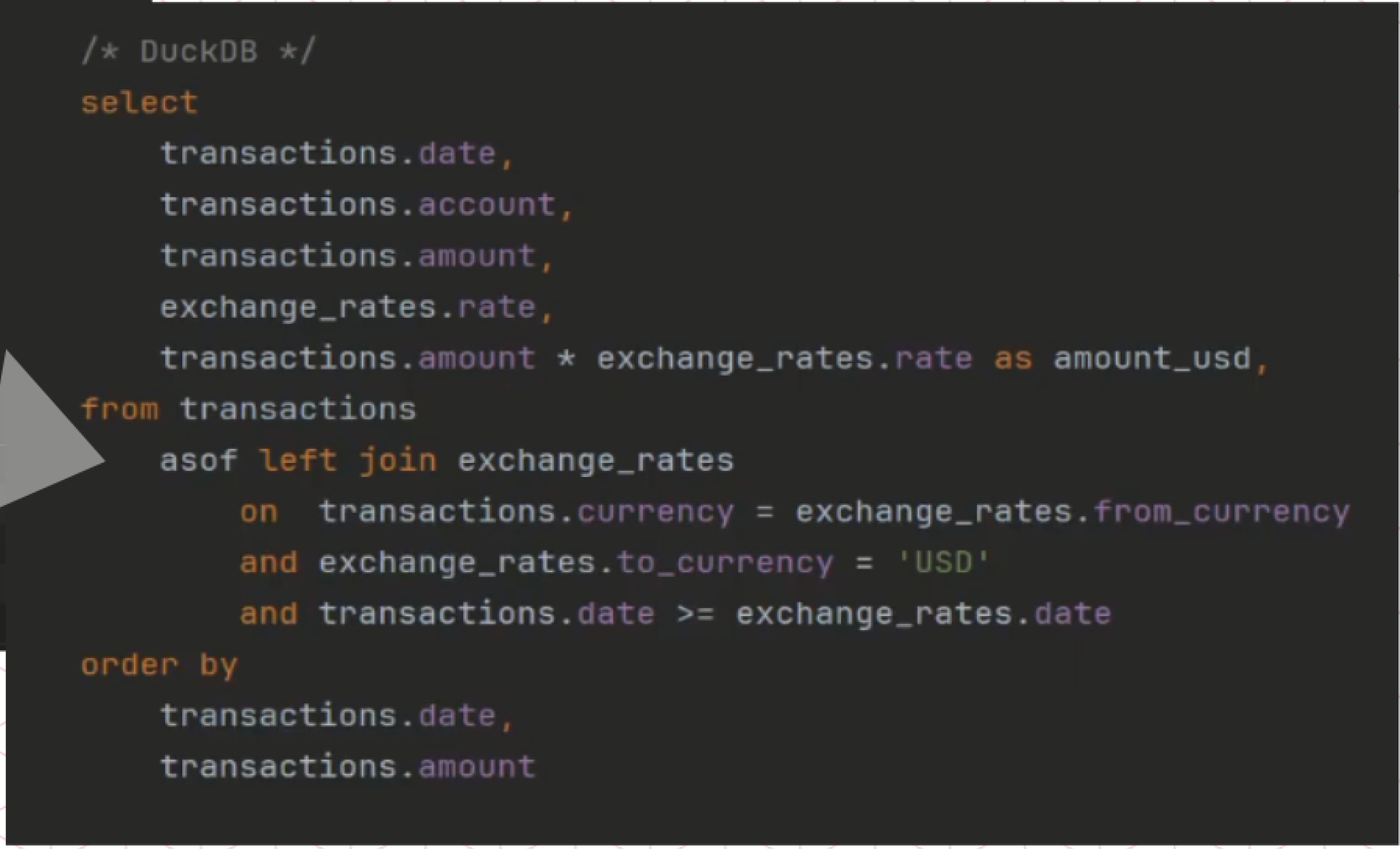
ASOF JOIN illustrated

`asof left join` = `left join lateral` sugar + execution speedup



```
/* Lateral */
select
    users.user_id,
    users.last_review_datetime,
    latest_events.event_id,
    latest_events.event_datetime,
    latest_events.event_type
from users
    left join lateral (
        select event_id, event_datetime, event_type
        from events
        where users.user_id = events.user_id
          and users.last_review_datetime >= events.event_datetime
        order by events.event_datetime desc
        limit 1
    ) as latest_events on true
order by users.user_id, latest_events.event_id
```

	user_id	last_review_datetime	event_id	event_datetime	event_type
1	1	2023-12-15 03:20:00	<null>	<null>	<null>
2	2	2024-01-24 03:30:00	32	2024-01-23 15:00:00	logout
3	3	2024-02-01 04:10:00	38	2024-01-31 21:00:00	login
4	4	2024-02-04 02:50:00	<null>	<null>	<null>



```
/* DuckDB */
select
    transactions.date,
    transactions.account,
    transactions.amount,
    exchange_rates.rate,
    transactions.amount * exchange_rates.rate as amount_usd,
from transactions
    asof left join exchange_rates
        on  transactions.currency = exchange_rates.from_currency
        and exchange_rates.to_currency = 'USD'
        and transactions.date >= exchange_rates.date
order by
    transactions.date,
    transactions.amount
```

See also "row_number () over"

SQL ASOF Join (DuckDB, Snowflake), Bill Wallis @ YouTube
https://bilbottom.github.io/sql-learning-materials/everything-about-joins/syntax/timestamp-joins/

Bitemporal Databases

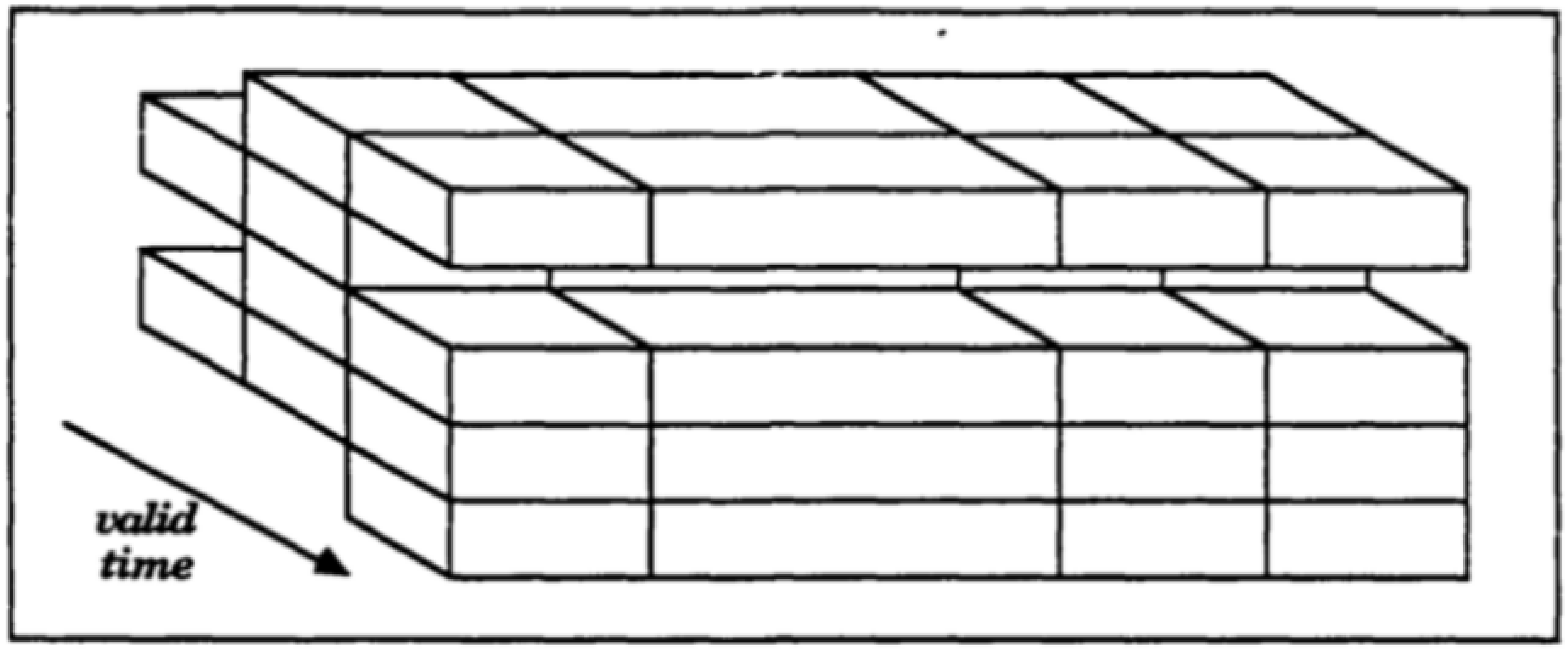


Figure 3: Historical Relation

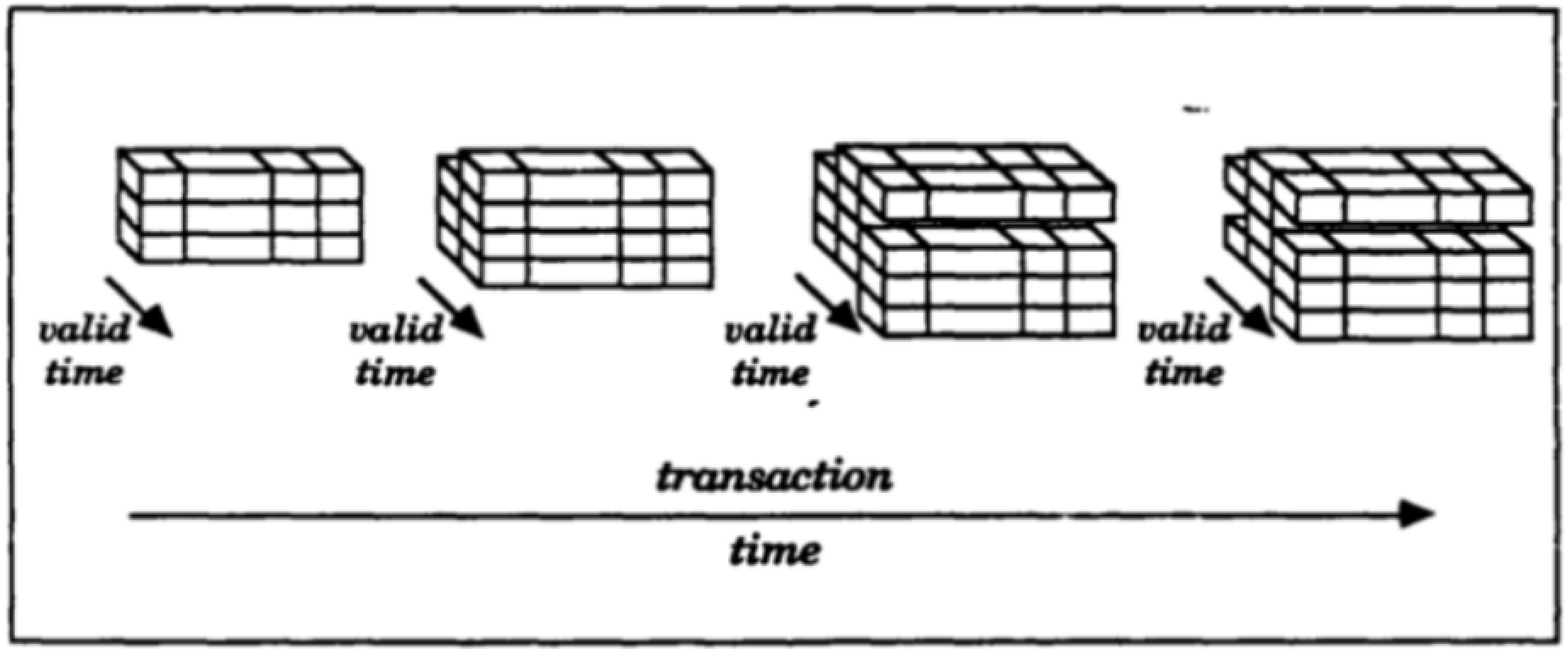


Figure 4: Temporal Relation

Scheme Evolution and
the Relational Algebra
May 1988

SAFE
& CORRECT

Sources of **essential** complexity:

Sources of **essential** complexity:

Your business logic.

Sources of **incidental** complexity:

Legacy Systems Integration

Regulatory Compliance and Audit Requirements

Complex Security and Access Control

Data Consistency and Synchronization Across Systems

High Availability and Low Latency Requirements

Changing Business Requirements

Vendor and Third-party Dependencies

Inconsistent Data Models

Cross-jurisdictional Requirements

Distributed Teams and Communication Challenges

Version Control and Change Management

Dependency on Batch Processing Systems

Complex Infrastructure and Cloud Migration

Tooling Fragmentation

Unclear or Shifting Responsibilities

Lack of Unified Documentation

Strict Performance Optimization Requirements

Testing and QA Challenges

Multi-language Support

Complex Release Management

Governance and Risk Management Overheads

Cultural and Organizational Resistance

•••

Sources of **incidental** complexity:

Legacy Systems Integration
Regulatory Compliance and Audit Requirements
Complex Security and Access Control
Data Consistency and Synchronization Across Systems
High Availability and Low Latency Requirements
Changing Business Requirements
Vendor and Third-party Dependencies
Inconsistent Data Models
Cross-jurisdictional Requirements
Distributed Teams and Communication Challenges
Version Control and Change Management

Dependency on Batch Processing Systems
Complex Infrastructure and Cloud Migration
Tooling Fragmentation
Unclear or Shifting Responsibilities
Lack of Unified Documentation
Strict Performance Optimization Requirements
Testing and QA Challenges
Multi-language Support
Complex Release Management
Governance and Risk Management Overheads
Cultural and Organizational Resistance

•••

Bitemporal =

Ingestion timestamp +

Reporting timestamp

Bitemporal =

Ingestion timestamp +

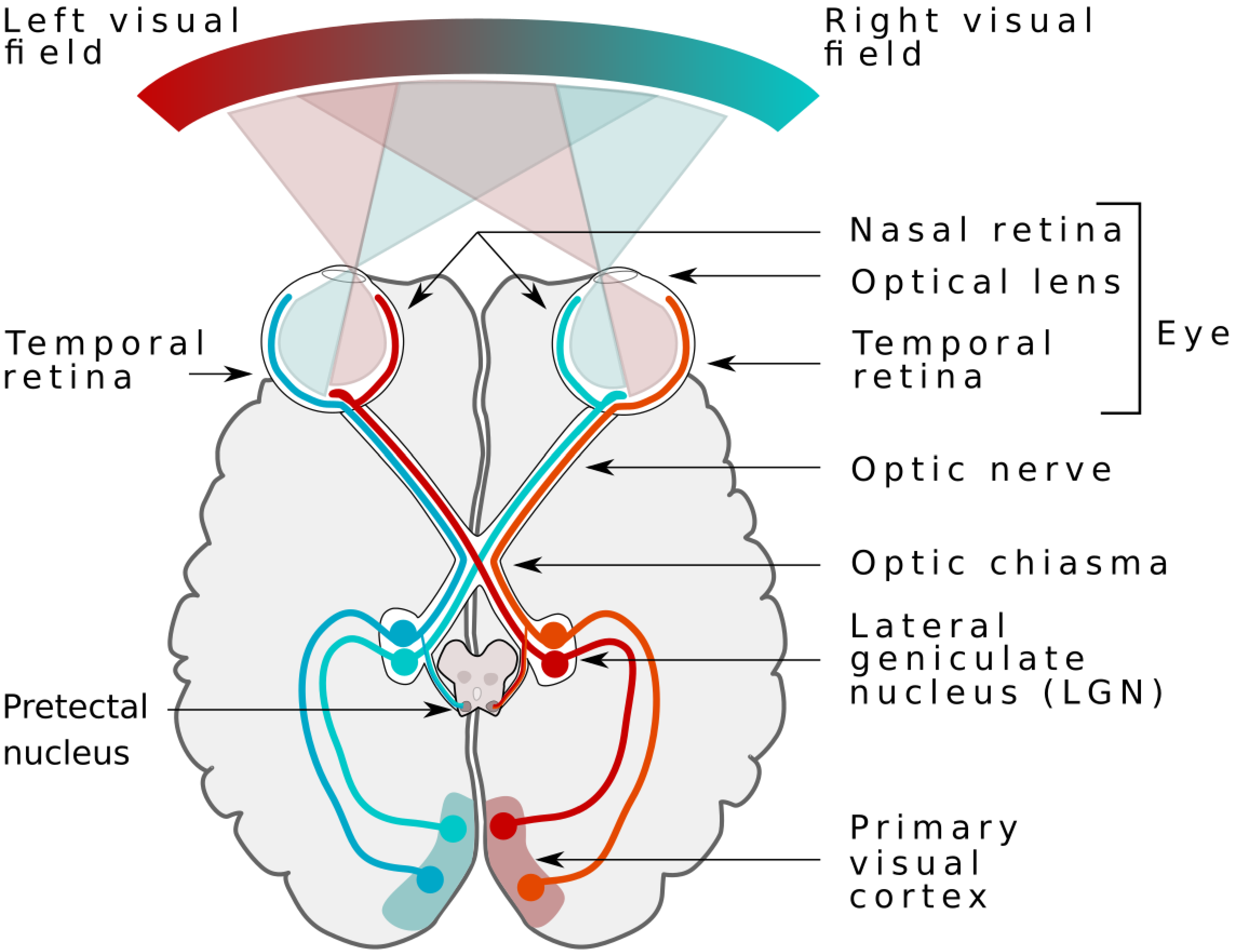
Reporting timestamp

Safe ("immutable")

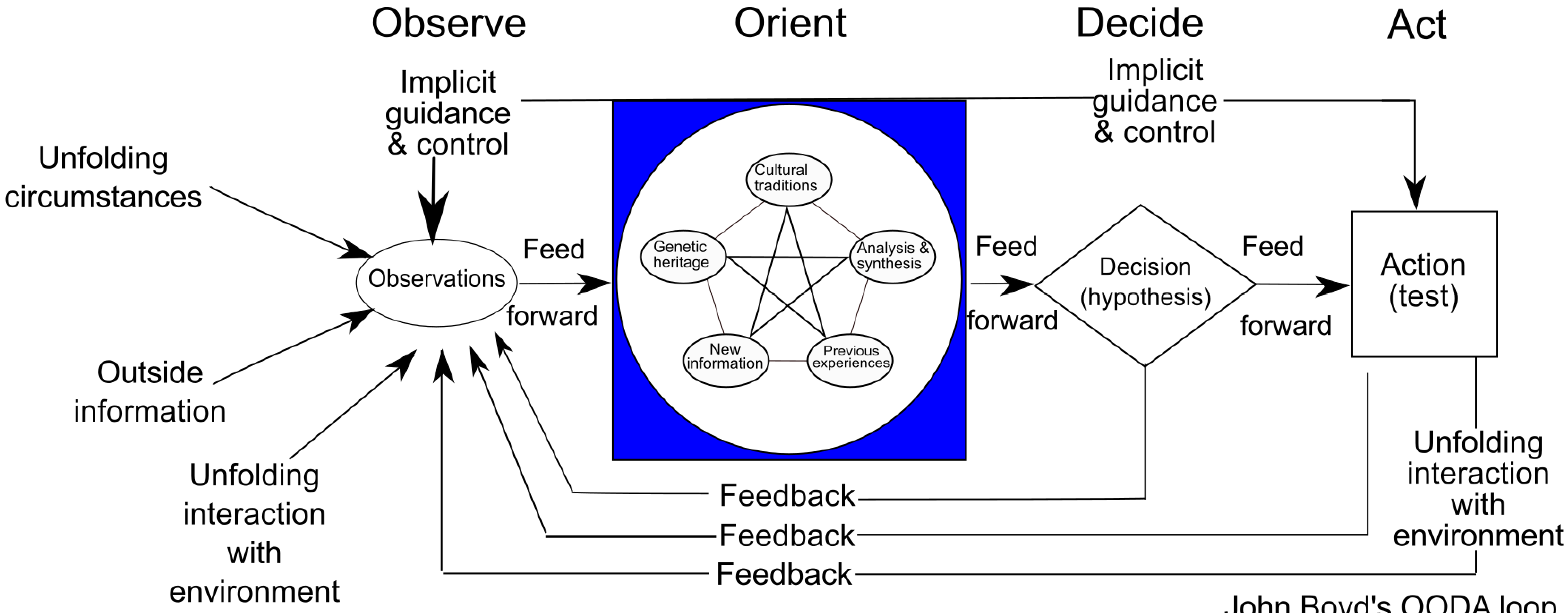
The timeline of the system

Correct

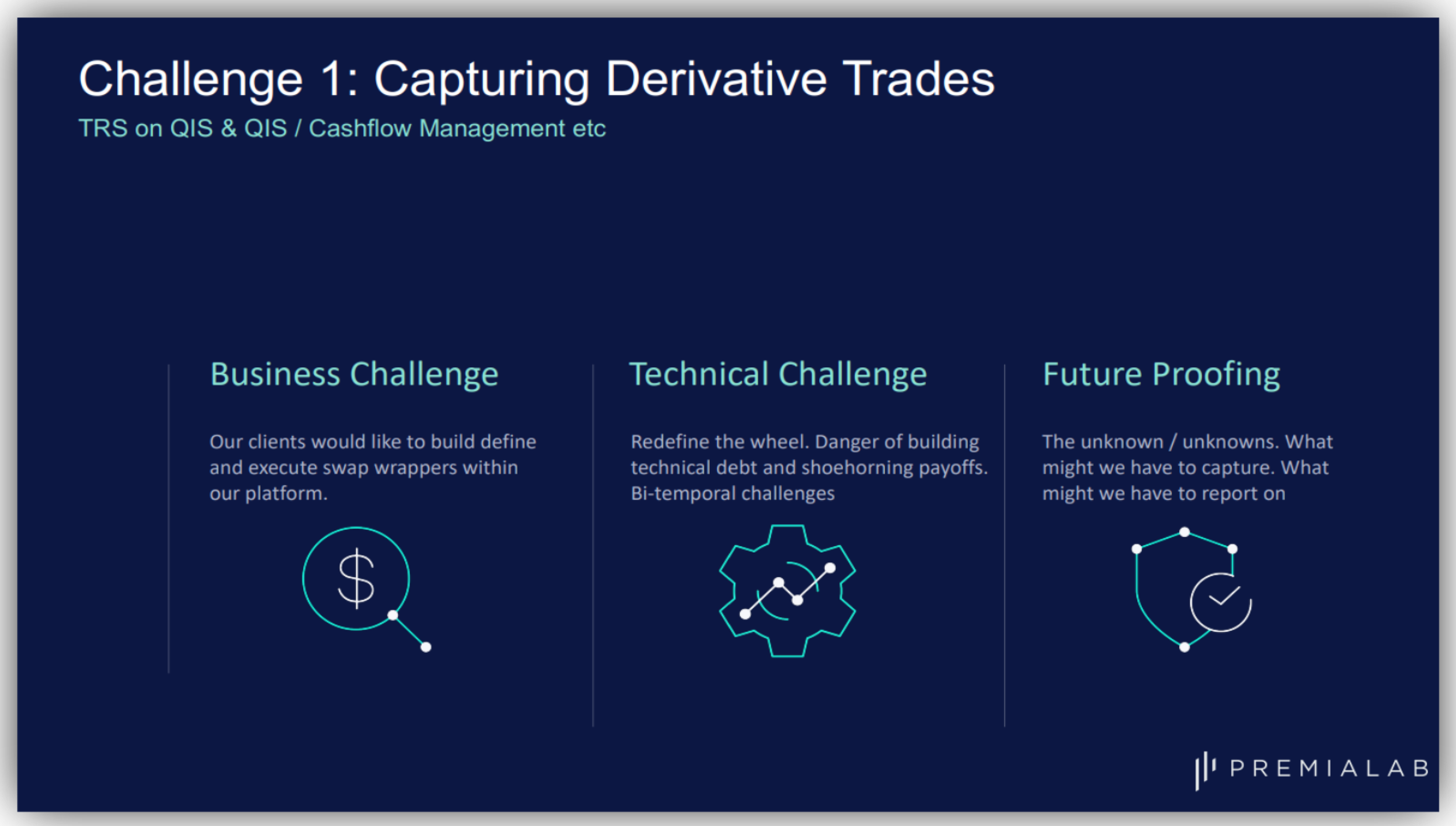
The timeline of the data



Miquel Perello Nieto, CC BY-SA 4.0



John Boyd's OODA loop



Challenge 1: Capturing Derivative Trades

TRS on QIS & QIS / Cashflow Management etc

Business Challenge

Our clients would like to build define and execute swap wrappers within our platform.

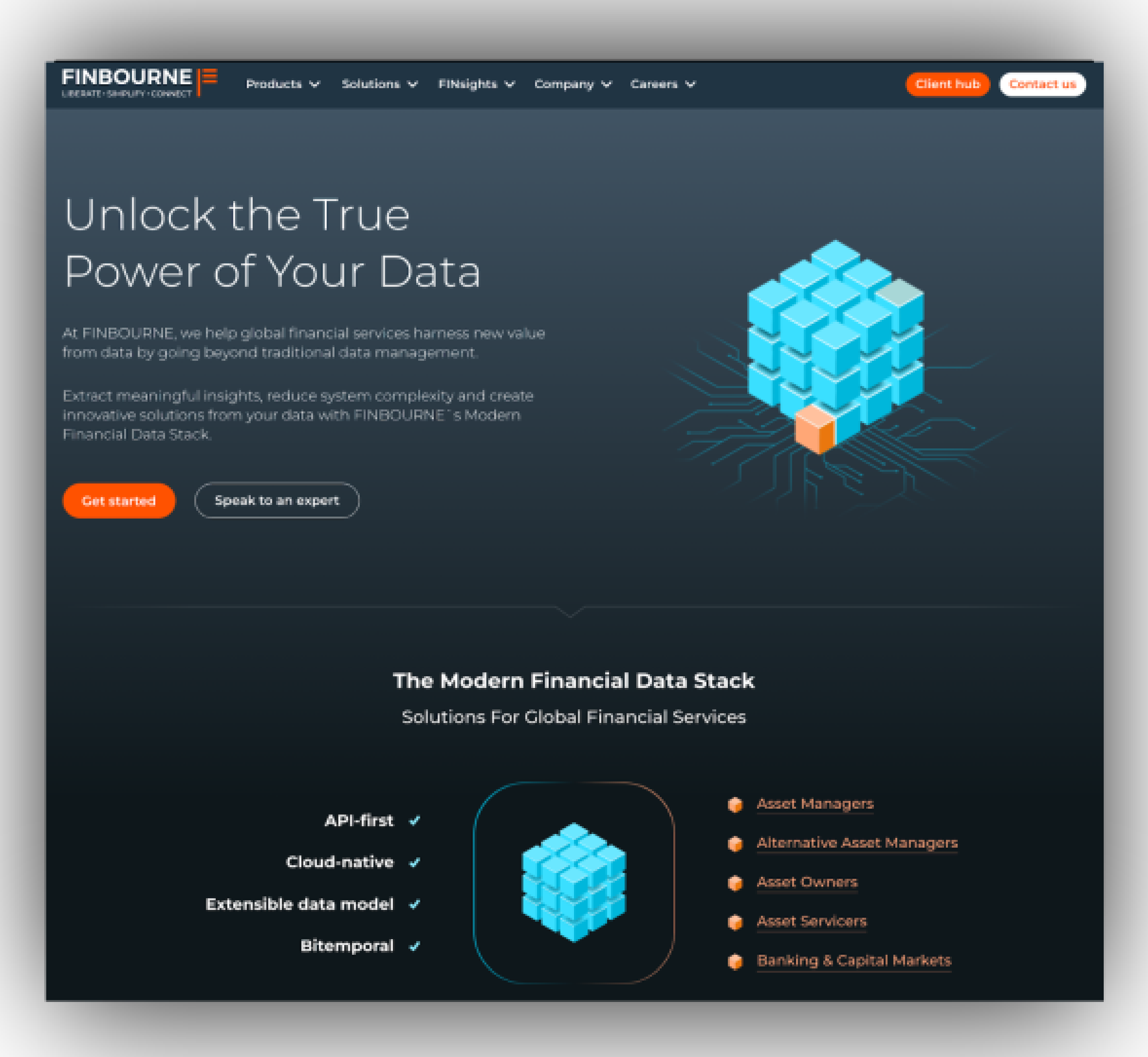
Technical Challenge

Redefine the wheel. Danger of building technical debt and shoehorning payoffs. Bi-temporal challenges

Future Proofing

The unknown / unknowns. What might we have to capture. What might we have to report on

PREMIALAB



Unlock the True Power of Your Data

At FINBOURNE, we help global financial services harness new value from data by going beyond traditional data management.

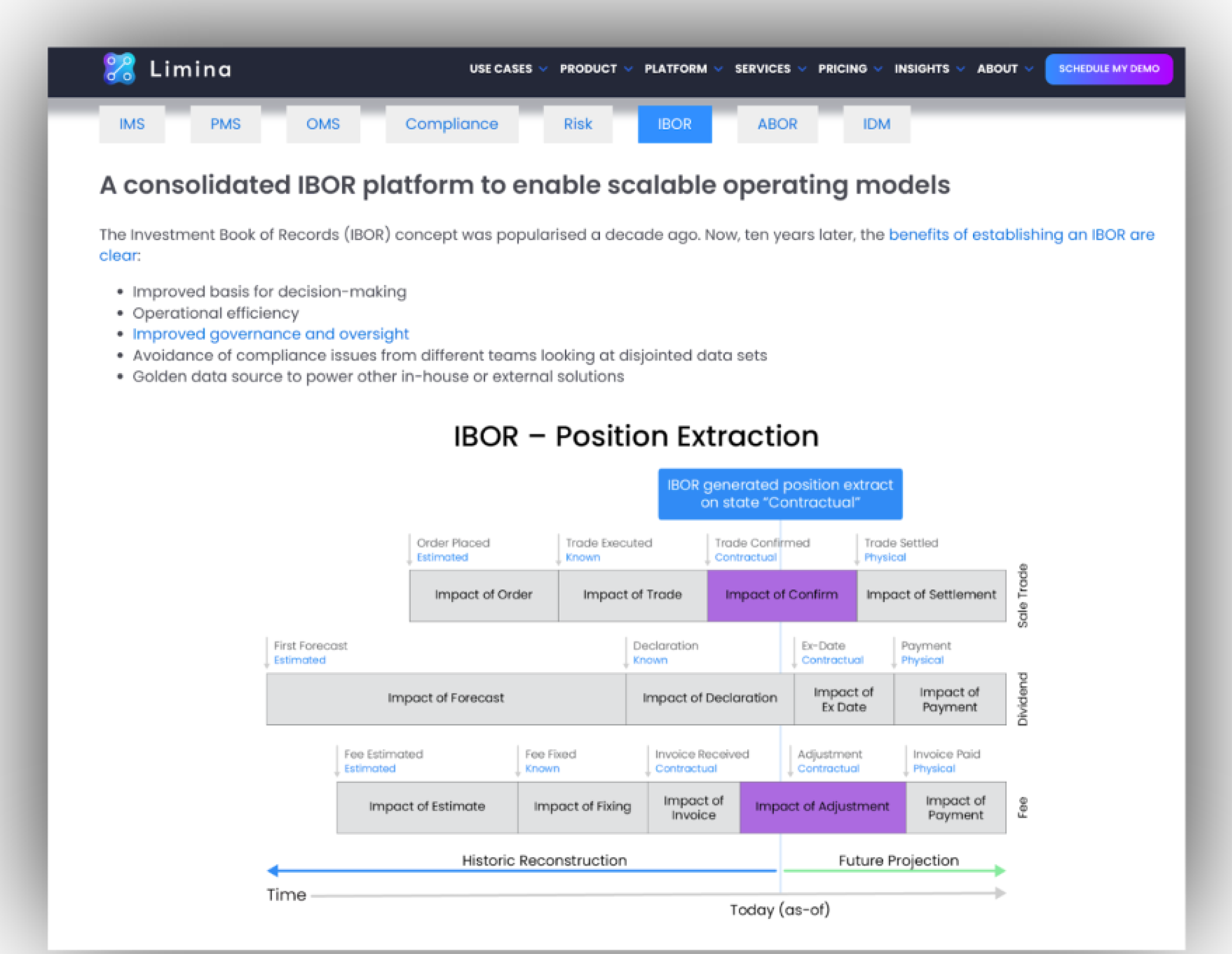
Extract meaningful insights, reduce system complexity and create innovative solutions from your data with FINBOURNE`s Modern Financial Data Stack.

The Modern Financial Data Stack

Solutions For Global Financial Services

- API-first ✓
- Cloud-native ✓
- Extensible data model ✓
- Bitemporal ✓

- Asset Managers
- Alternative Asset Managers
- Asset Owners
- Asset Servicers
- Banking & Capital Markets

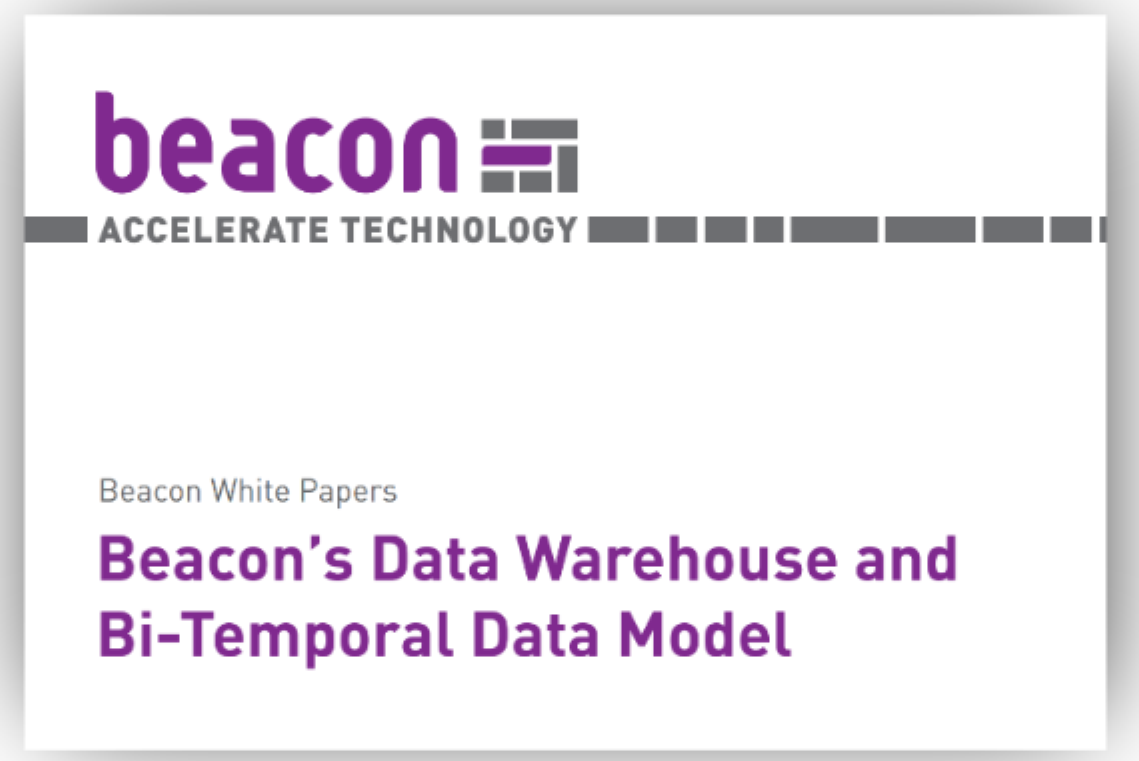


A consolidated IBOR platform to enable scalable operating models

The Investment Book of Records (IBOR) concept was popularised a decade ago. Now, ten years later, the benefits of establishing an IBOR are clear:

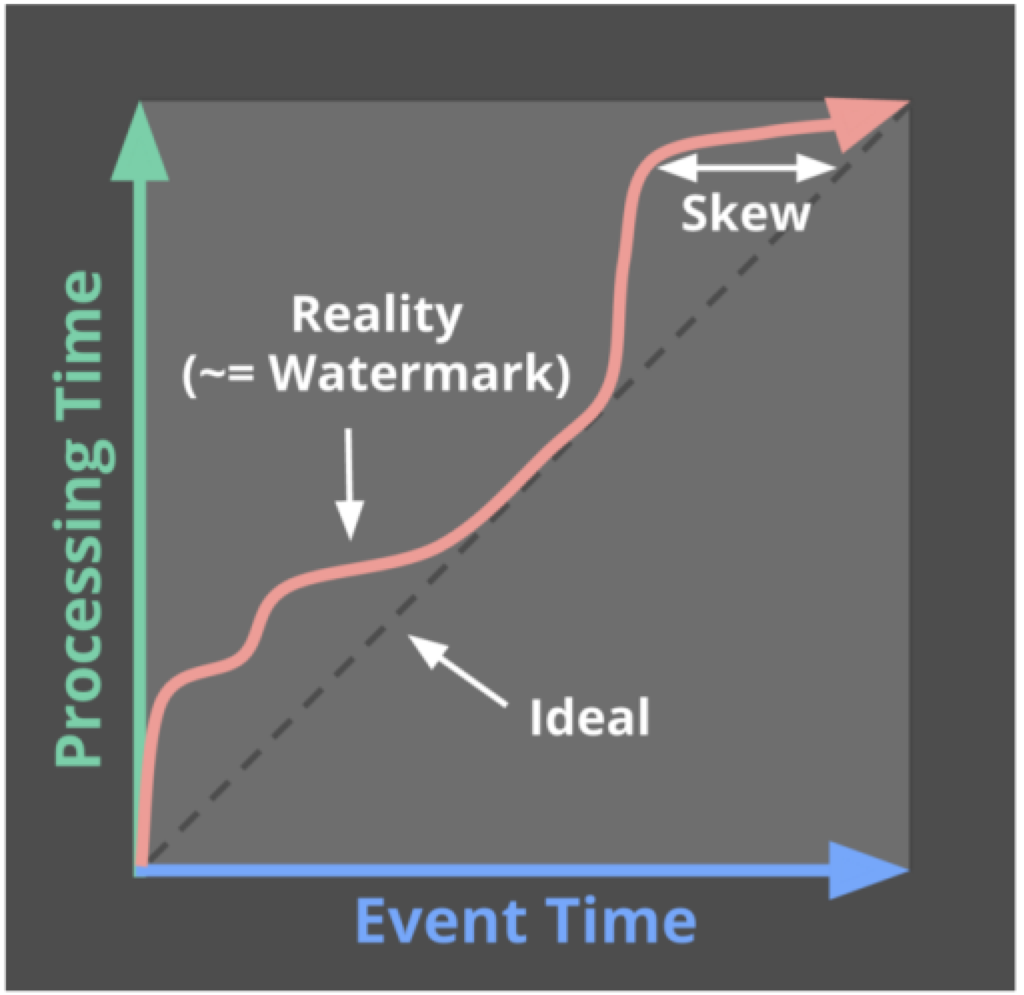
- Improved basis for decision-making
- Operational efficiency
- Improved governance and oversight
- Avoidance of compliance issues from different teams looking at disjointed data sets
- Golden data source to power other in-house or external solutions

IBOR – Position Extraction



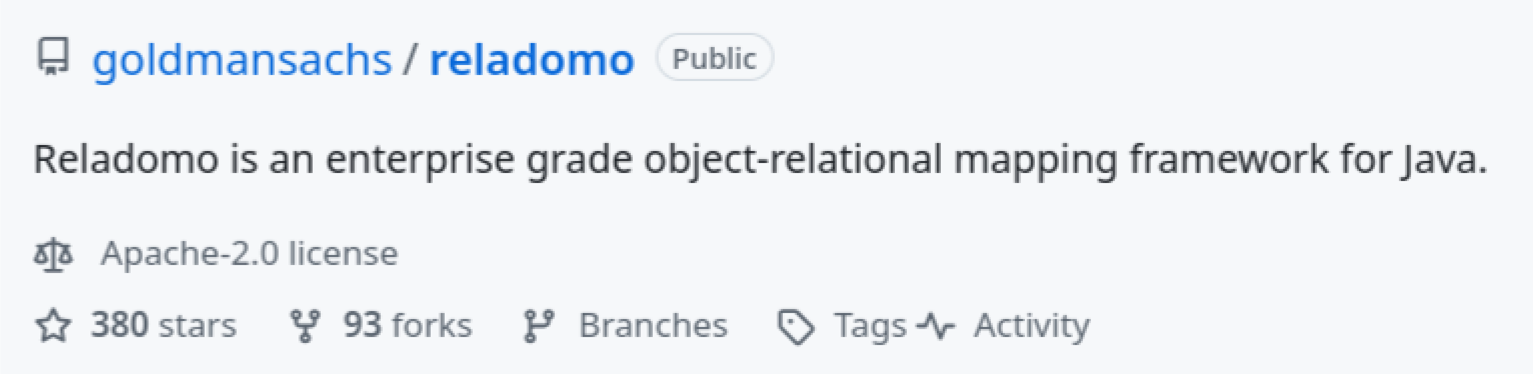
Beacon White Papers

Beacon's Data Warehouse and Bi-Temporal Data Model



Streaming Data Processing Systems
Tyler Akidau, Google

There are many relevant tools for developers and data teams...

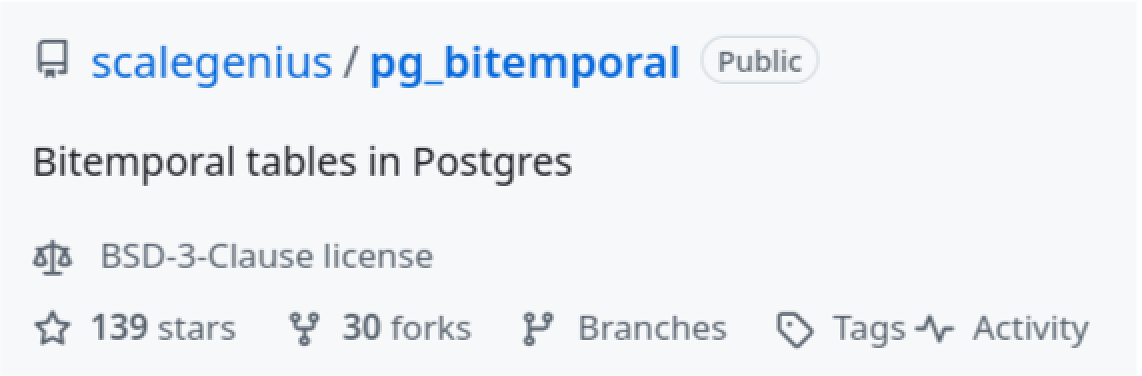

goldmansachs / reladomo Public

Reladomo is an enterprise grade object-relational mapping framework for Java.

Apache-2.0 license

380 stars 93 forks Branches Tags Activity

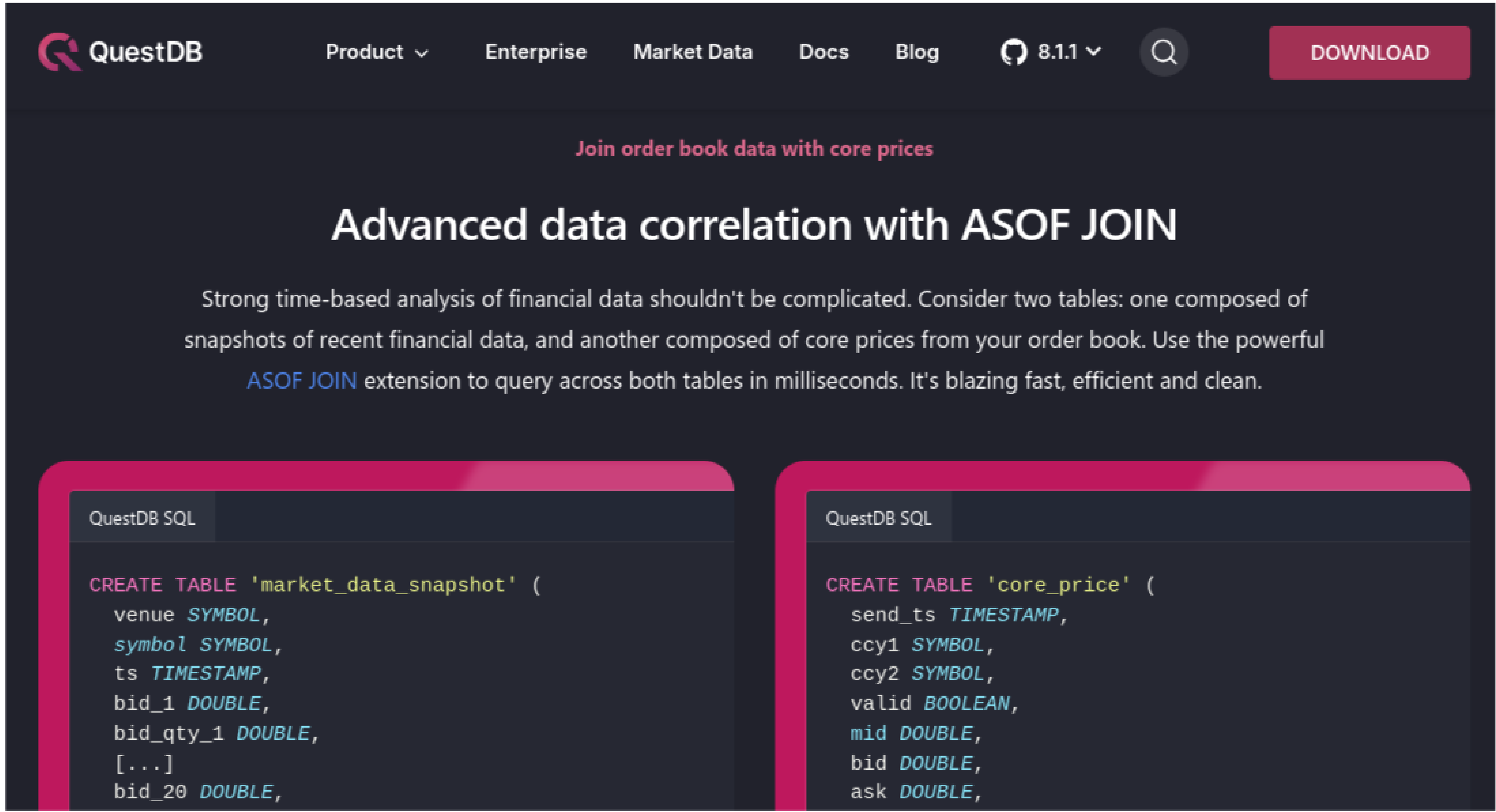
DRW


scalegenius / pg_bitemporal Public

Bitemporal tables in Postgres

BSD-3-Clause license

139 stars 30 forks Branches Tags Activity



Join order book data with core prices

Advanced data correlation with ASOF JOIN

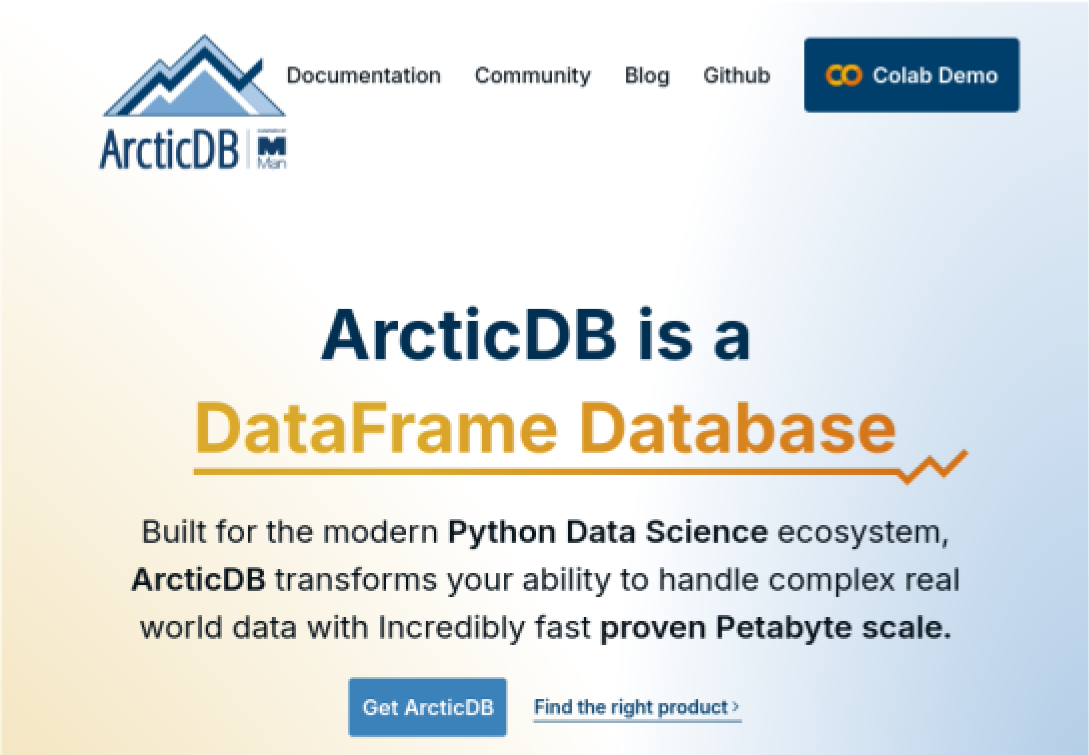
Strong time-based analysis of financial data shouldn't be complicated. Consider two tables: one composed of snapshots of recent financial data, and another composed of core prices from your order book. Use the powerful ASOF JOIN extension to query across both tables in milliseconds. It's blazing fast, efficient and clean.

QuestDB SQL

```sql
CREATE TABLE 'market_data_snapshot' (
  venue SYMBOL,
  symbol SYMBOL,
  ts TIMESTAMP,
  bid_1 DOUBLE,
  bid_qty_1 DOUBLE,
  [...]
  bid_20 DOUBLE,
```

QuestDB SQL

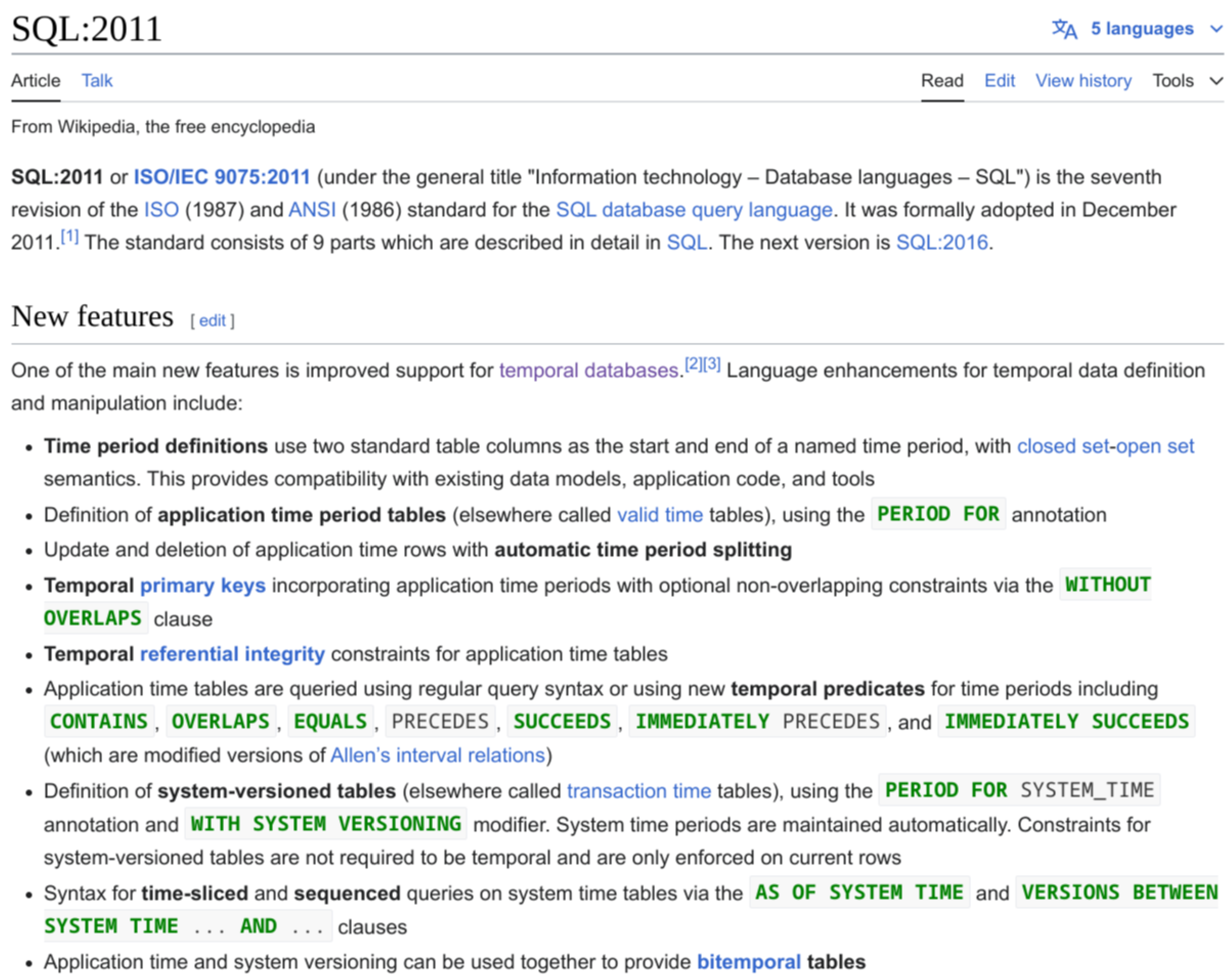
```sql
CREATE TABLE 'core_price' (
  send_ts TIMESTAMP,
  ccy1 SYMBOL,
  ccy2 SYMBOL,
  valid BOOLEAN,
  mid DOUBLE,
  bid DOUBLE,
  ask DOUBLE,
```



ArcticDB is a DataFrame Database

Built for the modern **Python Data Science** ecosystem, **ArcticDB** transforms your ability to handle complex real world data with Incredibly fast **proven Petabyte scale.**

Is your database pulling its weight?



SQL:2011

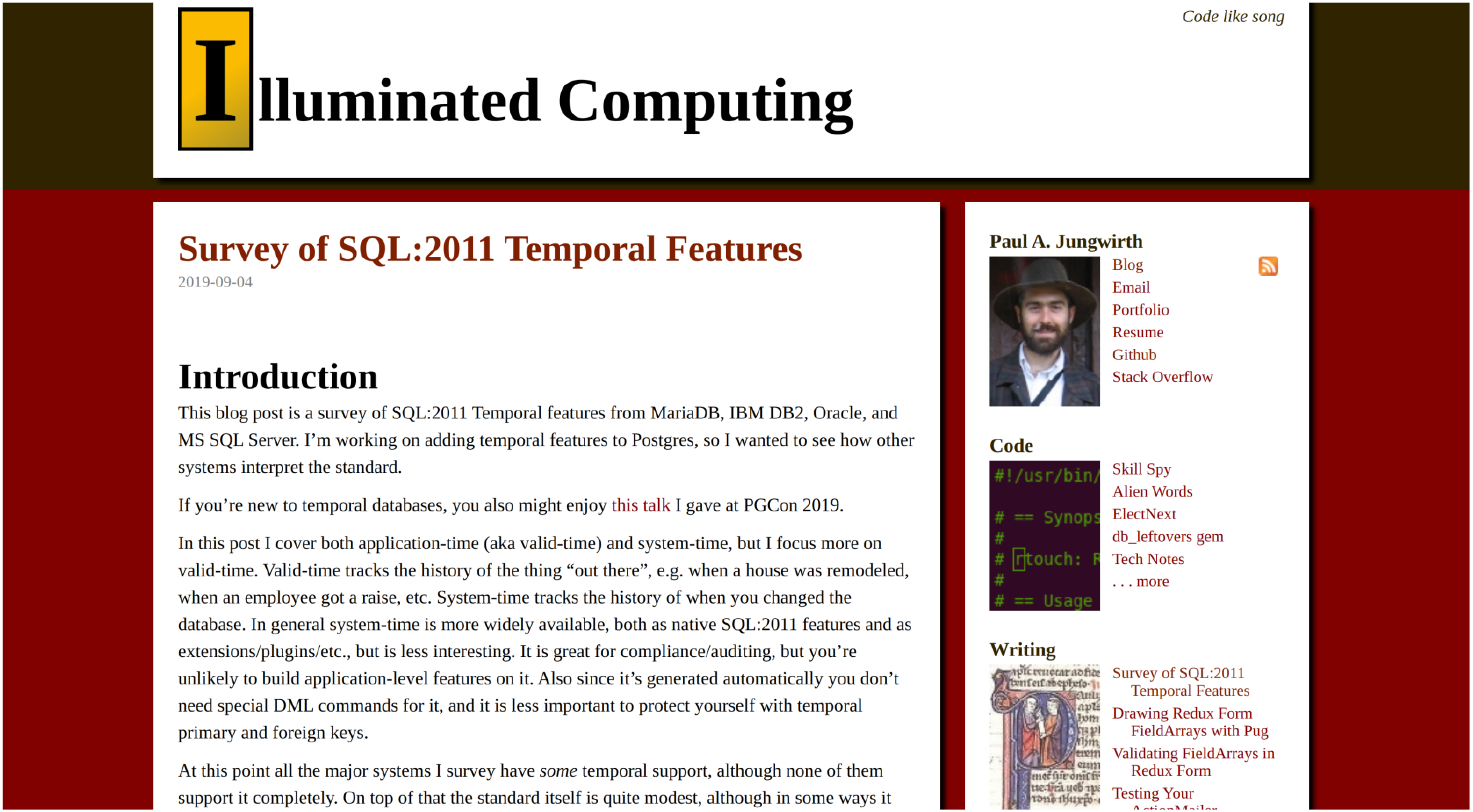
From Wikipedia, the free encyclopedia

SQL:2011 or **ISO/IEC 9075:2011** (under the general title "Information technology – Database languages – SQL") is the seventh revision of the ISO (1987) and ANSI (1986) standard for the SQL database query language. It was formally adopted in December 2011.[1] The standard consists of 9 parts which are described in detail in SQL. The next version is SQL:2016.

New features [edit]

One of the main new features is improved support for temporal databases.[2][3] Language enhancements for temporal data definition and manipulation include:

- **Time period definitions** use two standard table columns as the start and end of a named time period, with closed set-open set semantics. This provides compatibility with existing data models, application code, and tools
- Definition of **application time period tables** (elsewhere called valid time tables), using the `PERIOD FOR` annotation
- Update and deletion of application time rows with **automatic time period splitting**
- **Temporal primary keys** incorporating application time periods with optional non-overlapping constraints via the `WITHOUT OVERLAPS` clause
- **Temporal referential integrity** constraints for application time tables
- Application time tables are queried using regular query syntax or using new **temporal predicates** for time periods including `CONTAINS`, `OVERLAPS`, `EQUALS`, `PRECEDES`, `SUCCEEDS`, `IMMEDIATELY PRECEDES`, and `IMMEDIATELY SUCCEEDS` (which are modified versions of Allen's interval relations)
- Definition of **system-versioned tables** (elsewhere called transaction time tables), using the `PERIOD FOR SYSTEM_TIME` annotation and `WITH SYSTEM VERSIONING` modifier. System time periods are maintained automatically. Constraints for system-versioned tables are not required to be temporal and are only enforced on current rows
- Syntax for **time-sliced** and **sequenced** queries on system time tables via the `AS OF SYSTEM TIME` and `VERSIONS BETWEEN SYSTEM TIME ... AND ...` clauses
- Application time and system versioning can be used together to provide **bitemporal tables**



Illuminated Computing

Code like song

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

xtdb.com

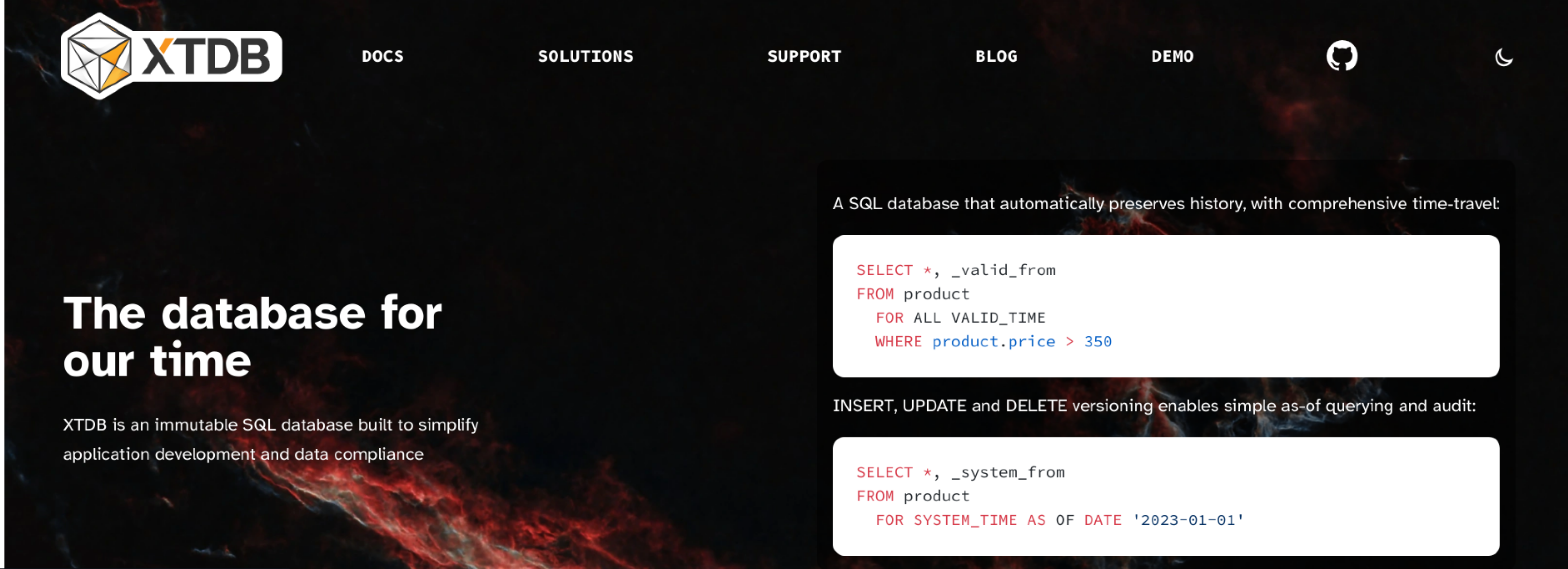
5 years

v2.0 Beta

2.6k stars

Built with:

APACHE ARROW



The database for our time

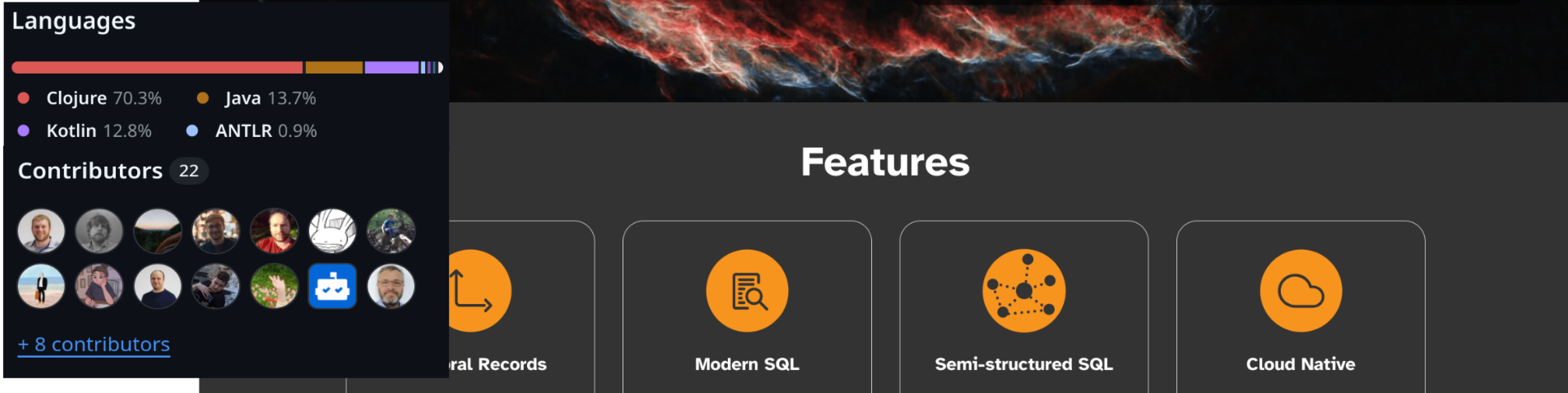
XTDB is an immutable SQL database built to simplify application development and data compliance

A SQL database that automatically preserves history, with comprehensive time-travel:

```sql
SELECT *, _valid_from
FROM product
  FOR ALL VALID_TIME
  WHERE product.price > 350
```

INSERT, UPDATE and DELETE versioning enables simple as-of querying and audit:

```sql
SELECT *, _system_from
FROM product
  FOR SYSTEM_TIME AS OF DATE '2023-01-01'
```



Languages

● Clojure 70.3% ● Java 13.7%
● Kotlin 12.8% ● ANTLR 0.9%

Contributors 22

+ 8 contributors

Features

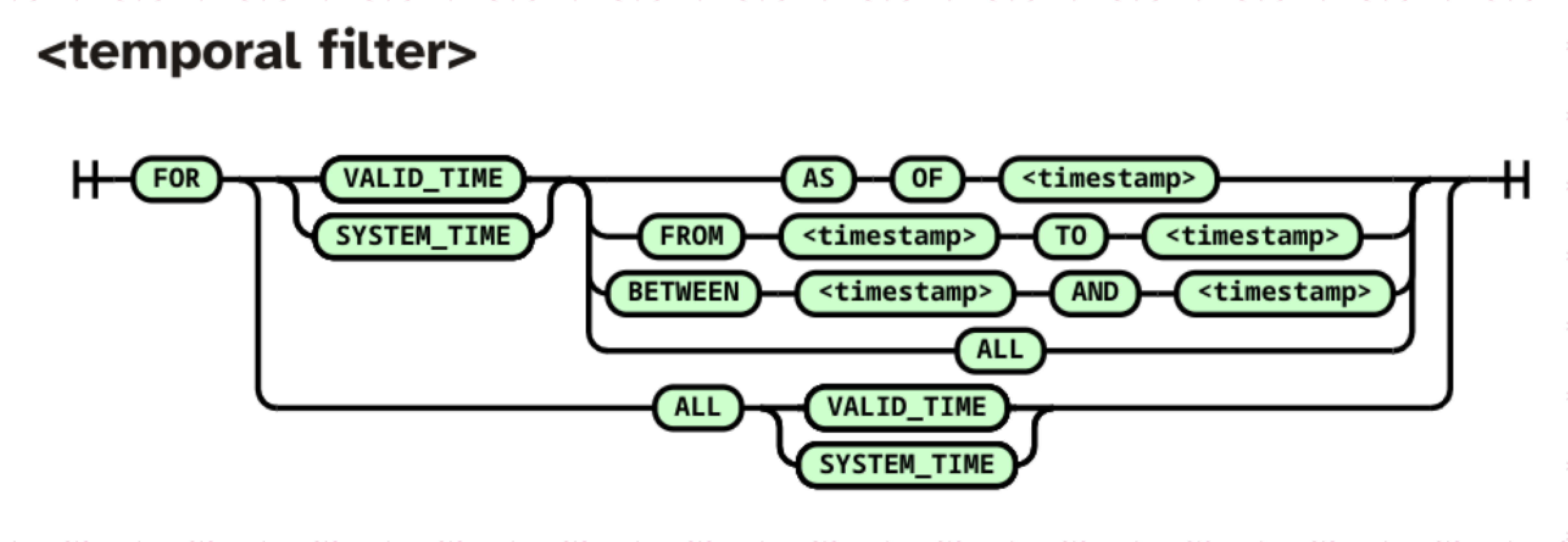
XTDB is built to solve Data Compliance problems

- Inconsistent reporting over historical data
  - Achieve consistency without snapshots or copying
- Inadequate auditing & compliance
  - Avoid ad hoc solutions that generate additional problems
- Challenging data integration & evolution
  - Manage changing semi-structured data with *gradual* schema

...XTDB helps when time is complex, and correctness matters.

SQL:2011 adds bitemporal versioning to tables:

- SYSTEM_TIME (auditing & debugging)
- VALID_TIME (reporting & versioning/corrections)
- Doesn't forget the past (e.g. DELETEs are 'soft')
- Various syntax to capture and query history...



It's taken how long just to add two timestamps??

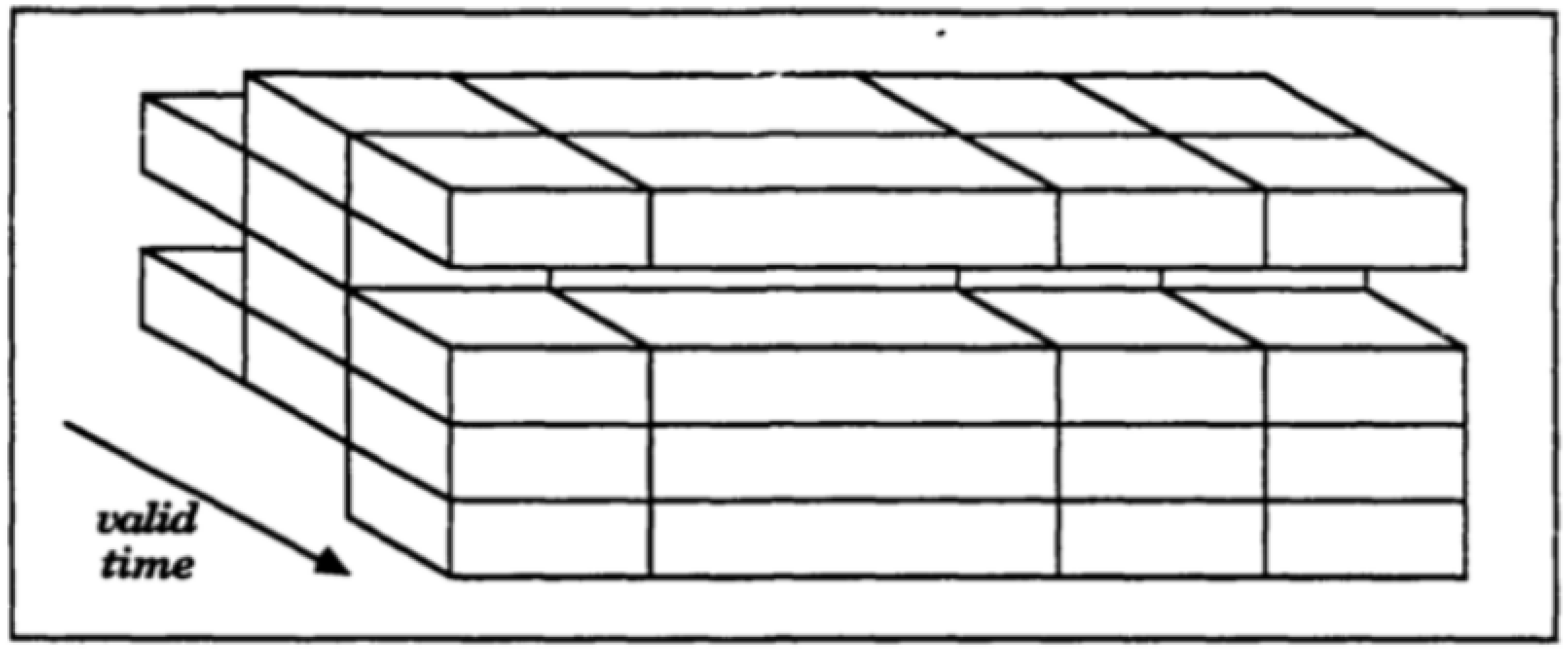


Figure 3: Historical Relation

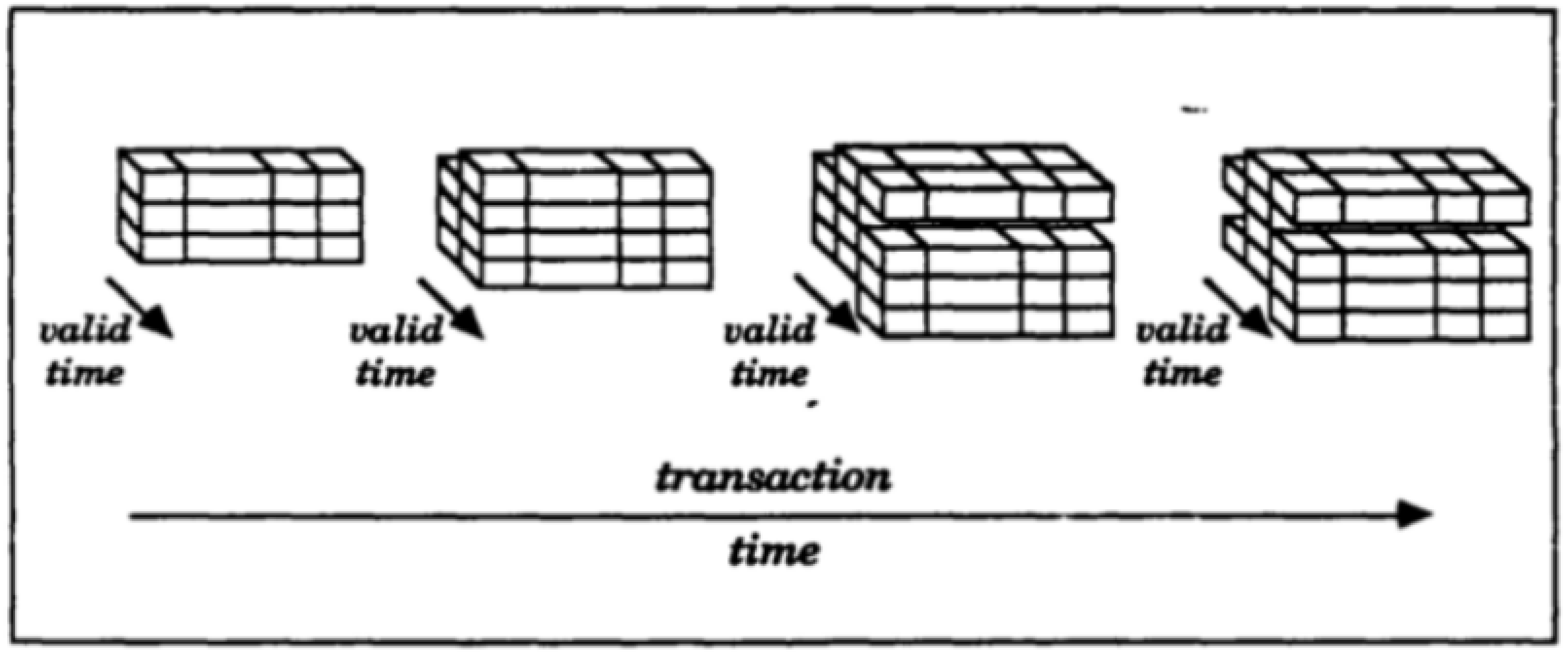
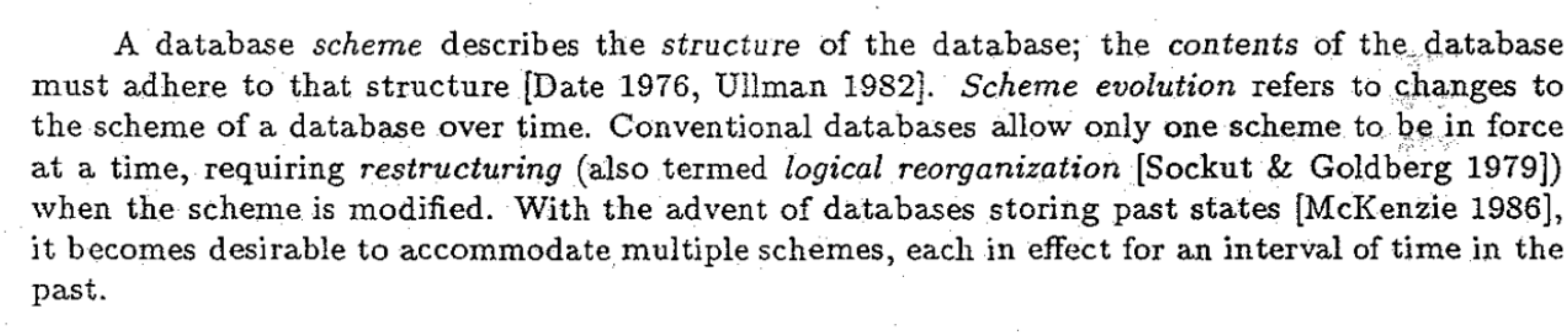


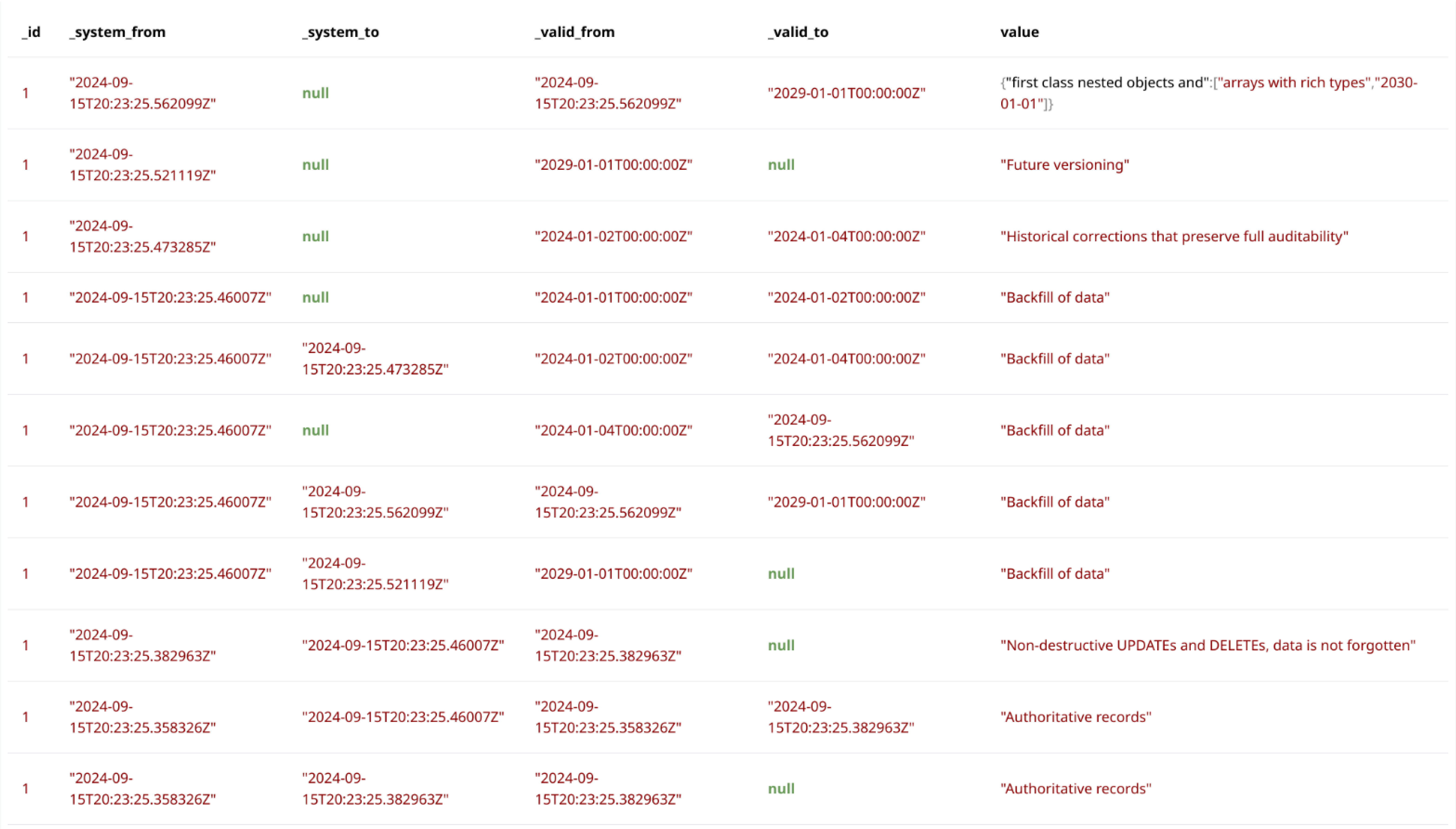
Figure 4: Temporal Relation

Scheme Evolution and the Relational Algebra
May 1988



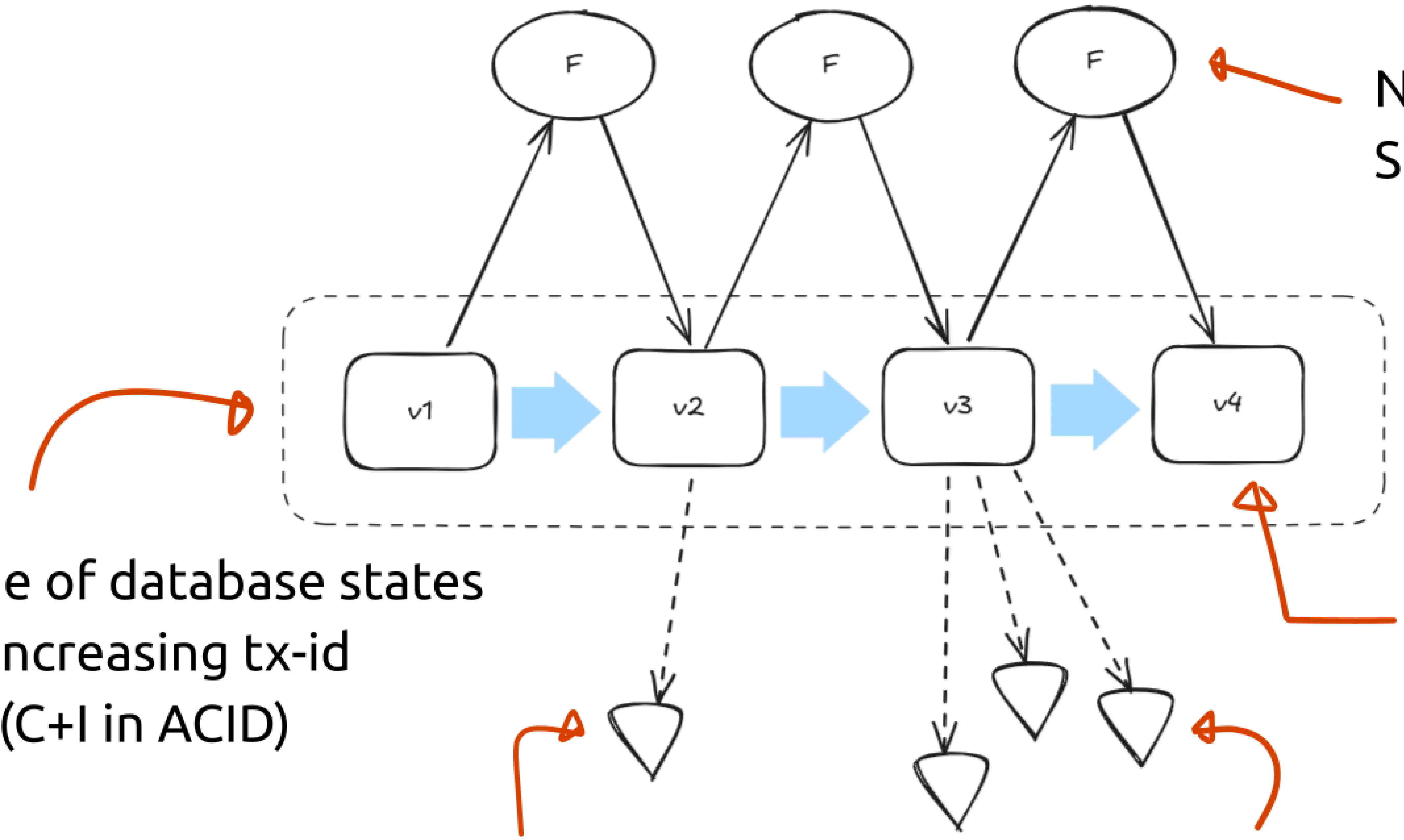
A database *scheme* describes the *structure* of the database; the *contents* of the database must adhere to that structure [Date 1976, Ullman 1982]. *Scheme evolution* refers to changes to the scheme of a database over time. Conventional databases allow only one scheme to be in force at a time, requiring *restructuring* (also termed *logical reorganization* [Sockut & Goldberg 1979]) when the scheme is modified. With the advent of databases storing past states [McKenzie 1986], it becomes desirable to accommodate multiple schemes, each in effect for an interval of time in the past.

A principled model of change



_id	_system_from	_system_to	_valid_from	_valid_to	value
1	"2024-09-15T20:23:25.562099Z"	null	"2024-09-15T20:23:25.562099Z"	"2029-01-01T00:00:00Z"	{"first class nested objects and":["arrays with rich types","2030-01-01"]}
1	"2024-09-15T20:23:25.521119Z"	null	"2029-01-01T00:00:00Z"	null	"Future versioning"
1	"2024-09-15T20:23:25.473285Z"	null	"2024-01-02T00:00:00Z"	"2024-01-04T00:00:00Z"	"Historical corrections that preserve full auditability"
1	"2024-09-15T20:23:25.46007Z"	null	"2024-01-01T00:00:00Z"	"2024-01-02T00:00:00Z"	"Backfill of data"
1	"2024-09-15T20:23:25.46007Z"	"2024-09-15T20:23:25.473285Z"	"2024-01-02T00:00:00Z"	"2024-01-04T00:00:00Z"	"Backfill of data"
1	"2024-09-15T20:23:25.46007Z"	null	"2024-01-04T00:00:00Z"	"2024-09-15T20:23:25.562099Z"	"Backfill of data"
1	"2024-09-15T20:23:25.46007Z"	"2024-09-15T20:23:25.562099Z"	"2024-09-15T20:23:25.562099Z"	"2029-01-01T00:00:00Z"	"Backfill of data"
1	"2024-09-15T20:23:25.46007Z"	"2024-09-15T20:23:25.521119Z"	"2029-01-01T00:00:00Z"	null	"Backfill of data"
1	"2024-09-15T20:23:25.382963Z"	"2024-09-15T20:23:25.46007Z"	"2024-09-15T20:23:25.382963Z"	null	"Non-destructive UPDATEs and DELETEs, data is not forgotten"
1	"2024-09-15T20:23:25.358326Z"	"2024-09-15T20:23:25.46007Z"	"2024-09-15T20:23:25.358326Z"	"2024-09-15T20:23:25.382963Z"	"Authoritative records"
1	"2024-09-15T20:23:25.358326Z"	"2024-09-15T20:23:25.382963Z"	"2024-09-15T20:23:25.382963Z"	null	"Authoritative records"

In XTDB, SQL:2011 SYSTEM_TIME provides database-wide immutability ("The Epochal Time Model"):



Non-interactive
SQL transactions

A single timeline of database states
Monotonically increasing tx-id
Fully serialized (C+I in ACID)

Granular versioning
Every table
Universal query 'basis'
AS OF <TIMESTAMP>
Repeatable queries

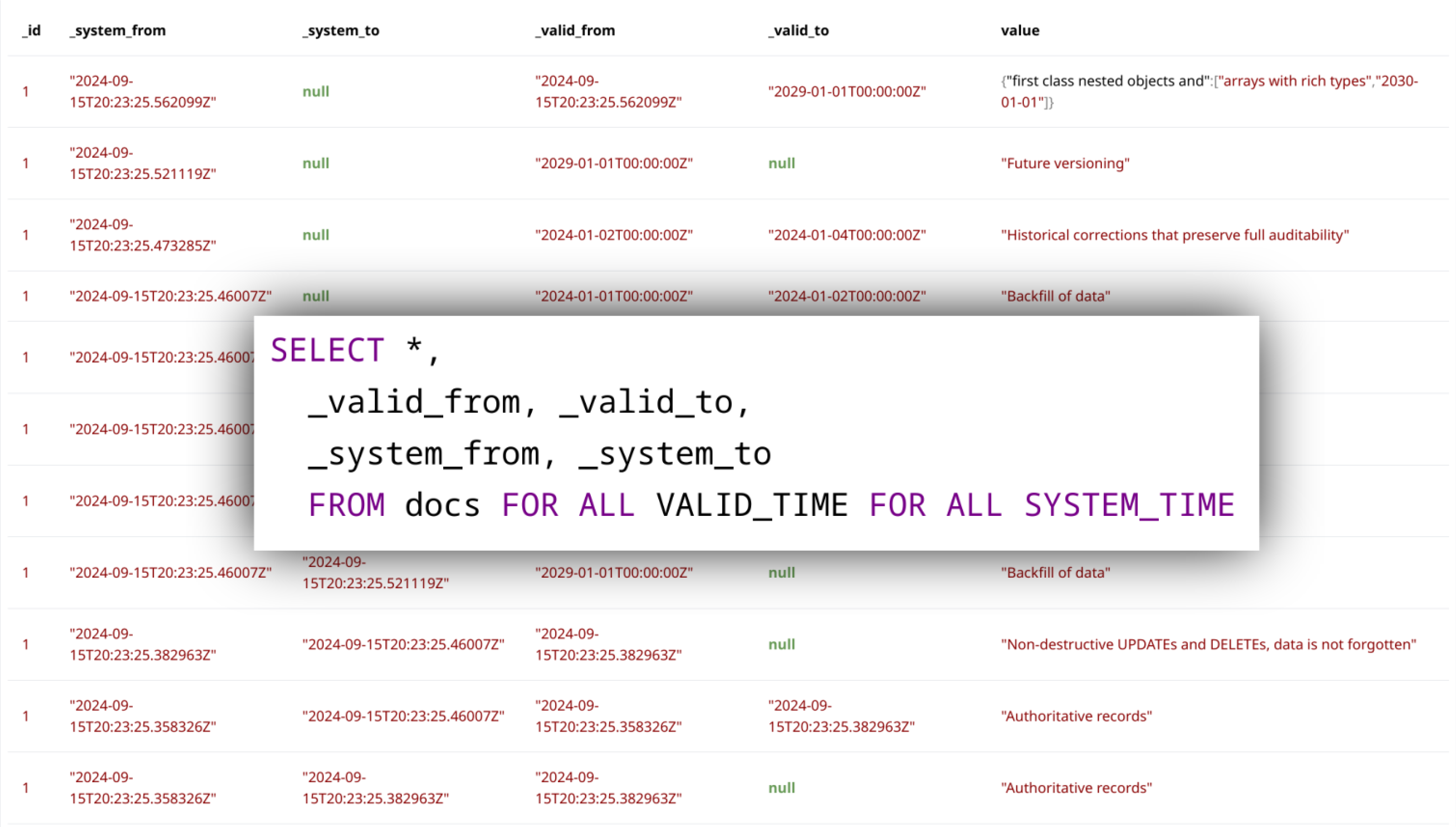
Unlimited horizontal scaling of read-only 'transactions'
No locking or other impact on writes

Everything is implicitly filtered "as-of now", by default.

```
SELECT *,
  _valid_from, _valid_to,
  _system_from, _system_to
  FROM docs
```

_id	_system_from	_valid_from	_valid_to	value
1	"2024-09-15T20:25:13.515361Z"	"2024-09-15T20:25:13.515361Z"	"2029-01-01T00:00:00Z"	{"first class nested objects and":["arrays with rich types","2030-01-01"]}

Specify a filter to override the as-of now filter:

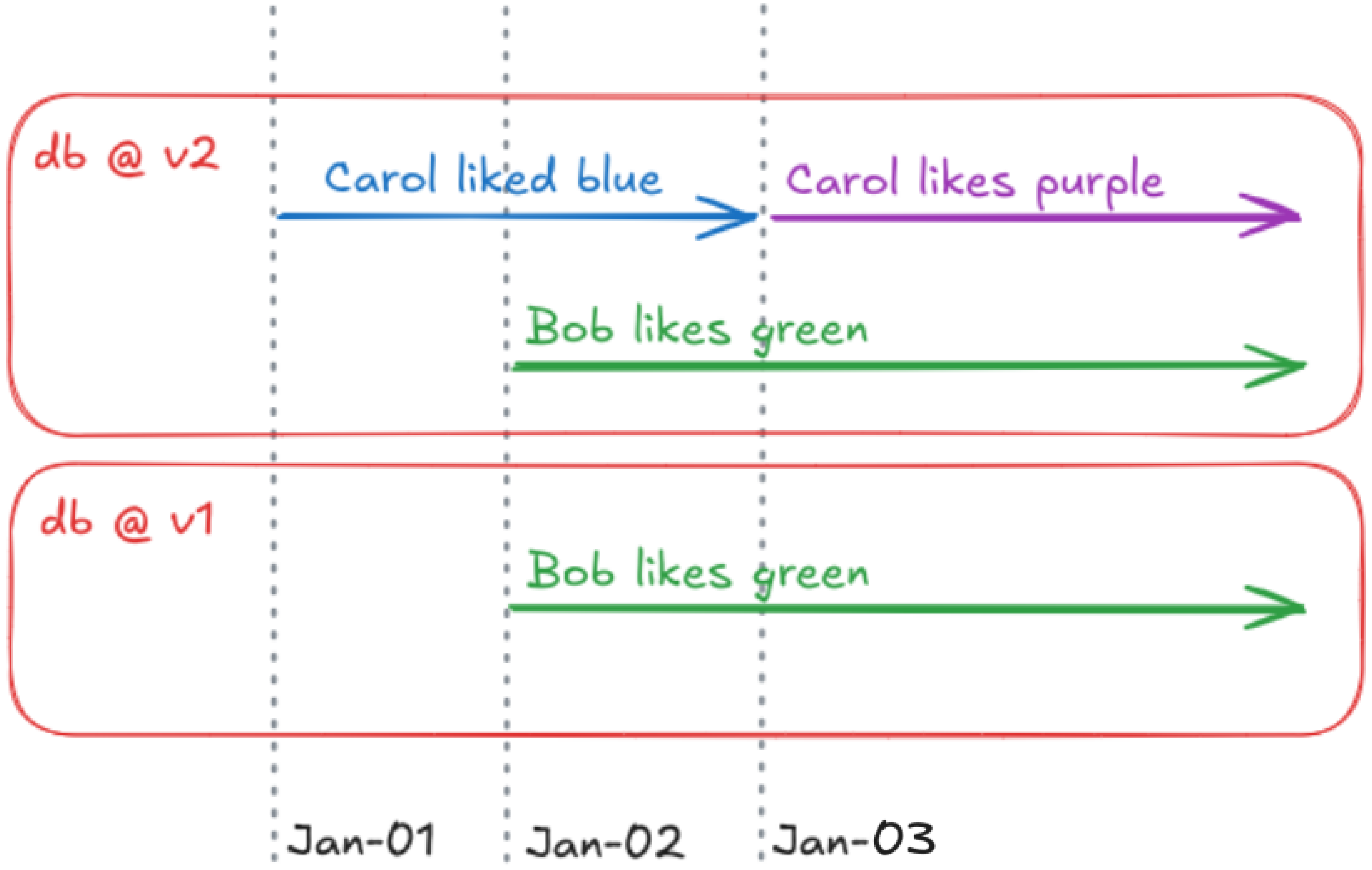


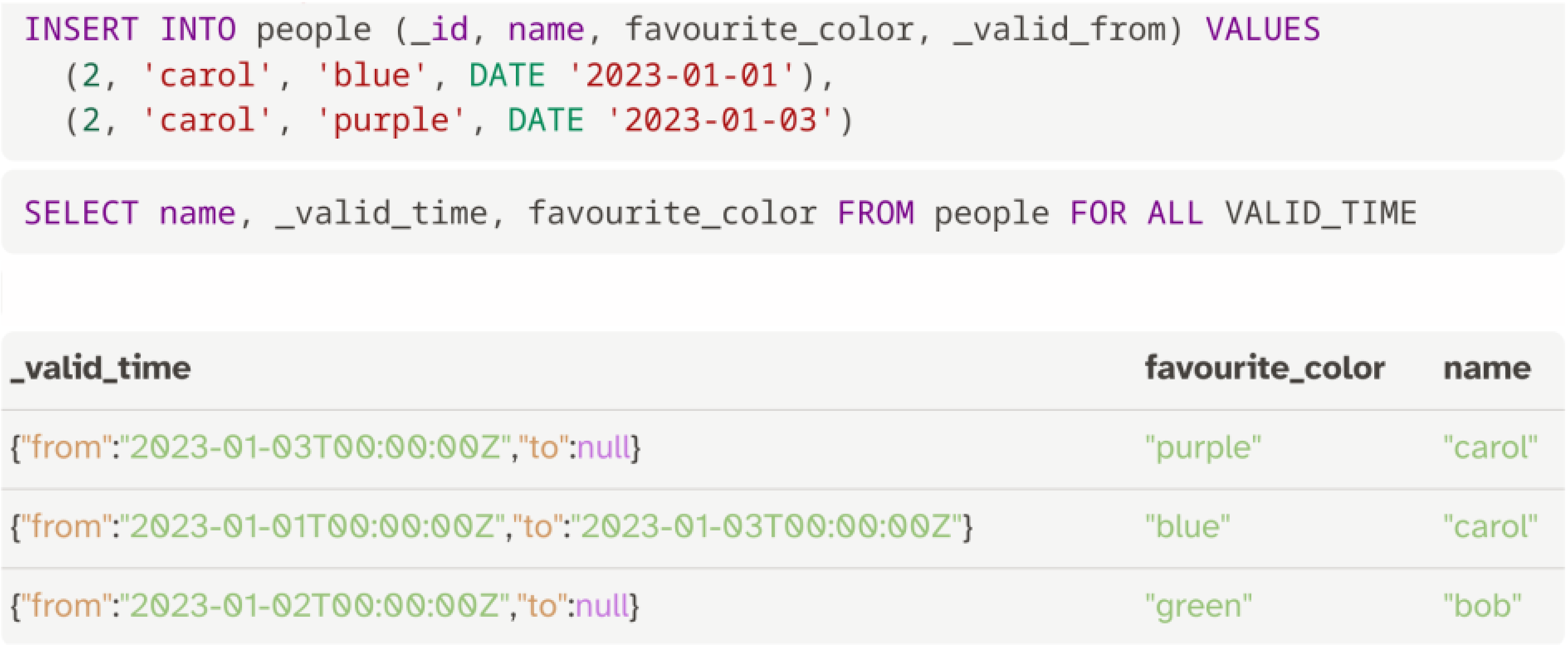
_id	_system_from	_system_to	_valid_from	_valid_to	value
1	"2024-09-15T20:23:25.562099Z"	null	"2024-09-15T20:23:25.562099Z"	"2029-01-01T00:00:00Z"	{"first class nested objects and":["arrays with rich types","2030-01-01"]}
1	"2024-09-15T20:23:25.521119Z"	null	"2029-01-01T00:00:00Z"	null	"Future versioning"
1	"2024-09-15T20:23:25.473285Z"	null	"2024-01-02T00:00:00Z"	"2024-01-04T00:00:00Z"	"Historical corrections that preserve full auditability"
1	"2024-09-15T20:23:25.46007Z"	null	"2024-01-01T00:00:00Z"	"2024-01-02T00:00:00Z"	"Backfill of data"
1	"2024-09-15T20:23:25.4600				
1	"2024-09-15T20:23:25.4600				
1	"2024-09-15T20:23:25.4600				
1	"2024-09-15T20:23:25.46007Z"	"2024-09-15T20:23:25.521119Z"	"2029-01-01T00:00:00Z"	null	"Backfill of data"
1	"2024-09-15T20:23:25.382963Z"	"2024-09-15T20:23:25.46007Z"	"2024-09-15T20:23:25.382963Z"	null	"Non-destructive UPDATEs and DELETEs, data is not forgotten"
1	"2024-09-15T20:23:25.358326Z"	"2024-09-15T20:23:25.46007Z"	"2024-09-15T20:23:25.358326Z"	"2024-09-15T20:23:25.382963Z"	"Authoritative records"
1	"2024-09-15T20:23:25.358326Z"	"2024-09-15T20:23:25.382963Z"	"2024-09-15T20:23:25.382963Z"	null	"Authoritative records"

```
SELECT *,
  _valid_from, _valid_to,
  _system_from, _system_to
  FROM docs FOR ALL VALID_TIME FOR ALL SYSTEM_TIME
```

The mutable VALID_TIME dimension enables:

- backfilling history from upstream sources
- corrections
- scheduled effectivity
- time-travel for business users (~debugging)



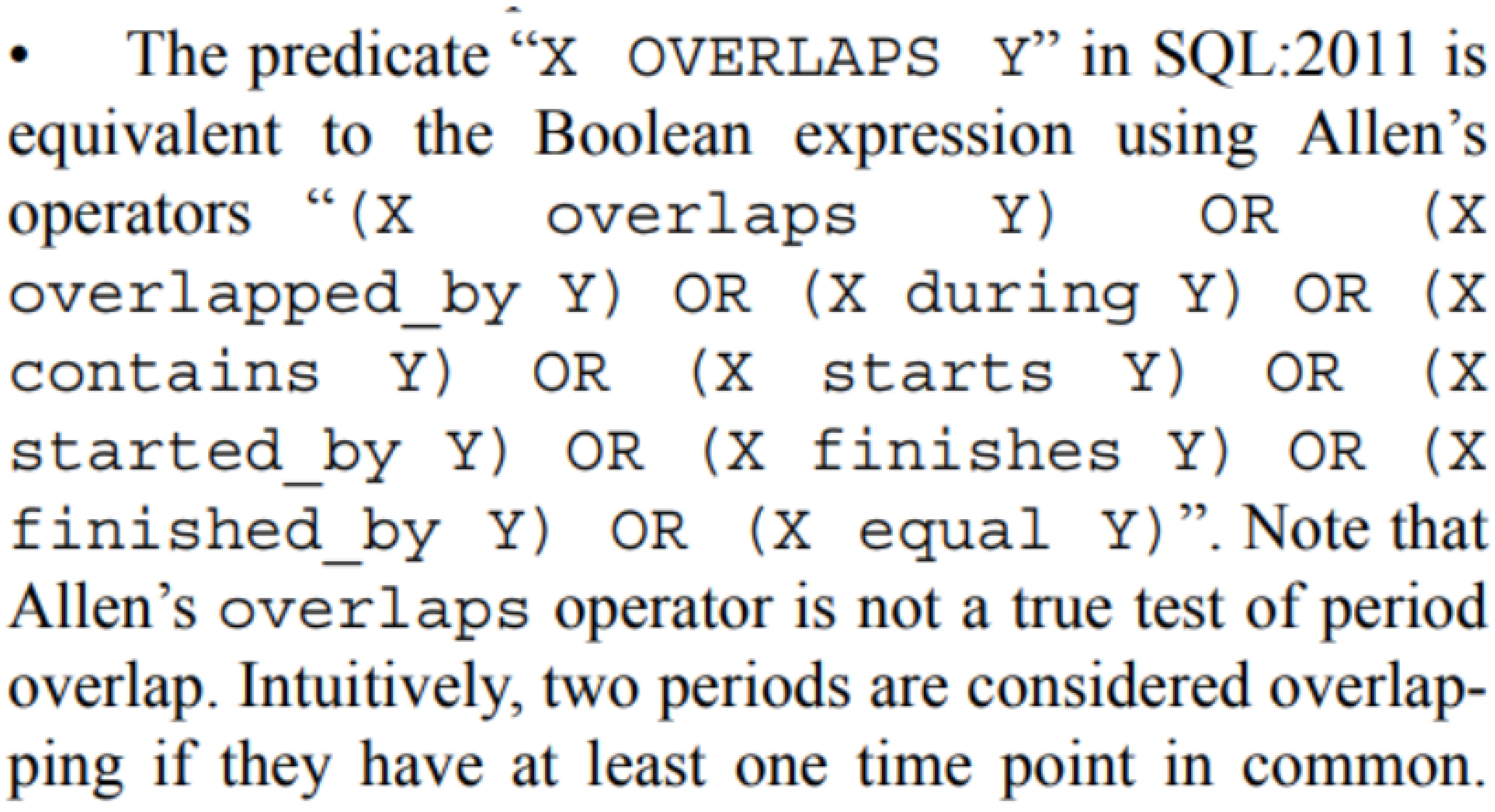


```sql
INSERT INTO people (_id, name, favourite_color, _valid_from) VALUES
  (2, 'carol', 'blue', DATE '2023-01-01'),
  (2, 'carol', 'purple', DATE '2023-01-03')
```

```sql
SELECT name, _valid_time, favourite_color FROM people FOR ALL VALID_TIME
```

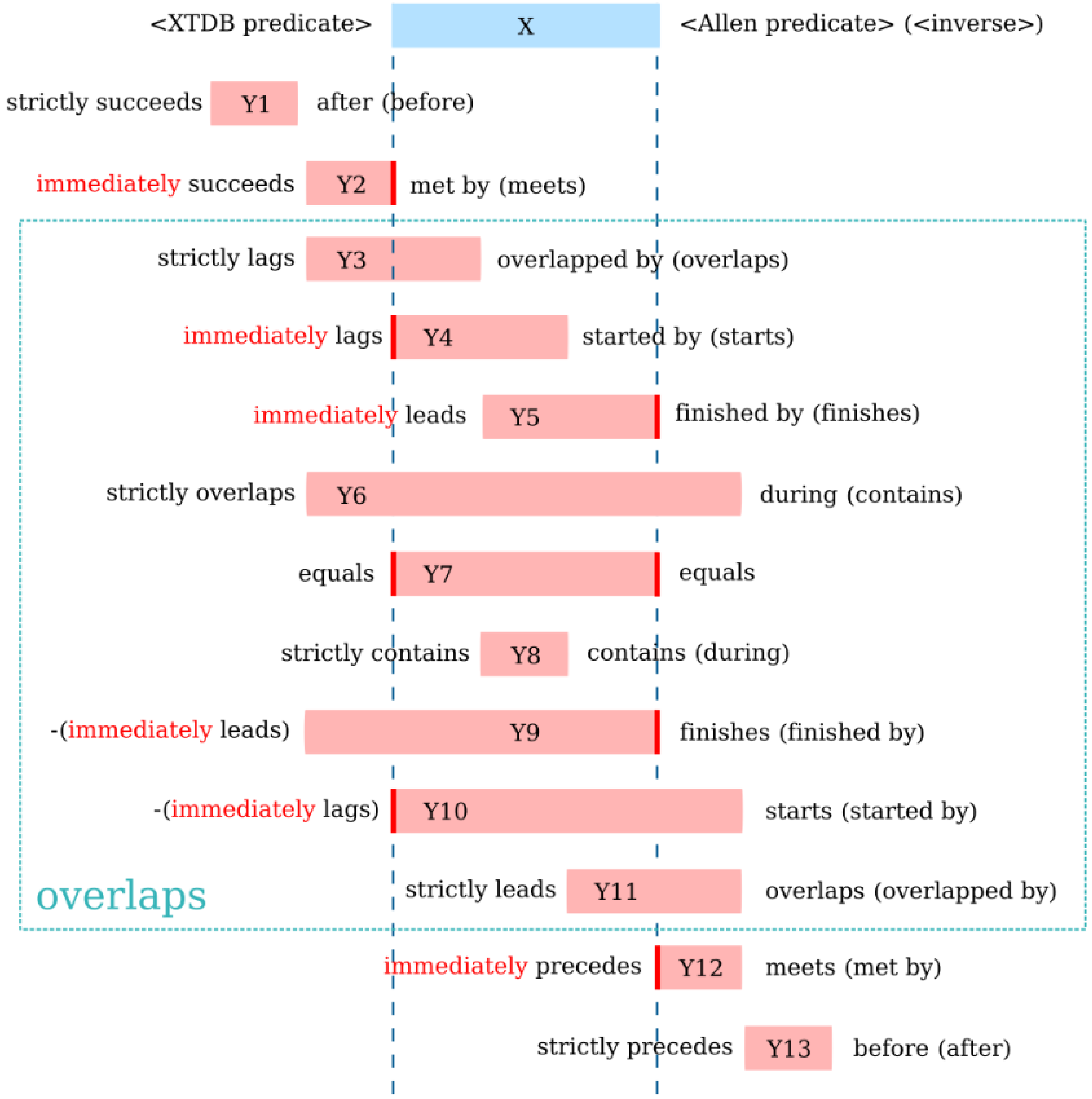
_valid_time	favourite_color	name
{"from":"2023-01-03T00:00:00Z","to":null}	"purple"	"carol"
{"from":"2023-01-01T00:00:00Z","to":"2023-01-03T00:00:00Z"}	"blue"	"carol"
{"from":"2023-01-02T00:00:00Z","to":null}	"green"	"bob"

"Change Data Integration"



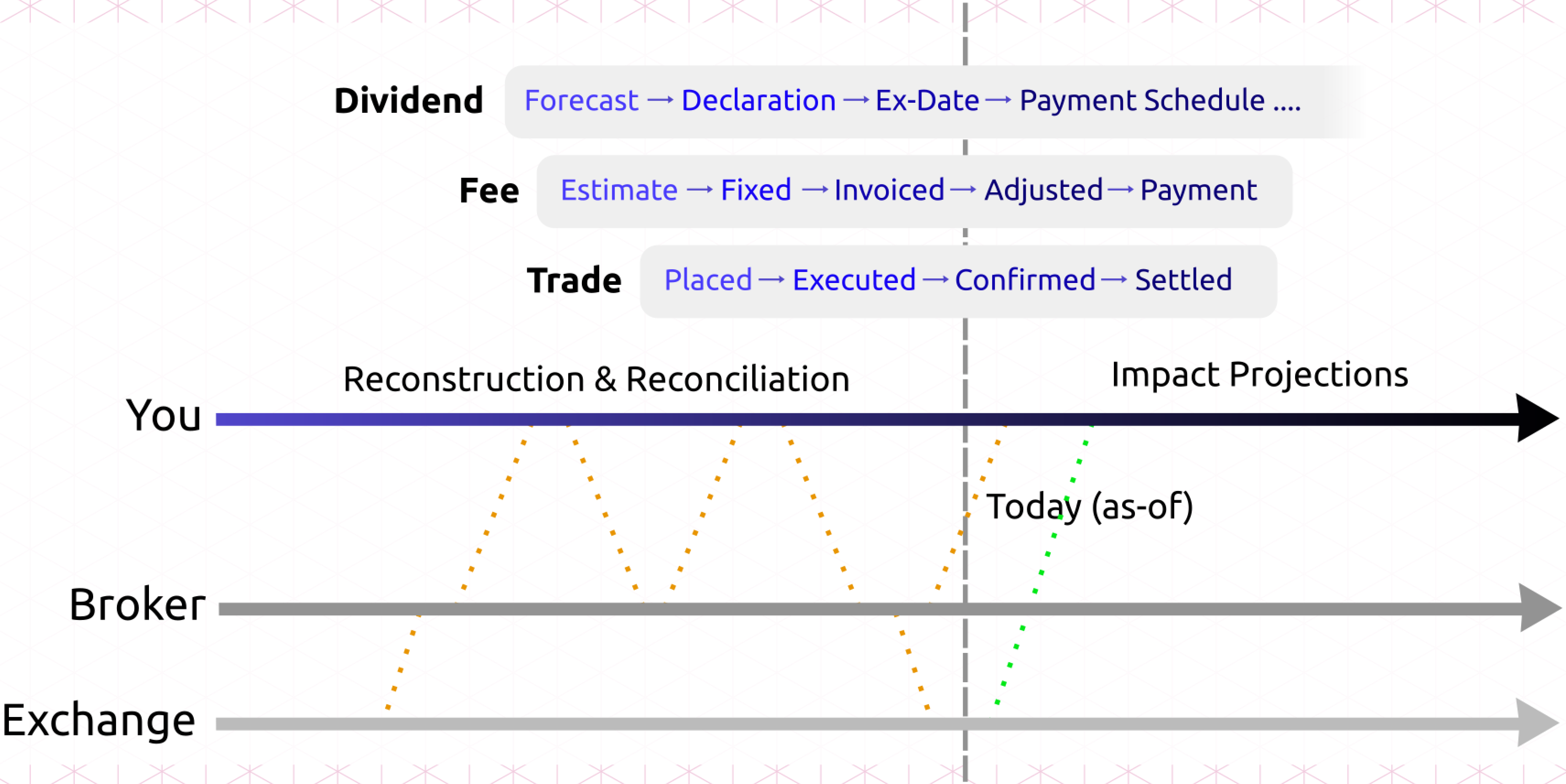
- The predicate "X OVERLAPS Y" in SQL:2011 is equivalent to the Boolean expression using Allen's operators "(X overlaps Y) OR (X overlapped_by Y) OR (X during Y) OR (X contains Y) OR (X starts Y) OR (X started_by Y) OR (X finishes Y) OR (X finished_by Y) OR (X equal Y)". Note that Allen's overlaps operator is not a true test of period overlap. Intuitively, two periods are considered overlapping if they have at least one time point in common.

"Temporal features in SQL:2011"
(Kulkarni, Michels)



Reconciliation

Track Complex, Evolving States & Positions



Dividend Forecast → Declaration → Ex-Date → Payment Schedule ....

Fee Estimate → Fixed → Invoiced → Adjusted → Payment

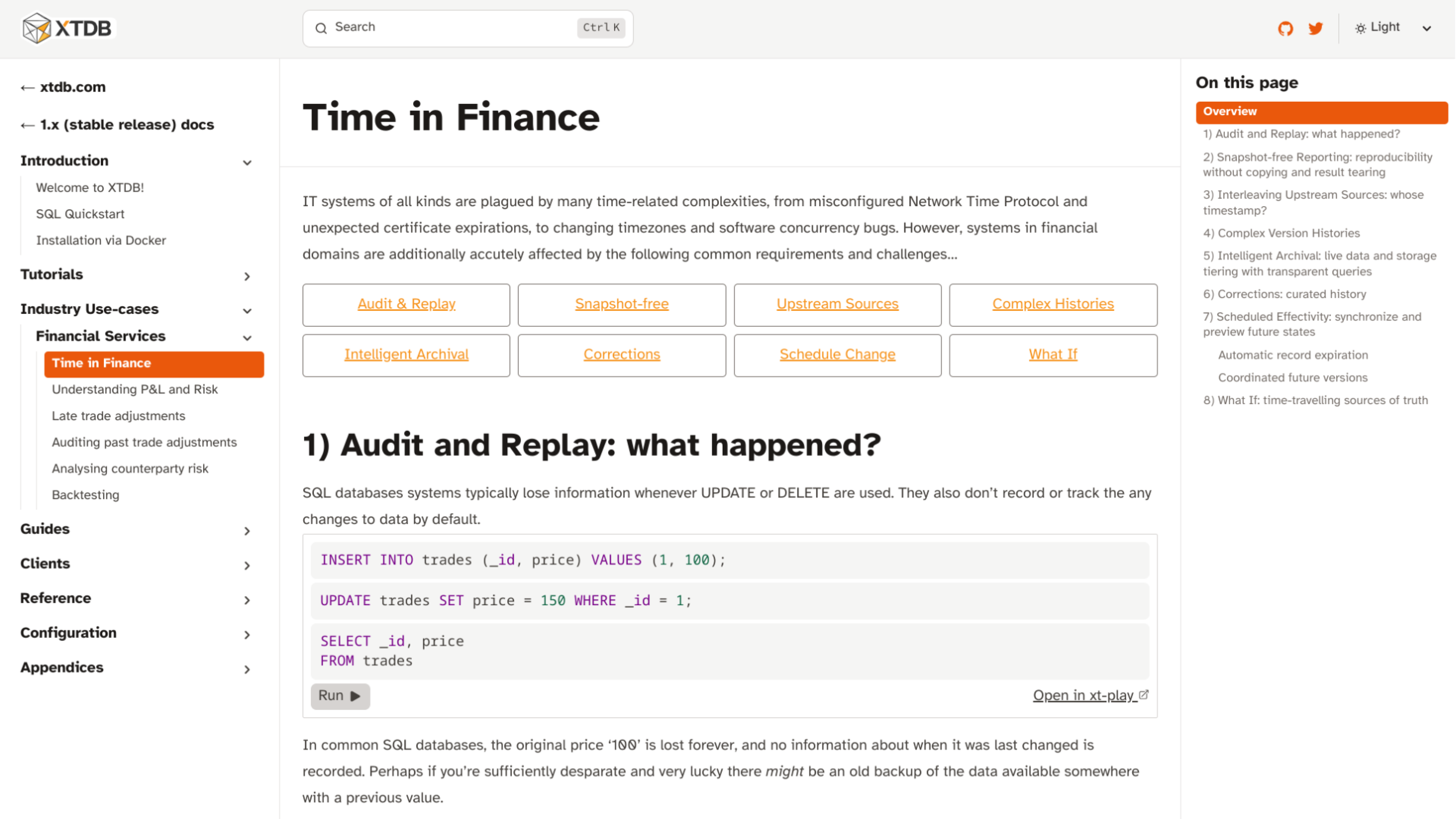
Trade Placed → Executed → Confirmed → Settled

Reconstruction & Reconciliation

Impact Projections

Today (as-of)

Risk



← xtdb.com
← 1.x (stable release) docs
Introduction
Welcome to XTDB!
SQL Quickstart
Installation via Docker
Tutorials
Industry Use-cases
Financial Services
Time in Finance
Understanding P&L and Risk
Late trade adjustments
Auditing past trade adjustments
Analysing counterparty risk
Backtesting
Guides
Clients
Reference
Configuration
Appendices

Time in Finance

IT systems of all kinds are plagued by many time-related complexities, from misconfigured Network Time Protocol and unexpected certificate expirations, to changing timezones and software concurrency bugs. However, systems in financial domains are additionally accutely affected by the following common requirements and challenges…

Audit & Replay	Snapshot-free	Upstream Sources	Complex Histories
Intelligent Archival	Corrections	Schedule Change	What If

1) Audit and Replay: what happened?

SQL databases systems typically lose information whenever UPDATE or DELETE are used. They also don't record or track the any changes to data by default.

```
INSERT INTO trades (_id, price) VALUES (1, 100);
```

```
UPDATE trades SET price = 150 WHERE _id = 1;
```

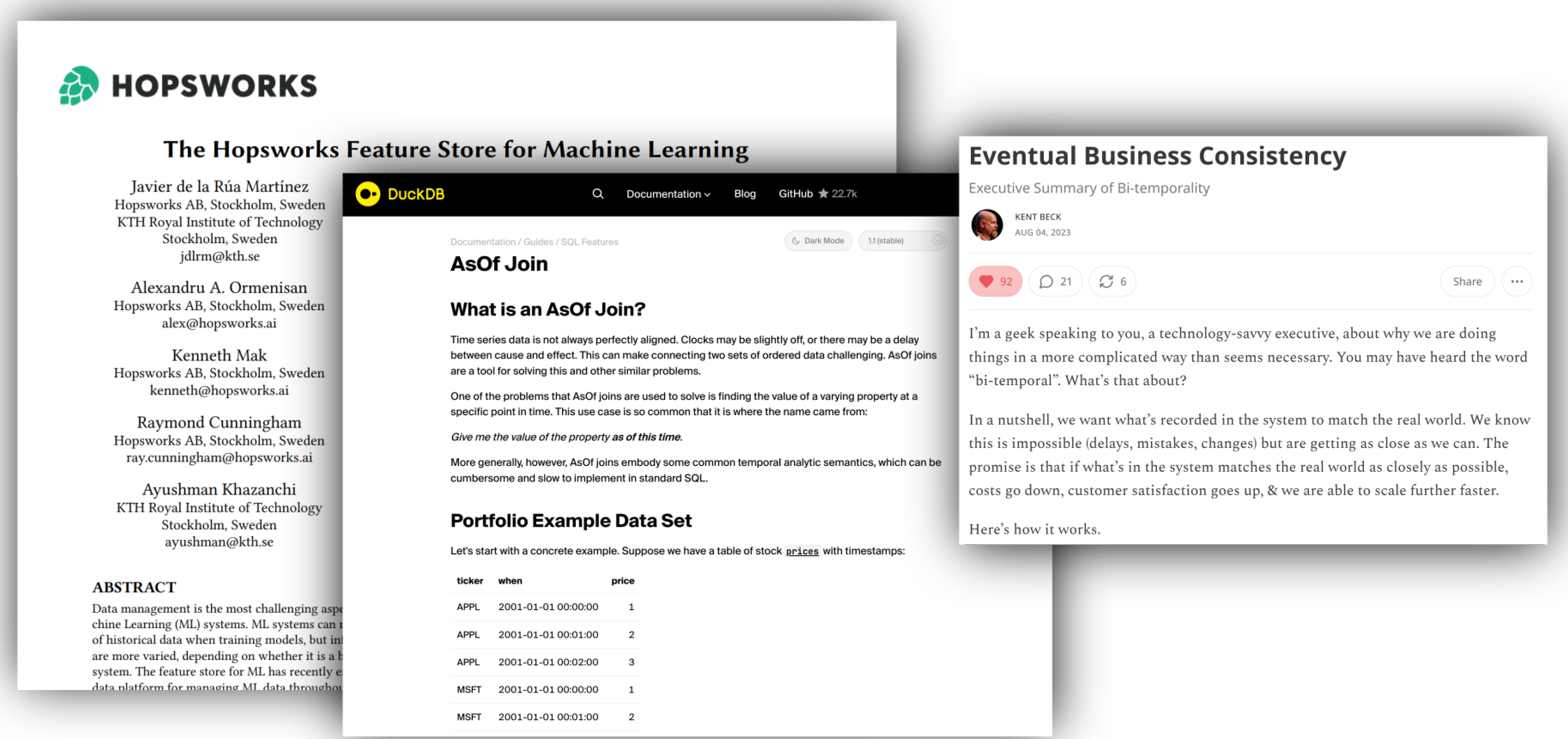
```
SELECT _id, price
FROM trades
```

In common SQL databases, the original price '100' is lost forever, and no information about when it was last changed is recorded. Perhaps if you're sufficiently desparate and very lucky there *might* be an old backup of the data available somewhere with a previous value.

On this page
Overview
1) Audit and Replay: what happened?
2) Snapshot-free Reporting: reproducibility without copying and result tearing
3) Interleaving Upstream Sources: whose timestamp?
4) Complex Version Histories
5) Intelligent Archival: live data and storage tiering with transparent queries
6) Corrections: curated history
7) Scheduled Effectivity: synchronize and preview future states
Automatic record expiration
Coordinated future versions
8) What If: time-travelling sources of truth

Machine Learning

Many people need *accurate* time-travel.



HOPSWORKS

The Hopsworks Feature Store for Machine Learning

Javier de la Rúa Martínez
Hopsworks AB, Stockholm, Sweden
KTH Royal Institute of Technology
Stockholm, Sweden
jdlrm@kth.se

Alexandru A. Ormenisan
Hopsworks AB, Stockholm, Sweden
alex@hopsworks.ai

Kenneth Mak
Hopsworks AB, Stockholm, Sweden
kenneth@hopsworks.ai

Raymond Cunningham
Hopsworks AB, Stockholm, Sweden
ray.cunningham@hopsworks.ai

Ayushman Khazanchi
KTH Royal Institute of Technology
Stockholm, Sweden
ayushman@kth.se

ABSTRACT

Data management is the most challenging aspe
chine Learning (ML) systems. ML systems can r
of historical data when training models, but inf
are more varied, depending on whether it is a b
system. The feature store for ML has recently e
data platform for managing ML data throughou

Documentation / Guides / SQL Features

AsOf Join

What is an AsOf Join?

Time series data is not always perfectly aligned. Clocks may be slightly off, or there may be a delay between cause and effect. This can make connecting two sets of ordered data challenging. AsOf joins are a tool for solving this and other similar problems.

One of the problems that AsOf joins are used to solve is finding the value of a varying property at a specific point in time. This use case is so common that it is where the name came from:

Give me the value of the property ***as of this time.***

More generally, however, AsOf joins embody some common temporal analytic semantics, which can be cumbersome and slow to implement in standard SQL.

Portfolio Example Data Set

Let's start with a concrete example. Suppose we have a table of stock `prices` with timestamps:

ticker	when	price
APPL	2001-01-01 00:00:00	1
APPL	2001-01-01 00:01:00	2
APPL	2001-01-01 00:02:00	3
MSFT	2001-01-01 00:00:00	1
MSFT	2001-01-01 00:01:00	2

Eventual Business Consistency

Executive Summary of Bi-temporality

KENT BECK
AUG 04, 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?

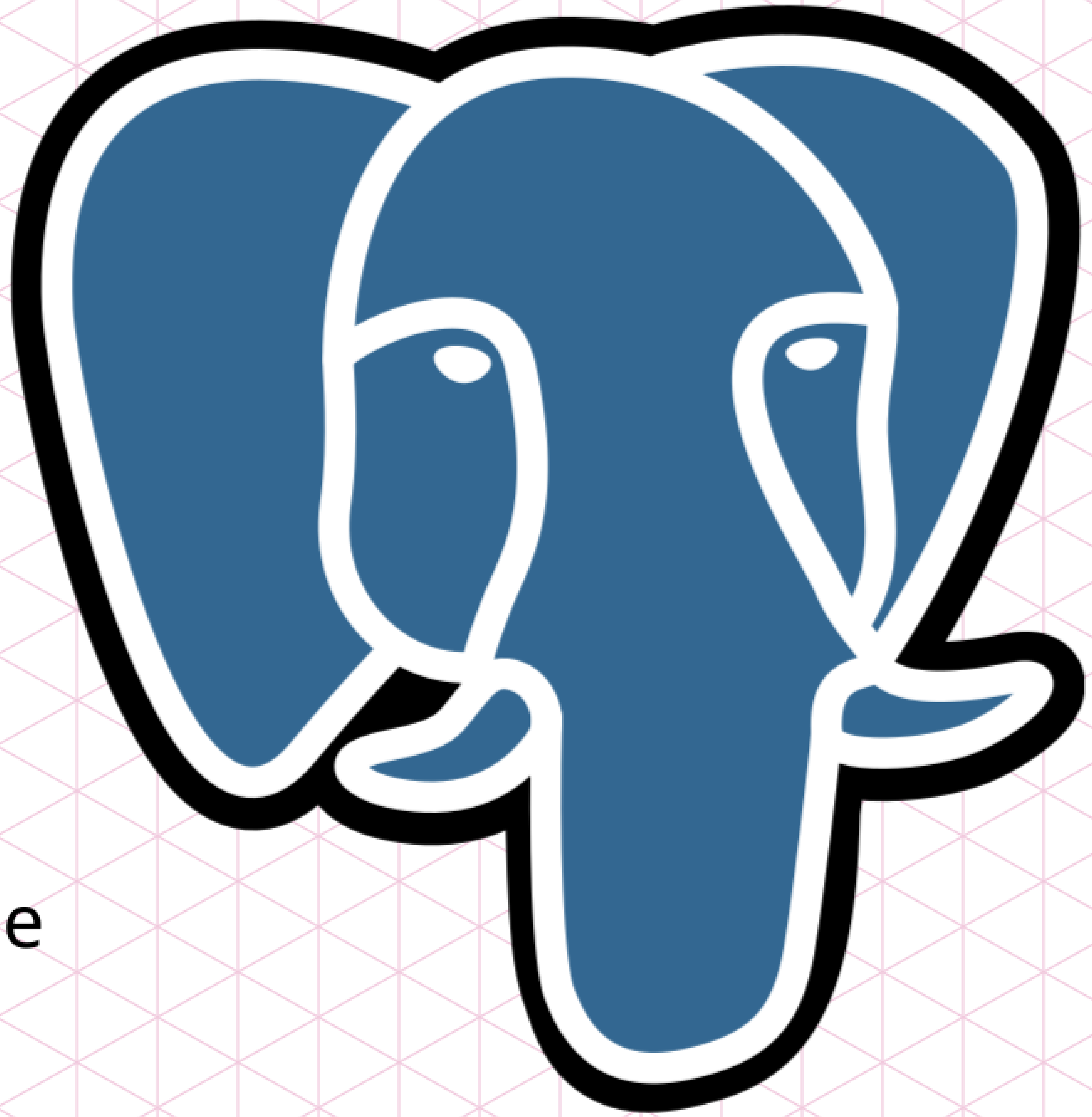
In a nutshell, we want what's recorded in the system to match the real world. We know this is impossible (delays, mistakes, changes) but are getting as close as we can. The promise is that if what's in the system matches the real world as closely as possible, costs go down, customer satisfaction goes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

Here's how it works.

PostgreSQL

✓ ~Low cost and ubiquitous

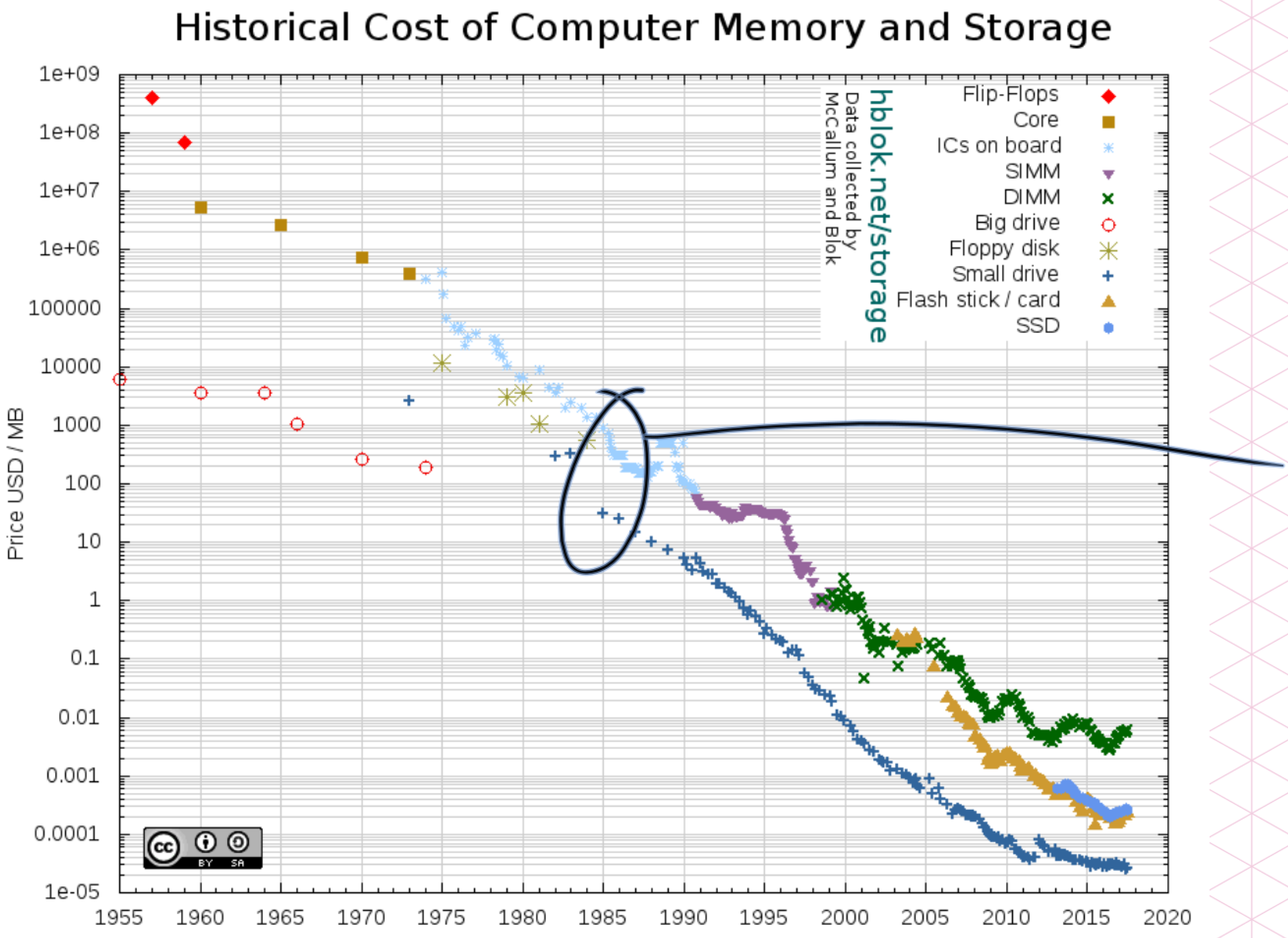
✓ The popular default



✓ Battle hardened
and highly extensible

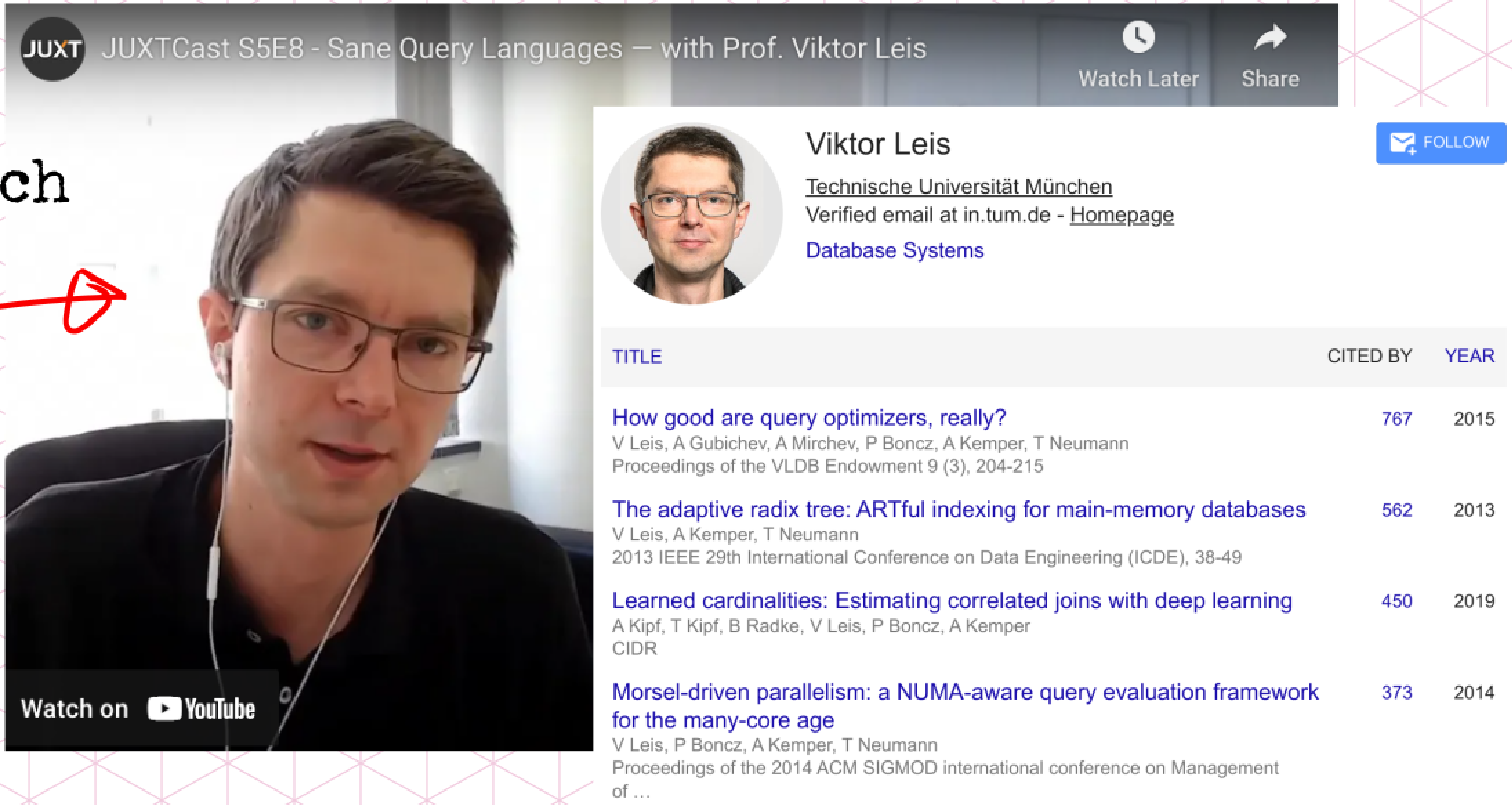
✓ Architected by
Michael Stonebraker

✗ Does not support bitemporal data



Postgres was originally designed in 1986

Prof. Viktor Leis
Technical University of Munich



JUXTCast S5E8 - Sane Query Languages — with Prof. Viktor Leis

Viktor Leis
Technische Universität München
Verified email at in.tum.de - Homepage
Database Systems

TITLE	CITED BY	YEAR
How good are query optimizers, really? V Leis, A Gubichev, A Mirchev, P Boncz, A Kemper, T Neumann Proceedings of the VLDB Endowment 9 (3), 204-215	767	2015
The adaptive radix tree: ARTful indexing for main-memory databases V Leis, A Kemper, T Neumann 2013 IEEE 29th International Conference on Data Engineering (ICDE), 38-49	562	2013
Learned cardinalities: Estimating correlated joins with deep learning A Kipf, T Kipf, B Radke, V Leis, P Boncz, A Kemper CIDR	450	2019
Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age V Leis, P Boncz, A Kemper, T Neumann Proceedings of the 2014 ACM SIGMOD international conference on Management of ...	373	2014

What do you think about the future of Postgres?

"It's just a very old system that's extremely inefficent [...]
there's many orders of magnitude for things to be more efficient usually [...]
there's no single kind of performance bottleneck, it's just that everything over time has accumulated so much performance overhead [...]

I don't think that it's really possible to save it."

```
$ docker run -p 5432:5432 ghcr.io/xtdb/xtdb:2.0.0b1
$ psql -h localhost
psql (16.3, server 16)
Type "help" for help.

jdt=> select xt.version();
    version
----------------
 XTDB @ 2.0.0b1
(1 row)

jdt=>
```

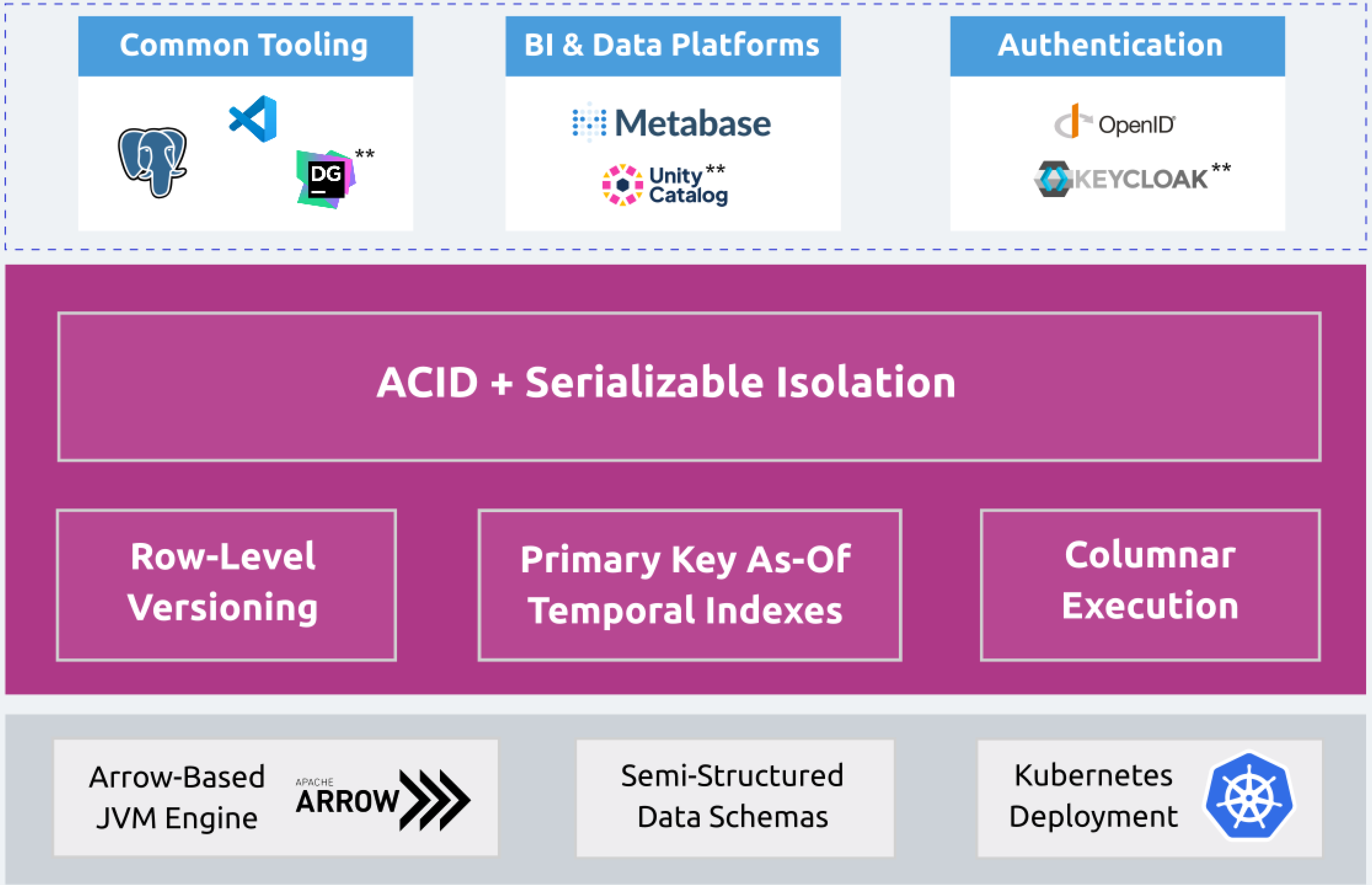
How XTDB Accelerates As-Of Reporting

Any language
Postgres Driver Compatibility

Standard SQL
Transactional Applications

All History
Hybrid Analytical Workloads

Any Cloud
Architected for Object Storage



Common Tooling **

BI & Data Platforms
Metabase
Unity Catalog **

Authentication
OpenID
KEYCLOAK **

ACID + Serializable Isolation

Row-Level Versioning

Primary Key As-Of Temporal Indexes

Columnar Execution

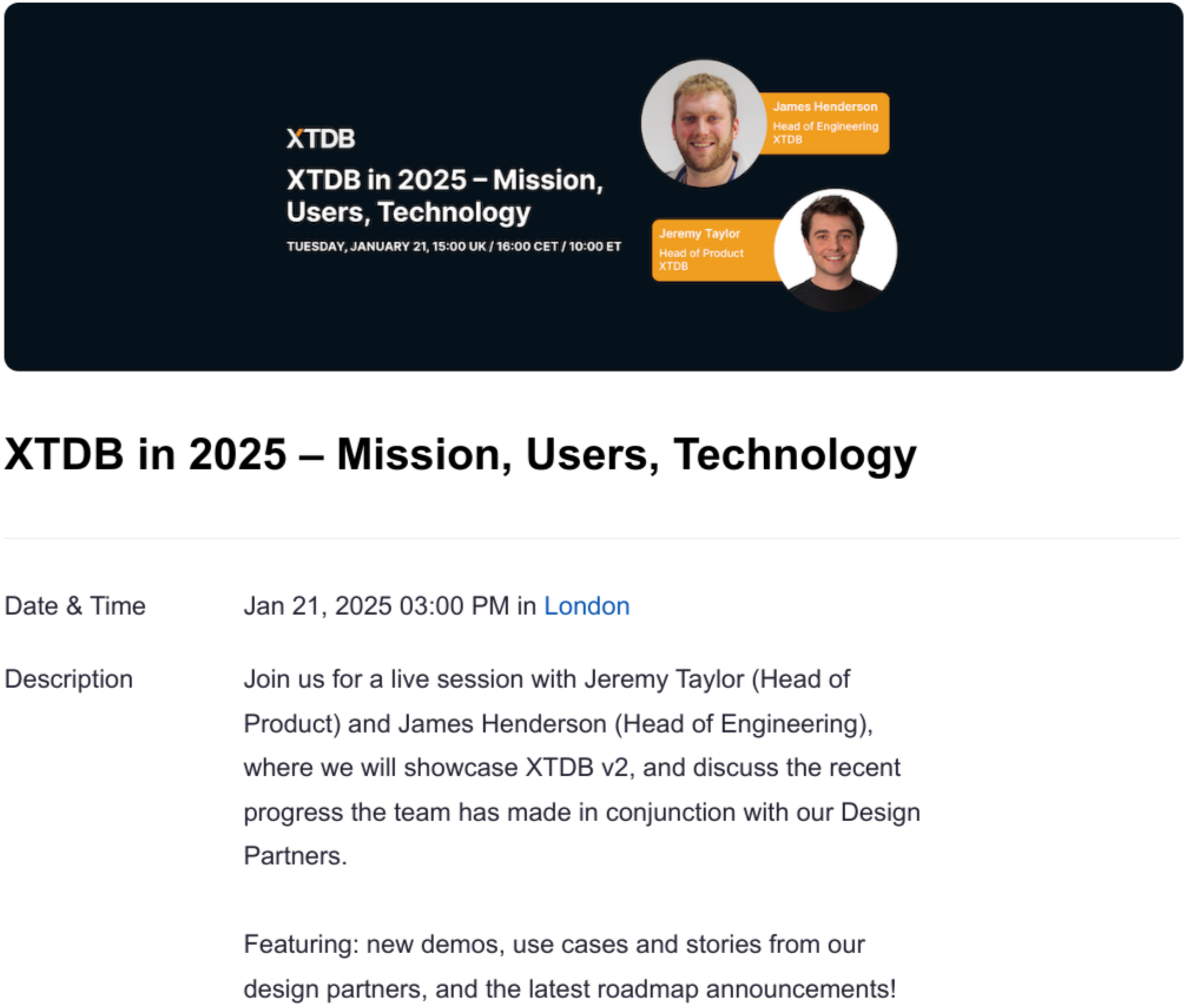
Arrow-Based JVM Engine
APACHE ARROW

Semi-Structured Data Schemas

Kubernetes Deployment

** Roadmap, Subject to Change (DISCLAIMER)

Join us again in January to hear from our users and discover our plans for 2025.


XTDB
XTDB in 2025 – Mission, Users, Technology
TUESDAY, JANUARY 21, 15:00 UK / 16:00 CET / 10:00 ET
James Henderson
Head of Engineering
XTDB
Jeremy Taylor
Head of Product
XTDB

XTDB in 2025 – Mission, Users, Technology

Date & Time Jan 21, 2025 03:00 PM in London

Description Join us for a live session with Jeremy Taylor (Head of Product) and James Henderson (Head of Engineering), where we will showcase XTDB v2, and discuss the recent progress the team has made in conjunction with our Design Partners.

Featuring: new demos, use cases and stories from our design partners, and the latest roadmap announcements!

https://us06web.zoom.us/webinar/register/4817339038090/WN_QqygMHteRvmPo0O4aQDW5Q

Thanks for listening! Any questions?

Try out XTDB v2 live in your browser right now:
https://docs.xtdb.com

Do you have (bi)temporal problems?
We're looking for Design Partners - let's chat:
>> **hello@xtdb.com** <<

Jeremy Taylor | JUXT | @refset

FINOS

JUXT
A GRID DYNAMICS COMPANY