XTDB

A First Look at XTDB v2

Join James Henderson — Head of Engineering at XTDB — as he lifts the lid on XT2 development

TUESDAY, DEC 19 @ 11:00 ET / 16:00 GMT / 17:00 CET

James Henderson
Head of Engineering
XTDB

Jeremy Taylor
Head of Product
XTDB

For the interactive demo:

1. `git clone https://github.com/xtdb/sakila-playground.git`
2. `cd sakila-playground; clojure -P  # download deps`

Coming up:

- What is XTDB v2?
- An interactive introduction to **XTQL**
- The lifecycle of XT2 data
- How you can get involved in the journey to GA

Primary aims for XTDB v2

- Hybrid transactional/analytical processing ('HTAP')
  - Apache Arrow
  - 'Separating storage from compute'
- Across-time bitemporal queries
  - 'Full' bitemporality
- Addition of first-class SQL:2011 support
  - Introducing XTQL!

Primary aims for XTQL

- Data-oriented query language
- Composable
- JSON + EDN dialects
  - client libraries for Java, JavaScript, Python etc to follow soon
- Comparable to SQL, familiar to EDN Datalog

Demo: ‘Sakila’ (MySQL) film rental store playground

1. `git clone https://github.com/xtdb/sakila-playground.git`
2. `clj -M:xtdb:nrepl`
   - connect to the nREPL in your favourite editor
   - Or, in Emacs: `cider-jack-in-clj`
3. `user.clj, (xt/q xt-node ‘(<your-xtql-query>))`

Guide: https://docs.xtdb.com/intro/what-is-xtql.html

Docs: https://docs.xtdb.com/reference/main/xtql/queries.html

from

```
SELECT f.title, f.length, f.rating FROM film f
```

```
(from :film [title length rating])
```

https://docs.xtdb.com/intro/what-is-xtql.html#_from

Pipelining

```
SELECT f.title, f.length, f.rating FROM film f ORDER BY f.title LIMIT 10
```

```
(-> (from :film [title length rating])
    (order-by title)
    (limit 10))
```

https://docs.xtdb.com/intro/what-is-xtql.html#_pipelines

Filtering

```
SELECT f.* FROM film f WHERE f.title = 'AFRICAN EGG'
```

```
(from :film [{:title "AFRICAN EGG"} *])
```

https://docs.xtdb.com/reference/main/xtql/queries.html#_from

Parameters

```
(xt/q xt-node "SELECT f.* FROM film f WHERE f.title = ?"
      {:args ["AFRICAN EGG"]})

(xt/q xt-node '(from :film [{:title $title} *])
      {:args {:title "AFRICAN EGG"}})
```

https://docs.xtdb.com/reference/main/xtql/queries.html#_query_options

Joins - `unify` (SQL)

```
SELECT f.xt$id AS film_id, f.title, f.length, f.rating,
       a.xt$id AS actor_id, a.first_name, a.last_name
FROM film f
  JOIN film_actor fa ON (fa.film_id = f.xt$id)
  JOIN actor a ON (fa.actor_id = a.xt$id)
WHERE f.title = 'AFRICAN EGG'
```

Joins - `unify` (XTQL)

```
(unify (from :film [{:xt/id film-id, :title "AFRICAN EGG"}
                    title length rating])
       (from :film-actor [{:film-id film-id, :actor-id actor-id}])
       (from :actor [{:xt/id actor-id} first-name last-name]))
```

https://docs.xtdb.com/intro/what-is-xtql.html#unify

Projections (SQL)

```
SELECT f.xt$id AS film_id, f.title, f.length, f.rating,
       a.xt$id AS actor_id,
       (a.first_name || ' ' || a.last_name) AS name
FROM film f
  JOIN film_actor fa ON (fa.film_id = f.xt$id)
  JOIN actor a ON (fa.actor_id = a.xt$id)
WHERE f.title = 'AFRICAN EGG'
```

Projections (XTQL)

```
(-> (unify (from :film [{:xt/id film-id, :title "AFRICAN EGG"}
                        title length rating])
           (from :film-actor [{:film-id film-id, :actor-id actor-id}])
           (from :actor [{:xt/id actor-id} first-name last-name]))
    (return title length rating
            {:name (concat first-name " " last-name)}))
```

https://docs.xtdb.com/reference/main/xtql/queries.html#_return

Nested projections (desired result)

```
[{:title "AFRICAN EGG",
  :length 130,
  :actors [{:name "MATTHEW CARREY"}
           {:name "MATTHEW LEIGH"}
           {:name "GARY PHOENIX"}
           {:name "DUSTIN TAUTOU"}
           {:name "THORA TEMPLE"}],
  :rating "G"}]
```

Nested projections (SQL)

```
SELECT f.xt$id AS film_id, f.title, f.length, f.rating,
       (SELECT ARRAY_AGG(
                 OBJECT(
                   'name': (a.first_name || ' ' || a.last_name)))
        FROM film_actor fa
          JOIN actor a ON (fa.actor_id = a.xt$id)
        WHERE fa.film_id = f.xt$id
        ORDER BY a.last_name) AS actor
FROM film f
WHERE f.title = 'AFRICAN EGG'
```

Nested projections - ‘pull’ (XTQL)

```
(-> (from :film [{:xt/id film-id, :title "AFRICAN EGG"} title length rating])
    (with
     {:actors (pull* (-> (unify (from :film-actor [{:film-id $film-id, :actor-id actor-id}])
                                (from :actor [{:xt/id actor-id} first-name last-name]))
                         (order-by last-name)
                         (return {:name (concat first-name " " last-name)})))
                    {:args [film-id]})})
    (return title length actors rating))
```

https://docs.xtdb.com/reference/main/xtql/queries.html#_subqueries

Aggregations (SQL)

Get me the top 5 frequently starring actors:

```
SELECT a.first_name, a.last_name, COUNT(*) AS film_count
FROM actor a
  JOIN film_actor fa ON (fa.actor_id = a.xt$id)
GROUP BY a.first_name, a.last_name
ORDER BY film_count DESC
LIMIT 5
```

Aggregations (XTQL)

Get me the top 5 frequently starring actors:

```
(-> (unify (from :actor [{:xt/id actor-id} first-name last-name])
           (from :film-actor [actor-id]))
    (aggregate first-name last-name {:film-count (row-count)})
    (order-by {:val film-count, :dir :desc})
    (limit 5))
```

https://docs.xtdb.com/reference/main/xtql/queries.html#_aggregate

Aggregations 2 (Desired result)

Get me the frequency distribution of how many films each actor has appeared in:

```
[{:film-count 14, :frequency 1}
 {:film-count 15, :frequency 2}
 ...
 {:film-count 24, :frequency 14}
 {:film-count 25, :frequency 19}
 {:film-count 26, :frequency 14}
 {:film-count 27, :frequency 17}
 ...
 {:film-count 41, :frequency 1}
 {:film-count 42, :frequency 1}]
```

Aggregations 2 (SQL)

Get me the frequency distribution of how many films each actor has appeared in:

```
SELECT fc.film_count, COUNT(*) AS frequency
FROM (SELECT fa.actor_id, COUNT(*) AS film_count
      FROM film_actor fa
      GROUP BY fa.actor_id) fc
GROUP BY fc.film_count
ORDER BY film_count
```

Aggregations 2 (XTQL)

Get me the frequency distribution of how many films each actor has appeared in:

```
(-> (from :film-actor [actor-id])
    (aggregate actor-id {:film-count (row-count)})
    (aggregate film-count {:frequency (row-count)})
    (order-by film-count))
```

https://docs.xtdb.com/reference/main/xtql/queries.html#_aggregate

Bitemporality (XTQL)

Find me who's ever rented 'African Egg':

```
(unify (from :film [{:xt/id film-id, :title "AFRICAN EGG"}])
       (from :inventory [{:xt/id inventory-id} film-id])
       (from :rental {:for-valid-time :all-time
                      :bind [{:xt/valid-time rented-during}
                             inventory-id customer-id]})
       (from :customer [{:xt/id customer-id} email]))
```

https://docs.xtdb.com/reference/main/xtql/queries.html#_temporal_filters

Bitemporality (XTQL)

Find me who's ever rented 'African Egg' - and what they also rented while they had 'African Egg' out:

```
(-> (unify (from :film [{:xt/id film-id, :title "AFRICAN EGG"}])
           (from :inventory [{:xt/id inventory-id} film-id])
           (from :rental {:for-valid-time :all-time
                          :bind [{:xt/valid-time ae-during} inventory-id customer-id]})
           (from :customer [{:xt/id customer-id} email])

           (from :rental {:for-valid-time :all-time
                          :bind [{:xt/valid-time other-during} customer-id {:inventory-id
i2}]})
           (from :inventory [{:xt/id i2, :film-id f2}])
           (from :film [{:xt/id f2} title])
           (where (<> title "AFRICAN EGG")
                  (overlaps? ae-during other-during)))

    (return email ae-during other-during title))
```

XTDB

XTQL DML

XTQL also supports DML - with the full power of the XTQL query language.

```
(xt/submit-tx xt-node
  (xt/sql-op "
    UPDATE users SET first_name = 'Sue' WHERE first_name = 'Susan'"))

(xt/submit-tx xt-node
  (xt/update-table :users
    {:bind [{:first-name "Susan"}]
     :set {:first-name "Sue"}}))
```

https://docs.xtdb.com/intro/what-is-xtql.html#dml

XTQL DML

XTQL also supports DML - with the full power of the XTQL query language.

```
(xt/submit-tx xt-node
  (xt/sql-op "
    UPDATE users SET version = version + 1 WHERE first_name = 'Sue'"))

(xt/submit-tx xt-node
  (xt/update-table :users
    {:bind [{:first-name "Sue"} version]
     :set {:version (+ version 1)}}))
```

https://docs.xtdb.com/intro/what-is-xtql.html#dml

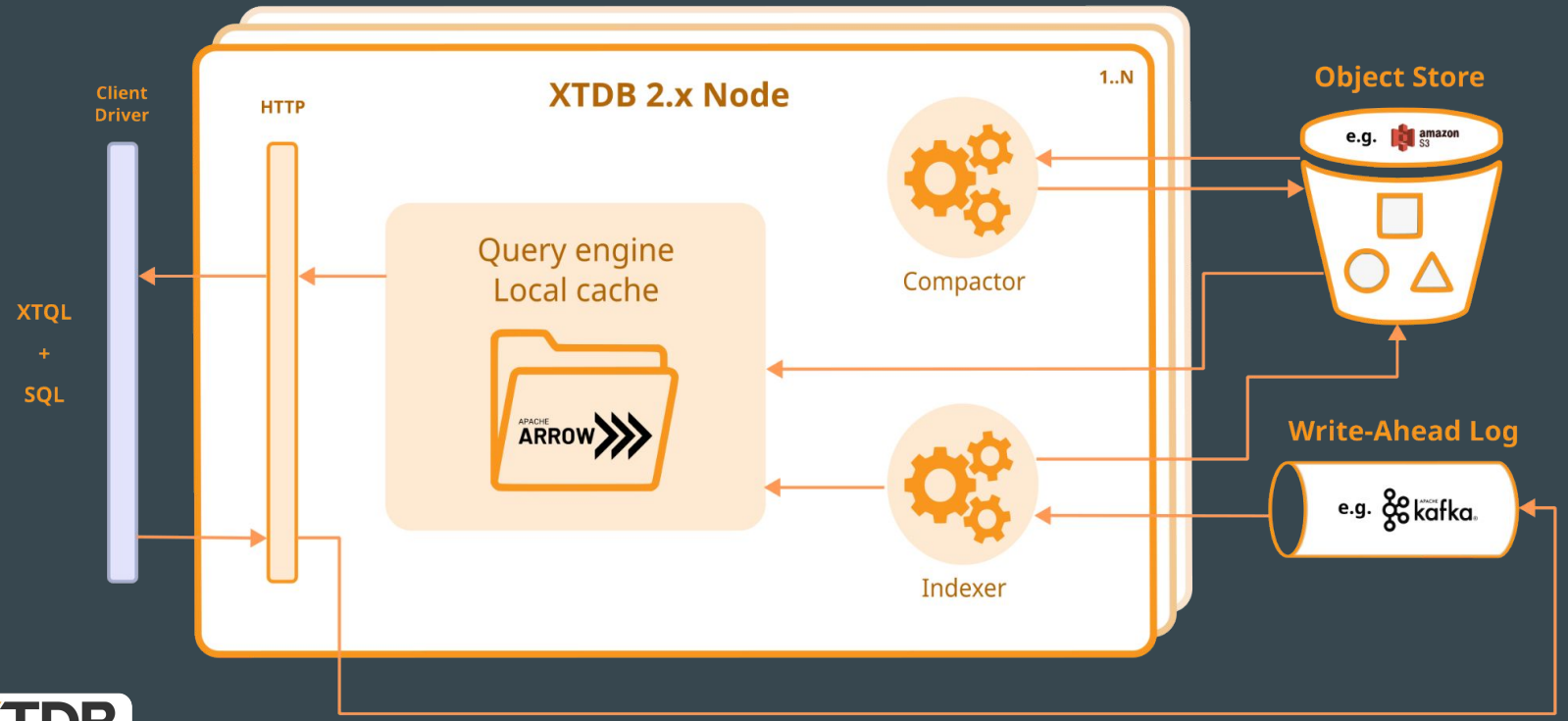
Dynamically generating queries - `template`

`template` is a helper macro to dynamically generate XTQL queries, inspired by backtick

Within it, you can use Clojure's 'unquote' (~) and 'unquote-splicing' (~@) forms.

```
(defn build-posts-query [{:keys [with-author? popular?]}]
  (xt/template (-> (from :posts [{:xt/id id} text
                                 ~@(when with-author?
                                     '[author-name])
                                 ~@(when popular?
                                     '[likes])])
                   ~@(when popular?
                       ['(where (> likes 100))]))))
```

The lifecycle of XT2 data



Interlude: Apache Arrow

- Columnar - documents broken down into attributes.
  - Faster lookup for individual values & column scanning, especially fixed-width primitives
- Off-heap - manual memory management (on our part)
  - Reduced GC pressure
- Memory layout == disk layout
  - Significantly reduced serialisation/deserialisation overhead
- Harder to update
  - Less of a problem for an immutable database :)

Interlude: Apache Arrow

- Fixed width:

```
[{:a 1, :b 1}, {:a 2, :b 3}, {:a 5, :b 8}]

⇒ {:a [1 2 5], :b [1 3 8]}
```

- Variable width:

```
[{:msg “hello”, ...}, {:msg “from”, ...}, {:msg “XTDB”, ...}]

⇒ {:msg {:offsets [0 5 9 13], :data “hellofromXTDB”}}
```

Interlude: Apache Arrow

- Structs:

```
[{:a {:a1 1, :a2 2}, ...}, {:a {:a1 3, :a2 4}, ...]
```

```
⇒ {:a {:a1 [1 3], :a2 [2 4]}}
```

- Lists:

```
[{:a [1 1 2], ...}, {:a [3 5 8 13], ...}]
```

```
⇒ {:a {:offsets [0 3 7], :data [1 1 2 3 5 8 13]}}
```

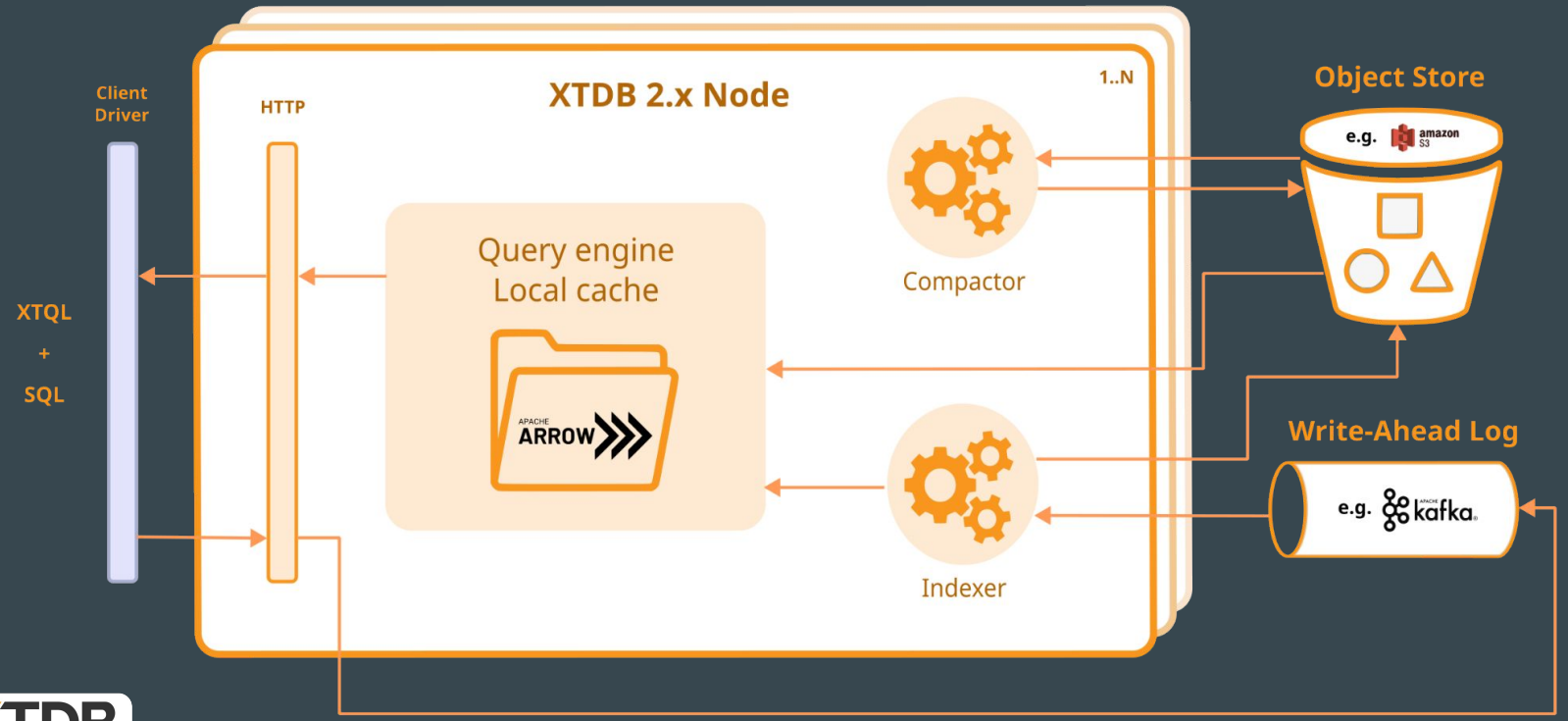
XTDB

Interlude: Apache Arrow

- Unions:

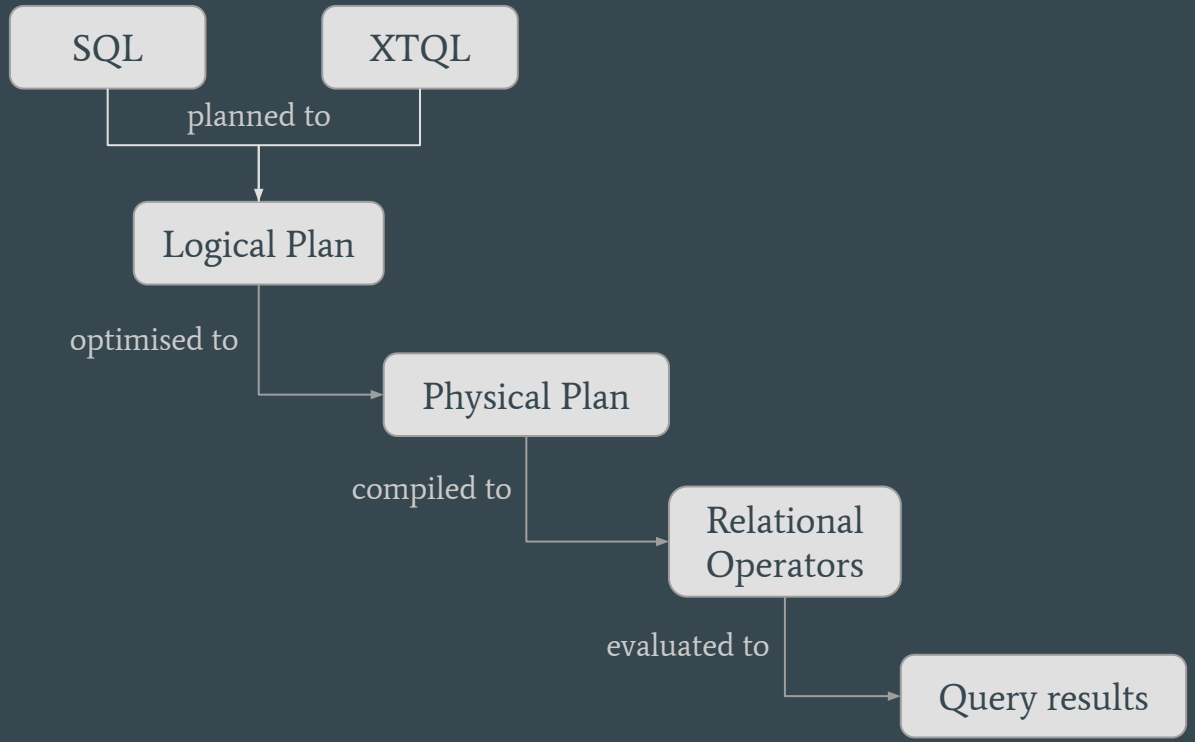
```
[{:a 4, ...}, {:a “a string”}, {:a 5, ...}, {:a 8, ...}]

⇒

{:a {:type-ids [0 1 0 0]
     :offsets [0 0 1 2]
     :data [[4 5 8]
            {:offsets [0 8], :data [“a string”]}]}}
```

The lifecycle of XT2 data



XTDB

The lifecycle of a query



XTDB

How does it perform?

- Performance optimisation will be a primary focus of the pre-release process
  - OLTP, particularly
  - More work to do optimising our ‘cold’ (object-store) reads, cache policies, etc.

XTDB

What's next?

- Your initial impressions and feedback, please! https://discuss.xtdb.com, hello@xtdb.com
- The road to GA:
  - Performance, stability, operational considerations - making it production-ready
  - Client libraries: Java, JavaScript, et al.
  - On our current wishlist:
    - Subscribing to incoming transactions
    - Speculative transactions (`with-tx`)
    - Schema introspection - `information_schema`
    - Reusable query fragments ('rules')
    - Window functions, and other next-level SQL functionality
  - ... but please let us know what's on *your* wishlist!

XTDB

Operations

Jon Pither
CEO
GitHub

Jon has led projects at Tier-1 Investment Banks, an online newspaper website, a major property portal, and an international public electric bikes scheme. Although he has largely traded Emacs hacking for Zoom meetings in his role as CEO, he is still deeply in touch with XTDB's implementation.

Malcolm Sparks
CTO
GitHub

With over 40 years of applied Computer Science experience, Malcolm cut his teeth on the ZX81. He has a long history of building database systems and services. He has worked with SQLWindows, Oracle Forms, and Oracle CASE tools, and even spent time as an Oracle DBA. Malcolm is also developing a resource server.

Kath Read
CFO
GitHub

Kath previously worked as a financial accountant and now brings her broad experience of oversight in the software industry to ensure that XTDB maintains a healthy balance as an open source product with long-term commercial sustainability.

Jeremy Taylor
Head of Product
GitHub

Jeremy became obsessed with databases while working as a solution engineer at IBM. He joined the XTDB team in 2019, prior to the initial public release, and now applies his passion to helping users and customers solve their problems.

Engineering

James Henderson
Head of Engineering
GitHub

James has spent his career building data-oriented systems in companies large and small, across a variety of domains. Eventually he realised that a lot of the domain complexities he'd faced had 'time' in common, so he came to JUXT to work on a bitemporal database.

Matt Butler
Senior Engineer
GitHub

During his long career at JUXT Matt has delivered numerous complex projects in multiple sectors, including telecommunications, infosec, and banking. Matt's affinity for hard problems now has him reading database white papers on weekends.

Dan Mason
Engineer
GitHub

During his time at JUXT, Dan has worked on a number of diverse consulting projects across sectors ranging from Finance to Medtech. He has been a contributor to XTDB since 2019 and is currently working on cloud operations.

Dan Stone
Senior Engineer
GitHub

Dan is a senior Clojure developer who spent significant time at Riverford, helping them revolutionize sustainable farming. Now he wrangles query optimizers to revolutionize the database industry.

Finn Völkel
Senior Engineer
GitHub

Finn is a keen student of computer science who previously worked at Google and Nextjournal. Working on databases was an inevitable career trajectory.

Consulting

Johanna Antonelli
GitHub

Alex Davis
GitHub

Tim Greene
GitHub

James Simpson
GitHub

Neale Swinnerton
GitHub

XTDB

A First Look at XTDB v2

https://xtdb.com/v2

https://discuss.xtdb.com

hello@xtdb.com

James Henderson

Head of Engineering
XTDB

Jeremy Taylor

Head of Product
XTDB