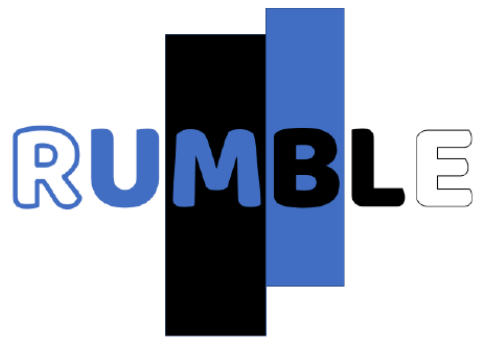
Full XQuery 3.1 support in RumbleDB

and a language server for JSONiq and XQuery



Jimmy Cai, ETH Zurich

Master's thesis, supervised by Dr. Ghislain Fourny

Declarative Amsterdam, 2026

RumbleDB

Thousands of XML files, one query

```
for $a in xml-files("hdfs:///papers/*.xml")
          /articles/article
where $a/year = 2024
group by $j := string($a/journal)
order by count($a) descending
return <journal
            name="{ $j }"
            articles="{ count($a) }"
        />
```

The same query on a laptop or a Spark cluster.

JSONiq and XQuery 3.1

One open-source engine for both languages.

Large, heterogeneous data

JSON, XML, CSV, Parquet, text, and more. From local files, HDFS, S3, Azure Blob Storage, read-only HTTP, PostgreSQL, MongoDB, and more.

Local or on Spark

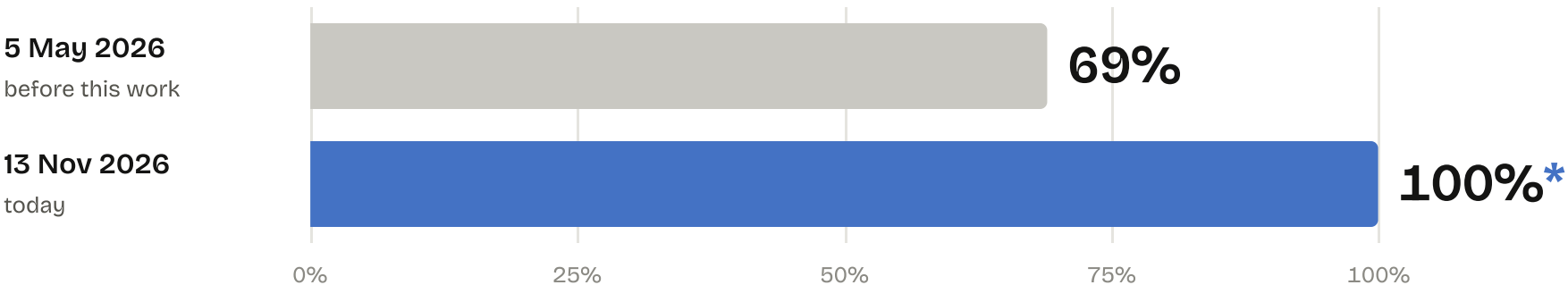
Queries over `xml-files()` or `json-lines()` run on Spark; over `doc()` and `json-doc()`, locally.

Easy to install

`brew install rumbledb`, or `pip install jsoniq` for Python and Jupyter, with `%%xquery` cells. Or as a Maven dependency.

Measuring progress with QT3

QT3 is the W3C test suite for XQuery and XPath (31,821 test cases). We run it on every pull request, with a regression report.



Pass rate of the 31,389 tests that apply to XQuery 3.1 (XQuery parser), the same tests on both dates.

Applicable tests

Not counted 432

Reason	Tests
Other XQuery and XPath versions	272
`fn:transform` (needs an XSLT processor)	124
XSD 1.1 schema validation (we support XSD 1.0)	6
Not applicable to RumbleDB (expect an optional feature to be absent, or an older Unicode or XML edition)	30

Not counted in any pass rate.

Failing 1

Feature	Tests
Remote HTTP in `fn:unparsed-text` (we don't support proving that our engine is a human, yet.)	1

Counted as not passing.

Window clauses

```
let $temps := (24, 31, 33, 32, 27, 29, 34, 35, 26)

(: 3-day moving average :)
for sliding window $w in $temps
    start at $s when true()
    only end at $e when $e - $s eq 2
return round(avg($w), 1)
```

Daily maximum temperature, °C

24	31	33	32	27	29	34	35	26
1	2	3	4	5	6	7	8	9



```
29.3  32  30.7  29.3  30  32.7  31.7
```

Tumbling windows do not overlap

Sliding windows can overlap, e.g. a 3-day moving average



XML Schema support

`invoice.xml`, with the types it gets from `shop.xsd`

```
<invoice xmlns="urn:shop"
         date="2026-11-05"> <!-- xs:date -->
  <line>
    <sku>AB-123</sku>      <!-- s:Sku -->
    <price>12.50</price>   <!-- xs:decimal -->
    <quantity>4</quantity> <!-- xs:positiveInteger -->
  </line>
  <line>…</line>
</invoice>                 <!-- s:InvoiceType -->
```

Query

```
import schema namespace s = "urn:shop"
  at "shop.xsd";

validate { doc("invoice.xml")/s:invoice }
```

With `<sku>CD456</sku>` in the second line:

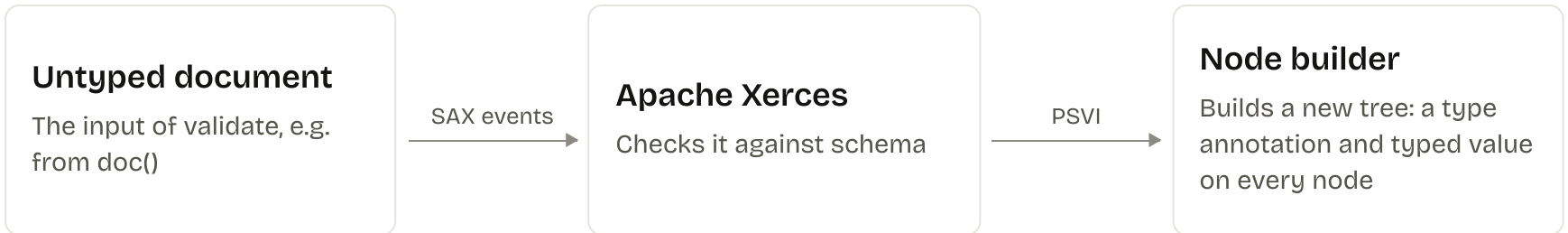
```
[XQDY0027] Value 'CD456' is not facet-valid
with respect to pattern '[A-Z]{2}-[0-9]{3}'
for type 'Sku'.
```

Validated documents are typed

```
let $invoice := validate { doc("invoice.xml")/s:invoice }
return (
    $invoice/@date + xs:dayTimeDuration("P30D"),            (: 2026-12-05 :)
    sum($invoice//s:line ! (s:price * s:quantity)),         (: 82 :)
    $invoice instance of element(s:invoice, s:InvoiceType)  (: true :)
)
```

Without `validate`, the values are untyped: the date addition fails with `FORG0001`, and the `instance of` test is `false`.

XML Schema: how it works



Node in `invoice.xml`	Without `validate`	After `validate`
`<invoice>...</invoice>`	`xs:untyped`	`s:InvoiceType`
`<price>12.50</price>`	`xs:untyped`, value `"12.50"`	`xs:decimal`, value `12.5`
`@date="2026-11-05"`	`xs:untypedAtomic`	`xs:date`

Casts, comparisons and type tests read these annotations, so **the rest of the engine just works**.

More of XQuery 3.1

Node set operators

```
let $i := <x><i>A</i><i>B</i><i>C</i></x>/i
return ($i except $i[2]) ! string()
```

→ A C

Partial application of built-ins

```
let $prefix := substring(?, 1, 2)
return $prefix("AB-123")
```

→ AB

Error variables in `catch`

```
try { 1 div 0 }
catch * { $err:code }
```

→ err:FOAR0001

String constructors

```
let $city := "Amsterdam"
return ``[Hello, `{ $city }`!]``
```

→ Hello, Amsterdam!

More built-in functions, by Henrik Pätzold

JSON to XML

```
json-to-xml('{"city": "Amsterdam"}')
```

```
→ <map xmlns="…/xpath-functions">
    <string key="city">Amsterdam</string>
  </map>
```

Numbers in words, in Dutch

```
format-integer(2026, "w", "nl")
```

```
→ tweeduizendzesentwintig
```

Localized dates

```
format-date(xs:date("2026-11-05"),
  "[Fn] [D] [Mn] [Y]", "nl", (), ())
```

```
→ donderdag 5 november 2026
```

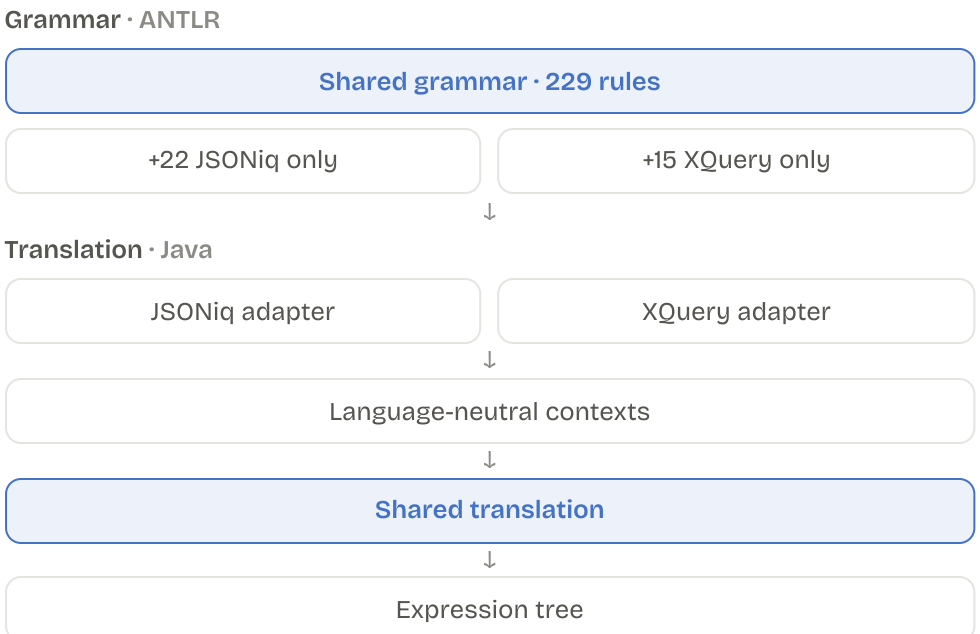
Innermost nodes

```
let $d := <a><b><c/></b><b/></a>
return innermost($d//*) ! name()
```

```
→ c b
```

Also: `fn:outermost`, `fn:lang`, `parse-json` options, collations, regular-expression errors, and faster `fold`, formatting and JSON parsing.

Unifying behavior of JSONiq and XQuery



JSONiq and XQuery are close siblings, yet RumbleDB had everything twice: two grammars and two translators.

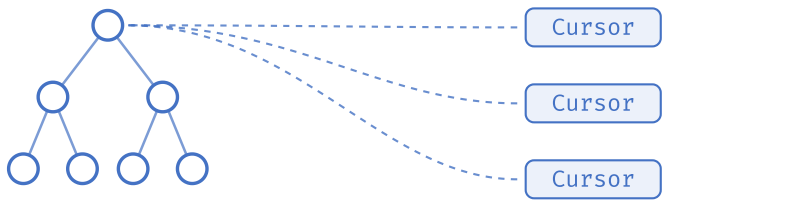
Grammar: the common rules live in one ANTLR grammar that both languages import.

Translation: ANTLR still generates separate classes per language, so a layer of language-neutral contexts lets the logic be written once.

Both languages now behave **uniformly and consistently**.

Faster function calls and maps

Iterators mixed the structure of a query plan with the state of one evaluation, so **every function call deep-copied the function's whole tree.**



Now: plans are built once; each evaluation gets a fresh cursor.

Recursive Fibonacci, `fib(26)`

392,835 function calls

Before		178 s
Runtime plans	≈ 2,000× faster	**0.08 s**

Building a map, one `map:put` at a time

4,000 entries

Before		84 s
Hash trie	≈ 4,000× faster	**0.02 s**

Laptop (Apple M3), Spark local mode. Median of 3 JVMs × 5 runs, including parsing and compilation.

A language server for JSONiq and XQuery

RumbleDB's types and errors, in your editor

Writing queries as a beginner

Before: write → run → read the error → repeat

```
for $sale in $sales
group by $shop := $slaes.shop
return $shop
```

```
⚠ There was an error on line 2
Code: [XPST0008]
Message: Uninitialized variable reference: slaes
```

Now: while you type

```
for $sale in $sales
group by $shop := $slaes.shop
return $shop
```

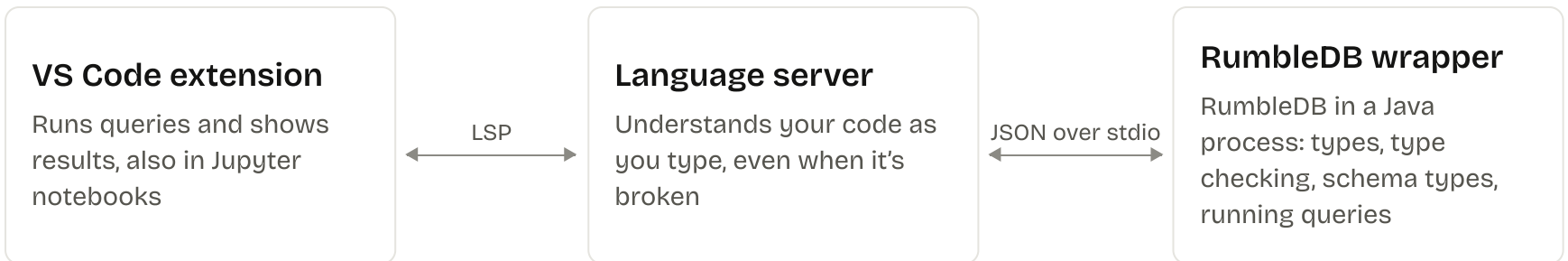
Reference to undefined variable `'$slaes'`

In the Big Data course at ETH Zurich, about 500 students this semester write JSONiq and XQuery, most of them for the first time.

Live demo

VS Code with "JSONiq and XQuery Language Support" extension

Architecture



Works **without Java**: syntax, scopes, navigation, rename, formatting

Engine knowledge with Java

Features

Language server alone

Diagnostics	syntax, names, imports
Completion	variables, functions, keywords
Hover	W3C function docs
Navigation	definitions, rename
Formatting	JSONiq and XQuery

With RumbleDB

Type errors	before running
Inferred types	hover and completion
Schema types	from imported XSDs
Execution	results with column types

What's next

XQuery at scale

Improvements on parallel XPath

We want to push RumbleDB to its limits and beyond

Distributed validation

Ship the schema to every machine

XSD 1.1 validation

Waiting for Xerces for 1.1 support in Maven

Going further

XQuery Update Facility

On top of RumbleDB's existing JSONiq updates

Apache Hudi

Next to Delta Lake and Apache Iceberg

More editors

The language server in Zed, JetBrains, anything with LSP

Full XQuery 3.1 support in RumbleDB

100%* of the W3C test suite, and a new language server

RumbleDB

rumbledb.org

```
brew install rumbledb
```

```
pip install jsoniq
```

VS Code extension

"JSONiq and XQuery Language Support" on the Marketplace and Open VSX

github.com/RumbleDB/language-tools