JSONiq Updates for RumbleDB

An investigation into updating JSON

David Loughlin

Declarative Amsterdam
LSDS, Imperial College London
(Work done at: Systems Group, ETH)

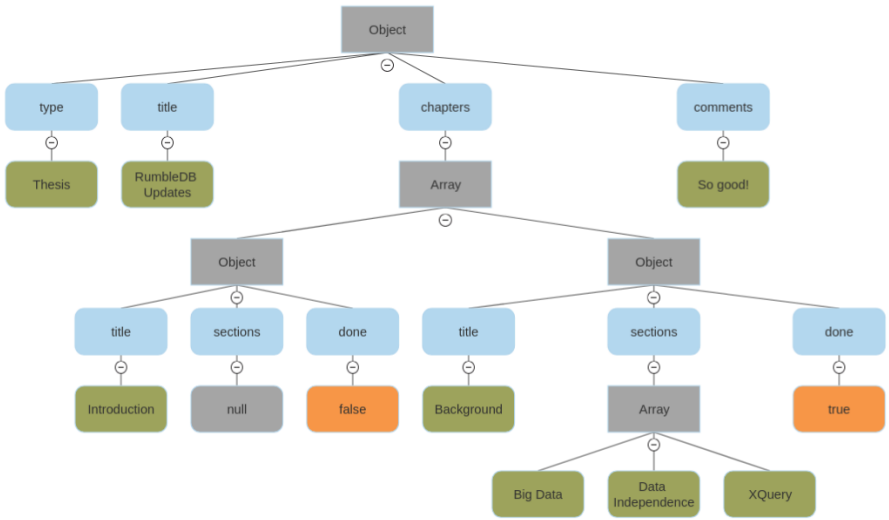
8th November 2024

Outline

JSON & Heterogeneous Data

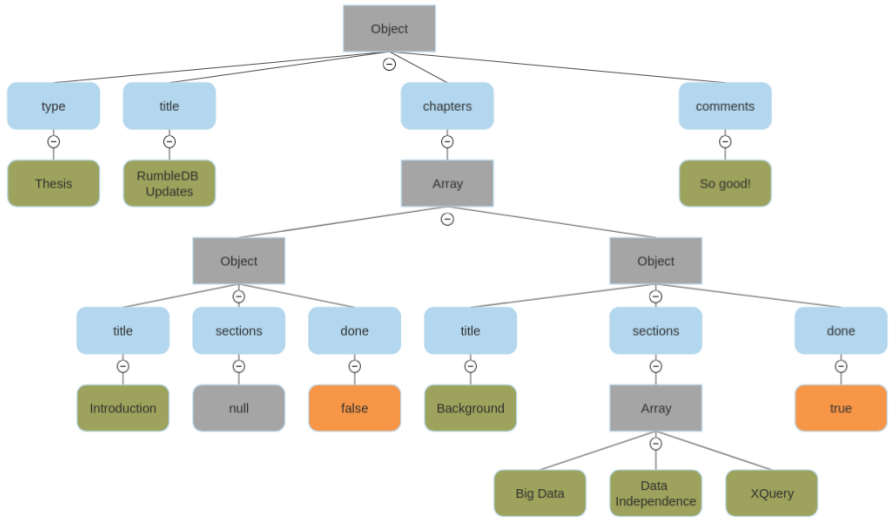
JSON & Heterogeneous Data

```json
{
  "type": "Thesis",
  "title": "RumbleDB Updates",
  "chapters": [
    {
      "title": "Introduction",
      "sections": null,
      "done": false
    },
    {
      "title": "Background",
      "sections": [
        "Big Data",
        "Data Independence",
        "XQuery"
      ],
      "done": true
    }
  ],
  "comments": "So good!"
}
```

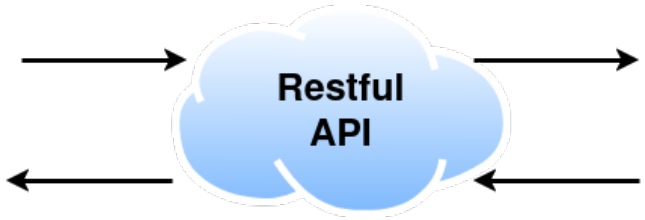


JSON & Heterogeneous Data

```json
{
  "type": "Thesis",
  "title": "RumbleDB Updates",
  "chapters": [
    {
      "title": "Introduction",
      "sections": null,
      "done": false
    },
    {
      "title": "Background",
      "sections": [
        "Big Data",
        "Data Independence",
        "XQuery"
      ],
      "done": true
    }
  ],
  "comments": "So good!"
}
```

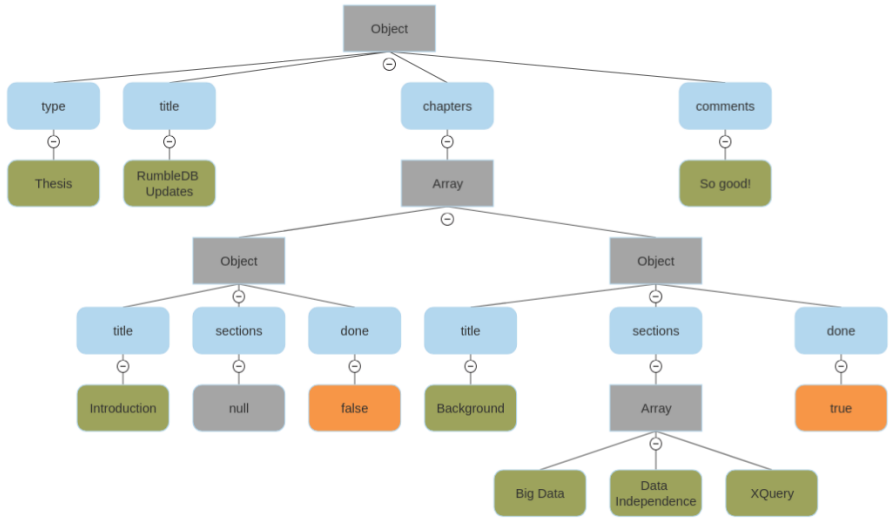


Variety in Big Data

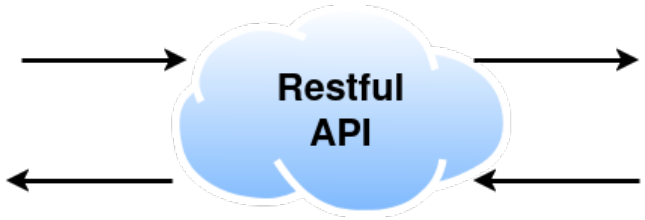


JSON & Heterogeneous Data

```json
{
  "type": "Thesis",
  "title": "RumbleDB Updates",
  "chapters": [
    {
      "title": "Introduction",
      "sections": null,
      "done": false
    },
    {
      "title": "Background",
      "sections": [
        "Big Data",
        "Data Independence",
        "XQuery"
      ],
      "done": true
    }
  ],
  "comments": "So good!"
}
```

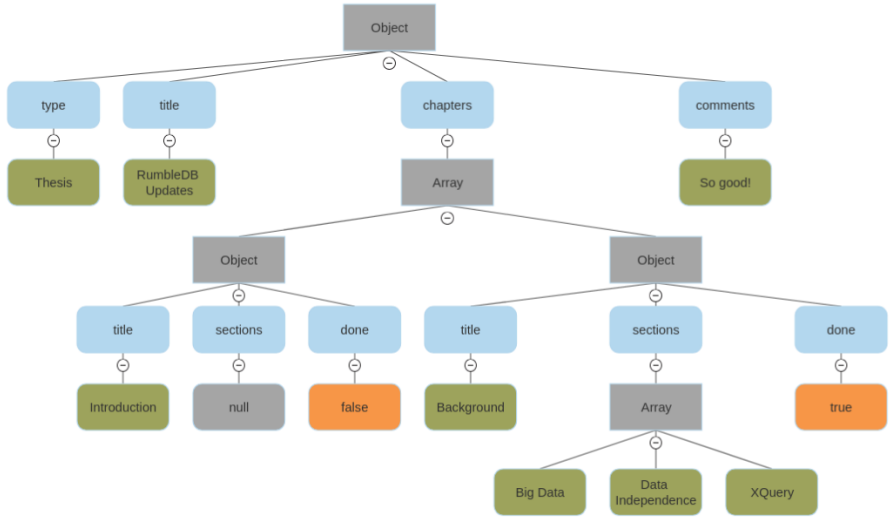


Variety in Big Data



JSON & Heterogeneous Data

```json
{
  "type": "Thesis",
  "title": "RumbleDB Updates",
  "chapters": [
    {
      "title": "Introduction",
      "sections": null,
      "done": false
    },
    {
      "title": "Background",
      "sections": [
        "Big Data",
        "Data Independence",
        "XQuery"
      ],
      "done": true
    }
  ],
  "comments": "So good!"
}
```



Mismatch with Tables

Table		

Analysing JSON

Back to SQL

```
SELECT fbs.foo, orders
FROM foos_with_bars fbs
NEST bars_with_foos bfs
ON KEYS ARRAY f.bar_id FOR f IN fbs.bars
```

Back to SQL

```
SELECT fbs.foo, orders
FROM foos_with_bars fbs
NEST bars_with_foos bfs
ON KEYS ARRAY f.bar_id FOR f IN fbs.bars
```

```
SELECT foo.bar
FROM table
UNNEST foobar
WHERE fizz <> 'buzz'
```

Back to SQL

```
SELECT fbs.foo, orders
FROM foos_with_bars fbs
NEST bars_with_foos bfs
ON KEYS ARRAY f.bar_id FOR f IN fbs.bars
```

```
SELECT foo.bar
FROM table
UNNEST foobar
WHERE fizz <> 'buzz'
```

- Lacks expressivity of heterogeneity.

JSONiq

```
for $foo in $foos[]
let $bars-for-foo := $bars[][ $$.foo_id eq $foo.id ]
return
  {
    "foo_id": $foo.id,
    "bars": [ $bars-for-foo.bar_id ]
  }
```

JSONiq

```
for $foo in $foos[]
let $bars-for-foo := $bars[][ $$.foo_id eq $foo.id ]
return
  {
    "foo_id": $foo.id,
    "bars": [ $bars-for-foo.bar_id ]
  }
```

- **Set-Oriented**
- **Declarative**

JSONiq

```
for $foo in $foos[]
let $bars-for-foo := $bars[][ $$.foo_id eq $foo.id ]
return
  {
    "foo_id": $foo.id,
    "bars": [ $bars-for-foo.bar_id ]
  }
```

- **Set-Oriented**
- **Declarative**
- Sequences are flat; items are everything a sequence is not.

JSONiq

```
for $foo in $foos[]
let $bars-for-foo := $bars[][ $$.foo_id eq $foo.id ]
return
  {
    "foo_id": $foo.id,
    "bars": [ $bars-for-foo.bar_id ]
  }
```

- **Set-Oriented**
- **Declarative**
- Sequences are flat; items are everything a sequence is not.

Problem: Data can be analysed but cannot be modified without complete overwrite.

JSONiq Updates

Introducing *Pending Update Lists*.

Definition (Pending Update Lists/PULs)

An unordered collection of *Update Primitives* within the same snapshot

Definition (Update Primitives)

Changes of state *to be* applied to an item.
(deletions, insertions, replacements, and renamings)

New expressions: `delete`, `insert`, `replace`, `rename`, `append`, `transforms`

JSONiq Updates

Introducing *Pending Update Lists*.

Definition (Pending Update Lists/PULs)

An unordered collection of *Update Primitives* within the same snapshot

Definition (Update Primitives)

Changes of state *to be* applied to an item.
(deletions, insertions, replacements, and renamings)

New expressions: `delete`, `insert`, `replace`, `rename`, `append`, `transforms`

JSONiq Updates

Introducing *Pending Update Lists*.

Definition (Pending Update Lists/PULs)

An unordered collection of *Update Primitives* within the same snapshot

Definition (Update Primitives)

Changes of state *to be* applied to an item.
(deletions, insertions, replacements, and renamings)

New expressions: `delete`, `insert`, `replace`, `rename`, `append`, `transforms`

JSONiq Updates

Introducing *Pending Update Lists*.

Definition (Pending Update Lists/PULs)

An unordered collection of *Update Primitives* within the same snapshot

Definition (Update Primitives)

Changes of state *to be* applied to an item.
(deletions, insertions, replacements, and renamings)

New expressions: `delete`, `insert`, `replace`, `rename`, `append`, `transforms`

JSONiq Updates

Returning a sequence and a PUL!

Definition (Expression Classification)

All expressions are classified as one of:

- **Simple**: not updating.
- **Basic-updating**: new expressions (except transforms).
- **Updating**: either basic-updating, or nested updating expressions.
- **Vacuous**: edge cases.

JSONiq Updates

Returning a sequence and a PUL!

Definition (Expression Classification)

All expressions are classified as one of:

- **Simple**: not updating.
- **Basic-updating**: new expressions (except transforms).
- **Updating**: either basic-updating, or nested updating expressions.
- **Vacuous**: edge cases.

JSONiq Updates

Returning a sequence and a PUL!

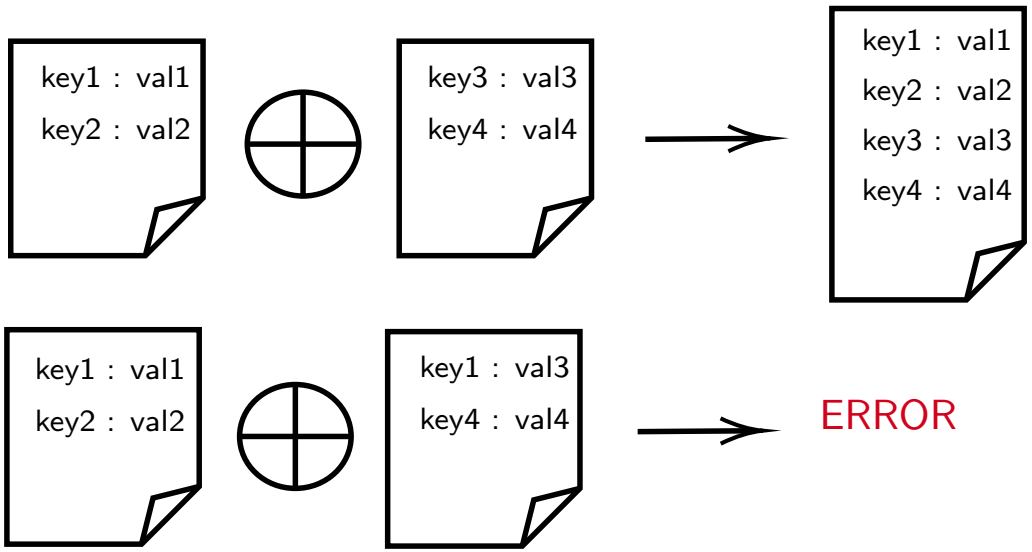
Definition (Expression Classification)

All expressions are classified as one of:

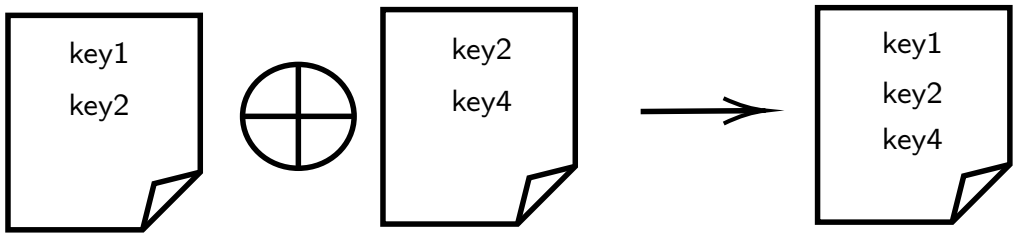
- **Simple**: not updating.
- **Basic-updating**: new expressions (except transforms).
- **Updating**: either basic-updating, or nested updating expressions.
- **Vacuous**: edge cases.

Merging Updates

Object Insertion



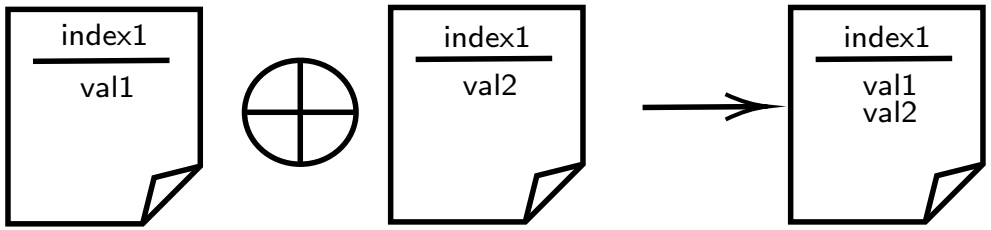
Object Deletion



Object Replacement & Renaming

Merging Updates

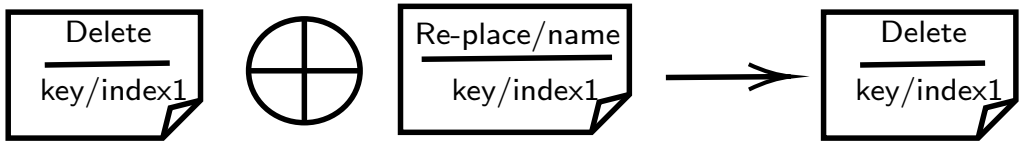
Array Insertion



Array Deletion

Array Replacement

Object/Array Deletion & Replacement/Renaming



Applying Updates

- For each update primitive, "Lock onto" the target item
- Do what it says on the tin: apply the update
- Order should not matter
- Upon completion the current snapshot ends

Applying Updates

- For each update primitive, “Lock onto” the target item
- Do what it says on the tin: apply the update
- Order should not matter
- Upon completion the current snapshot ends

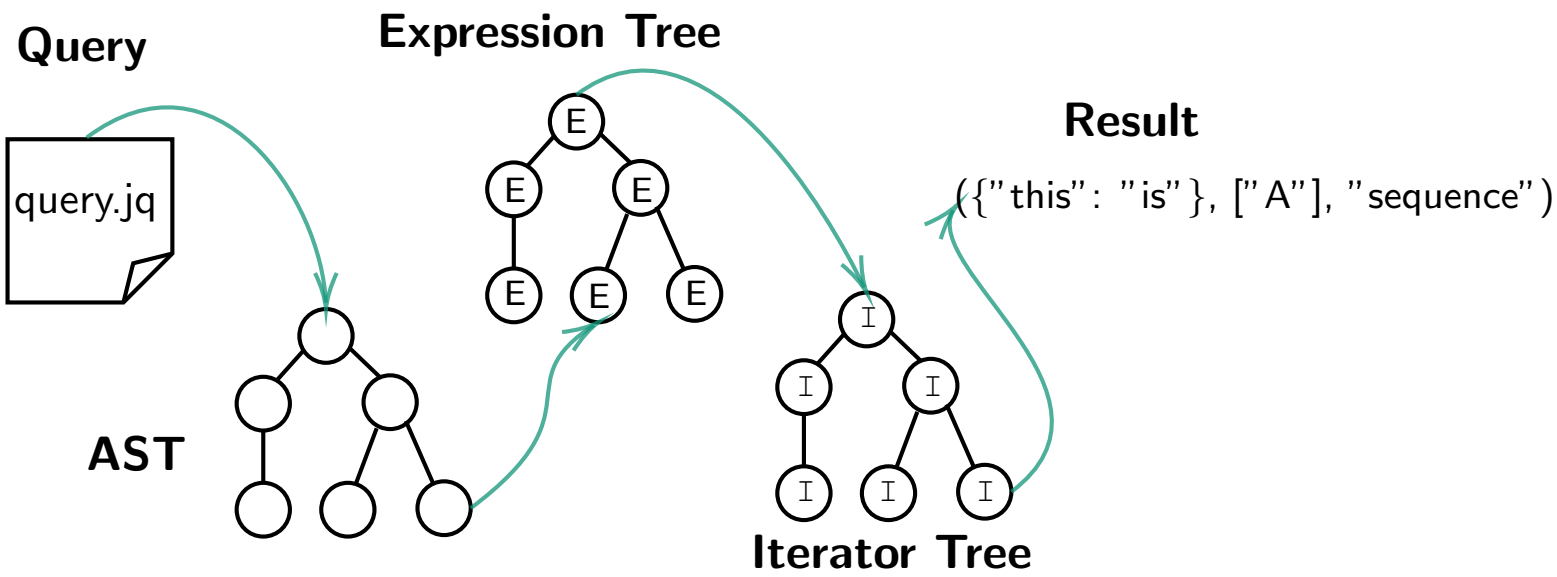
Applying Updates

- For each update primitive, "Lock onto" the target item
- Do what it says on the tin: apply the update
- Order should not matter
- Upon completion the current snapshot ends

RumbleDB

- A query execution engine for heterogeneous datasets using JSONiq
- Locally executable and Big Data compatible via Spark

RumbleDB



- A query execution engine for heterogeneous datasets using JSONiq
- Locally executable and Big Data compatible via Spark

Persisting Updates

Transactions

What are updates without persistence?

Delta Lakes

- Abstraction over parquet files for ACID-compliant storage.
- A write-ahead log in an object store maintains metadata about underlying parquet files.
- Allowing for ACID-compliant:
  - **UPSERTs**
  - **DELETEs**
  - **MERGEs**
  - **Schema Evolution**
- Delta lakes can be used via the Spark API!

Delta Lakes

- Abstraction over parquet files for ACID-compliant storage.
- A write-ahead log in an object store maintains metadata about underlying parquet files.
- Allowing for ACID-compliant:
  - **UPSERTs**
  - **DELETEs**
  - **MERGEs**
  - **Schema Evolution**
- Delta lakes can be used via the Spark API!

Delta Lakes

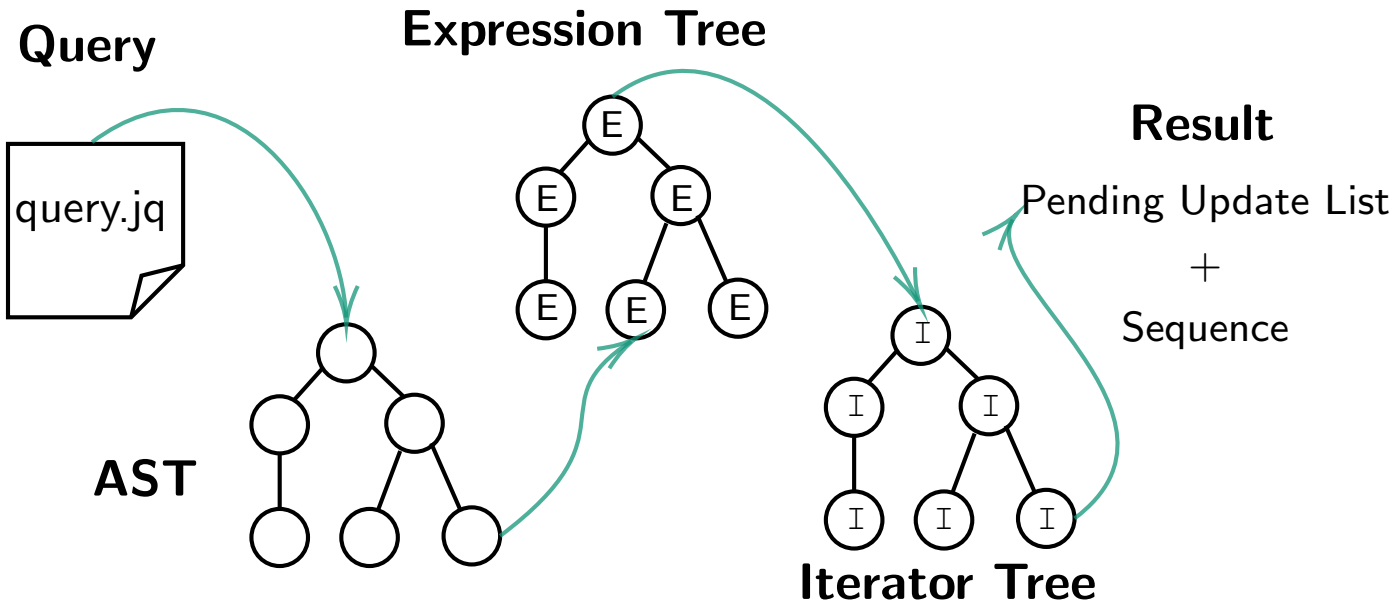
- Abstraction over parquet files for ACID-compliant storage.
- A write-ahead log in an object store maintains metadata about underlying parquet files.
- Allowing for ACID-compliant:
  - **UPSERTs**
  - **DELETEs**
  - **MERGEs**
  - **Schema Evolution**
- Delta lakes can be used via the Spark API!

Delta Lakes

- Abstraction over parquet files for ACID-compliant storage.
- A write-ahead log in an object store maintains metadata about underlying parquet files.
- Allowing for ACID-compliant:
  - **UPSERTs**
  - **DELETEs**
  - **MERGEs**
  - **Schema Evolution**
- Delta lakes can be used via the Spark API!

Methodology

Methodology



Grammar

```
deleteExpr              : Kdelete updateLocator;

updateLocator           : main_expr=primaryExpr ( arrayLookup | objectLookup )+;
```

- RumbleDB uses ANTLR4 for parsing and lexing to pass information to the AST.

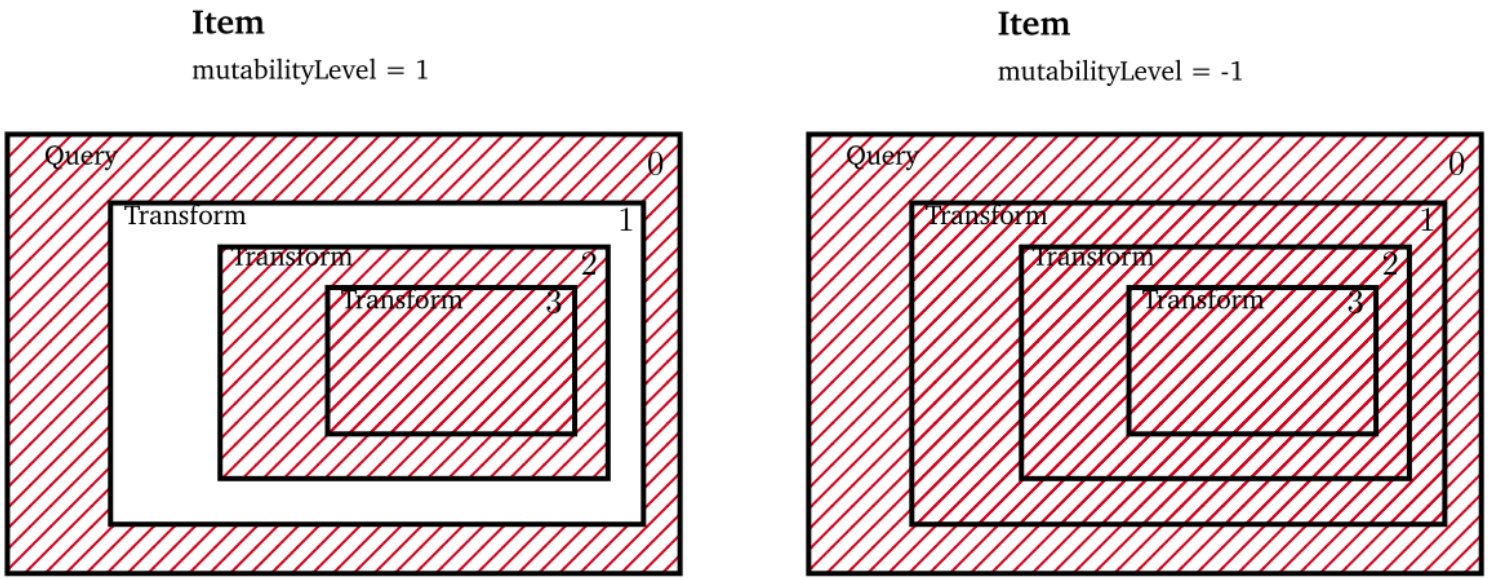
Expressions

- AST is translated to a tree of `Expression` instances.
- Using visitor pattern to populate `Expression` instances with information.
- Updated Expression Classifications to include **Unset**.

Iterators

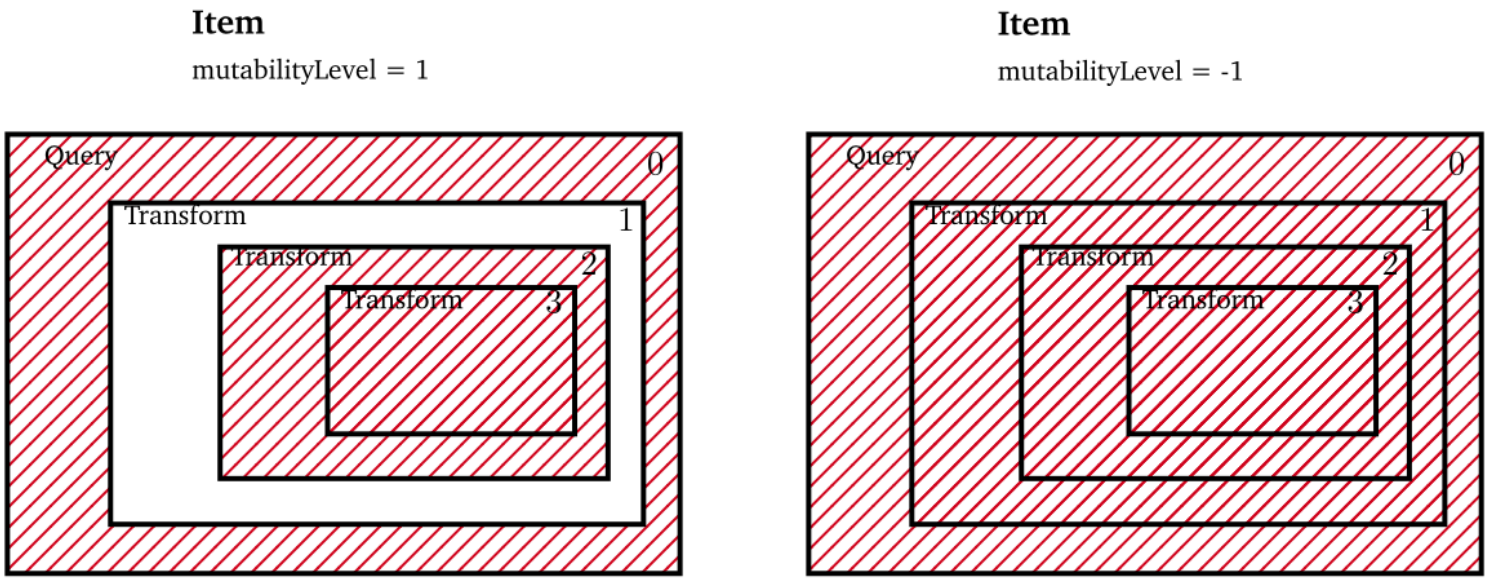
- `Expression` tree is translated to a tree of `RuntimeIterator` instances
- Extend `RuntimeIterator` interface to include `isUpdating` and `getPendingUpdateList` methods

Mutability Scoping



- Included information when making expressions

Mutability Scoping



- Included information when making expressions

Update Primitives

InsertIntoObject(Object, Object)

InsertIntoArray(Array, Integer, Item*)

Update Primitives

Target

InsertIntoObject(Object, Object)

InsertIntoArray(Array, Integer, Item*)

Update Primitives

Target Selector

InsertIntoObject(Object, Object)

InsertIntoArray(Array, Integer, Item*)

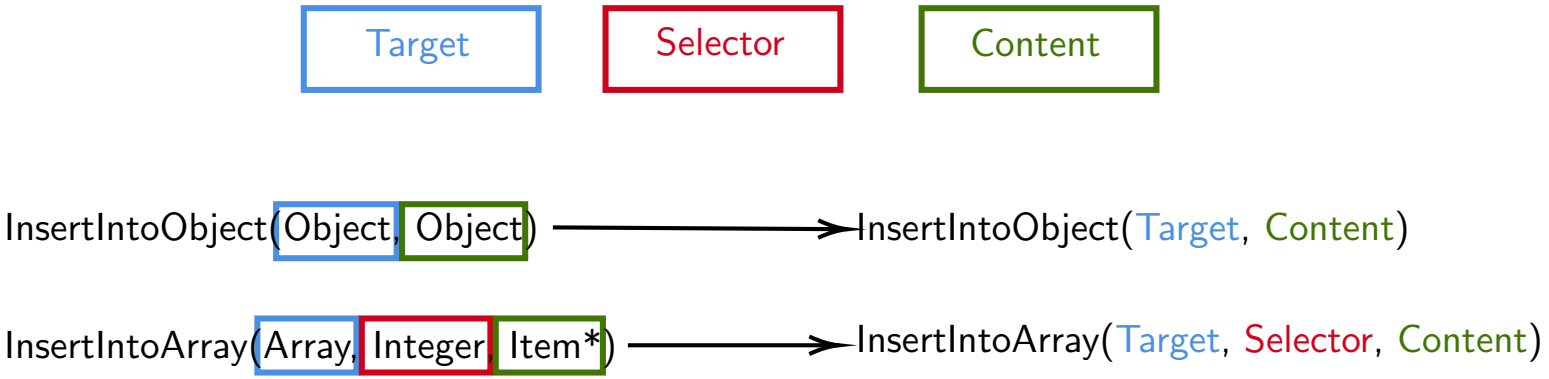
Update Primitives

Target Selector Content

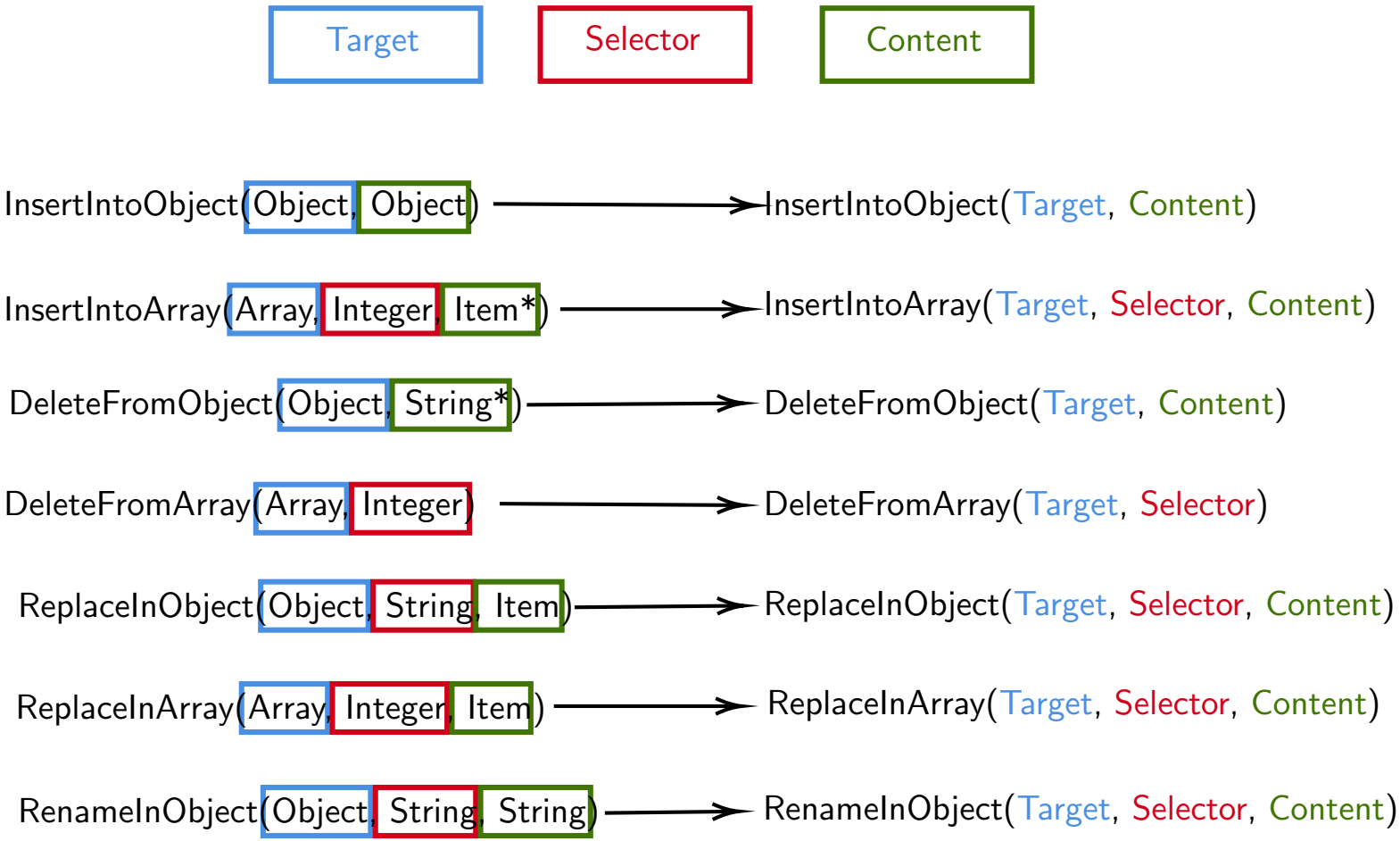
InsertIntoObject(Object, Object)

InsertIntoArray(Array, Integer, Item*)

Update Primitives



Update Primitives



Pending Update Lists

Decomposed In Map	Composed Equivalent
insertObjMap Target → Content Object Object	InsertIntoObject(Target, Content)
renameObjMap Target → Selector → Content Object String String	ReplaceInObject(Target, Selector, Content)

delReplaceObjMap
Target → Selector → Content
Object String Item
	if Content == null:
	DeleteFromObject(Target, List[Selector])
	
	if Content != null:
	ReplaceInObject(Target, Selector, Content)

Pending Update Lists

Decomposed In Map

insertArrayMap
Target→Selector→Content
Array Int List[Item]

delReplaceArrayMap
Target→Selector→Content
Array Int Item

Composed Equivalent

InsertIntoArray(Target, Selector, Content)

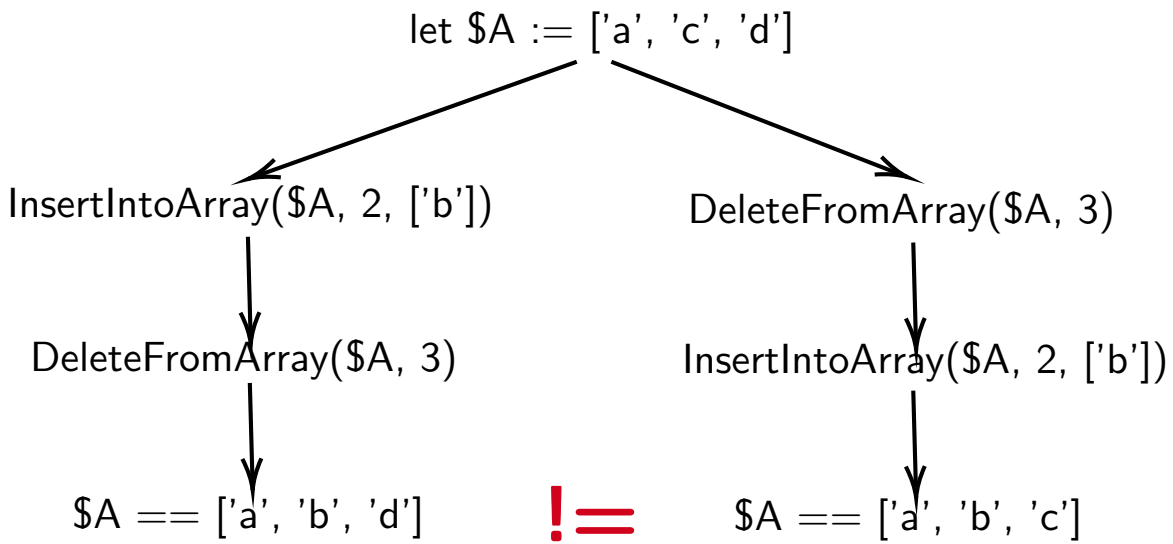
if Content == null:
    DeleteFromArray(Target, Selector)

if Content != null:
    ReplaceInArray(Target, Selector, Content)

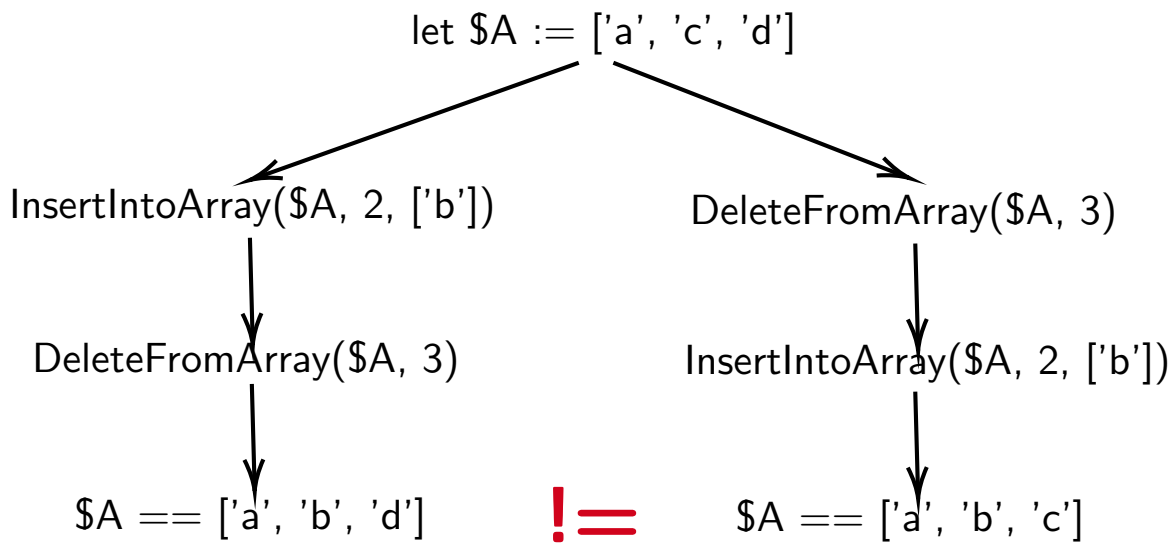
Merging PULs

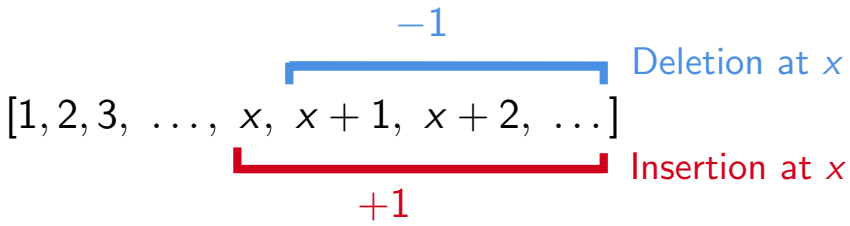
Iterate and merge using the Maps!
(according to the JSONiq specification)

Applying PULs

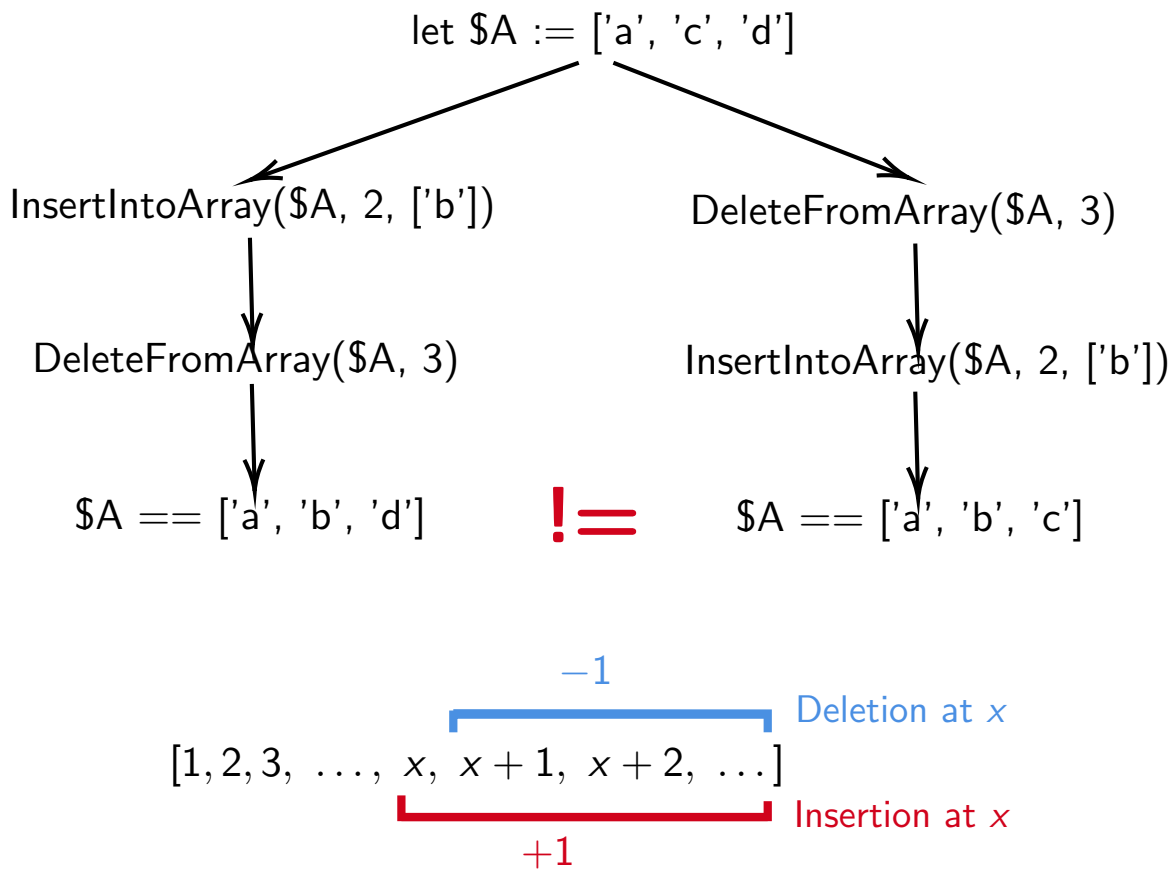


Applying PULs



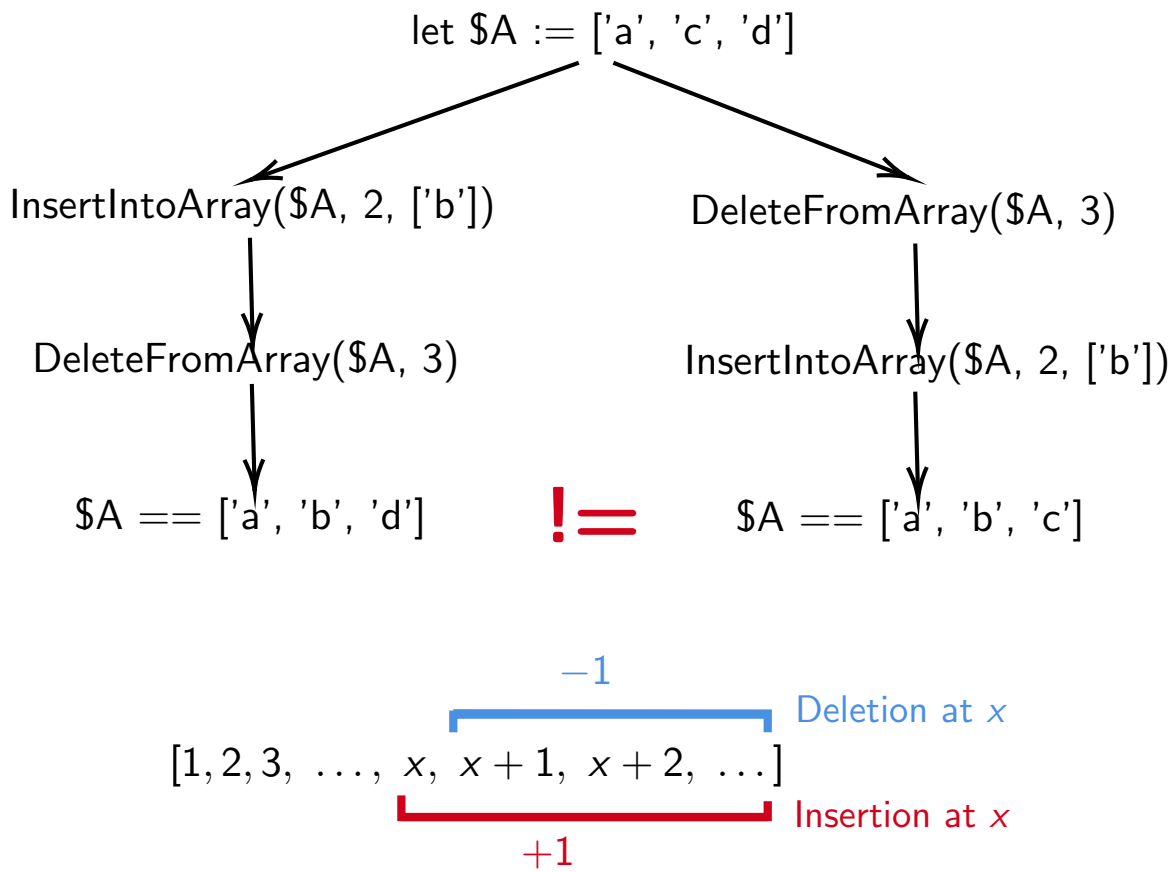


Applying PULs



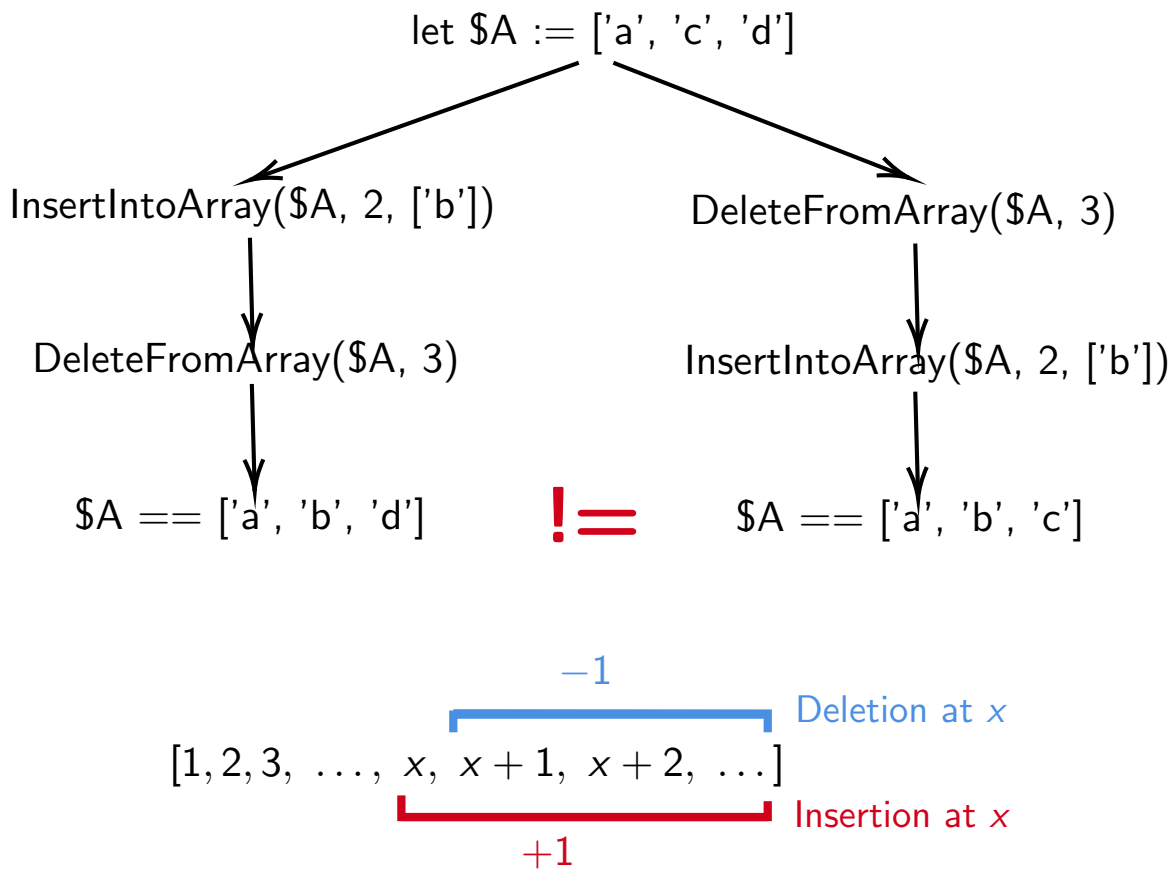
- Array insertions come last.

Applying PULs



- Array insertions come last.
- Array updates in descending order by *Selector* index.

Applying PULs

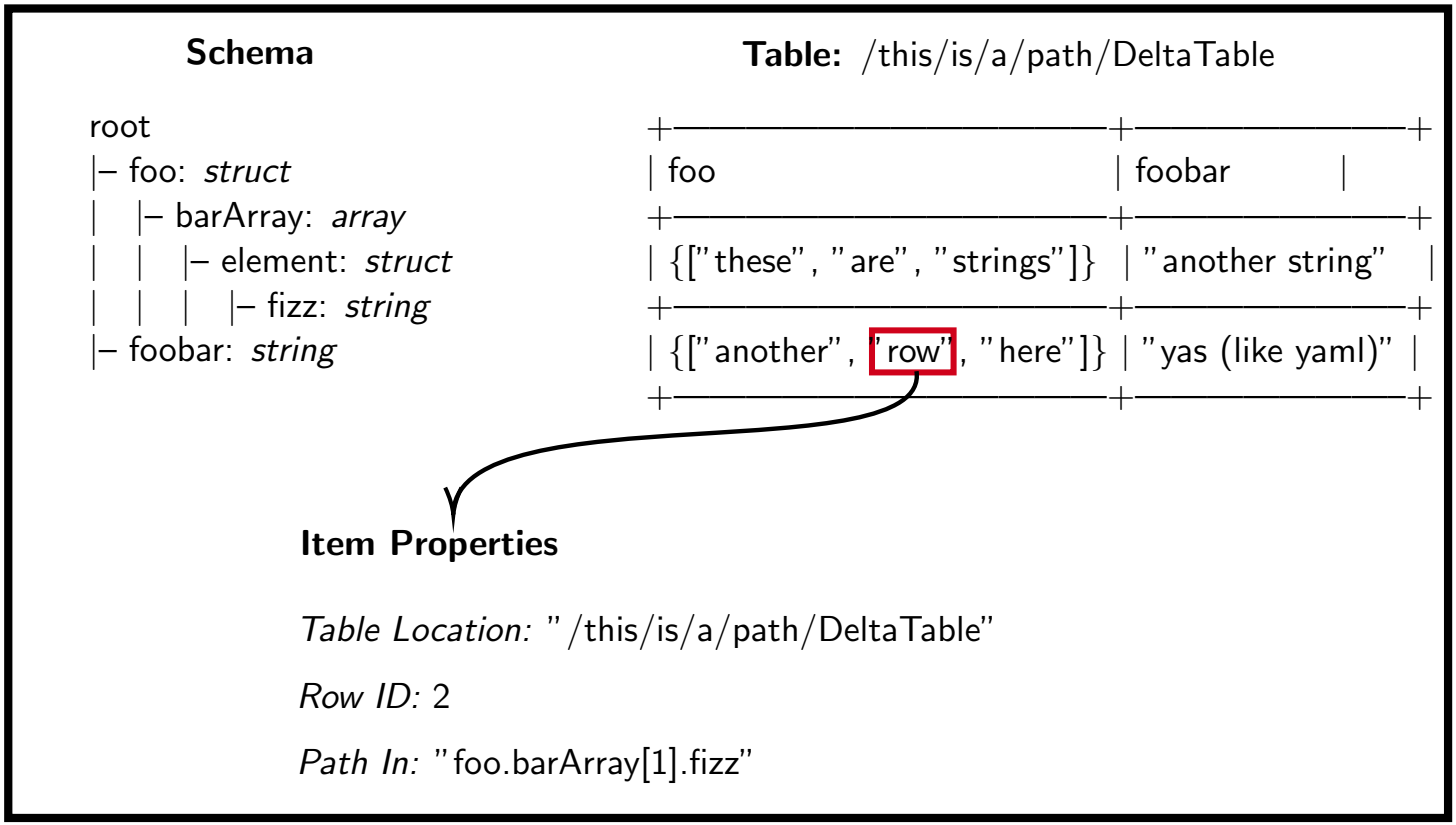


- Array insertions come last.
- Array updates in descending order by *Selector* index.
- Rebuild update primitives by iterating over PUL Maps.

Delta Tables

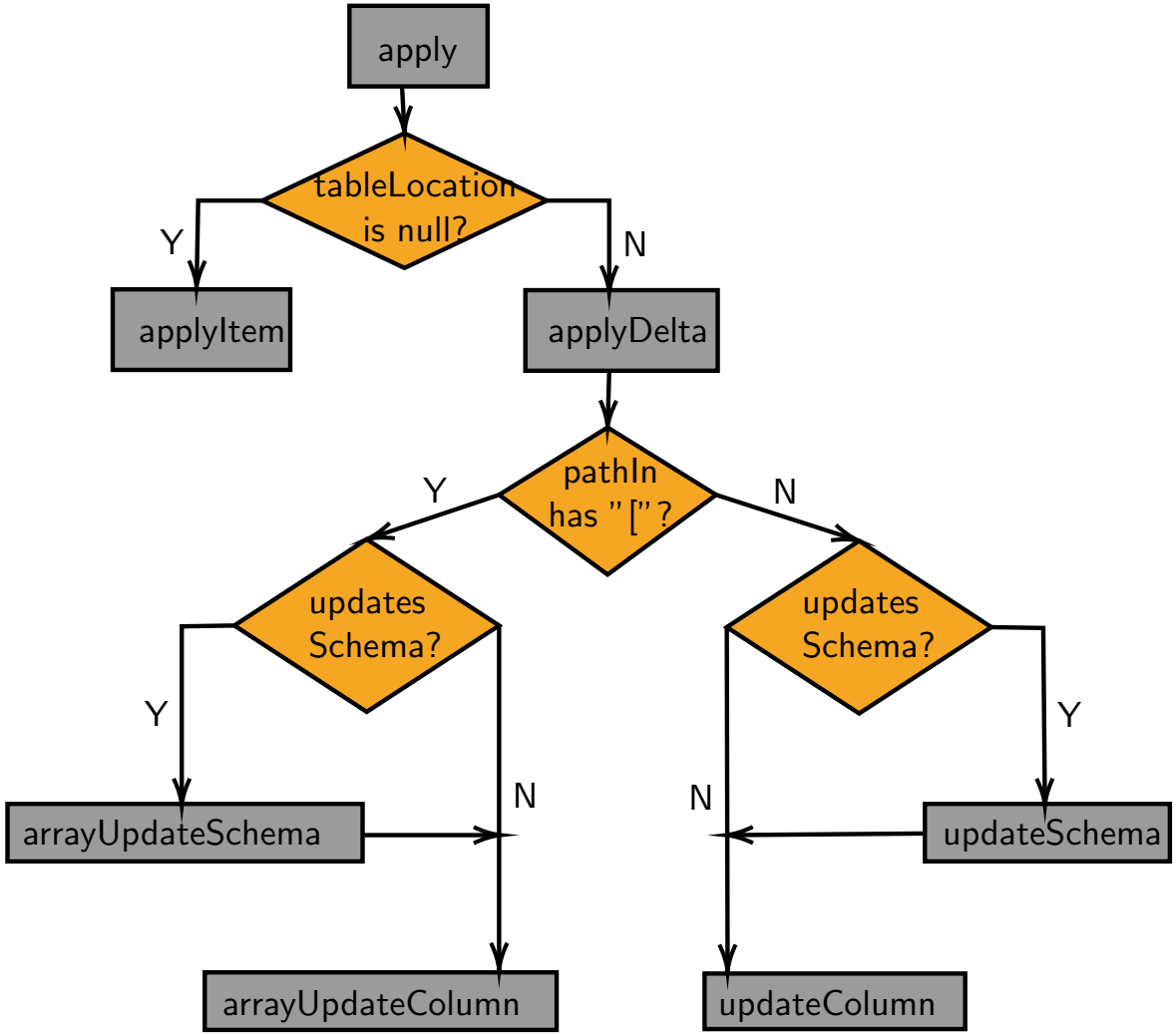
Thanks to Spark, reading and writing from and to Delta files are exactly like Parquet files. We add a `delta-file()` method and a corresponding iterator to create a dataframe.

Identifying Items



- Identify each Item in a Delta Table despite nesting

Applying to Delta



Results

Datasets

	TPC-C	**GitHub Archive**
Record Shape	Flat, Tabular	Extremely Nested
Column Shape	Homogeneous Column Types	Heterogeneous "payload" Field
Why?	Typical relational model	Emblematic of natural heterogeneity

TPC-C Insights

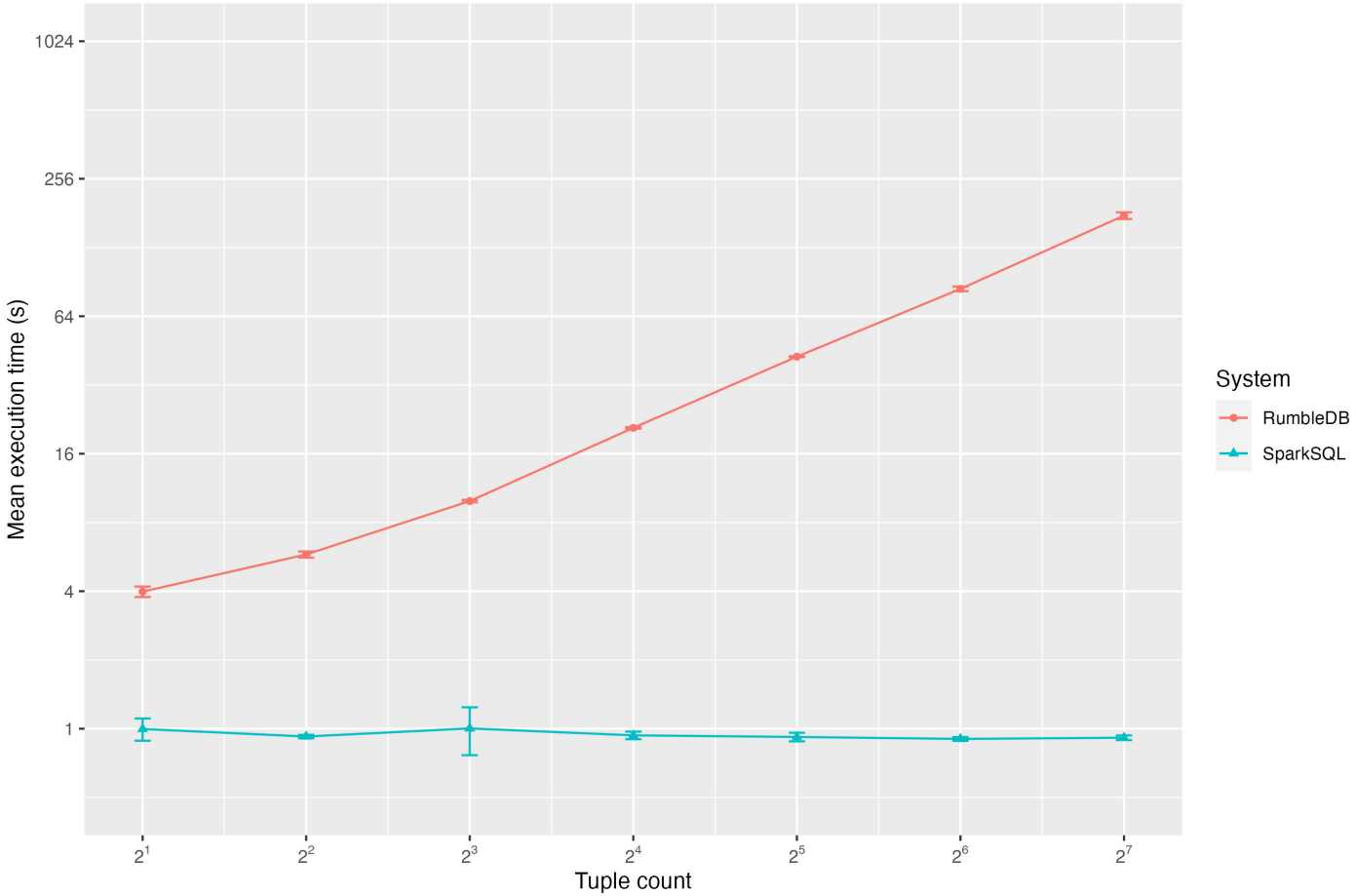


Figure: Comparison of performance for conditional replacement on TPC-C dataset between RumbleDB and SparkSQL, on $\log_2$ scales

GitHub Archive Insights

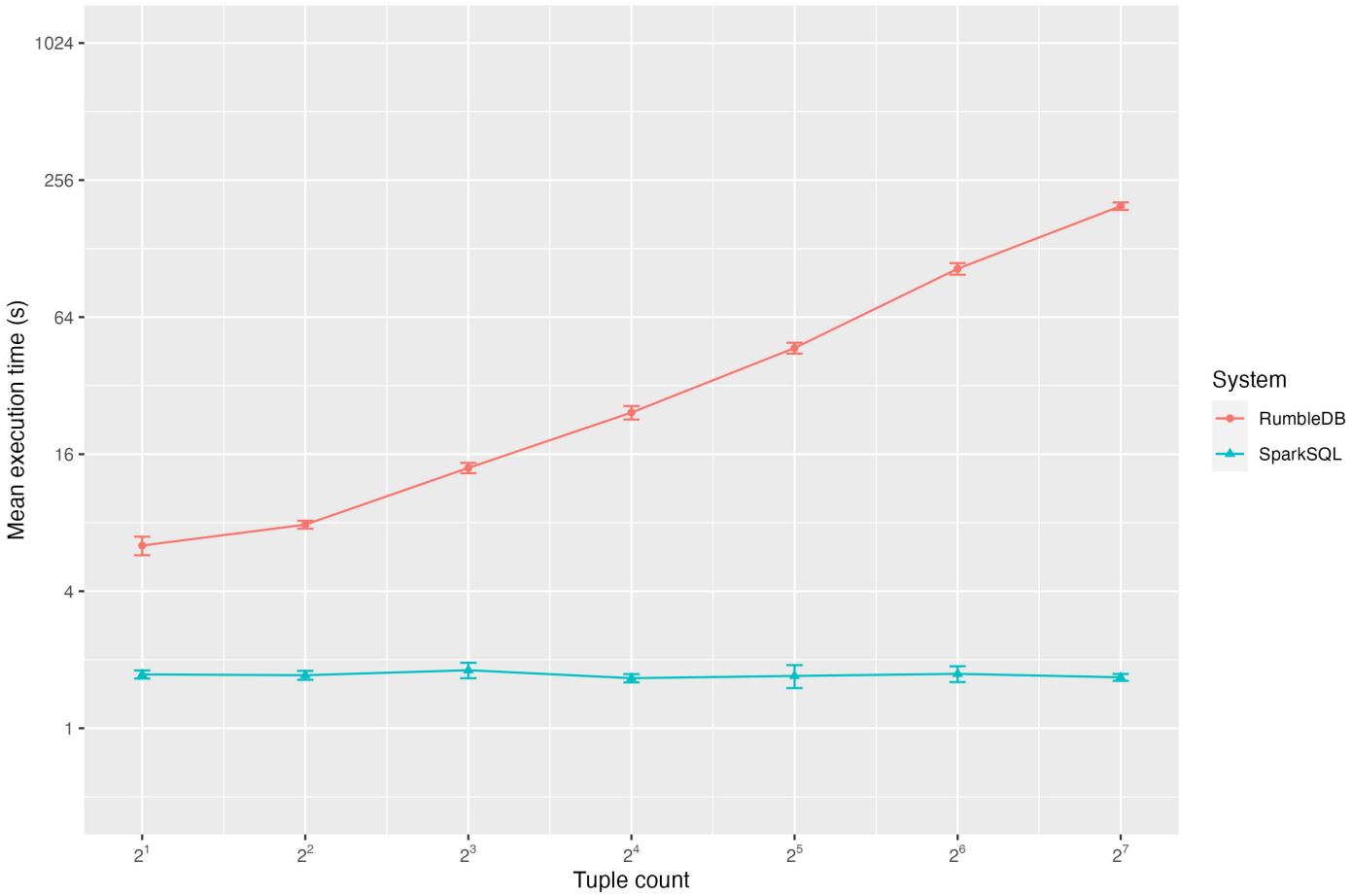


Figure: Comparison of performance for multi-nested insertion on GitHub Archive dataset between RumbleDB and SparkSQL, on $\log_2$ scales

RumbleDB Insights

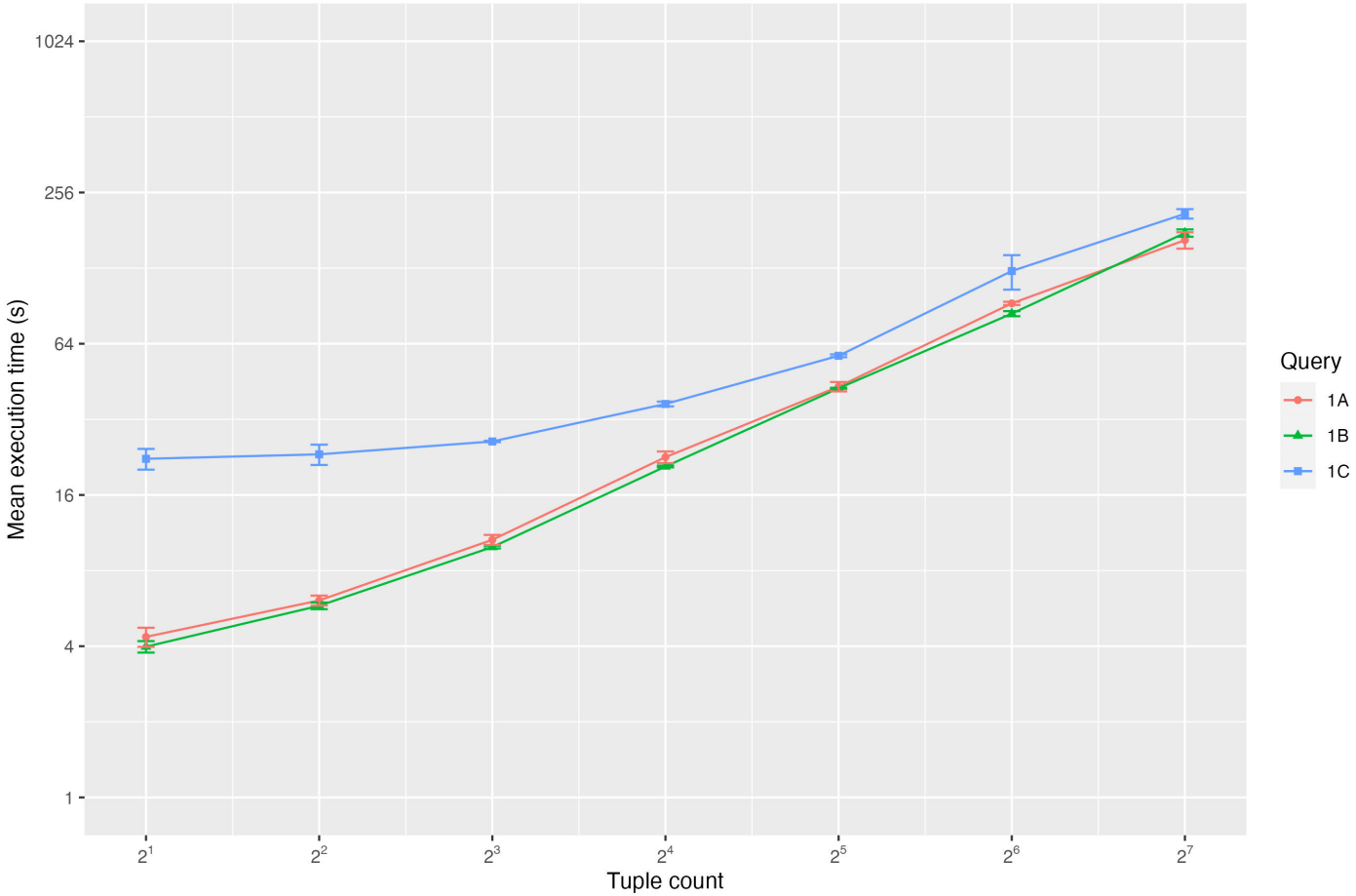


Figure: Comparison of performance of RumbleDB for simple replacement (1A), conditional replacement (1B), and replacement with a join (1C) on TPC-C dataset, on $\log_2$ scales

RumbleDB Insights

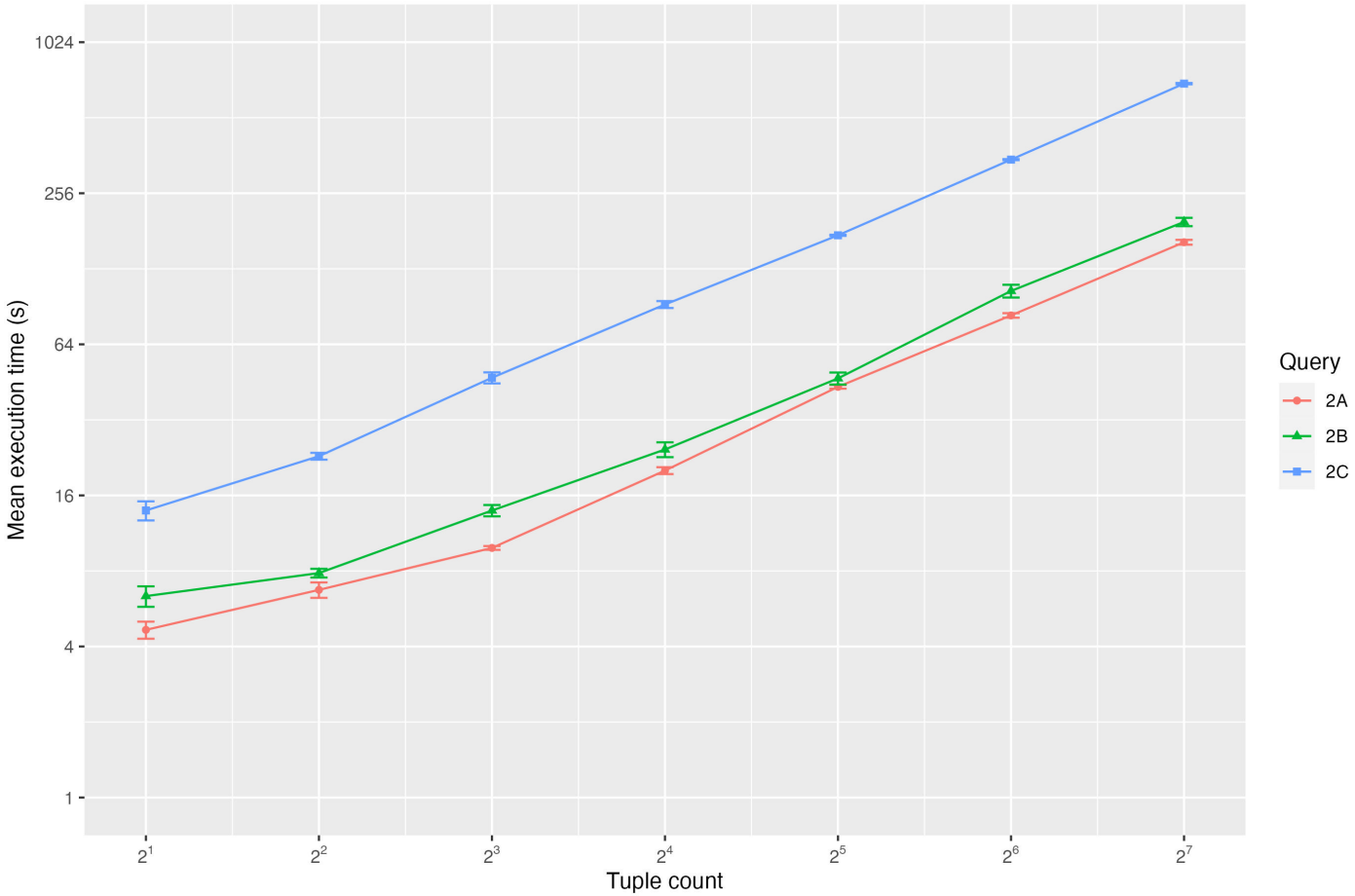


Figure: Comparison of performance of RumbleDB for simple replacement (2A), multi-nested insertion (2B), and nested array insertions and deletions (2C) on GitHub Archive dataset, on $\log_2$ scales

Conclusions

Conclusions

- Local execution for JSONiq updates is available in RumbleDB!

Conclusions

- Local execution for JSONiq updates is available in RumbleDB!
- Update primitive decomposition and mutability scoping models are crucial

Conclusions

- Local execution for JSONiq updates is available in RumbleDB!
- Update primitive decomposition and mutability scoping models are crucial
- JSONiq updates can be persisted to a Delta Table

Conclusions

- Local execution for JSONiq updates is available in RumbleDB!
- Update primitive decomposition and mutability scoping models are crucial
- JSONiq updates can be persisted to a Delta Table
- Performance against SparkSQL is poor and worsens linearly

Conclusions

- Local execution for JSONiq updates is available in RumbleDB!
- Update primitive decomposition and mutability scoping models are crucial
- JSONiq updates can be persisted to a Delta Table
- Performance against SparkSQL is poor and worsens linearly
- Much scope for improvement

Future Work

- Improved update primitives: target collections, target non-structured items
- Persisting updates in a document store, e.g. MongoDB

Thank you! Check out our work!

Work available in RumbleDB Version 1.22.0

–

The "Pyrenean oak" beta

Thank you! Any questions?

See title! Don't look down here!