Teaching Datafusion about Struct Fields

Matthew

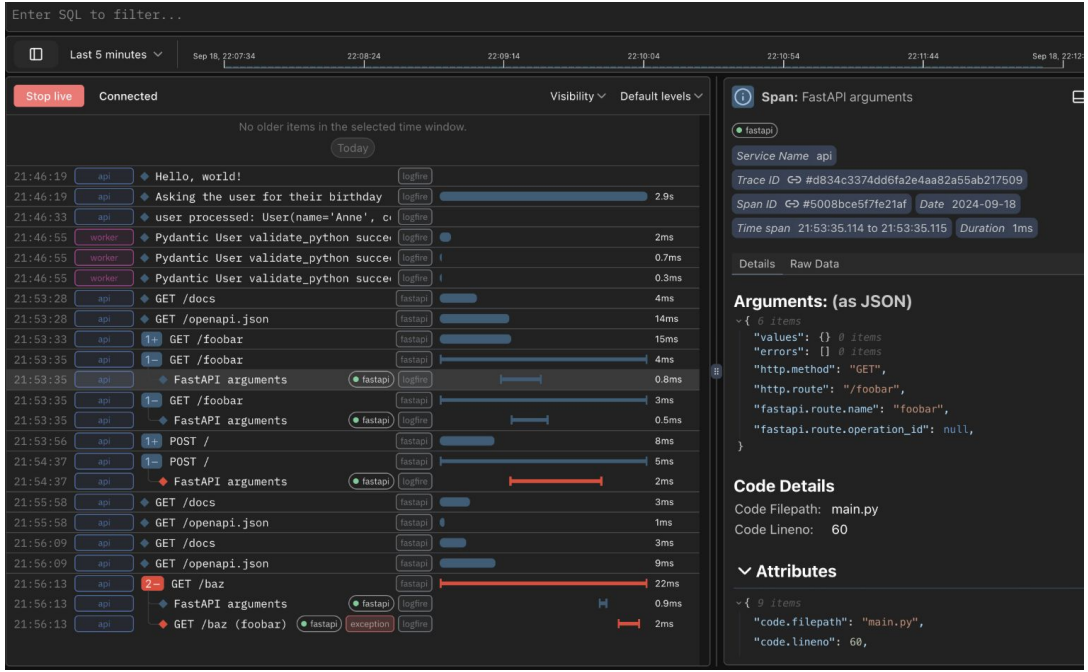
friendlymatthew on Github

I love file formats

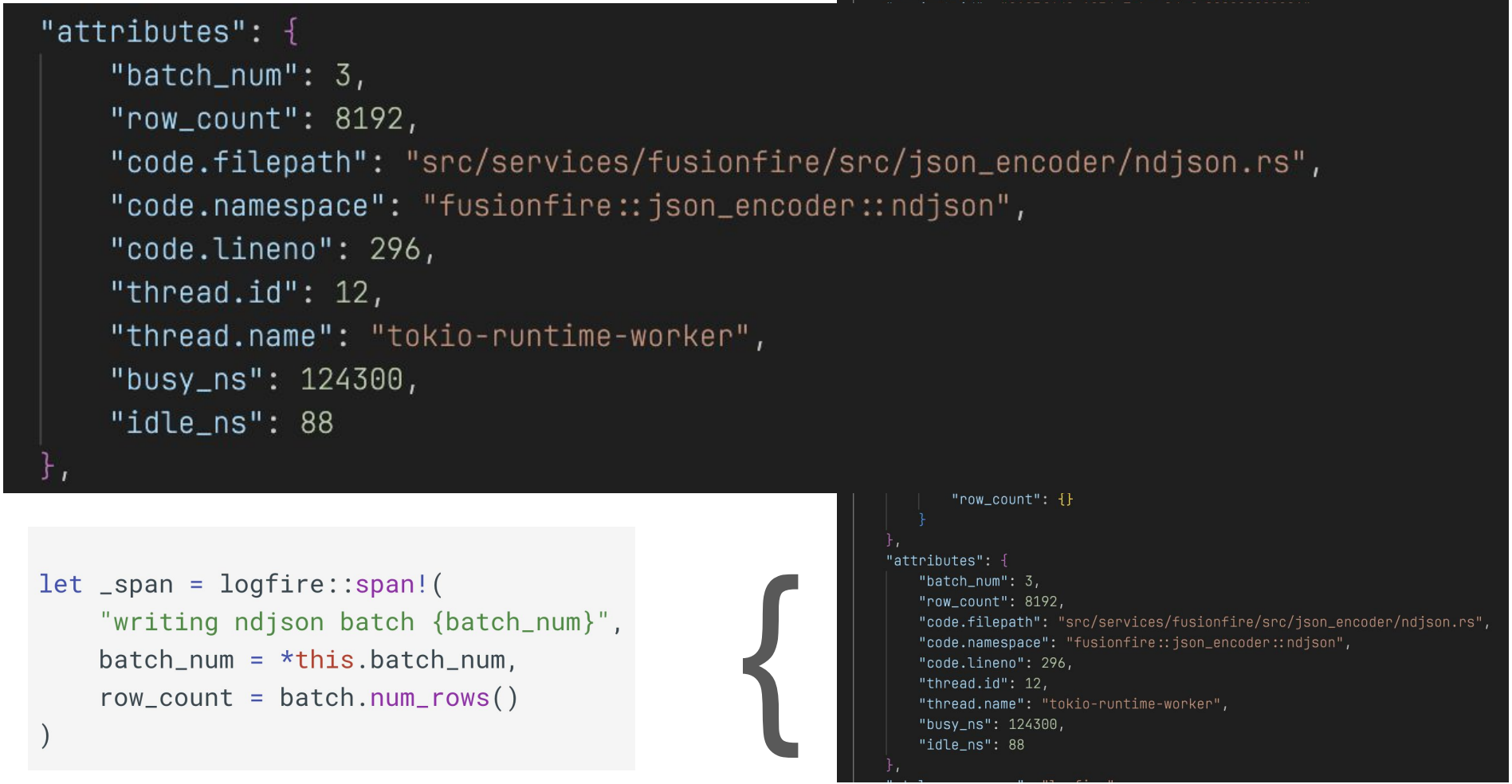
Work at Pydantic

We built our own database for our observability platform

```
let _span = logfire::span!(
    "writing ndjson batch {batch_num}",
    batch_num = *this.batch_num,
    row_count = batch.num_rows()
)
```



```rust
let _span = logfire::span!(
    "writing ndjson batch {batch_num}",
    batch_num = *this.batch_num,
    row_count = batch.num_rows()
)
```


"attributes": {
"batch_num": 3,
"row_count": 8192,
"code.filepath": "src/services/fusionfire/src/json_encoder/ndjson.rs",
"code.namespace": "fusionfire::json_encoder::ndjson",
"code.lineno": 296,
"thread.id": 12,
"thread.name": "tokio-runtime-worker",
"busy_ns": 124300,
"idle_ns": 88
},
let _span = logfire::span!(
"writing ndjson batch {batch_num}",
batch_num = *this.batch_num,
row_count = batch.num_rows()
)
"row_count": {}
},
"attributes": {
"batch_num": 3,
"row_count": 8192,
"code.filepath": "src/services/fusionfire/src/json_encoder/ndjson.rs",
"code.namespace": "fusionfire::json_encoder::ndjson",
"code.lineno": 296,
"thread.id": 12,
"thread.name": "tokio-runtime-worker",
"busy_ns": 124300,
"idle_ns": 88
},

There’s an impedance mismatch when we handle attributes

Existing solution: store attributes as a stringified JSON column

```
"attributes": {
    "batch_num": 3,
    "row_count": 8192,
    "code.filepath": "src/services/fusionfire/src/json_encoder/ndjson.rs",
    "code.namespace": "fusionfire::json_encoder::ndjson",
    "code.lineno": 296,
    "thread.id": 12,
    "thread.name": "tokio-runtime-worker",
    "busy_ns": 124300,
    "idle_ns": 88
},
```

```
select *
from records
where
    span_name = 'writing ndjson batch {batch_num}' and
    attributes->>'batch_num' < 3;
```

We can’t push the `attributes` filter down

We need to decode the entire JSON value just to evaluate one key

New datatype: Variant

“A better JSON data for OLAP workloads”

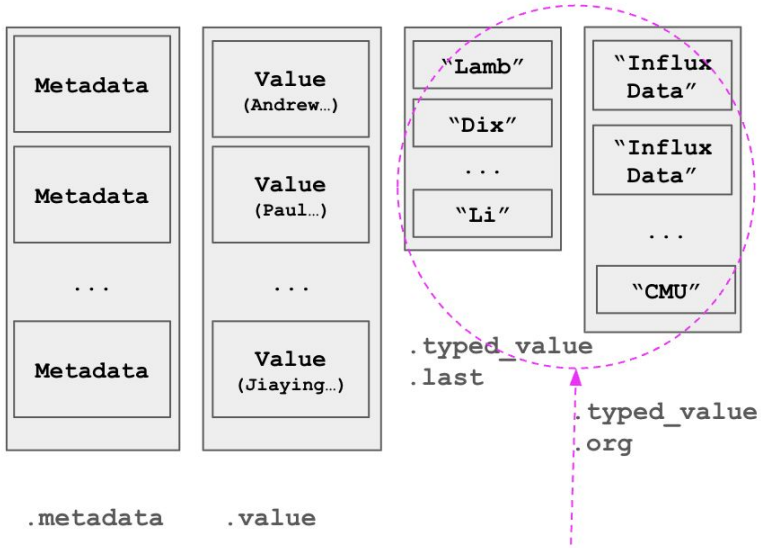
Variant Shredding in Parquet

```
GROUP
{
  metadata: binary,
  value: binary,
  typed_value: GROUP {
    last: string
    org: string
  }
} Some (not all) fields are "shredded"
```

Parquet Schema

```
[
 {
  "first": "Andrew",
  "last":  "Lamb",
  "org":   "InfluxData",
 },
 {
  "first": "Paul",
  "last":  "Dix",
  "org":   "InfluxData",
 },
...more here...
 {
  "first": "Jiaying",
  "last":  "Li",
  "org":   "CMU",
 },
]
```

JSON encoding



Parquet Storage (4 leaves)

See spec here: [VariantShredding.md]

From alamb's slides

We want to use Variant because it has better read/write performance

Variant data can be generalized to Struct data

We need to teach Datafusion how to read Struct data efficiently

What are struct columns?

Same data, two perspectives

How struct fields are represented

Arrow / logical view

attributes : struct
{ user_id: "u1", request_id: "r1",
http: { method: "GET", status: 200 } }
{ user_id: "u2", request_id: "r2",
http: { method: "POST", status: 201 } }
{ user_id: "u3", request_id: "r3",
http: { method: "GET", status: 404 } }

Parquet / physical layout — one column per leaf

user_id	request_id	http.method	http.status_code
u1	r1	GET	200
u2	r2	POST	201
u3	r3	GET	404
leaf 0	leaf 1	leaf 2	leaf 3

The struct "attributes" never touches disk. Only the four leaves do.

Only leaves have physical storage

```
attributes: struct          ← branch, no physical column
├── user_id:    string      ← leaf  → physical col 0
├── request_id: string      ← leaf  → physical col 1
└── http: struct            ← branch
    ├── method:      string ← leaf  → physical col 2
    └── status_code: int    ← leaf  → physical col 3
```

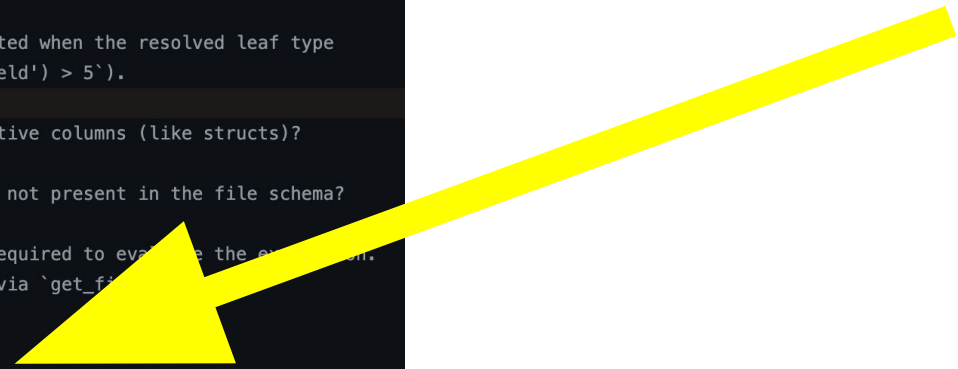
- Structs reconstructed at read time via repetition + definition levels
- Stats (min/max), dictionaries, page indices all live per-leaf

Teaching the pushdown checker about struct fields

```
select *
from records
where
    span_name = 'writing ndjson batch {batch_num}' and
    attributes->>'batch_num' < 3;
```

Teaching the pushdown checker about struct fields

```rust
/// Traverses a `PhysicalExpr` tree to determine if any column references would
/// prevent the expression from being pushed down to the parquet decoder.
///
/// An expression cannot be pushed down if it references:
/// - Unsupported nested columns (whole struct references or list fields that are
///   not covered by the supported predicate set)
/// - Columns that don't exist in the file schema
///
/// Struct field access via `get_field` is supported when the resolved leaf type
/// is primitive (e.g. `get_field(struct_col, 'field') > 5`).
struct PushdownChecker<'schema> {
    /// Does the expression require any non-primitive columns (like structs)?
    non_primitive_columns: bool,
    /// Does the expression reference any columns not present in the file schema?
    projected_columns: bool,
    /// Indices into the file schema of columns required to ev[illegible] the ex[illegible]n.
    /// Does not include struct columns accessed via `get_fi[illegible]
    required_columns: Vec<usize>,
    /// Struct field accesses via `get_field`.
    struct_field_accesses: Vec<StructFieldAccess>,
    /// Whether nested list columns are supported by the predicate semantics.
    allow_list_columns: bool,
    /// The Arrow schema of the parquet file.
    file_schema: &'schema Schema,
}
```



Teaching the pushdown checker about struct fields

```rust
impl TreeNodeVisitor<'_> for PushdownChecker<'_> {
    type Node = Arc<dyn PhysicalExpr>;

    fn f_down(&mut self, node: &Self::Node) -> Result<TreeNodeRecursion> {
        // Handle struct field access like `s['foo']['bar'] > 10`.
        //
        // DataFusion represents nested field access as `get_field(Column("s"), "foo")`
        // (or chained: `get_field(get_field(Column("s"), "foo"), "bar")`).
        //
        // We intercept the outermost `get_field` on the way *down* the tree so
        // the visitor never reaches the raw `Column("s")` node. Without this,
        // `check_single_column` would see that `s` is a Struct and reject it.
        //
        // The strategy:
        //   1. Match `get_field` whose first arg is a `Column` (the struct root).
        //   2. Check that the *resolved* return type is primitive — meaning we've
        //      drilled all the way to a leaf (e.g. `s['foo']` → Utf8).
        //   3. Record the root column index via `check_struct_field_column` and
        //      return `Jump` to skip visiting the children (the Column and the
        //      literal field-name args), since we've already handled them.
        //
        // If the return type is still nested (e.g. `s['nested_struct']` → Struct),
        // we fall through and let normal traversal continue, which will
        // eventually reject the expression when it hits the struct Column.
        if let Some(func) =
            ScalarFunctionExpr::try_downcast_func::<GetFieldFunc>(node.as_ref())
        {
```

Teaching the pushdown checker about struct fields

parquet_struct_projection bench, c4a-highmem-16, PR #20925

Single-field projection from a struct | branch vs. main

shape	bench	main	branch	speedup
Wide struct (5 leaves)	select_small_field	8.2 ms	419 µs	**19.5×**
Wide struct	sum_small_field	8.2 ms	638 µs	**12.8×**
Wide struct	select_one_string_field	8.3 ms	671 µs	**12.3×**
Nested struct	select_extra_string	3.4 ms	685 µs	**4.9×**
Nested struct	sum_inner_small_field	1.16 ms	662 µs	**1.75×**
Narrow struct (2 leaves)	select_small_field	654 µs	422 µs	**1.55×**
Wide struct	select_all	9.5 ms	10.0 ms	**≈1×**

Speedup scales with what gets skipped. select_all unchanged. nothing to skip, nothing to win.

thanks

Click to add text