

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()  
)
```

The screenshot displays the Logfire observability platform interface. The main view shows a timeline of events from 21:46:19 to 21:56:13. The events include:

- 21:46:19: api Hello, world! (logfire)
- 21:46:19: api Asking the user for their birthday (logfire) - 2.9s
- 21:46:33: api user processed: User(name='Anne', c... (logfire)
- 21:46:55: worker Pydantic User validate_python succe... (logfire) - 2ms
- 21:46:55: worker Pydantic User validate_python succe... (logfire) - 0.7ms
- 21:46:55: worker Pydantic User validate_python succe... (logfire) - 0.3ms
- 21:53:28: api GET /docs (fastapi) - 4ms
- 21:53:28: api GET /openapi.json (fastapi) - 14ms
- 21:53:33: api GET /foobar (fastapi) - 15ms
- 21:53:35: api GET /foobar (fastapi) - 4ms
- 21:53:35: api FastAPI arguments (fastapi) - 0.8ms
- 21:53:35: api GET /foobar (fastapi) - 3ms
- 21:53:35: api FastAPI arguments (fastapi) - 0.5ms
- 21:53:56: api POST / (fastapi) - 8ms
- 21:54:37: api POST / (fastapi) - 5ms
- 21:54:37: api FastAPI arguments (fastapi) - 2ms
- 21:55:58: api GET /docs (fastapi) - 3ms
- 21:55:58: api GET /openapi.json (fastapi) - 1ms
- 21:56:09: api GET /docs (fastapi) - 3ms
- 21:56:09: api GET /openapi.json (fastapi) - 9ms
- 21:56:13: api GET /baz (fastapi) - 22ms
- 21:56:13: api FastAPI arguments (fastapi) - 0.9ms
- 21:56:13: api GET /baz (foobar) (fastapi) exception (logfire) - 2ms

The right-hand panel shows a detailed view of a span titled "Span: FastAPI arguments". The span details include:

- Service Name: api
- Trace ID: #d834c3374dd6fa2e4aa82a55ab217509
- Span ID: #5008bce5f7fe21af, Date: 2024-09-18
- Time span: 21:53:35.114 to 21:53:35.115, Duration: 1ms

The "Arguments: (as JSON)" section shows the following JSON structure:

```
{  
  "values": {},  
  "errors": [],  
  "http.method": "GET",  
  "http.route": "/foobar",  
  "fastapi.route.name": "foobar",  
  "fastapi.route.operation_id": null,  
}
```

The "Code Details" section shows the code filepath as "main.py" and the code line number as 60.

The "Attributes" section shows the following JSON structure:

```
{  
  "code.filepath": "main.py",  
  "code.lineno": 60,  
}
```

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

```
"project_id": "0193f4d0-1234-7abc-9def-000000000001",
"day": "2026-05-11",
"start_timestamp": "2026-05-11T14:23:01.482931Z",
"end_timestamp": "2026-05-11T14:23:01.483055Z",
"duration": 0.000124,
"trace_id": "01972f5c1d8a7a3c9e0b4f2a1d8e3c5a",
"span_id": "9e0b4f2a1d8e3c5a",
"parent_span_id": "7c1a3f5b2d4e6080",
"kind": "span",
"level": 9,
"span_name": "writing ndjson batch {batch_num}",
"message": "writing ndjson batch 3",
"otel_status_code": null,
"otel_status_message": null,
"otel_links": null,
"otel_events": null,
"is_exception": false,
"exception_message": null,
"exception_type": null,
"exception_stacktrace": null,
"attributes_json_schema": {
  "type": "object",
  "properties": {
    "batch_num": {},
    "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
},
```

```
"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
  }
}
```

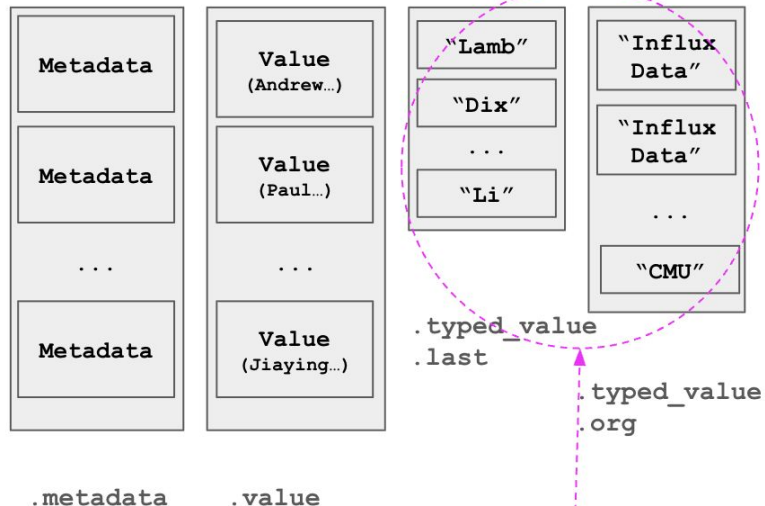
} Some (not all) fields are "shredded"

Parquet Schema

See spec here: [VariantShredding.md](#)

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

JSON encoding



And stored as separate columns

Parquet Storage (4 leaves)

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
```

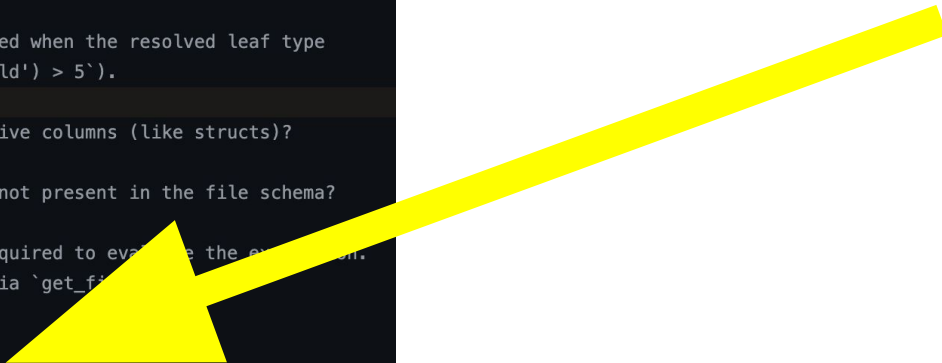
- Structs reconstructed at read time via repetition + definition levels
- Stats (min/max), dictionaries, page indices all live per-leaf

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

```
/// 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 evaluate the expression.
    /// Does not include struct columns accessed via `get_field`.
    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

```

    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 μ s	19.5x
Wide struct	sum_small_field	8.2 ms	638 μ s	12.8x
Wide struct	select_one_string_field	8.3 ms	671 μ s	12.3x
Nested struct	select_extra_string	3.4 ms	685 μ s	4.9x
Nested struct	sum_inner_small_field	1.16 ms	662 μ s	1.75x
Narrow struct (2 leaves)	select_small_field	654 μ s	422 μ s	1.55x
Wide struct	select_all	9.5 ms	10.0 ms	\approx 1x

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

thanks

Click to add text