FACTSET

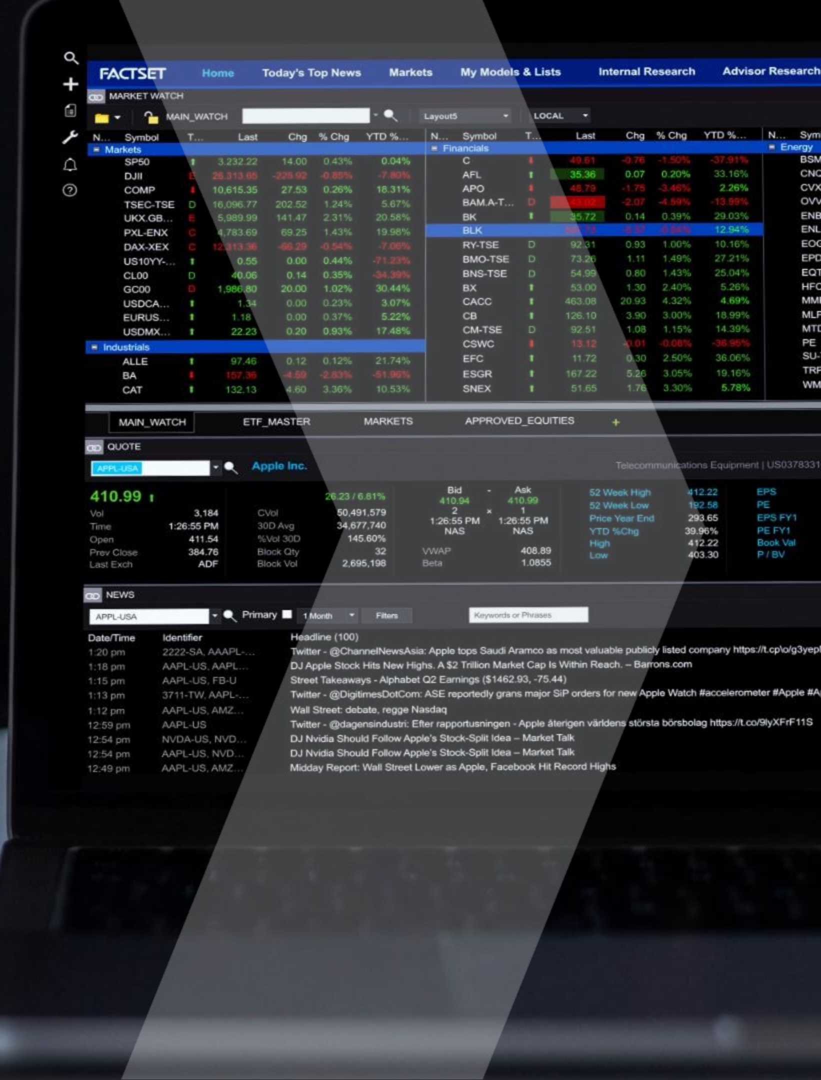
March 2 - 3, 2022 – Subsurface LIVE Winter 2022

Powering a Data Mesh with Dynamic GraphQL Schema Generation

Matthew Topol, Principal Software Architect

Steve Perkins, Lead Software Engineer

Content Engineering



FACTSET

Agenda

- Why would you need this? (What were we trying to solve?)
- What about Federation?
- The Tech We're Using
- Flexibility Through Metadata
- Benefits of Dynamic Schema Generation
- Future Optimizations and Tooling
- Q&A

FACTSET

Why would you need this?

Consistent New Data Points

The data points that product developers want to expose is continuously expanding. Onboarding new data points needs to be quick and painless.

Multiple Different Data Stores

The data that we need to expose is stored in multiple different technologies (postgres, MSSQL, parquet) and can not currently be migrated to something consistent.

Non-Standard Schemas

The schemas for the various data stores are non-consistent. To ease data fetches for consumers, these should be standardized.

Multiple Internal Consumers

Not only do different internal consumers exist for this data, but their query patterns also vary. Fetching the data needs to be straightforward and not require large changes on the consumer side. Data also needs to be consistent across all consumers

Public

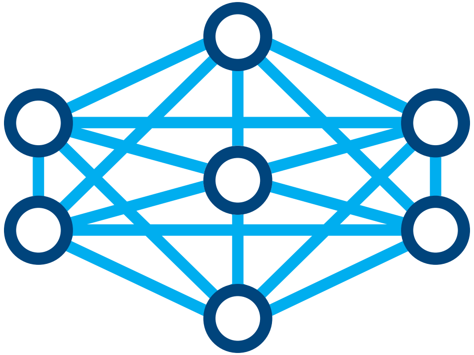
FACTSET

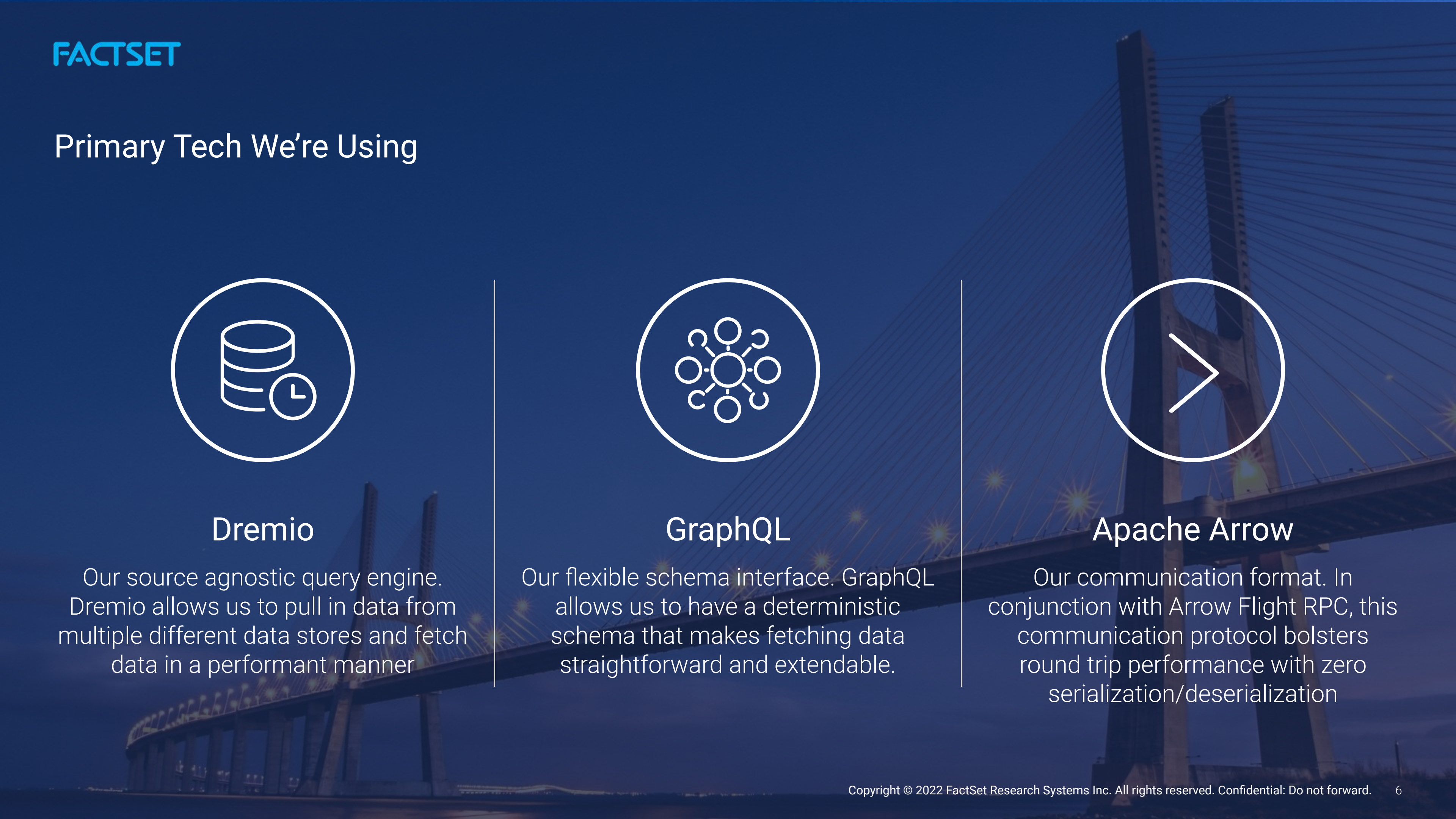
Additional Considerations

- Abstraction from data source
  - Hide the data mesh complexity
- Data consistency and timeliness
- Cross dataset queries
- Performance
  - Query performance
  - Data transfer performance

Why didn’t we create a Federated GraphQL Service? (It’s all the rage right now)

- A Federated Schema != A Standard Schema
- Maintaining multiple different services amongst various teams proved difficult over time
- Ensuring conformance updates were implemented across multiple services was not reliable
- Single-entry-point infrastructure allows us to streamline automation, improve development time, and quality assurance
  - Easier to enforce conformance to standards
- Push joins down to query engine
  - More performant





FACTSET

Primary Tech We’re Using

Dremio

Our source agnostic query engine. Dremio allows us to pull in data from multiple different data stores and fetch data in a performant manner

GraphQL

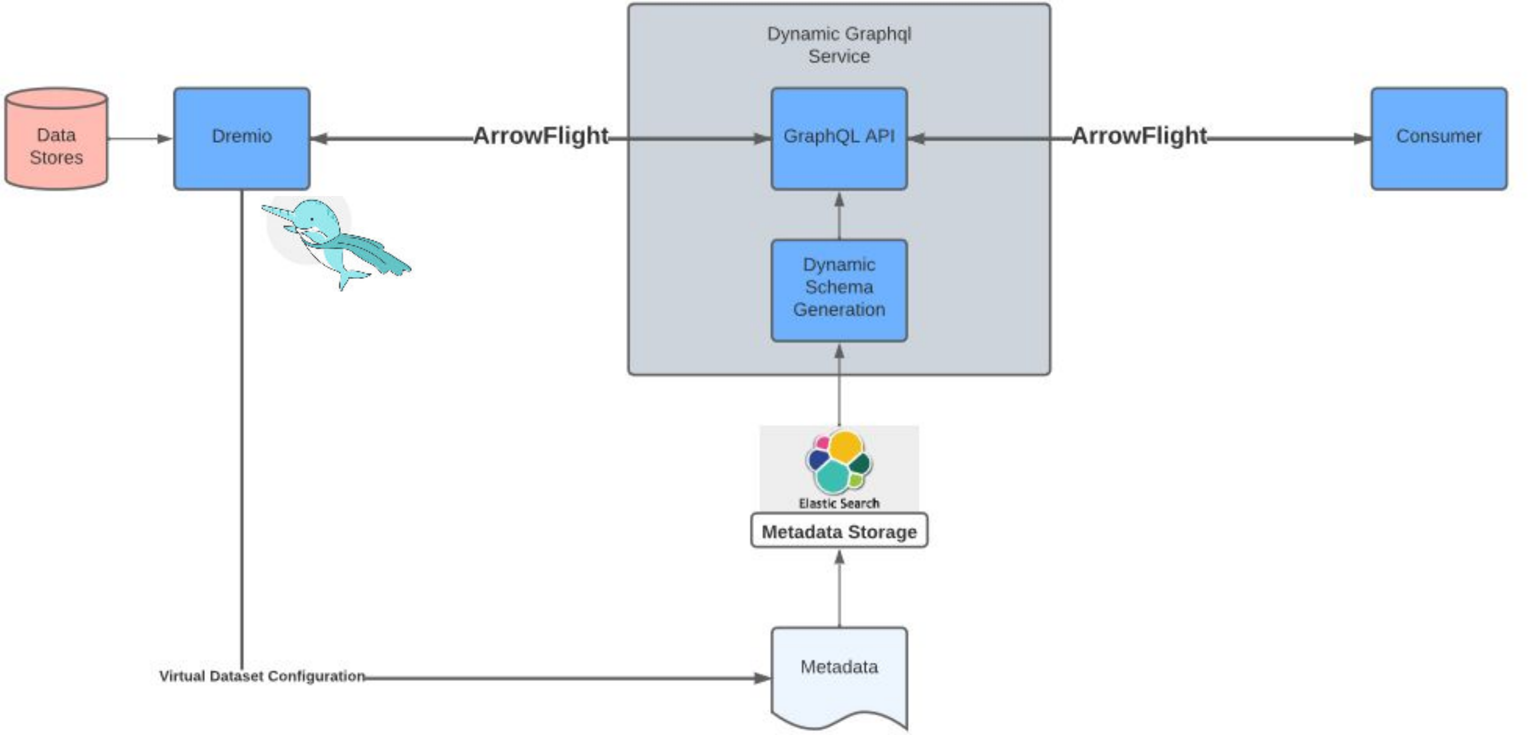
Our flexible schema interface. GraphQL allows us to have a deterministic schema that makes fetching data straightforward and extendable.

Apache Arrow

Our communication format. In conjunction with Arrow Flight RPC, this communication protocol bolsters round trip performance with zero serialization/deserialization

FACTSET

High Level Architecture



Additional Technologies We’re Using

- Programming language
  - Golang
- Communication framework
  - Arrow Flight RPC over gRPC
- Cluster Deployment
  - Kubernetes
- Golang Libraries
  - graphql-go-tools - github.com/jensneuse/graphql-go-tools
  - Apache Arrow - github.com/apache/arrow/go/v7/arrow

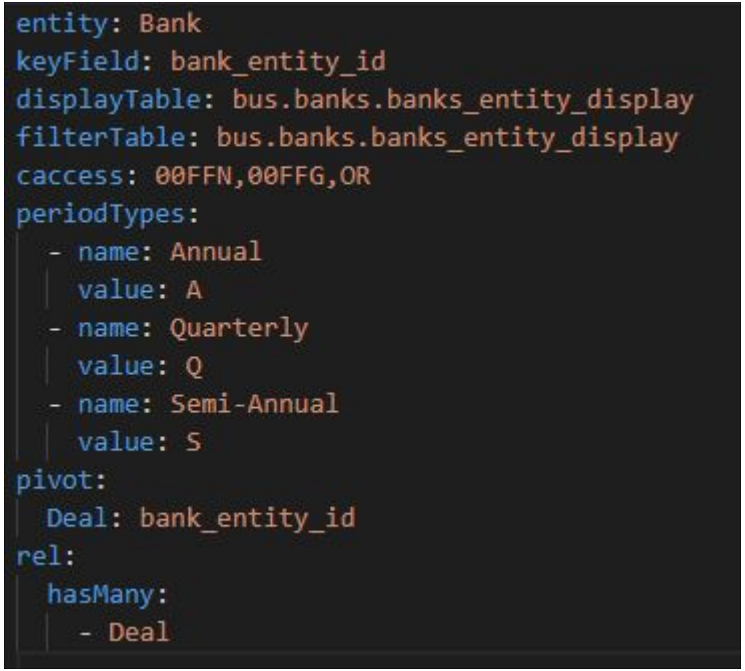
Flexibility through Metadata

Metadata Drives All

- Currently our metadata is driven by a fairly simple YAML configuration
  - Split into Entities and Data Items
- Multiple levels of automated schema validation to ensure expectations are met
  - Metadata schema defined using JSON Schema which can be applied to YAML
- Metadata can be owned and manipulated by content teams to provide a self-service API
  - Faster time-to-market for exposing new data!
- Schema generation is flexible enough to ignore unrecognized attributes until they are leveraged
- Deterministic schema generation leads to easily deterministic Query Generation!

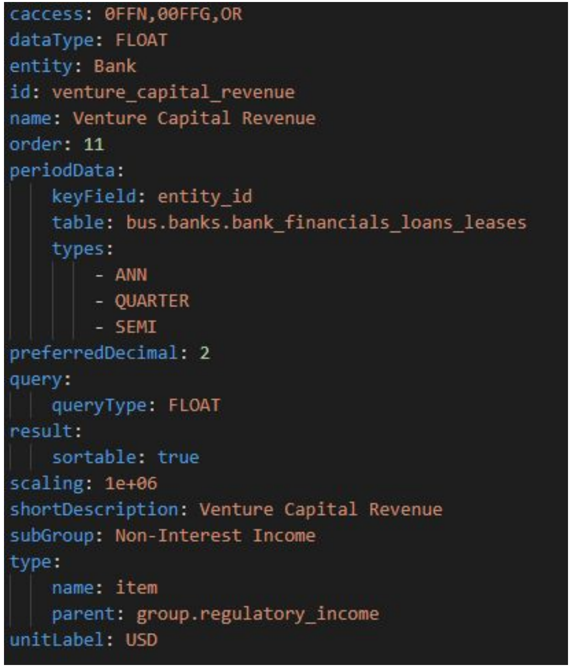
Metadata Example

Entity



```yaml
entity: Bank
keyField: bank_entity_id
displayTable: bus.banks.banks_entity_display
filterTable: bus.banks.banks_entity_display
caccess: 00FFN,00FFG,OR
periodTypes:
  - name: Annual
    value: A
  - name: Quarterly
    value: Q
  - name: Semi-Annual
    value: S
pivot:
  Deal: bank_entity_id
rel:
  hasMany:
    - Deal
```

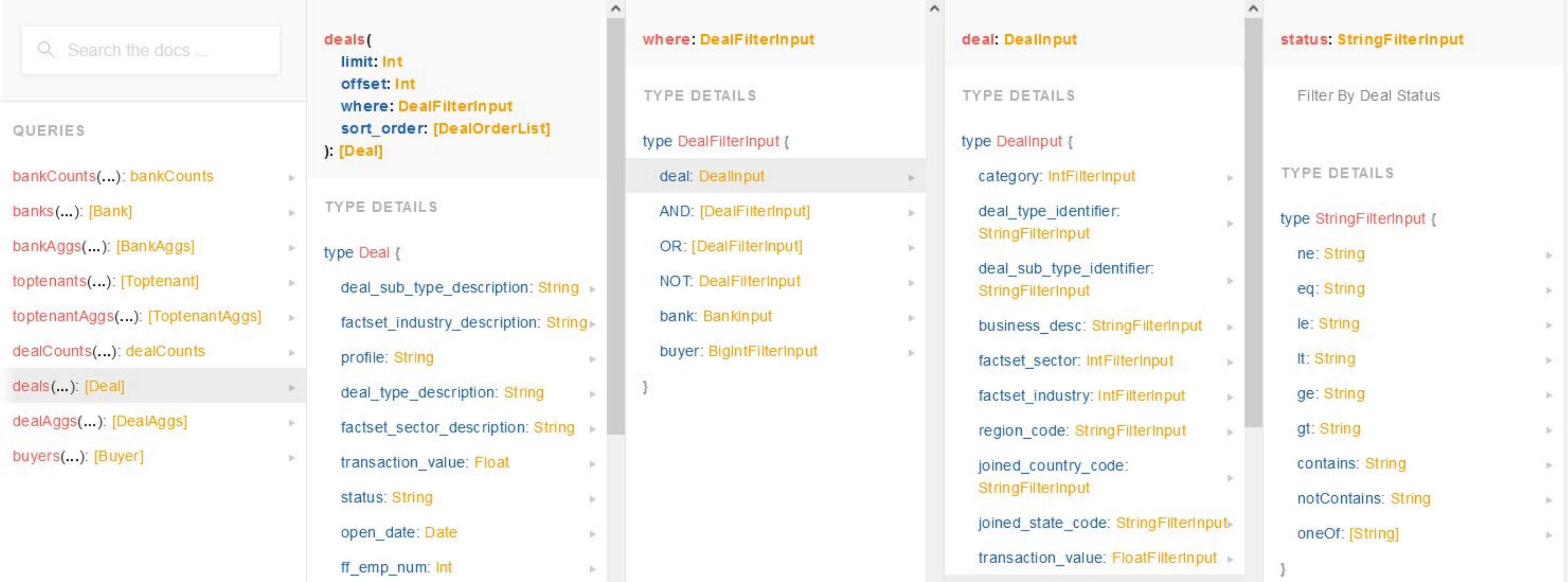
Data Item



```yaml
caccess: 0FFN,00FFG,OR
dataType: FLOAT
entity: Bank
id: venture_capital_revenue
name: Venture Capital Revenue
order: 11
periodData:
  keyField: entity_id
  table: bus.banks.bank_financials_loans_leases
  types:
    - ANN
    - QUARTER
    - SEMI
preferredDecimal: 2
query:
  queryType: FLOAT
result:
  sortable: true
scaling: 1e+06
shortDescription: Venture Capital Revenue
subGroup: Non-Interest Income
type:
  name: item
  parent: group.regulatory_income
unitLabel: USD
```

Public

Meshing Around With Different Schemas



```
Search the docs ...

QUERIES

bankCounts(...): bankCounts
banks(...): [Bank]
bankAggs(...): [BankAggs]
toptenants(...): [Toptenant]
toptenantAggs(...): [ToptenantAggs]
dealCounts(...): dealCounts
deals(...): [Deal]
dealAggs(...): [DealAggs]
buyers(...): [Buyer]
```

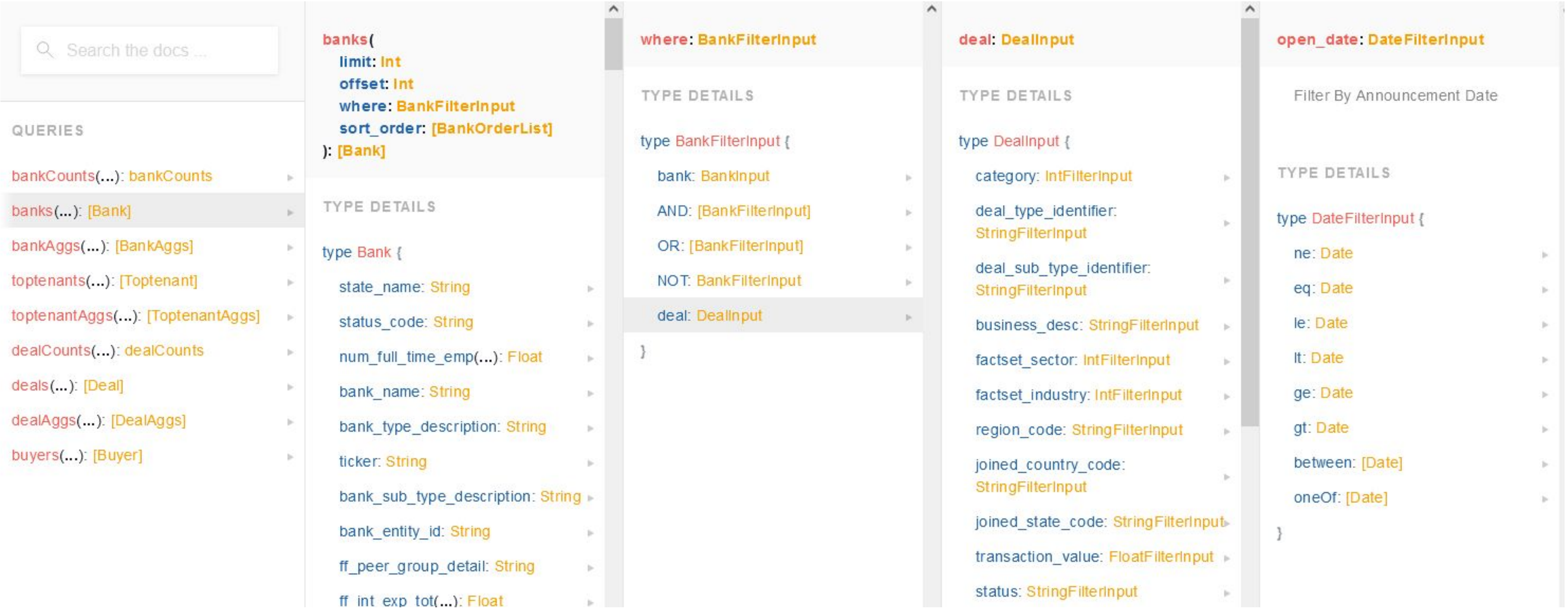
```
deals(
  limit: Int
  offset: Int
  where: DealFilterInput
  sort_order: [DealOrderList]
): [Deal]

TYPE DETAILS

type Deal {
  deal_sub_type_description: String
  factset_industry_description: String
  profile: String
  deal_type_description: String
  factset_sector_description: String
  transaction_value: Float
  status: String
  open_date: Date
  ff_emp_num: Int
```

```
where: DealFilterInput

TYPE DETAILS

type DealFilterInput {
  deal: DealInput
  AND: [DealFilterInput]
  OR: [DealFilterInput]
  NOT: DealFilterInput
  bank: BankInput
  buyer: BigIntFilterInput
}
```

```
deal: DealInput

TYPE DETAILS

type DealInput {
  category: IntFilterInput
  deal_type_identifier:
  StringFilterInput
  deal_sub_type_identifier:
  StringFilterInput
  business_desc: StringFilterInput
  factset_sector: IntFilterInput
  factset_industry: IntFilterInput
  region_code: StringFilterInput
  joined_country_code:
  StringFilterInput
  joined_state_code: StringFilterInput
  transaction_value: FloatFilterInput
```

```
status: StringFilterInput

Filter By Deal Status

TYPE DETAILS

type StringFilterInput {
  ne: String
  eq: String
  le: String
  lt: String
  ge: String
  gt: String
  contains: String
  notContains: String
  oneOf: [String]
}
```

Produced GraphQL Schema:



Search the docs ...

QUERIES

```
bankCounts(...): bankCounts
banks(...): [Bank]
bankAggs(...): [BankAggs]
toptenants(...): [Toptenant]
toptenantAggs(...): [ToptenantAggs]
dealCounts(...): dealCounts
deals(...): [Deal]
dealAggs(...): [DealAggs]
buyers(...): [Buyer]
```

```
banks(
  limit: Int
  offset: Int
  where: BankFilterInput
  sort_order: [BankOrderList]
): [Bank]
```

TYPE DETAILS

```
type Bank {
  state_name: String
  status_code: String
  num_full_time_emp(...): Float
  bank_name: String
  bank_type_description: String
  ticker: String
  bank_sub_type_description: String
  bank_entity_id: String
  ff_peer_group_detail: String
  ff_int_exp_tot(...): Float
```

```
where: BankFilterInput
```

TYPE DETAILS

```
type BankFilterInput {
  bank: BankInput
  AND: [BankFilterInput]
  OR: [BankFilterInput]
  NOT: BankFilterInput
  deal: DealInput
}
```

```
deal: DealInput
```

TYPE DETAILS

```
type DealInput {
  category: IntFilterInput
  deal_type_identifier:
  StringFilterInput
  deal_sub_type_identifier:
  StringFilterInput
  business_desc: StringFilterInput
  factset_sector: IntFilterInput
  factset_industry: IntFilterInput
  region_code: StringFilterInput
  joined_country_code:
  StringFilterInput
  joined_state_code: StringFilterInput
  transaction_value: FloatFilterInput
  status: StringFilterInput
```

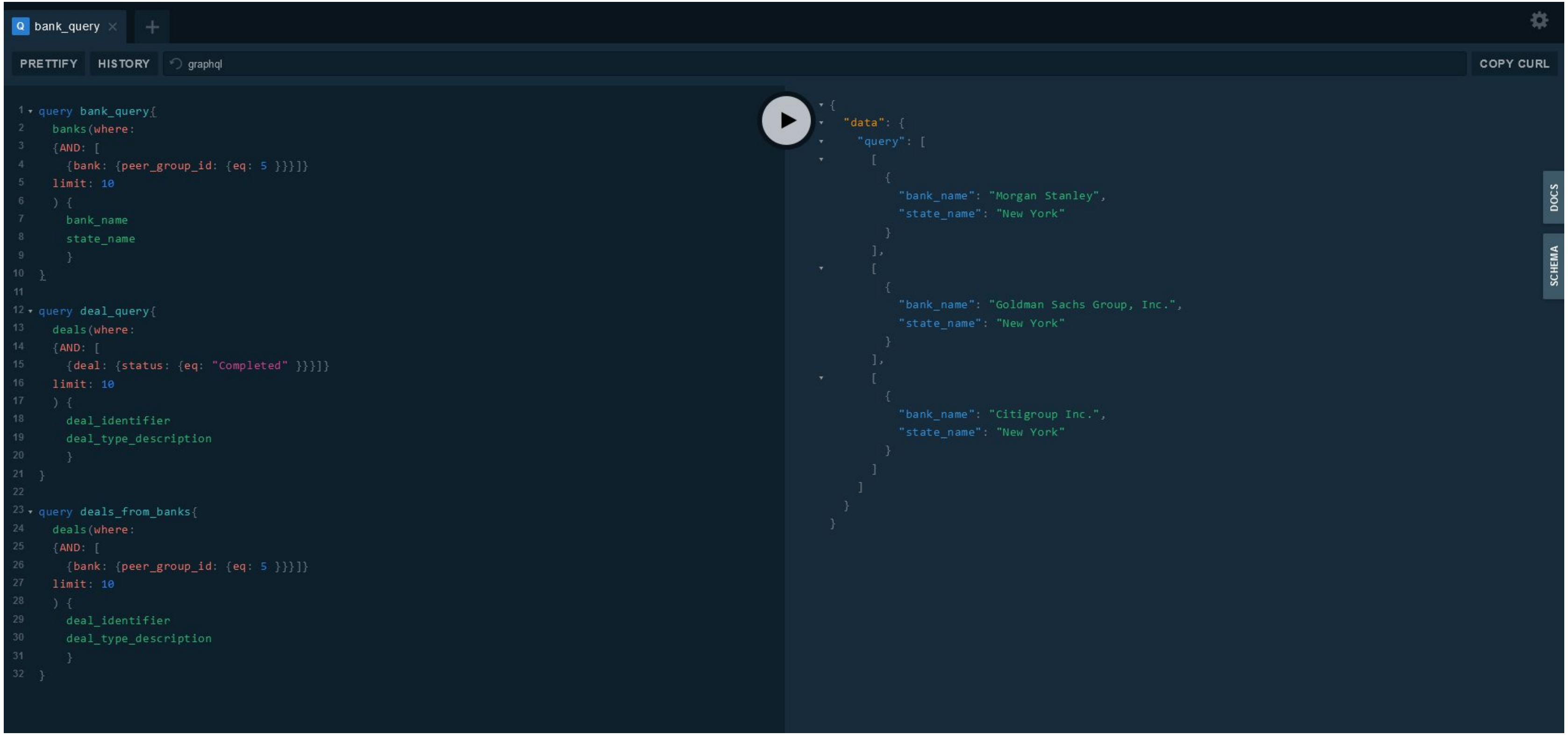
```
open_date: DateFilterInput
```

Filter By Announcement Date

TYPE DETAILS

```
type DateFilterInput {
  ne: Date
  eq: Date
  le: Date
  lt: Date
  ge: Date
  gt: Date
  between: [Date]
  oneOf: [Date]
}
```

One Query Pattern To Rule Them All



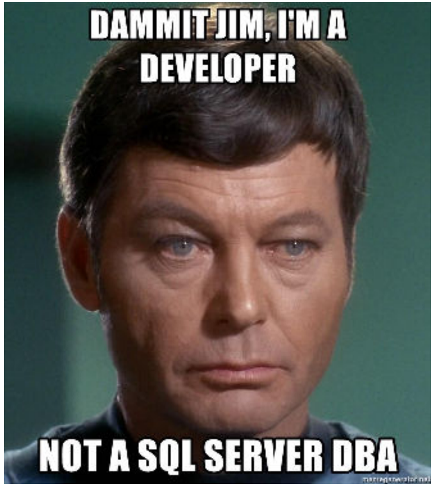
```
query bank_query{
  banks(where:
  {AND: [
    {bank: {peer_group_id: {eq: 5 }}}]}
  limit: 10
  ) {
    bank_name
    state_name
    }
}

query deal_query{
  deals(where:
  {AND: [
    {deal: {status: {eq: "Completed" }}}]}
  limit: 10
  ) {
    deal_identifier
    deal_type_description
    }
}

query deals_from_banks{
  deals(where:
  {AND: [
    {bank: {peer_group_id: {eq: 5 }}}]}
  limit: 10
  ) {
    deal_identifier
    deal_type_description
    }
}
```

```
{
  "data": {
    "query": [
      [
        {
          "bank_name": "Morgan Stanley",
          "state_name": "New York"
        }
      ],
      [
        {
          "bank_name": "Goldman Sachs Group, Inc.",
          "state_name": "New York"
        }
      ],
      [
        {
          "bank_name": "Citigroup Inc.",
          "state_name": "New York"
        }
      ]
    ]
  }
}
```

Public

Benefits of Dynamic Schema Generation

- Faster time to market for new data points
- Ease of automated extension
  - Define relationships in metadata for easy one-to-many / many-to-many query generation
- Data source abstraction
  - Consumers write GraphQL, not SQL
- Reduced engineering effort necessary for maintenance
- Consistent query patterns allow for templated query generation



The Query Template

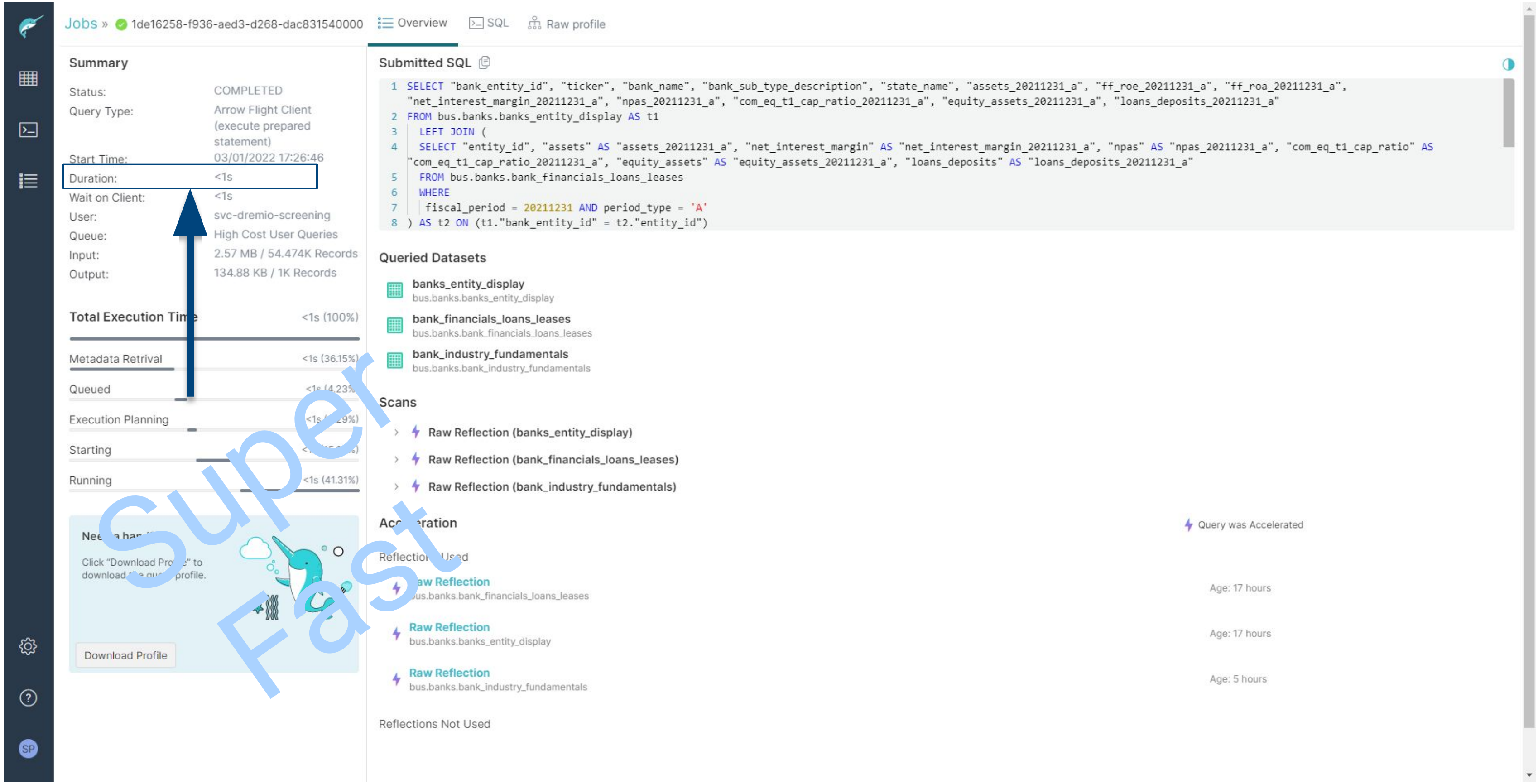
```
SELECT <display_columns>
FROM <display_table> AS t1
LEFT JOIN (
  SELECT <dimension_1_display_columns>
  FROM <dimension_1_display_table>
  WHERE <dimension_1_display_filters>) AS t2 ON <dimension_1_join_fields>
LEFT JOIN ( … )AS t2 ON <dimension_n_join_fields>
WHERE <display_key> IN (
  SELECT
    DISTINCT <display_key>
  FROM <filter_table> AS t1
  WHERE
    <filter_criteria> AND
    <display_key> IN (
        SELECT DISTINCT <display_key> FROM <dimension_table_1> AS t1 WHERE <dimension_1_filter_criteria>) AND
    <display_key> IN ( … ))
  ORDER BY LOWER(<display_key>) ASC
  LIMIT 1000
```

The Query

```
SELECT "bank_entity_id", "ticker", "bank_name", "bank_sub_type_description“,…
FROM bus.banks.banks_entity_display AS t1
LEFT JOIN (…) AS t2 ON (t1."bank_entity_id" = t2."entity_id")
LEFT JOIN (…) AS t3 ON (t1."bank_entity_id" = t3."entity_id")
WHERE t1."bank_entity_id" IN (
  SELECT
    DISTINCT t1."bank_entity_id"
  FROM bus.banks.banks_entity_display AS t1
  WHERE
    ((t1."bank_sub_type" = '0|1' OR t1."bank_type" = '0' OR t1."bank_type" = '1') AND t1."bank_entity_id" IN (
    SELECT DISTINCT "entity_id" FROM bus.banks.bank_financials_loans_leases AS t1 WHERE
      (t1."venture_capital_revenue" >= 10 AND period_type = 'A' AND fiscal_period = 20211231)
    )
   AND t1."status_code" IN ('Active','Extinct'))) ORDER BY LOWER("bank_entity_id") ASC LIMIT 1000
```

FACTSET

Dynamic Schemas -> Deterministic Queries



Jobs » 1de16258-f936-aed3-d268-dac831540000

Overview | SQL | Raw profile

Summary

Status:	COMPLETED
Query Type:	Arrow Flight Client (execute prepared statement)
Start Time:	03/01/2022 17:26:46
Duration:	<1s
Wait on Client:	<1s
User:	svc-dremio-screening
Queue:	High Cost User Queries
Input:	2.57 MB / 54.474K Records
Output:	134.88 KB / 1K Records

Total Execution Time	<1s (100%)
Metadata Retrival	<1s (36.15%)
Queued	<1s (4.23%)
Execution Planning	<1s ([illegible]29%)
Starting	<[illegible]
Running	<1s (41.31%)

Need a hand?

Click “Download Profile” to download the query profile.

Download Profile

Submitted SQL

```
SELECT "bank_entity_id", "ticker", "bank_name", "bank_sub_type_description", "state_name", "assets_20211231_a", "ff_roe_20211231_a", "ff_roa_20211231_a",
"net_interest_margin_20211231_a", "npas_20211231_a", "com_eq_t1_cap_ratio_20211231_a", "equity_assets_20211231_a", "loans_deposits_20211231_a"
FROM bus.banks.banks_entity_display AS t1
  LEFT JOIN (
  SELECT "entity_id", "assets" AS "assets_20211231_a", "net_interest_margin" AS "net_interest_margin_20211231_a", "npas" AS "npas_20211231_a", "com_eq_t1_cap_ratio" AS
"com_eq_t1_cap_ratio_20211231_a", "equity_assets" AS "equity_assets_20211231_a", "loans_deposits" AS "loans_deposits_20211231_a"
  FROM bus.banks.bank_financials_loans_leases
  WHERE
   fiscal_period = 20211231 AND period_type = 'A'
) AS t2 ON (t1."bank_entity_id" = t2."entity_id")
```

Queried Datasets

- banks_entity_display
  bus.banks.banks_entity_display
- bank_financials_loans_leases
  bus.banks.bank_financials_loans_leases
- bank_industry_fundamentals
  bus.banks.bank_industry_fundamentals

Scans

- Raw Reflection (banks_entity_display)
- Raw Reflection (bank_financials_loans_leases)
- Raw Reflection (bank_industry_fundamentals)

Acceleration

Query was Accelerated

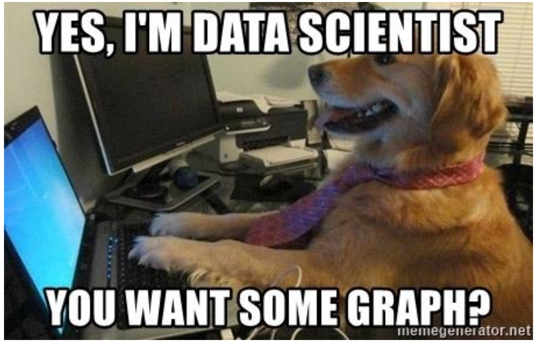
Reflections Used

Raw Reflection bus.banks.bank_financials_loans_leases	Age: 17 hours
Raw Reflection bus.banks.banks_entity_display	Age: 17 hours
Raw Reflection bus.banks.bank_industry_fundamentals	Age: 5 hours

Reflections Not Used

Public

Future Optimizations and Tooling



- Improve self-service tooling for adding new metadata
  - Web Platform for internal content teams to use
- More automated metadata generation from datasets
- Higher level GraphQL query planning for utilizing column-caches when available for super-low-latency queries
  - Provide query handling for subset of patterns which can't be served with Dremio SQL
- Computational capabilities in addition to fetches worked into dynamic schema and query generation

Questions?