Building a Domain Specific Query Language on top of Clickhouse

Clickhouse Spring Meetup, April 2023

Make Things. Better.

ODEN TECHNOLOGIES

Agenda

Introduction

Part 1.

- Why are we doing this?
- What does it mean to design a language?

Part 2.

- Oden's data model
- How it's used now

Part 3.

- The Implementation

Hi, I'm Russ!

- Sr TLM, Oden Technologies (Manager + IC)
- Manage the Integrations Team and the Core Data Team
- As an IC, primarily focused on query execution
- Before Oden, spent five years at Uber
- Before that, CS + math @ Rutgers University

Oden Technologies: Manufacturing Analytics

- Help manufacturers waste less material, improve quality, etc
- Mostly timeseries data from sensors combined with contextual information about some process
- SaaS product
- Tech stack: React, Golang, Apache Beam / Dataflow, Postgres, Clickhouse

Now | Oden

platform.oden.app/assets/e32d004e-ce8b-40bf-bbc1-14fd907d2f47/list?timeRange=last7day

Now | Dashboards | Explore | Discover | Labs | Data Status

Factory A

Last 7 days | All lines | Print | List

Line	LINE AGGREGATE State History	Util	Perf	Quality	OEE	Production	Scrap (Ft)	Yield (Ft)	CURRENT RUN ON LINE Work Order	Current Speed	Current Production	Current
Analytics Line	Uptime 1,060d 10h	100%	96%	84%	62%	1,404,514 ft	40,588	1,177,500	VEG-56420000 7h 44m Product EDE86783 Batch 188EDBED	170 164 +4 / -2	75,339 ft	
Compounding 1	Uptime 2h 53m	80%	112%	79%	70%	943,605 lb	66,392	744,208	VED-71120000 3d 3h Product 6D50D9B2 Batch 9270A025	7,123 6,300 +700 / -300	418,351 lb	3
Compounding 2	Not configured	-	-	0%	0%	0	53,792	943,008	VEG-64810000 5h 24m Product C0BE503C Batch 4541B911		0	
Compounding (test)	Not configured	-	-	-	-	-	-	-			-	
Extruder 1	Uptime 37m 51s	81%	89%	0%	0%	2,897,116 ft	-	-	123 267d 4h Product 109	426 400 +100 / -300	12,578,746 ft	
Extruder 2	Uptime 244d 5h	100%	0%	0%	0%	0 ft	-	-	XYZ-123 938d 11h Product 85EB50E1 Batch 22CD2E30	- 56 +4 / -6	0 ft	
Extruder 3	Uptime 43m 58s	63%	80%	0%	0%	913,287 ft	-	-	VEG-81001000 54m 31s Product B0DBE4C3	228 225 ± 10	8,056 ft	
		97%	0%	0%	0%	0 ft	-	-	line_demo_in_PROD 335d 14h	350	0 ft	

OEE	Utilization	Performance	Quality	Production Output	Scrap	Yield
0.8%	91.2%	51.9%	1.8%	–	160,772	5,256,479

Multiple Units

Dashboard | Oden

platform.oden.app/dashboard/ece73f30-872b-405f-99ec-0ea7a699d57d

Now | Dashboards | Explore | Discover | Labs | Data Status

Availability Trends

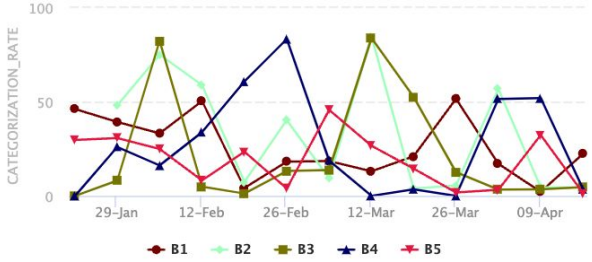
updated a few seconds ago

Availability Trends | Select Range | + OQL Module | + Module | Done

All Lines | All States | All Products | All Shifts | Operator Id

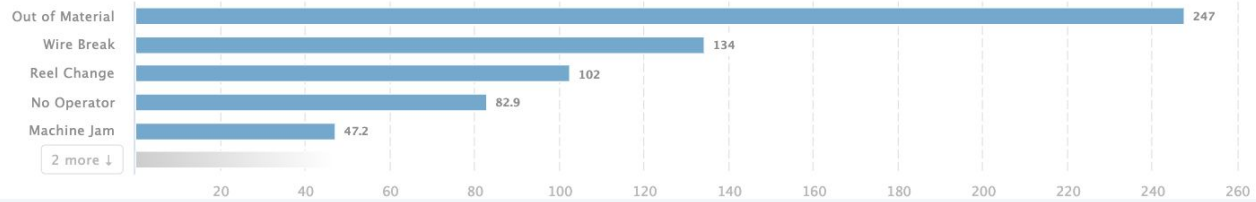
Weekly Downtime Categorization Rate

Weekly Downtime Categorization Rate by Line



Top Downtime Causes

Previous 4 weeks and current week



Dashboard | Oden

platform.oden.app/dashboard/ece73f30-872b-405f-99ec-0ea7a699d57d

Now | Dashboards | Explore | Discover | Labs | Data Status

Availabil

Availability Tre | All Lines | Done

Update Weekly Downtime Categorization Rate by Line

Information

Name

Weekly Downtime Categorization Rate by Line

Description

Module visualization

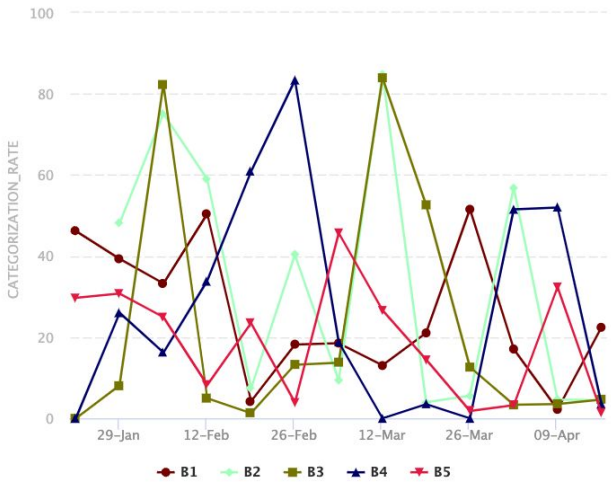
Line Trend

Configuration

Need help? Learn how to write OQL queries here

Query

```
aggregate timeInIf(interval.state.category.id = 2)/timeInIF(interval
    .state.category.id in (2,122,31,49,100,32))*100 as
    categorization_rate where time from now() - 3M to now() and
    factory.id = 'a6e8920a-71be-4ed3-afae-c74acc364b5e' group by line
    .name, bucket 1W
```



Cancel | Save

20 40 60 80 100 120 140 160 180 200 220 240 260

Part 1

Justifying OQL, Language Design

Google

site:newegg.com -pixel "android" $500...$1000

All | News | Images | Videos | Shopping

About 70,200 results (0.54 seconds)

https://www.newegg.com › d=android+one

android one - Newegg.com

Refurbished: WD 4TB My Cloud Home Duo Personal Cloud Storage (i
Compatible) - (WDBMUT0040JWT-NESN). $147. $49.99 Shipping.

https://www.newegg.com › d=android+12+tablet

android 12 tablet - Newegg.com

Samsung Galaxy Tab S7+ 12.4" 256GB **Android** Tablet w/ Snapdragon
Processor - Mystic Black. $1,165. Free Shipping. CompareAdd To Wisl

https://www.newegg.com › insider › why-android-phon...

Why Android Phones are Cheaper than Ever - Ne

EXPLORER | DIAGNOSTICS

CONFIGURATION | MQL | PROMQL

FORMAT | EXAMPLE QUERIES

Press Alt+F1 for Accessibility Options.

```
fetch dataflow_job
| metric 'dataflow.googleapis.com/job/elements_produced_count'
| filter (resource.job_name =~ '.*bifrost.*')
| align rate(1m)
| every 1m
| group_by [],
    [value_elements_produced_count_mean: mean(value.elements_produced_count)]
```

Assigned to current user, sprint is open ...

Save as | Details

assignee = currentuser() and (sprint in openSprints() or sprint in futureSprints()) and status != Released and status != Closed

1–8 of 8

T	Key	Summary
	RW-88	GCL to CLI: enable calculating percentiles for metrics

OQL Justifications

- **Densely** and completely represent what data we want for a particular use case
- Highly **expressive**; boolean and mathematical expressions
- **Domain knowledge** built into the language for ease of use: **inferred joins**, no worrying about different data sources
- Flexibility to **cache** or **optimize** in the backend
- **Parallelization** of work streams: backend engineers focus on fetching & aggregating data, frontend engineers focus on display
  - Corollary: immediate delivery of features
  - Faster iteration
- Provide a **powerful tool** to those with domain expertise

What does it mean to design a language?

- Formal grammar: describing languages based on production rules
- Each rule maps one string of symbols to another
- Originally applied to natural languages
- Now is used mostly in computer science to describe programming languages
- Concretely, designing a language results in a grammar file, which can be used by programs to parse strings of that language

```
Sentence → Subject “ ” Verb “ ” Object “.”
Subject  → (Article? Noun) / ProperNoun
Verb     → “eats” / “likes”
Article  → “a” / “the”
Object   → (Article? Noun) / ProperNoun
Noun     → “dog” / “cheese”

The dog likes cheese.
The cheese eats dog.
```

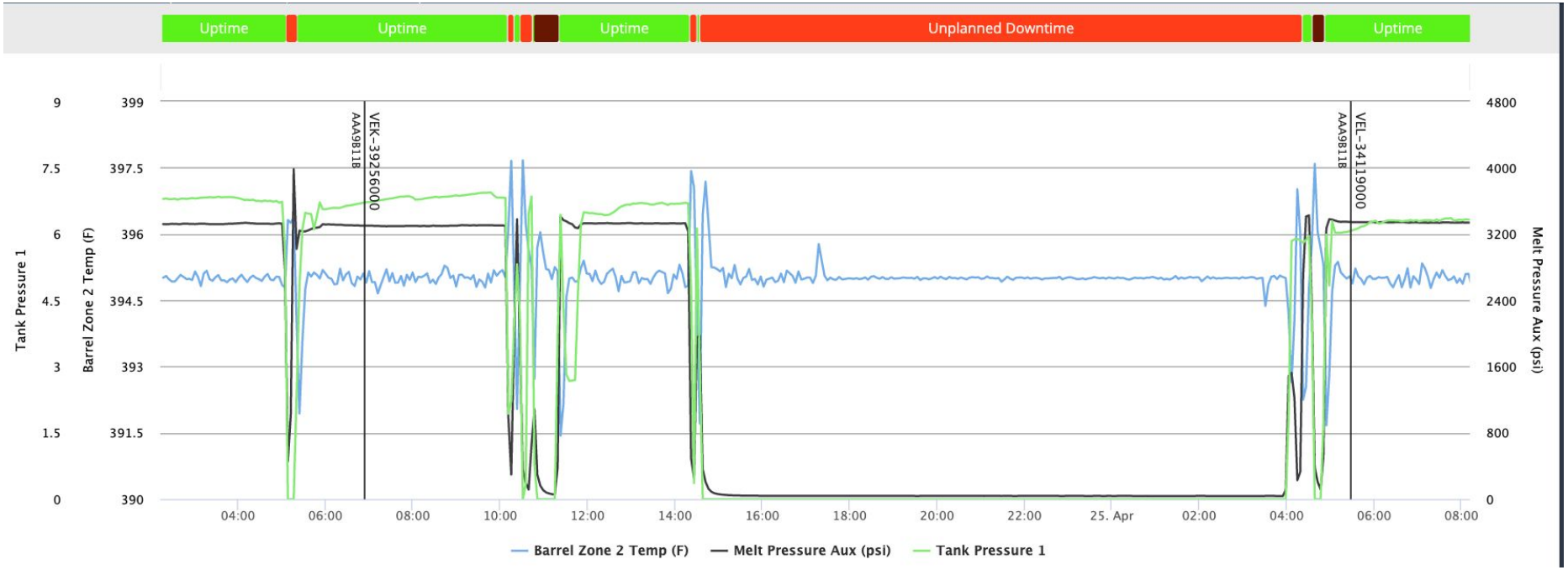
Simplified OQL-like Grammar

```
Query          → "aggregate" Spacing TargetList Spacing WhereClause
TargetList     → Expression (“,” Spacing Expression)*
Expression     → MetricID (Operator Spacing MetricID)*
Operator       → “+” / “-” / “*” / “/”
MetricID       → “metric(“ String “)”
WhereClause    → Key “=” Value
GroupBy        → “group by line”
```

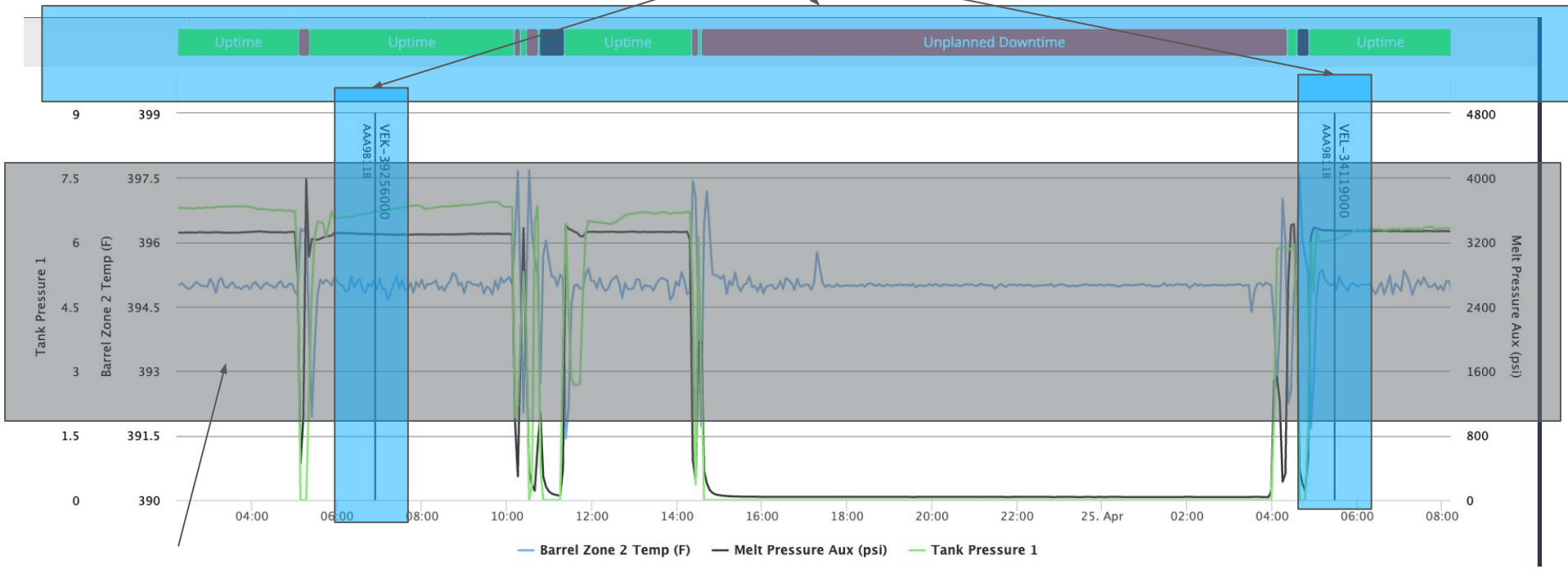
```
aggregate metric(“line_speed”) where state=”uptime” group by line

aggregate metric(“line_speed”) / metric(“output”), metric(“line_speed”) where
product=”blue 14awg” group by line
```

Part 2

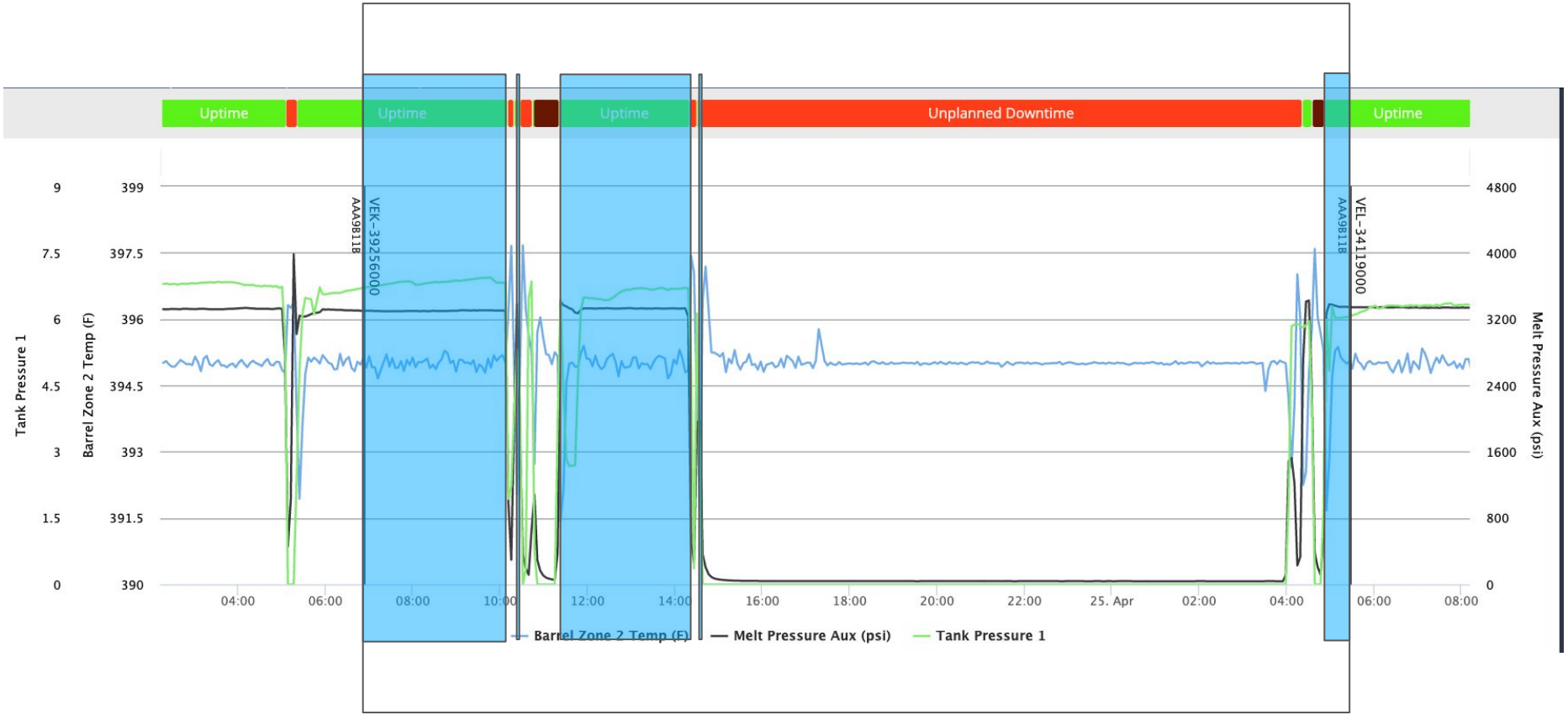
Oden's data,
How to use OQL


Uptime
Uptime
Uptime
Unplanned Downtime
Uptime
AAA98118
VEK-39256000
AAA98118
VEL-34119000
Tank Pressure 1
Barrel Zone 2 Temp (F)
Melt Pressure Aux (psi)
9
7.5
6
4.5
3
1.5
0
399
397.5
396
394.5
393
391.5
390
4800
4000
3200
2400
1600
800
0
04:00
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
25. Apr
02:00
04:00
06:00
08:00
Barrel Zone 2 Temp (F)
Melt Pressure Aux (psi)
Tank Pressure 1

Intervals



Metrics


Uptime
Uptime
Uptime
Unplanned Downtime
Uptime
AAA98118
VEK-39256000
AAA98118
VEL-34119000
Tank Pressure 1
Barrel Zone 2 Temp (F)
Melt Pressure Aux (psi)
9
7.5
6
4.5
3
1.5
0
399
397.5
396
394.5
393
391.5
390
4800
4000
3200
2400
1600
800
0
04:00
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00
25. Apr
02:00
04:00
06:00
08:00
Barrel Zone 2 Temp (F)
Melt Pressure Aux (psi)
Tank Pressure 1

What is Oden Query Language?

- OQL is an analytics language for describing queries about manufacturing data
- SQL-like: general syntax is inspired by SQL, some functions are the same as they are in SQL
- “aggregate” queries are the main use case since manufacturing data needs to be aggregated in order to be useful (i.e. line speed while in uptime)

OQL Basics

Measures: the actual aggregations we want.
Comma separate expressions.

```
aggregate ...
where time from ... to ...
and ...
group by ...
```

Time range of query

Additional filtering, e.g. on state = uptime

Dimensions: what we are grouping by; can be an entity or a time bucket, or omitted to return a single row

- The grouping keys are always included in the response; don't need to include them in the measures we actually want
- Expressions are supported over the measures: + - * / are all supported as well as () for grouping

Simple measures

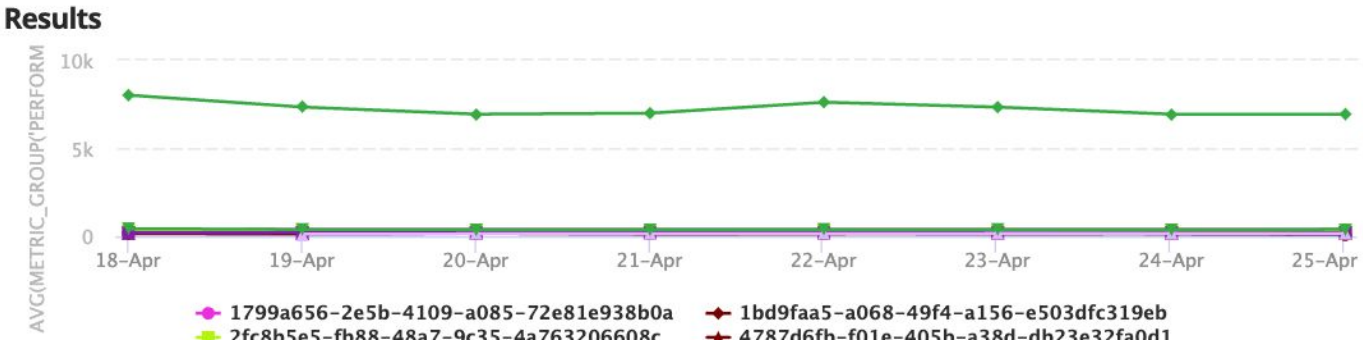
- High level KPIs:
  - `aggregation.output`
  - `aggregation.scrap`
  - `aggregation.yield`
  - `aggregation.oee`
  - `aggregation.availability`
  - `aggregation.quality`
  - `aggregation.performance`
  - `aggregation.utilization`
- Or, you can specify metrics directly & metric groups:
  - `sum(metric("uuid"))`
  - `count(metric_group("performance"))`
  - `deltaSum(metric_group("uuid"))`

```
aggregate aggregation.output
where time from now() - 48h to now()
and factory.id =
'a6e8920a-71be-4ed3-afae-c74acc364b5e'
group by line.id, bucket 1h
```

Line.Id	Bucket	Aggregation.Output
B3	Sun, 2AM	5842.91
B3	Sun, 1AM	11099.99
B3	Sun, 12AM	11099.99
B3	Sat, 11PM	11099.99
B3	Sat, 10PM	11099.99
B3	Sat, 9PM	1903.81
B3	Sat, 8PM	0
B2	Mon, 8PM	4743.75
B2	Mon, 7PM	13499.99
B2	Mon, 6PM	13499.99
B2	Mon, 5PM	13499.99

Aggregating a metric only when in uptime

```
aggregate avg(metric_group('performance'))
where time from now() - 1W to now()
and interval.state.category.id = 1
group by line.id, bucket 1D
```



Advanced Measures

- `countIf` can count the number of intervals matching a certain condition within each group
- `timeInIf` can sum the time spent in intervals matching a certain condition within each group
- How much time did we spend in uptime each day?
- How many downtime intervals happened each day?

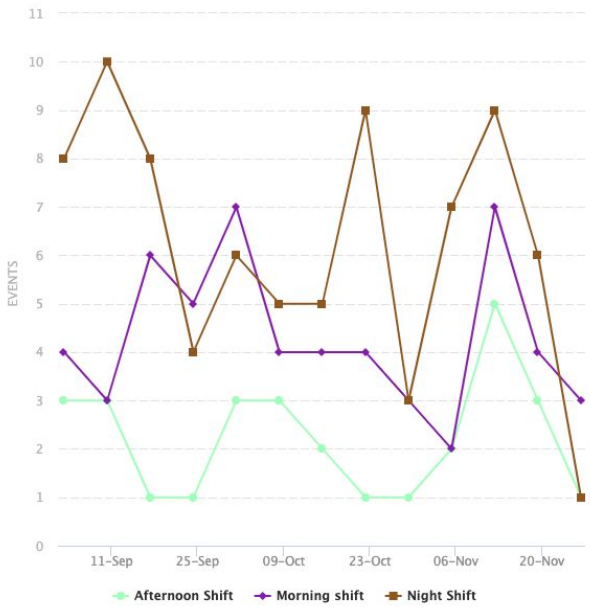
```
aggregate timeInIf(interval.state.category.name = 'Uptime')
where time from now() - 48h to now()
and factory.id = 'a6e8920a-71be-4ed3-afae-c74acc364b5e'
group by line.id, bucket 1D
```

Line.Id	Bucket	TimeInIf(Interval.State.Category.Name = "Uptime")
B5	Mon, 12AM	0
B5	Sun, 12AM	0
B5	Sat, 12AM	5351.34
B4	Mon, 12AM	0
B4	Sun, 12AM	32010
B4	Sat, 12AM	8421.34
B3	Mon, 12AM	0
B3	Sun, 12AM	9110
B3	Sat, 12AM	8000
B2	Mon, 12AM	77978.66
B2	Sun, 12AM	86400
B2	Sat, 12AM	8421.34
B1	Mon, 12AM	77978.66
B1	Sun, 12AM	74280
B1	Sat, 12AM	0

Other grouping types

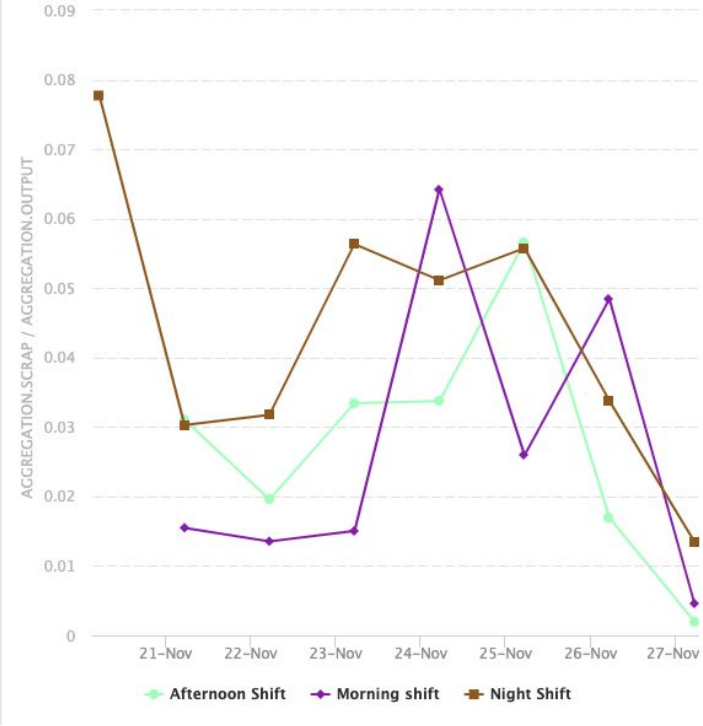
- bucket …
  - E.g. 1m 1h 1D 1M 1W
- line.id
- factory.id
- product.name
- shift.name
- interval.state.category
- interval.state.reason
- interval.<custom type>.value

```
aggregate countIf(interval.state.reason.name='Reel Change') as events
where time from now() - 3M to now()
and factory.id = 'a6e8920a-71be-4ed3-afae-c74acc364b5e'
group by shift.name, bucket 1W
```



Expressions over aggregates

```
aggregate aggregation.scrap / aggregation.output
where time from now() - 1W to now()
and factory.id="a6e8920a-71be-4ed3-afae-c74acc364b5e"
group by shift.name, bucket 1D
```



Part 3

How it’s done

High level: Three Phases

1. Parse the query into a concrete syntax tree
2. Translate syntax tree into an Intermediate Representation
3. Execute the IR in a backend; the Clickhouse backend will emit SQL
    a. Creates a Query Plan Tree
    b. Each node in the tree emits its own SQL

```go
// AST is an intermediate representation of a query
type AST struct {
    Type        QueryType     `json:"query_type"`
    Aggregates  []Aggregate   `json:"aggregates,omitempty"`
    Metrics     []Metric      `json:"metrics,omitempty"`
    TimeRange   TimeRange     `json:"time_range"`
    Filters     []Filter      `json:"filters,omitempty"`
    GroupBy     []GroupBy     `json:"group_bys,omitempty"`
    Expressions []*Expression `json:"expressions,omitempty"`

    // Query metadata
    UserID    uuid.UUID      `json:"user_id,omitempty"`     /
    UserOrgID uuid.UUID      `json:"user_org_id,omitempty"` /
    OrgID     uuid.UUID      `json:"org_id,omitempty"`      /
    Timezone  *time.Location `json:"timezone,omitempty"`    /
}
```

Why have an Intermediate Representation?

- Different backends
- Ease of testing
- Make changes to grammar without modifying the backend
- Make changes to the backend without modifying the grammar

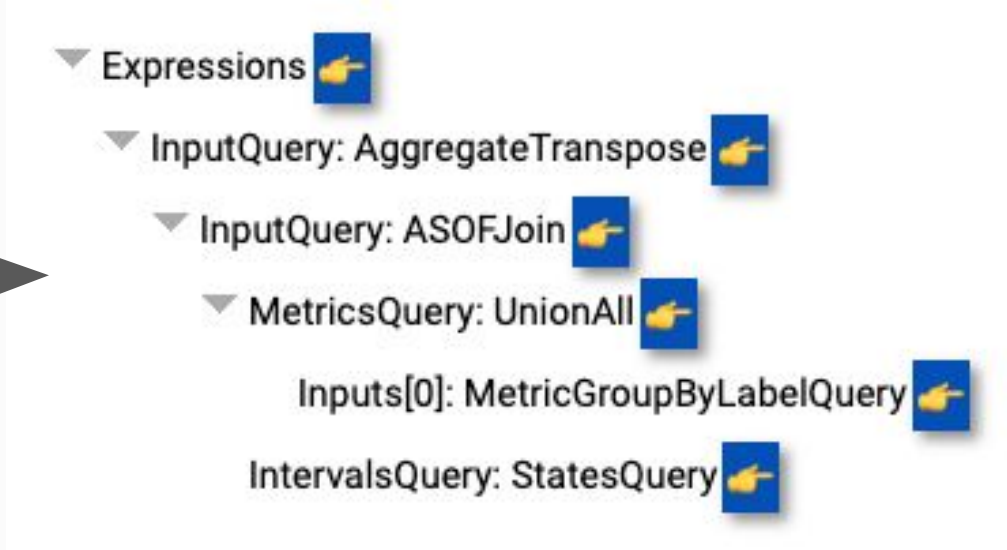
```
AGGREGATE
   avg(metric_group('performance'))
WHERE TIME FROM now() - 1W TO now()
AND interval.state.category.id = 1
GROUP BY line.id, bucket 1D
```

```
Grammar "AGGREGATE avg(metric_group('performance')) WHERE TIME FROM now() - 1
 Query "AGGREGATE avg(metric_group('performance')) WHERE TIME FROM now() - 1W
  AggregateQuery "AGGREGATE avg(metric_group('performance')) WHERE TIME FROM
   Spacing " "
    Space " "
   TargetList "avg(metric_group('performance'))"
    ANDExpression "avg(metric_group('performance'))"
     ORExpression "avg(metric_group('performance'))"
      LogicalAtom "avg(metric_group('performance'))"
       AggregateExpr "avg(metric_group('performance'))"
        AggregateTerm "avg(metric_group('performance'))"
         AggregateFactor "avg(metric_group('performance'))"
          AggregateOperand "avg(metric_group('performance'))"
           MetricFuncCall "avg(metric_group('performance'))"
            FuncName "avg"
            MetricRef "metric_group('performance')"
             MetricGroupTerm "metric_group('performance')"
              StringLiteral "'performance'"
               SingleQuoteString "'performance'"
   Spacing " "
    Space " "
   WhereClause "WHERE TIME FROM now() - 1W TO now() AND interval.state.catego
    Spacing " "
     Space " "
    ClauseBody "TIME FROM now() - 1W TO now() "
     TimeInClause "TIME FROM now() - 1W TO now() "
      TimeFromToClause "TIME FROM now() - 1W TO now() "
```

```
Grammar "AGGREGATE avg(metric_group('performance')) WHERE TIME FROM now() - 1W TO now() AND in
 Query "AGGREGATE avg(metric_group('performance')) WHERE TIME FROM now() - 1W TO now() AND int
  AggregateQuery "AGGREGATE avg(metric_group('performance')) WHERE TIME FROM now() - 1W TO now
   Spacing " "
    Space " "
   TargetList "avg(metric_group('performance'))"
    ANDExpression "avg(metric_group('performance'))"
     ORExpression "avg(metric_group('performance'))"
      LogicalAtom "avg(metric_group('performance'))"
       AggregateExpr "avg(metric_group('performance'))"
        AggregateTerm "avg(metric_group('performance'))"
         AggregateFactor "avg(metric_group('performance'))"
          AggregateOperand "avg(metric_group('performance'))"
           MetricFuncCall "avg(metric_group('performance'))"
            FuncName "avg"
            MetricRef "metric_group('performance')"
             MetricGroupTerm "metric_group('performance')"
              StringLiteral "'performance'"
               SingleQuoteString "'performance'"
   Spacing " "
    Space " "
   WhereClause "WHERE TIME FROM now() - 1W TO now() AND interval.state.category.id = 1"
    Spacing " "
     Space " "
    ClauseBody "TIME FROM now() - 1W TO now() "
     TimeInClause "TIME FROM now() - 1W TO now() "
      TimeFromToClause "TIME FROM now() - 1W TO now() "
```


```yaml
filters:
- comparator: '='
  property: interval.state.category.id
  type: interval
  values:
  - 1
group_bys:
- property: line.id
  type: line
- amount: 1
  type: bucket
  unit: D
lines:
- 6dc2fc41-b49f-4f26-a443-2f62c2cdb355
- b56b9924-d0ba-4e7c-b8b5-a5b6ec87e9c8
...
location: America/New_York
metrics:
- function: avg
  metric_group_label: performance
  type: metric_group_label
org_id: b949c5eb-449e-4e6b-8512-b1322f5b4c4
query_type: aggregate
settings:
  backend: v1
target_list:
- expressions:
  - terms:
    - factors:
      - metric_value:
          Index: 0
```

```yaml
filters
- comparator: '='
  property: interval.state.category.id
  type: interval
  values:
  - 1
group_bys:
- property: line.id
  type: line
- amount: 1
  type: bucket
  unit: D
lines:
- 6dc2fc41-b49f-4f26-a443-2f62c2cdb355
- b56b9924-d0ba-4e7c-b8b5-a5b6ec87e9c8
...
location: America/New_York
metrics:
- function: avg
  metric_group_label: performance
  type: metric_group_label
org_id: b949c5eb-449e-4e6b-8512-b1322f5b4c4a
query_type: aggregate
settings:
  backend: v1
target_list:
- expressions:
  - terms:
    - factors:
      - metric_value:
          Index: 0
```



```sql
-- name: ASOFJoin :sql-only
-- templater: text/template
-- interface: IntervalsQuery { Query() (string, error) }
-- interface: MetricsQuery { Query() (string, error) }
-- in: Metrics      MetricsQuery
-- in: Intervals    IntervalsQuery
SELECT * FROM (
    {{ .Metrics.Query }}
) metrics
ASOF JOIN (
    {{ .Intervals.Query }}
) intervals
ON metrics.timestamp ≥ intervals.start_timestamp AND metrics.line_id = intervals.line_id
ORDER BY timestamp ASC
```

Example OQL and SQL

```
AGGREGATE
   avg(metric_group('performance'))
WHERE TIME FROM now() - 1W TO now()
AND interval.state.category.id = 1
GROUP BY line.id, bucket 1D
```

```
SELECT
    line_id AS `line.id`,
    timestamp AS timestamp,
    avg_performance_highest AS `avg(metric_group('performance'))`
FROM
(
    SELECT
        toStartOfInterval(timestamp, toIntervalDay(1), 'America/New_York') AS timestamp,
        avgWeightedIf(value, weight, identifier = 'performance_highest') AS avg_performance_hi
        line_id
    FROM
    (
        SELECT *
        FROM
        (
            WITH defs AS
                (
                    SELECT
                        line_id,
                        argMin(metric_id, label_rank) AS metric_id,
                        argMin(display_name, label_rank) AS display_name
                    FROM metric_taxonomy
                    WHERE (line_id IN ('6dc2fc41-b49f-4f26-a443-2f62c2cdb355', 'b56b9924-d0bc
'4787d6fb-f01e-405b-a38d-db23e32fa0d1', '1bd9faa5-a068-49f4-a156-e503dfc319eb', '7c89b7d6-e9f
'b10378d2-aafb-4c5b-9906-e367a0284d4d', '3505d217-419a-4ee7-950d-6bb70089a961', '1799a656-2e5
'703c6a90-69a1-401b-8c37-f21ea7673c6b', 'ff2fec36-72bc-43a1-9ca7-3cc566348abd', '2fc8b5e5-fb8
'e5c9570e-22f5-48a9-9e48-66cdc0423ec5', '0a74ebb6-93ad-426f-b907-0b94d77405e7')) AND (label_r
                    GROUP BY line_id
                )
            SELECT
                m.timestamp,
                timestamp + toIntervalSecond(1) AS next_timestamp,
                1 AS weight,
                defs.line_id,
                m.metric_id,
                defs.display_name,
                value,
                'performance_highest' AS identifier
```

Example OQL and SQL

```
AGGREGATE
   avg(metric_group('performance'))
WHERE TIME FROM now() - 1W TO now()
AND interval.state.category.id = 1
GROUP BY line.id, bucket 1D
```

```
            FROM
            (
                SELECT
                    timestamp,
                    metric_id,
                    value
                FROM metrics
                FINAL
                PREWHERE (metric_id IN (
                    SELECT metric_id
                    FROM defs
                )) AND (date >= (toDate('2023-04-13') - toIntervalDay(1))) AND (date <= (toDa
toDateTime('2023-04-13 20:01:36', '')) AND (timestamp <= toDateTime('2023-04-20 20:01:36', ''
                SETTINGS do_not_merge_across_partitions_select_final = 1, max_final_threads = 1
            ) AS m
            ANY LEFT JOIN defs ON m.metric_id = defs.metric_id
        ) AS metrics
        ASOF INNER JOIN
        (
            SELECT
                line_id,
                timestamp AS start_timestamp,
                formatRow('JSONEachRow', category_id, reason_id, original_timestamp, id, type
            FROM
            (
                SELECT
                    *,
                    concat(toString(original_timestamp), toString(line_id)) AS id,
                    'state' AS type
                FROM interval_state
                WHERE (line_id IN ('6dc2fc41-b49f-4f26-a443-2f62c2cdb355', 'b56b9924-d0ba-4e7
'4787d6fb-f01e-405b-a38d-db23e32fa0d1', '1bd9faa5-a068-49f4-a156-e503dfc319eb', '7c89b7d6-e9f
'b10378d2-aafb-4c5b-9906-e367a0284d4d', '3505d217-419a-4ee7-950d-6bb70089a961', '1799a656-2e5
'703c6a90-69a1-401b-8c37-f21ea7673c6b', 'ff2fec36-72bc-43a1-9ca7-3cc566348abd', '2fc8b5e5-fb8
'e5c9570e-22f5-48a9-9e48-66cdc0423ec5', '0a74ebb6-93ad-426f-b907-0b94d77405e7')) AND ((timest
            )
            UNION ALL
            SELECT
```

Example OQL and SQL

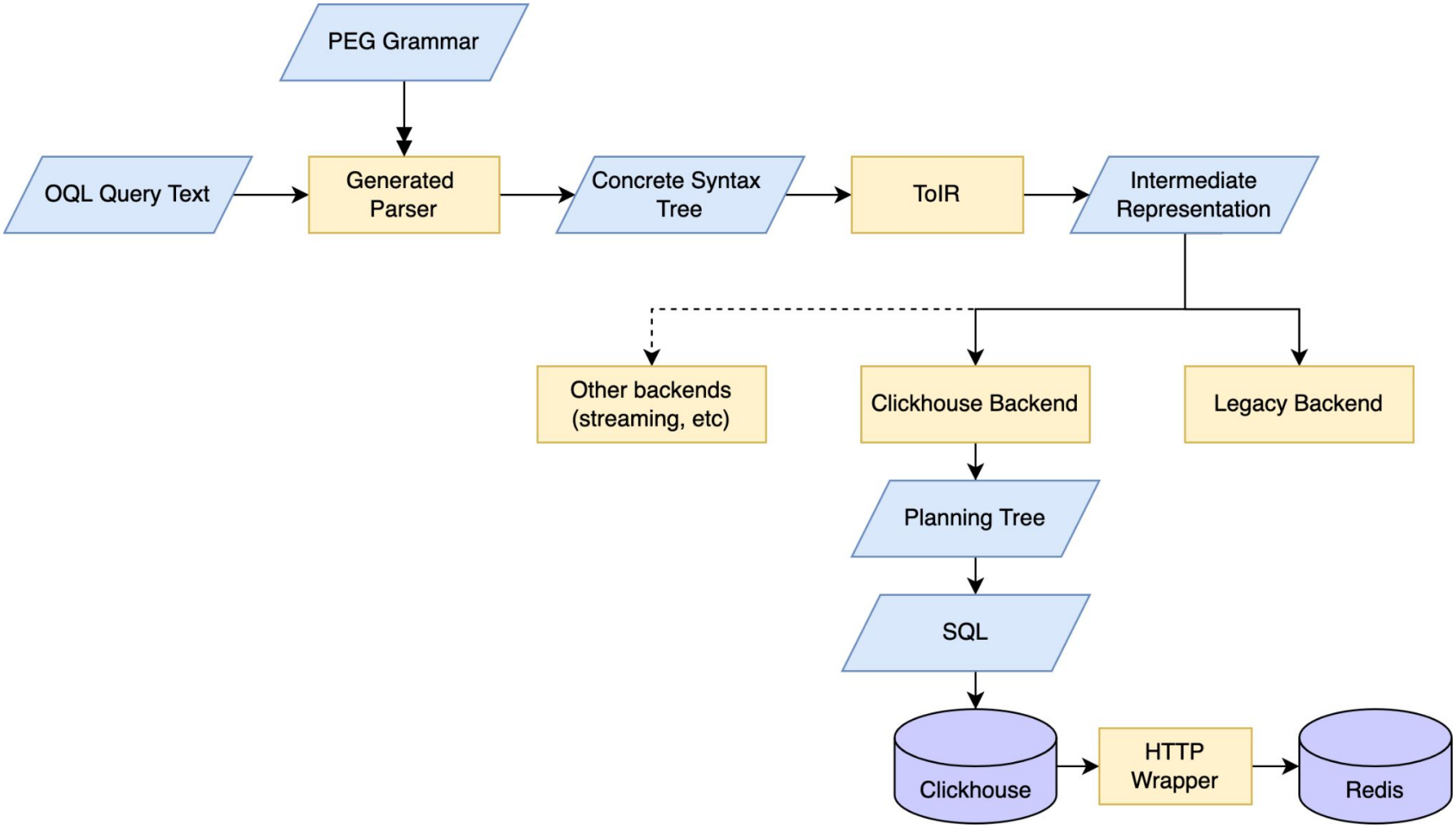
```
AGGREGATE
   avg(metric_group('performance'))
WHERE TIME FROM now() - 1W TO now()
AND interval.state.category.id = 1
GROUP BY line.id, bucket 1D
```

```
            SELECT
                line_id,
                max(timestamp) AS start_timestamp,
                argMax(formatRow('JSONEachRow', category_id, reason_id, original_timestamp, i
            FROM
            (
                SELECT
                    *,
                    concat(toString(original_timestamp), toString(line_id)) AS id,
                    'state' AS type
                FROM interval_state
                WHERE (line_id IN ('6dc2fc41-b49f-4f26-a443-2f62c2cdb355', 'b56b9924-d0ba-4e7
'4787d6fb-f01e-405b-a38d-db23e32fa0d1', '1bd9faa5-a068-49f4-a156-e503dfc319eb', '7c89b7d6-e9f
'b10378d2-aafb-4c5b-9906-e367a0284d4d', '3505d217-419a-4ee7-950d-6bb70089a961', '1799a656-2e5
'703c6a90-69a1-401b-8c37-f21ea7673c6b', 'ff2fec36-72bc-43a1-9ca7-3cc566348abd', '2fc8b5e5-fb8
'e5c9570e-22f5-48a9-9e48-66cdc0423ec5', '0a74ebb6-93ad-426f-b907-0b94d77405e7')) AND (timesta
            )
            GROUP BY line_id
        ) AS intervals ON (metrics.timestamp >= intervals.start_timestamp) AND (metrics.line_
        ORDER BY timestamp ASC
    )
    WHERE JSONExtractString(state_metadata, 'category_id') = '1'
    GROUP BY
        timestamp,
        line_id
    ORDER BY timestamp ASC
    SETTINGS enable_optimize_predicate_expression = 0
)
```

Conclusion

- Is a DSL the right approach for you?
- Design the language around the domain
- Use an intermediate / independent representation for flexibility of backend
- Create a query plan to manage the complexity of generating SQL

Thank you,
that’s all!
Questions?


PEG Grammar
OQL Query Text
Generated Parser
Concrete Syntax Tree
ToIR
Intermediate Representation
Other backends (streaming, etc)
Clickhouse Backend
Legacy Backend
Planning Tree
SQL
Clickhouse
HTTP Wrapper
Redis