GridGain

Apache Ignite SQL Troubleshooting

Vladimir Pligin, GridGain CS Team

GridGain

Apache Ignite SQL Troubleshooting

Feb, 2021

Vladimir Pligin

- Work at GridGain System (CS Team);
- Apache Ignite contributor

SQL Troubleshooting: Topics

- Ignite SQL Overview (for the current H2-based engine)
- Diagnostic tools
- Analysis examples for several cases
- SQL typical traps / optimization checklist

Ignite SQL Overview

Ignite SQL: Learn more

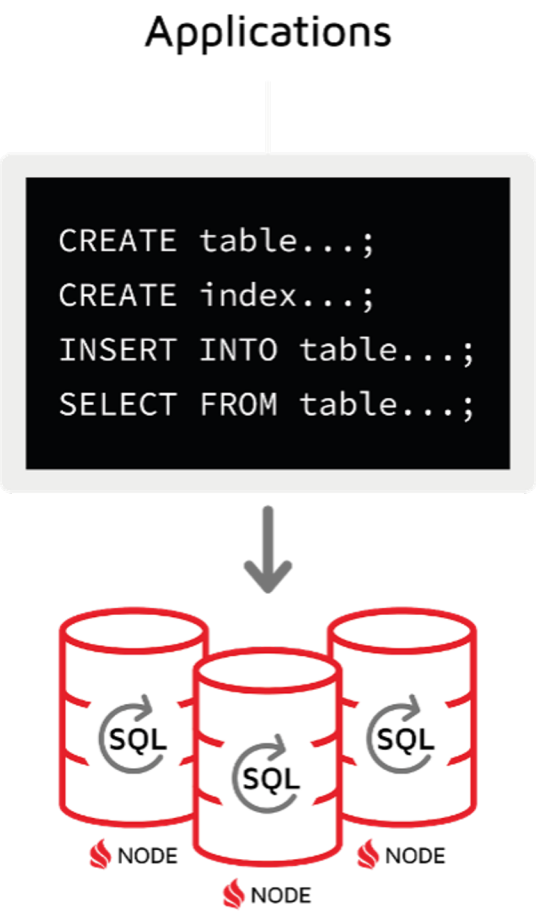
- How to start using Ignite SQL
  https://youtu.be/fwMRFA7BWTk

- Apache Ignite Troubleshooting
  https://youtu.be/QUH7vJXPXG0

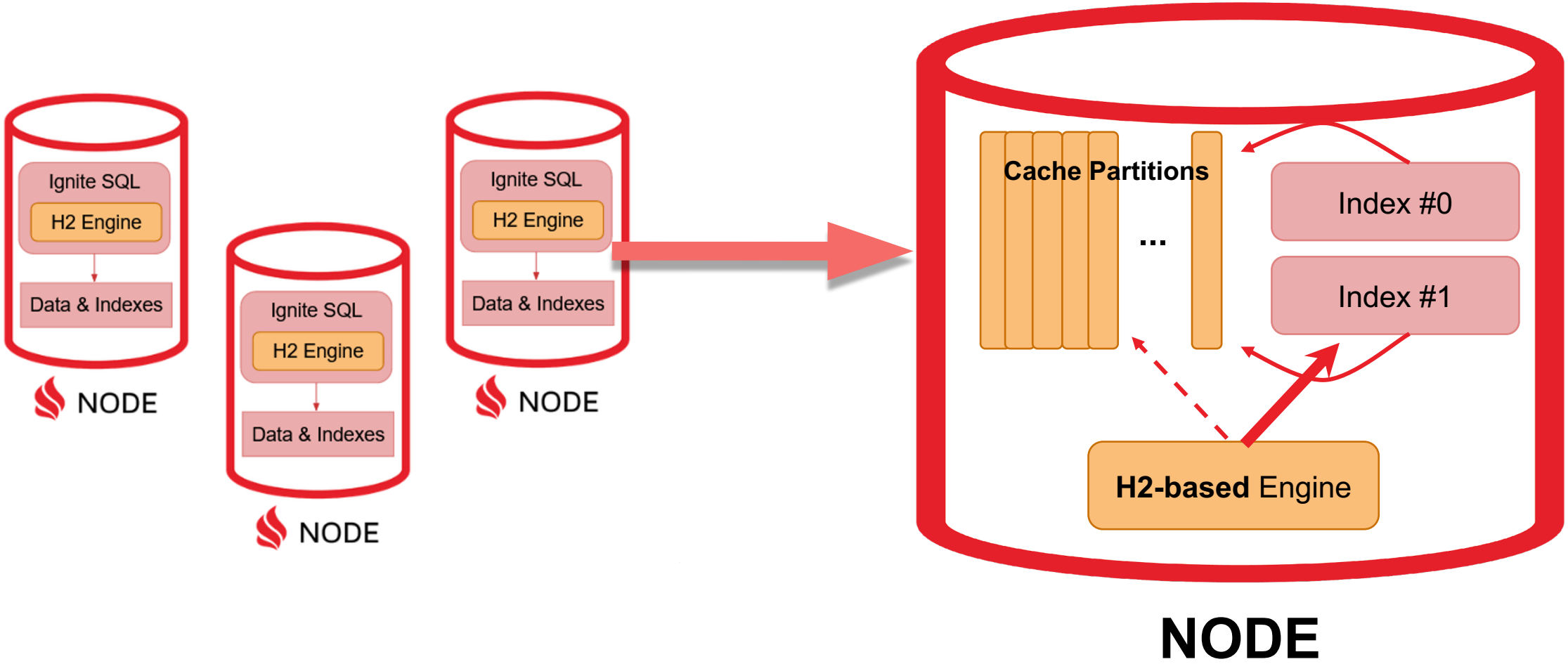
- Getting Started with Apache Ignite SQL
  https://www.youtube.com/watch?v=eYV-tNLzIts

Ignite SQL Overview

- ANSI-99 DML and DDL syntax
  - SELECT, UPDATE, CREATE…
- Distributed joins, grouping, sorting
- Schema changes in runtime
  - ALTER TABLE, CREATE/DROP INDEX
- Works with in-memory and disk-only records



Ignite SQL Engine Internals



Query Execution Phases

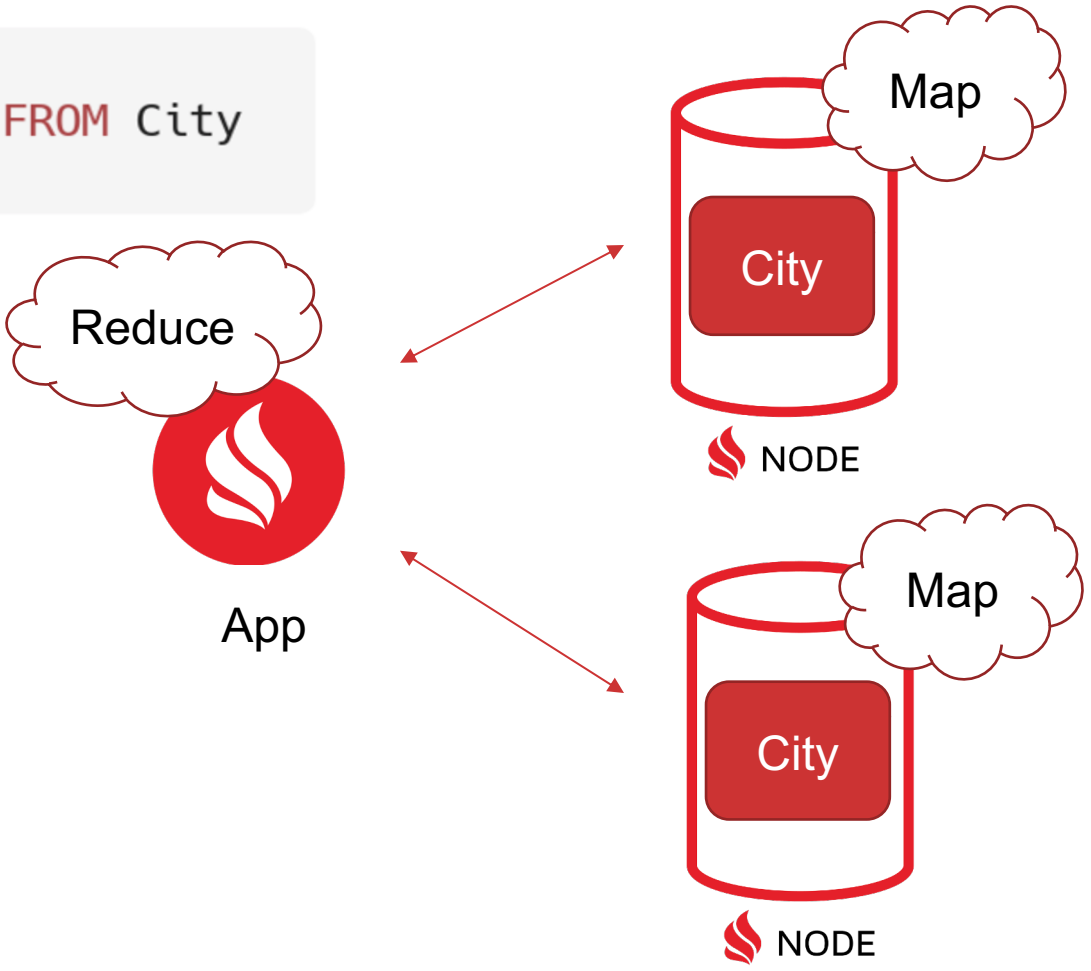
```
SELECT AVG(population) FROM City
```

Map

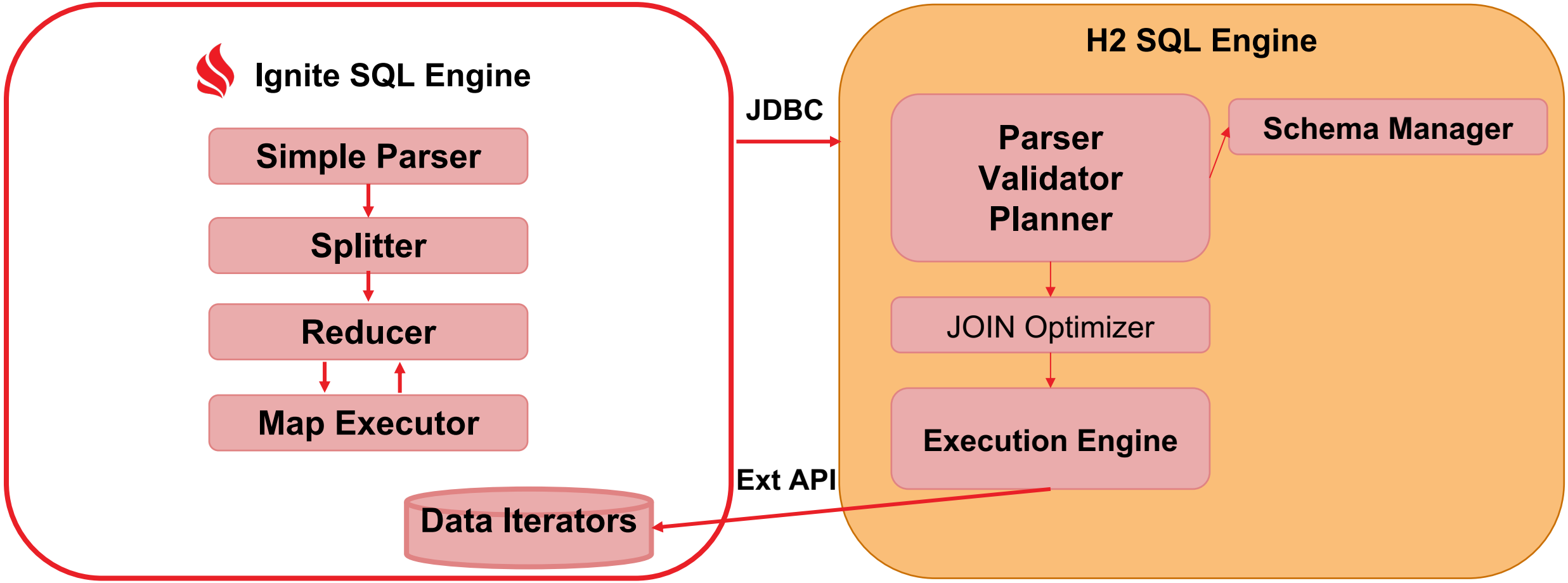
```
SELECT SUM(population) as sum0,
COUNT(population) as count0
FROM City
```

Reduce

```
SELECT SUM(sum0)/SUM(count0)
FROM resultTable
```

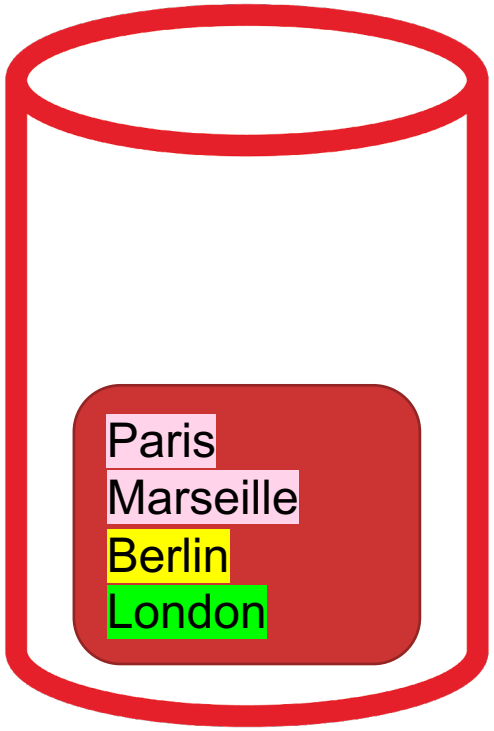


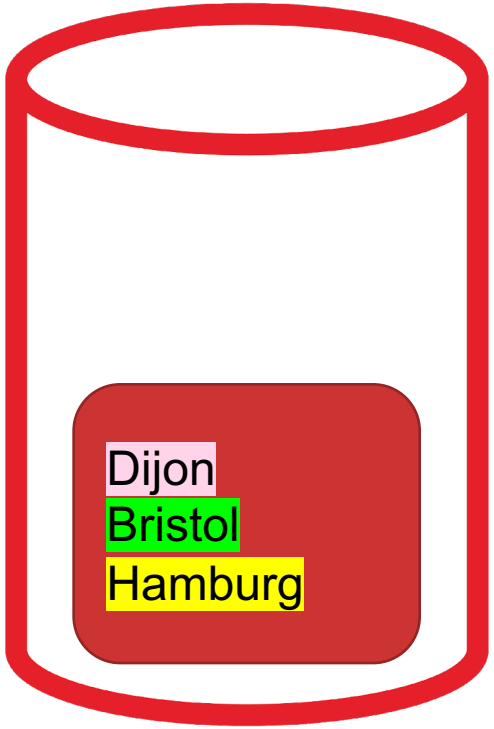
SQL Engine Components



Data distribution: default (PK)

```
CREATE TABLE CITY (
    id INT,
    countryId INT,
    name VARCHAR,
    population INT,
    PRIMARY KEY
    (id, countryId ))");
```



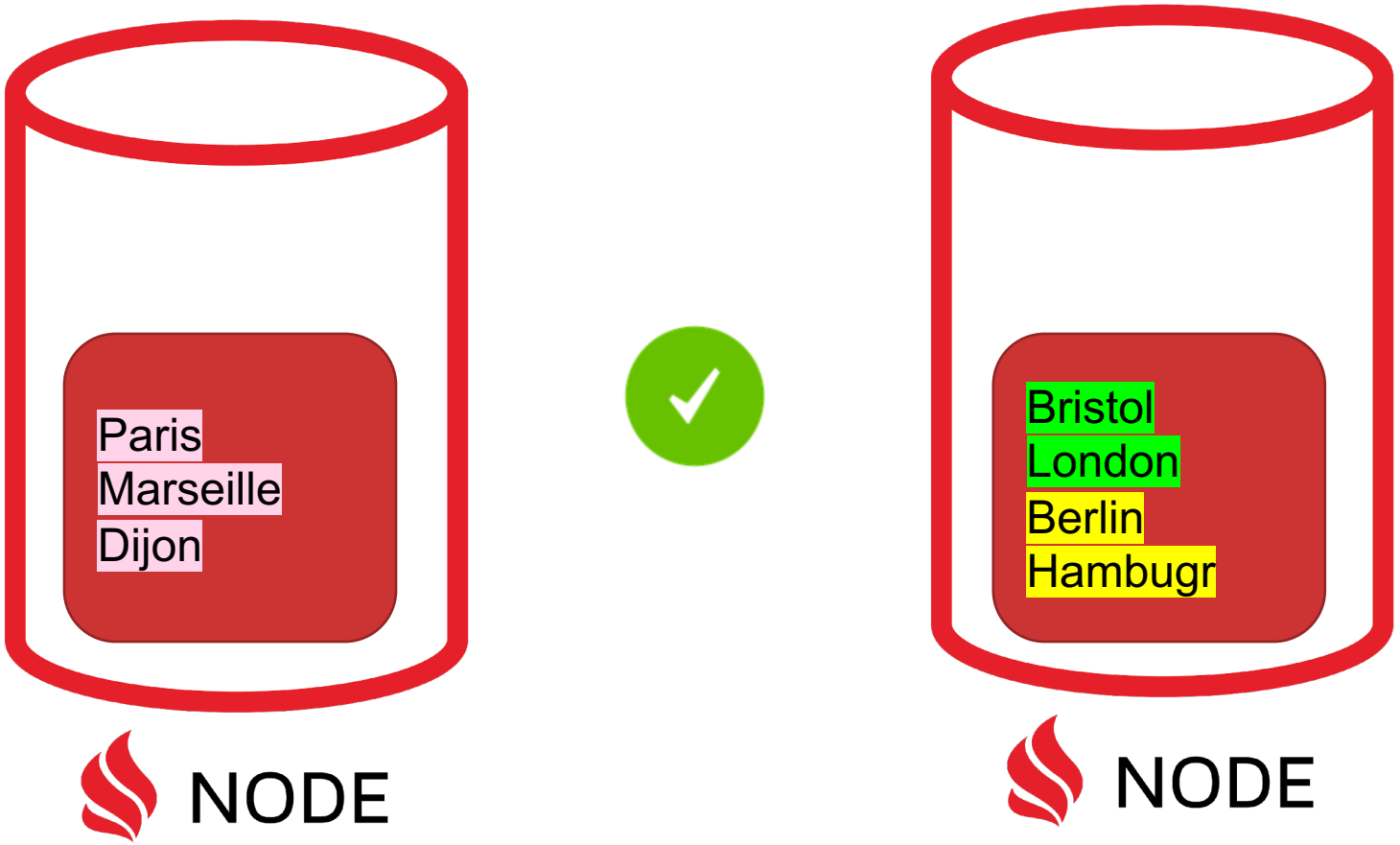


NODE

NODE

Data distribution: affinity column

```
CREATE TABLE CITY (
    id INT,
    countryId INT,
    name VARCHAR,
    population INT,
    PRIMARY KEY(id, countryId))
    WITH \"AFFINITY_KEY=countryId\"
```



Useful query execution hints

SqlFieldsQuery flags (take a look at the javadoc)

- lazy
- enforceJoinOrder
- distributedJoins
- collocated (read as collocated **GROUP BY**)
- skipReducerOnUpdate

Current capabilities of the H2-based engine

- Current capabilities of the Optimizer:
  - JOIN order (+ trivial cost model)
  - Index usage
  - Subquery to JOIN
- Memory Tracking + Offloading to disk
  - Supported only for GG CE
- Single MAP / REDUCE phase

IEP-37: New query execution engine

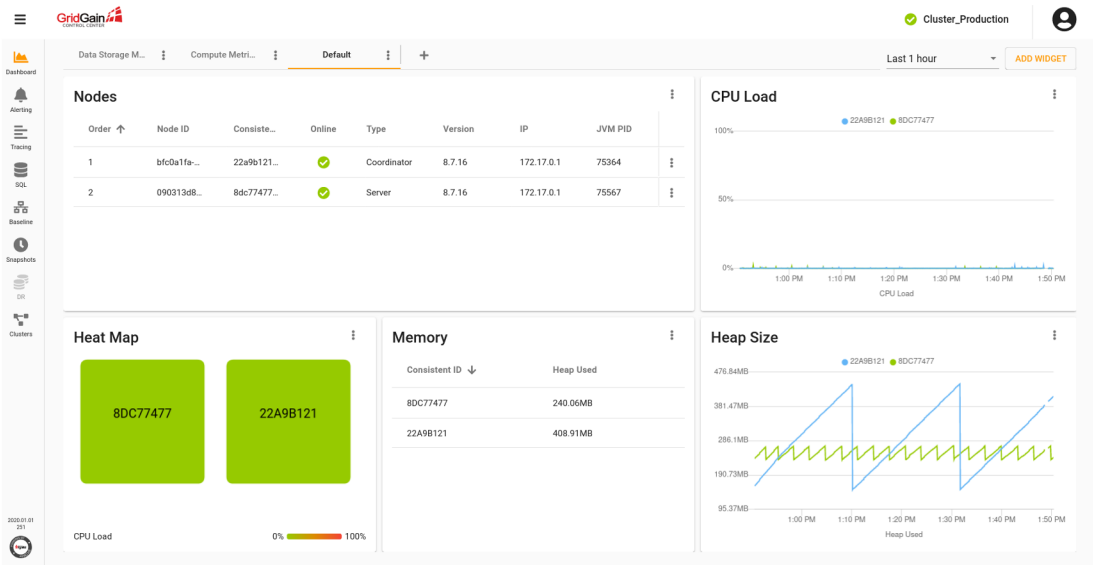
Based on Apache Calcite

Diagnostic tools

Diagnostic tools overview

- Query Plan
- System Views
- Metrics
- Tracing
- Logs

GridGain Control Center



Query Plan: Simple Plain Query

```
EXPLAIN SELECT ID FROM TEST WHERE val0 = ? AND val1 < ?
```

MAP (The First Rows)

```
SELECT
    "__Z0"."ID" AS "__C0_0"
FROM "PUBLIC"."TEST" "__Z0"
    /* PUBLIC.IDX0: VAL1 < ?1 */
WHERE ("__Z0"."VAL1" < ?1)
    AND ("__Z0"."VAL0" = ?2)
```

REDUCE (The Last Row)

```
SELECT
    "__C0_0" AS "ID"
FROM "PUBLIC"."__T0"
    /* PUBLIC.merge_scan */
```

Query Plan: Schema for complex query

Schema

```
CREATE TABLE COUNTRY (id INT PRIMARY KEY, name VARCHAR)");

CREATE TABLE CITY (
        id INT PRIMARY KEY,
        countryId INT,
        name VARCHAR,
        population INT)");

CREATE INDEX CityCountryIdx on CITY(countryId)");
```

Query

```
SELECT COUNTRY.name, cities.pop FROM
    (SELECT countryId, SUM(population) as pop
       FROM CITY
       GROUP BY countryId
       HAVING (AVG(population)) < ?
    ) AS cities "
    JOIN COUNTRY ON COUNTRY.id = cities.countryId
```

Query Plan: MAP queries for complex query

MAP #0

```
SELECT
    "__Z0"."COUNTRYID" AS "__C0_0",
    SUM("__Z0"."POPULATION") AS "__C0_1",
    AVG(CAST("__Z0"."POPULATION" AS DOUBLE)) AS "__C0_2",
    COUNT("__Z0"."POPULATION") AS "__C0_3"
FROM "PUBLIC"."CITY" "__Z0"
    /* PUBLIC.CITYCOUNTRYIDX */
GROUP BY "__Z0"."COUNTRYID"
/* group sorted */
```

MAP #1

```
SELECT
    "__Z2"."NAME" AS "__C1_0",
    "__Z2"."ID" AS "__C1_1"
FROM "PUBLIC"."COUNTRY" "__Z2"
    /* PUBLIC._key_PK_proxy */
ORDER BY 2
/* index sorted */
```

Query Plan: REDUCE queries for complex query

```sql
SELECT
    "__Z2__NAME" AS "NAME",
    "CITIES__Z1"."POP" AS "POP"
FROM (
    SELECT
        "__C0_0" AS "COUNTRYID",
        CAST(CAST(SUM("__C0_1") AS BIGINT) AS BIGINT) AS "POP"
    FROM "PUBLIC"."__T0"
    GROUP BY "__C0_0"
    HAVING CAST((CAST(SUM("__C0_2" * "__C0_3") AS BIGINT) / CAST(SUM("__C0_3") AS BIGINT)) AS INTEGER) < ?1
    ORDER BY 1
) "CITIES__Z1"
    /* SELECT
        __C0_0 AS COUNTRYID,
        CAST(CAST(SUM(__C0_1) AS BIGINT) AS BIGINT) AS POP
    FROM PUBLIC.__T0
        /++ PUBLIC.merge_scan ++/
    GROUP BY __C0_0
    HAVING CAST((CAST(SUM("__C0_2" * "__C0_3") AS BIGINT) / CAST(SUM("__C0_3") AS BIGINT)) AS INTEGER) < ?1
    ORDER BY 1
    */
INNER JOIN (
    SELECT
        "__C1_0" AS "__Z2__NAME",
        "__C1_1" AS "__Z2__ID"
    FROM "PUBLIC"."__T1"
    ORDER BY 2
) "__Z3"
    /* SELECT
        __C1_0 AS __Z2__NAME,
        __C1_1 AS __Z2__ID
    FROM PUBLIC.__T1
        /++ PUBLIC.merge_sorted: __C1_1 IS ?2 ++/
    WHERE __C1_1 IS ?2
    ORDER BY 2
    /++ index sorted ++/: __Z2__ID = CITIES__Z1.COUNTRYID
     */
    ON 1=1
WHERE "__Z2__ID" = "CITIES__Z1"."COUNTRYID"
```

Query Plan: REDUCE queries for complex query

```
    ORDER BY 1
     */
INNER JOIN (
    SELECT
        "__C1_0" AS "__Z2__NAME",
        "__C1_1" AS "__Z2__ID"
    FROM "PUBLIC"."__T1"
    ORDER BY 2
) "__Z3"
    /* SELECT
        __C1_0 AS __Z2__NAME,
        C1_1 AS  Z2__ID
```

MAP #1

```
SELECT
    "__Z2"."NAME" AS "__C1_0",
    "__Z2"."ID" AS "__C1_1"
FROM "PUBLIC"."COUNTRY" "__Z2"
    /* PUBLIC._key_PK_proxy */
ORDER BY 2
/* index sorted */
```

Query Plan: What is useful?

- JOIN order
- Index usage
- Distributed Query Split (MAP / REDUCE)

System Views

- Documentation:
  https://ignite.apache.org/docs/latest/SQL/schemas

- Short query to remember

```
SELECT * FROM SYS.VIEWS
```

- The most useful views for SQL:

  - TABLES
  - INDEXES
  - TABLE_COLUMNS
  - SQL_QUERIES
  - SQL_QUERIES_HISTORY
  - CLIENT_CONNECTIONS
  - METRICS

System Views: What is useful?

- TABLES / TABLE_COLUMNS:
  affinity, field types

- INDEXES
  fields, inline size

- SQL_QUERIES
  current running **on node** ⚠️

Metrics

- Documentation:
  https://ignite.apache.org/docs/latest/monitoring-metrics/new-metrics

- Exporters
  - JMX
  - SQL View: SELECT * FROM **SYS.METRICS**
  - OpenCensus
  - Logs

Metrics: What is useful?

- Index I/O

  io.statistics.sortedIndexes.*

- Prepared query

  sql.parser.cache.*

- Memory usage

  sql.memory.quotas.*

freeMem

OffloadedQueriesNumber

Tracing: Experimental

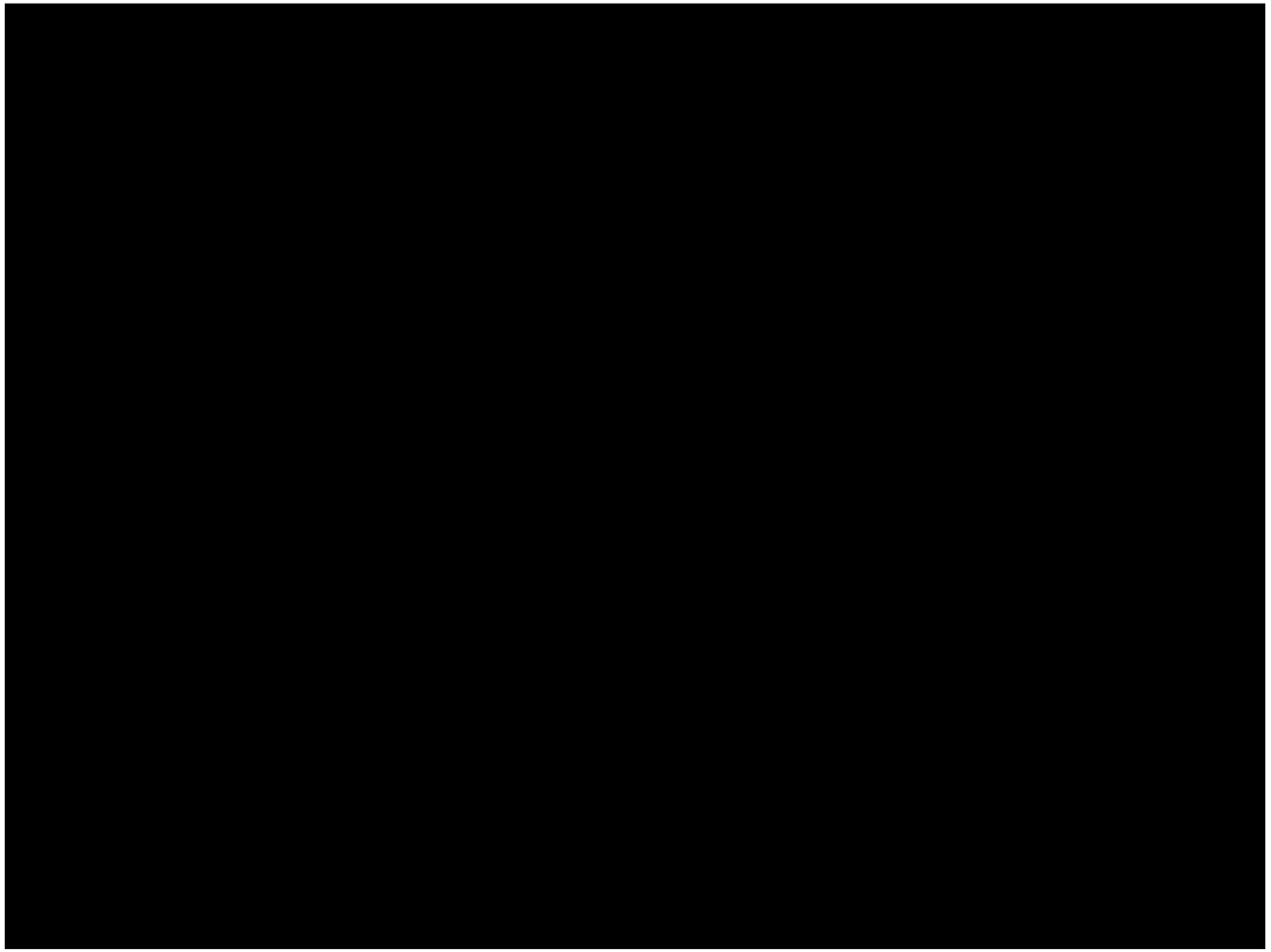
- Documentation: https://ignite.apache.org/docs/latest/monitoring-metrics/tracing

- OpenCensus

- Export to Zipkin

```
ZipkinTraceExporter.createAndRegister(
    ZipkinExporterConfiguration.builder().setV2Url("http://localhost:9411/api/v2/spans")
        .setServiceName("ignite-cluster").build());
```

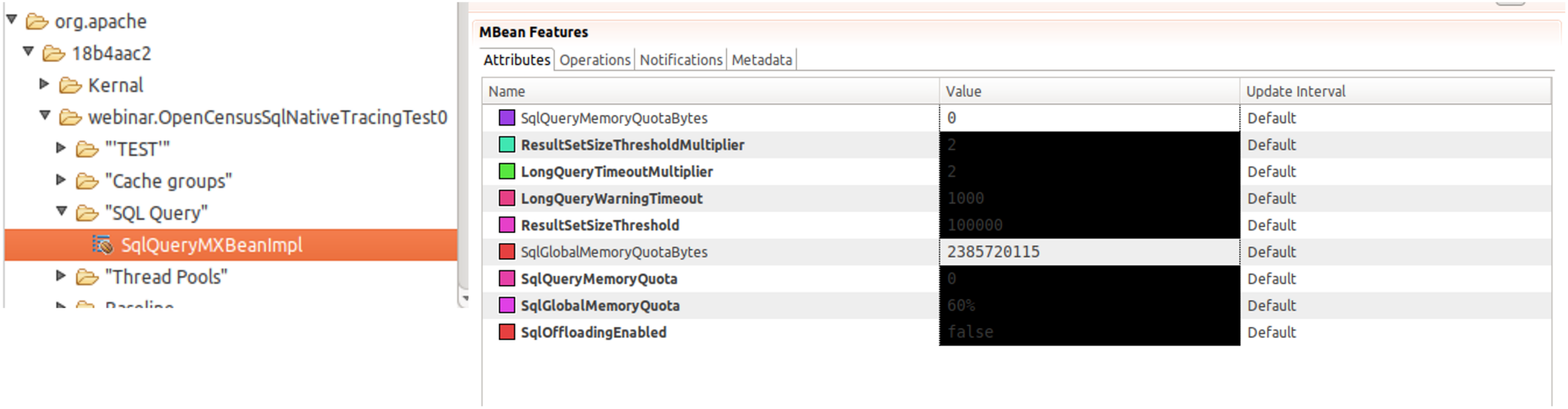
Tracing: Demo (video)



Logs

- Documentation: in progress
- Errors / Exceptions on MAP nodes
- Warnings:
  - Long running queries
  - Queries with huge result sets
  - Inline size

Logs: Tune in runtime by JMX



- Long running: timeout / multiplier
- Huge results: threshold / multiplier

Logs: Message example (Long running)

```
[2021-01-23 16:07:52,087][WARN ][long-qry-#31%webinar.OpenCensusSqlNativeTracingTest0%]
[LongRunningQueryManager] Query execution is too long [duration=2153ms, type=MAP,
distributedJoin=false, enforceJoinOrder=false, lazy=false, schema=PUBLIC, sql='SELECT
"__Z0"."ID" AS "__C0_0",
"PUBLIC".DELAY(1000) AS "__C0_1"
FROM "PUBLIC"."TEST" AS "__Z0"
WHERE "__Z0"."VAL0" > ?1', plan=SELECT
    __Z0.ID AS __C0_0,
    PUBLIC.DELAY(1000) AS __C0_1
FROM PUBLIC.TEST __Z0
    /* PUBLIC.TEST.__SCAN_ */
    /* scanCount: 4 */
    /* lookupCount: 1 */
WHERE __Z0.VAL0 > ?1, node=TcpDiscoveryNode [id=8401e400-b5f5-4040-9c73-70262460003,
consistentId=webinar.OpenCensusSqlNativeTracingTest3, addrs=ArrayList [127.0.0.1],
sockAddrs=HashSet [/127.0.0.1:0], discPort=0, order=4, intOrder=4,
lastExchangeTime=1611407258490, loc=false, ver=8.8.127#20210123-sha1:ef55c03f,
isClient=true], reqId=101, segment=0]
```

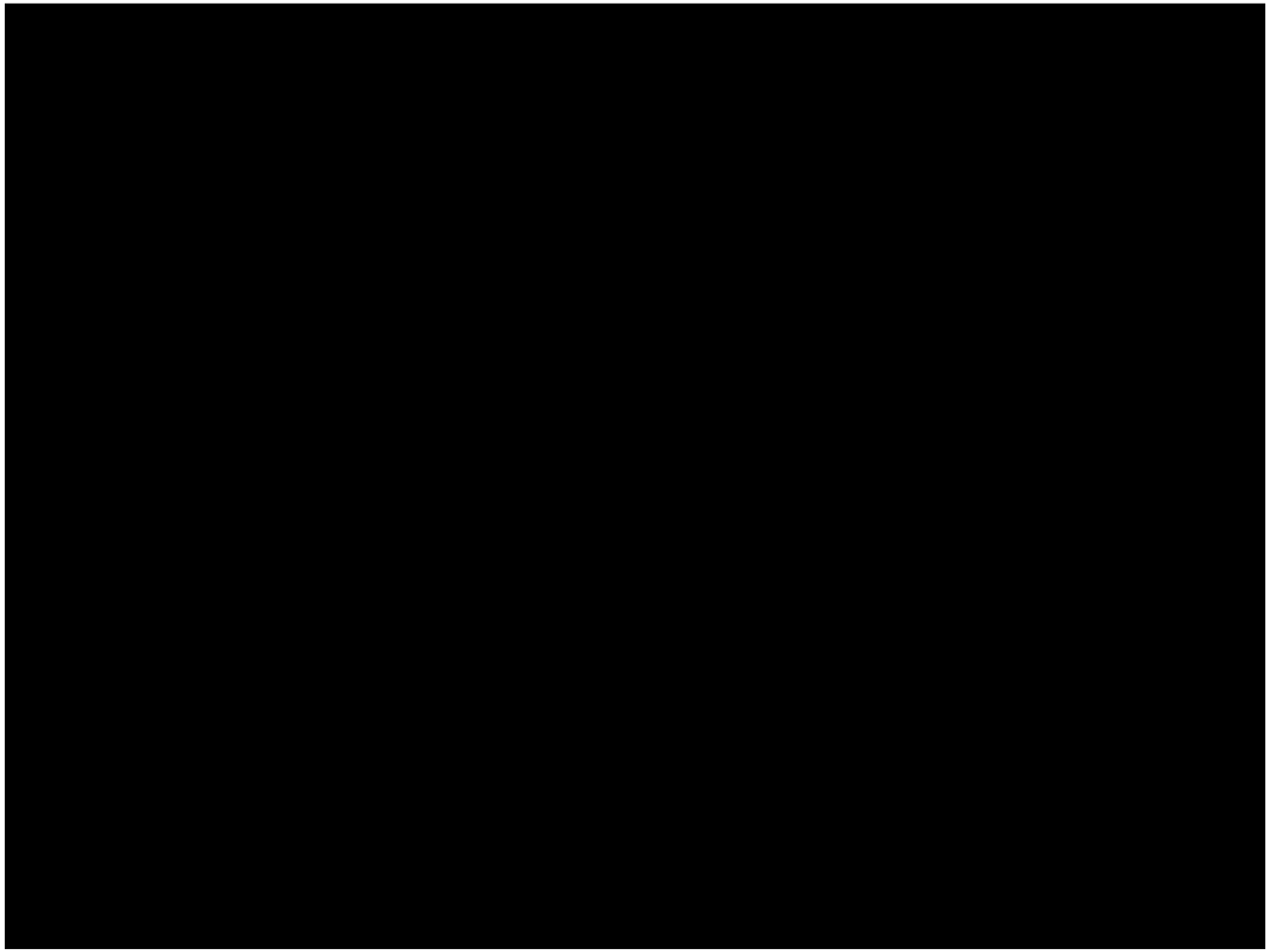
Logs: Message example (Huge results)

```
[2021-01-30 20:07:37,135][WARN ][test-runner-#55%webinar.LogTest%][GridMapQueryExecutor]
Query produced big result set.  [fetched=320000, duration=647ms, type=MAP,
distributedJoin=false, enforceJoinOrder=false, lazy=true, schema=PUBLIC, sql='SELECT
"T0__Z0"."NAME" AS "__C0_0"
FROM "PUBLIC"."TEST" AS "T0__Z0"
 INNER JOIN "PUBLIC"."TEST" AS "T1__Z1"
 ON TRUE
WHERE "T0__Z0"."PRICE" > 10', plan=SELECT
    T0__Z0.NAME AS __C0_0
FROM PUBLIC.TEST T0__Z0
    /* PUBLIC.TEST.__SCAN_ */
    /* WHERE T0__Z0.PRICE > 10
    */
    /* scanCount: 43 */
    /* lookupCount: 1 */
INNER JOIN PUBLIC.TEST T1__Z1
    /* PUBLIC.TEST.__SCAN_ */
    ON 1=1
    /* scanCount: 320031 */
    /* lookupCount: 32 */
```

Logs: Message example (Inline size)

```
[WARN ][test-runner-#55%webinar.LogTest%][IgniteH2Indexing]
Indexed columns of a row cannot be fully inlined into index
what may lead to slowdown due to additional data page reads,
increase index inline size if needed
(use INLINE_SIZE option for CREATE INDEX command,
QuerySqlField.inlineSize for annotated classes,
or QueryIndex.inlineSize for explicit QueryEntity configuration)
[cacheName=TEST, tableName=SQL_PUBLIC_TEST, idxName=IDX0,
idxCols=(TYPE, PRICE, TS, NAME, ID0, ID1, _KEY), idxType=SECONDARY,
curSize=10, recommendedInlineSize=98]
```

GridGain Control Center: All in one



Analysis examples

Case #0: Invalid JOIN results

Query

```
SELECT *
FROM CITY, COUNTRY
WHERE CITY.countryId=COUNTRY.id AND COUNTRY.name=?
```

OK on one-node cluster. INVALID on multiple nodes

Diagnostic

COLUMN_NAME	TABLE_NAME	AFFINITY_COLUMN	PK
_KEY	CITY	true	true
ID	CITY	false	true
_KEY	COUNTRY	true	true
ID	COUNTRY	false	true

Case #0: Invalid JOIN results (Single Node)

Schema

```
CREATE TABLE COUNTRY (id INT PRIMARY KEY, name VARCHAR)");

CREATE TABLE CITY (
        id INT PRIMARY KEY,
        countryId INT,
        name VARCHAR,
        population INT)");
```

Query

```
SELECT *
FROM CITY, COUNTRY
WHERE CITY.countryId=COUNTRY.id AND COUNTRY.id=?
```

INVALID

IEP-24: SQL Partition Pruning

Case #0: Valid schema

Schema #0 (Partitioned)

```
CREATE TABLE COUNTRY (id INT PRIMARY KEY, name VARCHAR)");

CREATE TABLE CITY (
        id INT,
        countryId INT,
        name VARCHAR,
        population INT,
        PRIMARY KEY(id, countryId))
WITH \"AFFINITY_KEY=countryId\"
```

Schema #1 (Replicated)

```
CREATE TABLE COUNTRY (id INT PRIMARY KEY, name VARCHAR)")
WITH \"TEMPLATE=replicated\";

CREATE TABLE CITY (
        id INT PRIMARY KEY,
        countryId INT,
        name VARCHAR,
        population INT)")
WITH \"TEMPLATE=replicated\";
```

Case #1: Be cautious while using SQL + cache API

```
CREATE TABLE TEST (
id0 INT, id1 INT, val VARCHAR,
PRIMARY KEY (id0, id1)
)
WITH\"KEY_TYPE=TEST_KEY,CACHE_NAME=TEST\"
```

```
BinaryObjectBuilder bob = ign.binary().builder(typeName:"TEST_KEY");
bob.setField(name:"ID0", val:0);
bob.setField(name:"ID1", val:0);
bob.setField(name:"hidden", val:0);
```

```
SELECT count(1) FROM TEST  ?  ign.cache(name:"TEST").size()
```

Case#2: Index performance & INLINE_SIZE

```
CREATE TABLE TEST (
id0 INT, id1 BIGINT, name VARCHAR, price INT,
PRIMARY KEY (id0, id1))
```

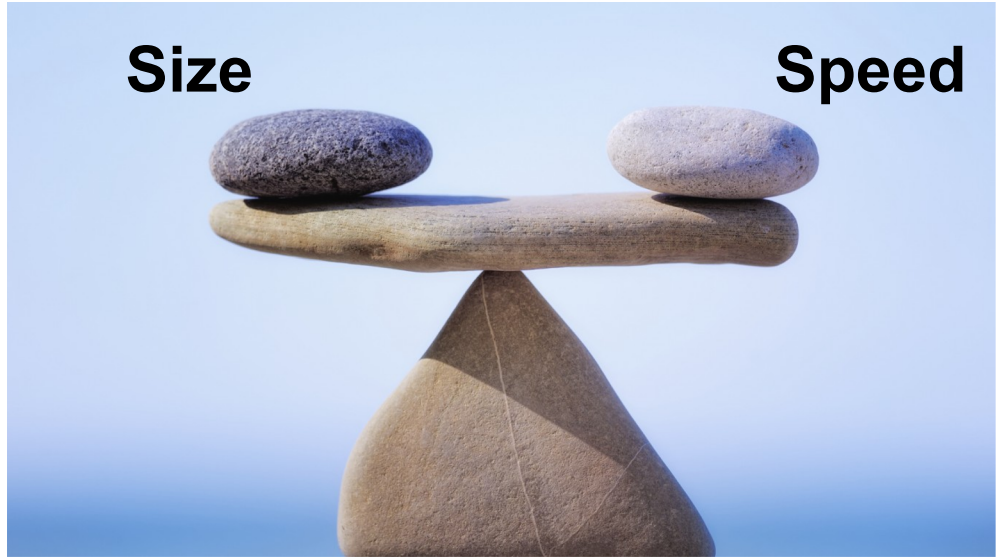
```
CREATE INDEX idx0 on TEST(price) INLINE_SIZE 5
```

VS

```
CREATE INDEX idx0 on TEST(price)
```

Small INLINE

Large INLINE



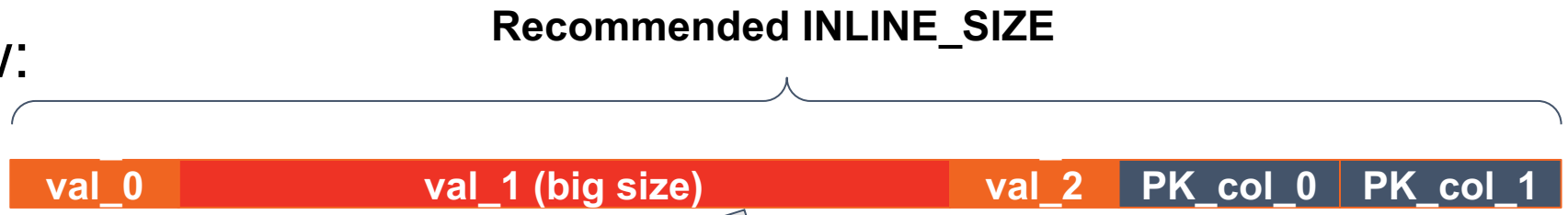
Case#2: Index inline values in deep

INT	tag (1) + value (4)	**5**
BIGINT (long)	tag (1) + value (8)	**9**
VARCHAR (String)	tag (1) + size (2) + UTF-8 value (trimmed)	**3 + len**
OTHER (BinaryObject)	tag (1) + hash (4)	**5**

Case#2: Index record in deep

Index structure:

Data row:



Fixed Size

Varsize trimmed to fixed size (prefix)

Inline values

Case#2: Index inline recommendation on examples

id	parent	path
root	NULL	/
folder_0	root	/folder_0
folder_1	toot	/folder_1
sub_01	folder_0	/folder_0/sub_01
sub_02	folder_0	/folder_0/sub_02
...	...	...
sub_NNN	folder_0	/folder_0/sub_NNN

Const prefix

CREATE INDEX <name> ON TBL (path)
INLINE SIZE ?

INLINE_SIZE < 17 - unuseful and it's a waste of memory
VARCHAR: 1 (tag) + 2 (size) + 14 (const chars)

Case#2: Index inline recommendation on examples

batch_id	my_id	order_num
BUR	74d7c5d4-b9d7-4ffc-8ac8-3bf5d8989c5c	0
BUR	11afda19-8b27-494e-9ff3-5a7078b2d699	1
...	...	...
SFO	11afda19-8b27-494e-9ff3-5a7078b2d699	0
SFO	74d7c5d4-b9d7-4ffc-8ac8-3bf5d8989c5c	2

Const prefix
few NDV

UUID - fixed size type

```
CREATE INDEX <name>
ON TBL (
    batch_id,
    my_id,
    order_num)
INLINE SIZE ?
```

INLINE_SIZE < 23 - unuseful and it's a waste of memory

Calculation: 1 (tag) + 2 (size) + 3 (const chars) + 1 (tag)+ 16 (uuid)

Recommendation: 23 + INLINE_SIZE (order_num) + INLINE_SIZE (PK)

Case#3: High memory consumption

```
SELECT  "SQL", DURATION_MIN, DURATION_MAX, MEMORY_MAX
FROM SYS.SQL_QUERIES_HISTORY WHERE MEMORY_MAX > 1000000;
```

	SQL	DURATION_MIN	DURATION_MAX	MEMORY_MAX
1	SELECT * FROM TEST T0 WHERE T0.type = 0 ORDER BY T0.name	4,606	20,113	5,354,052

```
SELECT *
FROM "PUBLIC"."TEST" "T0__Z0"
    /* PUBLIC.IDX_TYPE: TYPE = 0 */
WHERE "T0__Z0"."TYPE" = 0
ORDER BY 3
```

```
SELECT *
FROM "PUBLIC"."TEST" "T0__Z0" USE INDEX ("IDX_NAME")
    /* PUBLIC.IDX_NAME */
WHERE "T0__Z0"."TYPE" = 0
ORDER BY 3
/* index sorted */
```

Case#3: `lazy` execution mode

lazy = false	lazy = true
• Accumulate **`ResultSet`** in the heap on query execution. • Query exec finishes before **`ResultSet`** is ready to read	• Use **`Iterator`** pattern • (When H2 engine can do it)

- **Not a silver bullet** to reduce memory usage
- Performance impact for tiny result set (~10-15%)
- Not implemented for **Reduce** phase yet

Case#3: Configure GridGain SQL memory tracking

- SqlConfiguration
  - Object
  - defaultQueryTimeout: long
  - longQueryWarningTimeout: long
  - sqlGlobalMemoryQuota: String
  - sqlOffloadingEnabled: boolean
  - sqlQueryHistorySize: int
  - sqlQueryMemoryQuota: String
  - sqlSchemas: String...
  - sqlSchemas: String[]

Default: **60% Heap**

⚠️ **Pay attention to GROUP BY**
Default: **OFF**

Default: **OFF**

SQL typical traps / optimization checklist

SQL Troubleshooting Checklist

- Data **Colocation** (JOIN) ✅
- **Lazy** execution mode ✅
- Index usage: plan + hint **USE INDEX(<idx_name>)** ✅
- JOIN order (use **enforceJoinOrder** if need) ✅
- Index **INLINE** & Index structure ✅
- Simple optimization (e.g. OR -> UNION ALL) ✅

Useful Links:

SQL Performance Tuning

https://ignite.apache.org/docs/latest/perf-and-troubleshooting/sql-tuning

New SQL Engine (based on Apache Calcite framework)

IEP-37: New query execution engine

Q & A Session