SQL for NoSQL and how Apache Calcite can help

FOSDEM 2017
Christian Tzolov

Engineer at Pivotal
BigData, Hadoop, Spring Cloud Dataflow
Apache Committer, PMC member
Apache {Crunch, Geode, HAWQ, ...}

Disclaimer This talk expresses my personal opinions. It is not read or approved by Pivotal and does not necessarily reflect the views and opinions of Pivotal nor does it constitute any official communication of Pivotal. Pivotal does not support any of the code shared here.
“It will be interesting to see what happens if an established NoSQL database decides to implement a reasonably standard SQL; The only predictable outcome for such an eventuality is plenty of argument.”

2012, Martin Fowler, P.J. Sadalage, NoSQL Distilled
Data Big Bang

Why?
NoSQL Driving Forces

• **Rise of Internet Web, Mobile, IoT – Data Volume, Velocity, Variety challenges**
  
  *ACID & 2PC clash with Distributed architectures. CAP, PAXOS instead.*

• **Row-based Relational Model. Object-Relational Impedance Mismatch**
  
  *More convenient data models: Datastores, Key/Value, Graph, Columnar, Full-text Search, Schema-on-Load…*

• **Infrastructure Automation and Elasticity (Cloud Computing)**
  
  *Eliminate operational complexity and cost. Shift from Integration to application databases …*
Data Big Bang Implications

- Over 150 commercial NoSQL and BigData systems.
- Organizations will have to mix data storage technologies!
- How to integrate such multitude of data systems?
“Standard” Data Process/Query Language?

- **Functional - Unified Programming Model**
  - Apache {Beam, Spark, Flink, Apex, Crunch}, Cascading
  - Converging around Apache Beam

- **Declarative - SQL**
  - Adopted by many NoSQL Vendors
  - Most Hadoop tasks: Hive and SQL-on-Hadoop
  - Spark SQL - most used production component for 2016
  - Google F1

```
pcollection.apply(Read.from("in.txt"))
  .apply(FlatMapElements.via((String word) ->
    asList(word.split("[^a-zA-Z]+"))
  )
  .apply(Filter.by((String word)->!word.isEmpty())
  .apply(Count.<String>perElement())
```

**Batch & Streaming, OLTP**

```
SELECT b."totalPrice", c."firstName"
FROM "BookOrder" as b
INNER JOIN "Customer" as c
  ON b."customerNumber" = c."customerNumber"
WHERE b."totalPrice" > 0;
```

**OLAP, EDW, Exploration**
SQL for NoSQL?

- Extended Relational Algebra - already present in most NoSql data system
- Relational Expression Optimization – Desirable but hard to implement
Organization Data - Integrated View

Single Federated DB (M:1:N)

- HAWQ FDBS
  - Apache HAWQ
    - Optimizer, Columnar (HDFS)
    - PXF 1
    - PXF 2
    - PXF n
- Native API 1
- Native API 2
- Native API n
- SQL/JDBC
- NoSQL 1
- NoSQL 1
- NoSQL n

Direct (M:N)

- Organization Data Tools
  - SQL/JDBC
  - Calcite SQLAdapter 1
    - NoSQL 1
  - Calcite SQLAdapter 2
    - NoSQL 2
  - Calcite SQLAdapter n
    - NoSQL n

https://issues.apache.org/jira/browse/HAWQ-1235
Single Federated Database

Federated External Tables with Apache HAWQ - MPP, Shared-Noting, SQL-on-Hadoop

CREATE EXTERNAL TABLE MyNoSQL
(
    customer_id TEXT,
    first_name TEXT,
    last_name TEXT,
    gender TEXT
)
LOCATION ('pxf://MyNoSQL-URL>?

FRAGMENTER=MyFragmenter&
ACCESSOR=MyAccessor&
RESOLVER=MyResolver&

FORMAT
'custom'(formatter='pxfwritable_import');
Apache Calcite?

Java framework that allows SQL interface and advanced query optimization, for virtually any data system

- Query Parser, Validator and Optimizer(s)
- JDBC drivers - local and remote
- Agnostic to data storage and processing
Calcite Application

- Apache Apex
- Apache Drill
- Apache Flink
- Apache Hive
- Apache Kylin
- Apache Phoenix
- Apache Samza
- Apache Storm
- Cascading
- Qubole Quark
- SQL-Gremlin
- ...
SQL Adapter Design Choices

*SQL completeness vs. NoSql design integrity*

- Data Type Conversion

- Move Computation to Data

- (simple) Predicate Pushdown: **Scan, Filter, Projection**

- (complex) Custom Relational Rules and Operations: **Sort, Join, GroupBy ...**

**Catalog** – namespaces accessed in queries

**Schema** - collection of schemas and tables

**Table** - single data set, collection of rows

**RelDataType** – SQL fields types in a Table
Geode to Calcite Data Types Mapping

Geode Cache is mapped into Calcite Schema

Create Column Types (RelDataType) from Geode Value class (JavaTypeFactoryImpl)

Geode Key/Value is mapped into Table Row

Regions are mapped into Tables
**Geode Adapter - Overview**

- **Parse SQL, converts into relational expression and optimizes**
- **Spring Data API for interacting with Geode**
- **Push down the relational expressions supported by Geode OQL and falls back to the Calcite Enumerable Adapter for the rest**
- **Convert SQL relational expressions into OQL queries**

**Diagram:**
- Geode API and OQL
- SQL/JDBC/ODBC
- Geode Adapter (Geode Client)
- Apache Calcite
- Enumerable Adapter
- Spring Data API for interacting with Geode
- Geode Server
- Geode Server
- Geode Server
- Data
- Data
- Data
Simple SQL Adapter

Initialize

```java
connect jdbc:calcite:model=path-to-model.json
```

defaultSchema: 'MyNoSQL',
schemas: [{
  name: 'MyNoSQLAdapter,'
  factory: MySchemaFactory,
  operand: { myNoSqlUrl: ..., }
}]

```
ScannableTable,
FilterableTable,
ProjectableFilterableTable
```

MyTable

+getRowType(RelDataTypeFactor)
+scan(ctx):Enumerator<Object[]>

Returns an Enumeration over the entire target data store

Uses reflection to builds RelDataType from your value's class type

Query

```sql
SELECT b."totalPrice" FROM "BookOrder" as b
WHERE b."totalPrice" > 0;
```

```
on scan() create
```

MyEnummerator

+moveNext()
+convert(Object):E

Defined in the Linq4j sub-project

Converts MyNoSQL value response into Calcite row data

```
Get all Data
```

My NoSQL
Non-Relational Tables (Simple)

Scanned without intermediate relational expression.

- **ScannableTable** - can be scanned
  ```csharp
  Enumerable<Object[]> scan(DataContext root);
  ```

- **FilterableTable** - can be scanned, applying supplied filter expressions
  ```csharp
  Enumerable<Object[]> scan(DataContext root, List<RexNode> filters);
  ```

- **ProjectableFilterableTable** - can be scanned, applying supplied filter expressions and projecting a given list of columns
  ```csharp
  Enumerable<Object[]> scan(DataContext root, List<RexNode> filters, int[] projects);
  ```
Calcite Ecosystem

Several “semi-independent” projects.

- JDBC and Avatica
- SQL Parser & AST
  - Converts SQL queries into AST (SqlNode ...)
  - Relational Algebra, expressions, optimizations ...
- Linq4j
  - Expression Tree
- Interpreter
- Enumerable Adapter
- 3rd party Adapters

Local and Remote JDBC driver

Port of LINQ (Language-Integrated Query) to Java.

Method for translating executable code into data (LINQ/MSN port)

Complies Java code generated from linq4j "Expressions". Part of the physical plan implementer

Default (in-memory) Data Store Adapter implementation. Leverages Linq4j
Calcite SQL Query Execution Flow

1. On new SQL query JDBC delegates to **Prepare** to prepare the query execution

2. Parse SQL, convert to rel. expressions. **Validate** and **Optimize** them

3. Start building a physical plan from the relation expressions

4. Implement the Geode relations and encode them as **Expression tree**

5. Pass the Expression tree to the **Interpreter** to generate Java code

6. Generate and Compile a **Binder** instance that on ‘bind()’ call runs Geodes’ query method

7. JDBC uses the newly compiled **Binder** to perform the query on the Geode Cluster
Query optimizer: Transforms a relational expression according to a given set of rules and a cost model.

RelOptRule
+ onMatch(call)

RelOptRuleOperand

ConverterRule
+ RelNode convert(RelNode)

Convertor

Convention

RelTrait

RexNode

MyDBConvention

NONE

EnumerableConvention

Rule transforms an expression into another. It has a list of Operands, which determine whether the rule can be applied to a particular section of the tree.

calling convention used to represent a single data source.
Inputs to a relational expression must be in the same convention.
Calcite Adapter Implementation Patterns

1. ConventionImpl("MyAdapter")
   - Common interface for all MyAdapter Relation Expressions. Provides implementation callback method called as part of physical plan implementation.

2. MyAdapterRel
   - + implement(implContext)
   - ImplContext
     - + implParm1
     - + implParm2 ...

3. MyAdapterTable
   - + toRel(optTable).
   - + asQueryable(provider, ...)

4. AbstractTableQueryable
   - + myQuery(params) : Enumerator

5. MyAdapterQueryable
   - myQuery() implements the call to your DB. It is called by the auto-generated code. It must return an Enumerable instance.

6. MyAdapterToEnumerableConverterRule
   - operands: (RelNode.class, MyAdapterConvention, EnumerableConvention)

7. MyAdapterToEnumerableConverter
   - + implement(EnumerableRelImplementor) {
     - ctx = new MyAdapterRel.ImplContext()
     - getImputs().implement(ctx)
     - return BlockBuild.append(MY_QUERY_REF, Expressions.constant(ctx.implParms1), Expressions.constant(ctx.implParms2))
   }

8. RelOptRule
   - MyAdapterProjectRule
   - MyAdapterFilterRule
   - MyAdapterXXXRule

9. RelNode
   - MyAdapterProject
   - MyAdapterFilter
   - MyAdapterXXX

MyAdapterAdapter components

TranslatableTable
AbstractQueryableTable
Convention.Impl("MyAdapter")
RelNode

Can convert queries in Expression
Common interface for all MyAdapter Relation Expressions. Provides implementation callback method called as part of physical plan implementation

<instance of>

"create on match"
Relational Algebra

SELECT b."totalPrice", c."firstName"
FROM "BookOrder" as b
INNER JOIN "Customer" as c ON b."customerNumber" = c."customerNumber"
WHERE b."totalPrice" > 0;
SELECT b."totalPrice", c."firstName"
FROM "BookOrder" as b
INNER JOIN "Customer" as c ON b."customerNumber" = c."customerNumber"
WHERE b."totalPrice" > 0;

PLAN

'LogicalProject(totalPrice=[$6], firstName=[$8])
  LogicalFilter(condition=[>($6, 0)])
    LogicalJoin(condition=[=(5, 7)], joinType=[inner])
      LogicalTableScan(table=[[TEST, BookOrder]])
    LogicalTableScan(table=[[TEST, Customer]])
Calcite with Geode – Scannable Table (Simple)

```
SELECT b."totalPrice", c."firstName"
FROM "BookOrder" as b
INNER JOIN "Customer" as c ON b."customerNumber" = c."customerNumber"
WHERE b."totalPrice" > 0;
```
Calcite with Geode – Relational (Complex)

```sql
SELECT b."totalPrice", c."firstName"
FROM "BookOrder" as b
INNER JOIN "Customer" as c ON b."customerNumber" = c."customerNumber"
WHERE b."totalPrice" > 0;
```

```javascript
'PLAN'
'EnumerableCalc(expr#0..3={[inputs]}, totalPrice=[$t0], firstName=[$t3])
  EnumerableJoin(condition=[=($1, $2)], joinType=[inner])
    GeodeToEnumerableConverterRel
      GeodeProjectRel(totalPrice=[$3], customerNumber=[$6])
      GeodeFilterRel(condition=[>($3, 0)])
      GeodeTableScanRel(table=[[TEST, BookOrder]])
    GeodeToEnumerableConverterRel
      GeodeProjectRel(customerNumber=[$0], firstName=[$1])
      GeodeTableScanRel(table=[[TEST, Customer]])
'"
```
References

• Big Data is Four Different Problems, 2016, M.Stonebraker: https://www.youtube.com/watch?v=S79-buNhdhI
• NoSQL Distilled, 2012 (Pramod J. Sadalage and M.Fowler) https://martinfowler.com/books/nosql.html
• Apache Geode Project (2016) : http://geode.apache.org
• Apache Calcite Project (2016) : https://calcite.apache.org
• Apache Geode Adapter for Apache Calcite: https://github.com/tzolov/calcite
Thanks!
Apache Geode?

“… in-memory, distributed database with strong consistency built to support low latency transactional applications at extreme scale”
Why Apache Geode?

China Railway

5,700 train stations
4.5 million tickets per day
20 million daily users
1.4 billion page views per day
40,000 visits per second

Indian Railways

7,000 stations
72,000 miles of track
23 million passengers daily
120,000 concurrent users
10,000 transactions per minute

Geode Features

• In-Memory Data Storage
  – Over 100TB Memory
  – JVM Heap + Off Heap
• Any Data Format
  – Key-Value/Object Store
• ACID and JTA Compliant Transactions
• HA and Linear Scalability
• Strong Consistency
• Streaming and Event Processing
  – Listeners
  – Distributed Functions
  – Continuous OQL Queries
• Multi-site / Inter-cluster
• Full Text Search (Lucene indexes)
• Embedded and Standalone
• Top Level Apache Project
Apache Geode Concepts

Locator - tracks system members and provides membership information

Client - read and modify the content of the distributed system

CacheServer - process connected to the distributed system with created Cache

Region - consistent, distributed Map (key-value), Partitioned or Replicated

Cache - in-memory collection of Regions

Listener - event handler. Registers for one or more events and notified when they occur

Functions - distributed, concurrent data processing

Cache Server (member)

Locator (member)  Client

Listeners

Functions
Geode Topology

Peer-to-Peer

Client-Server

Multi-Site