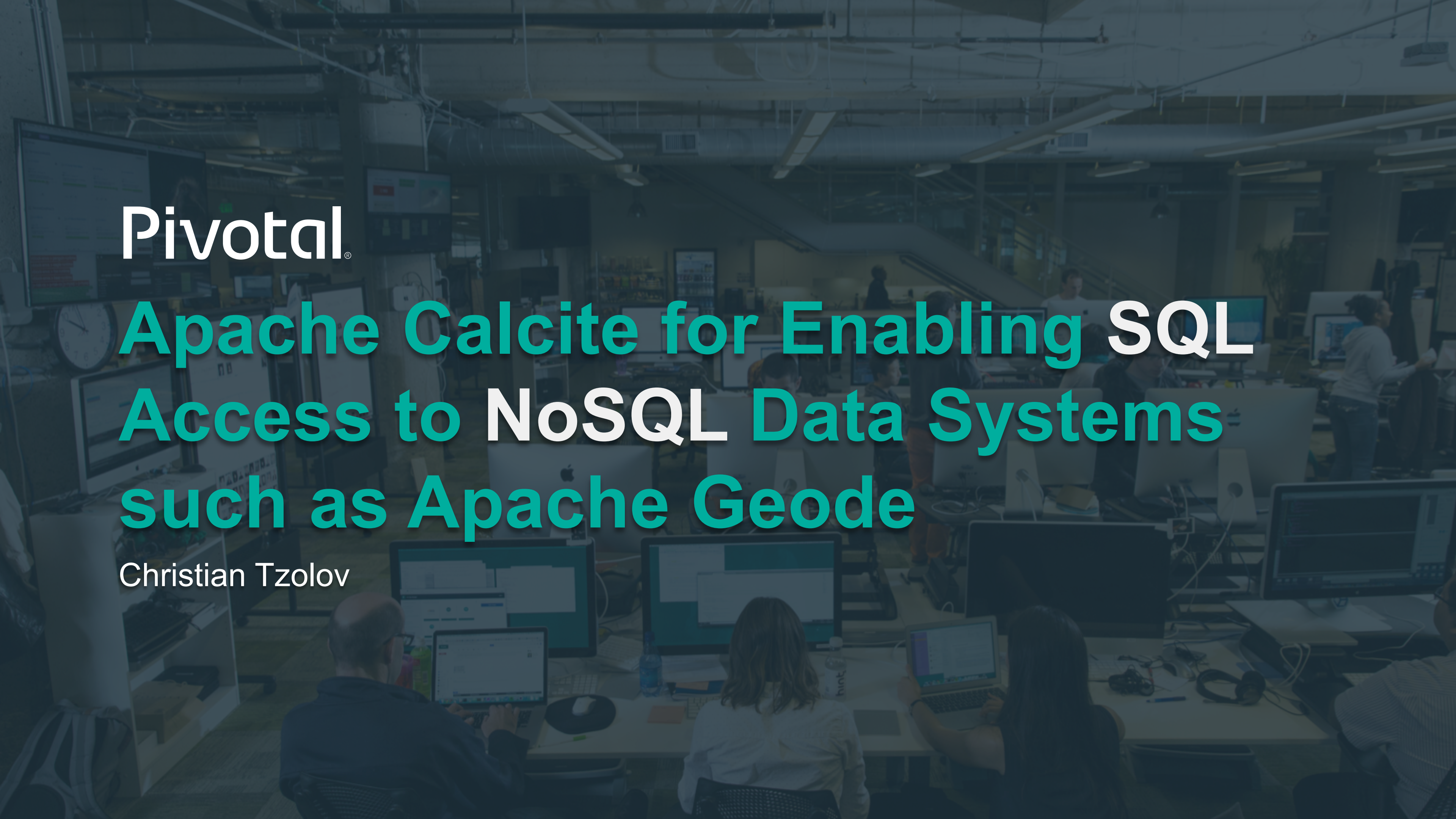
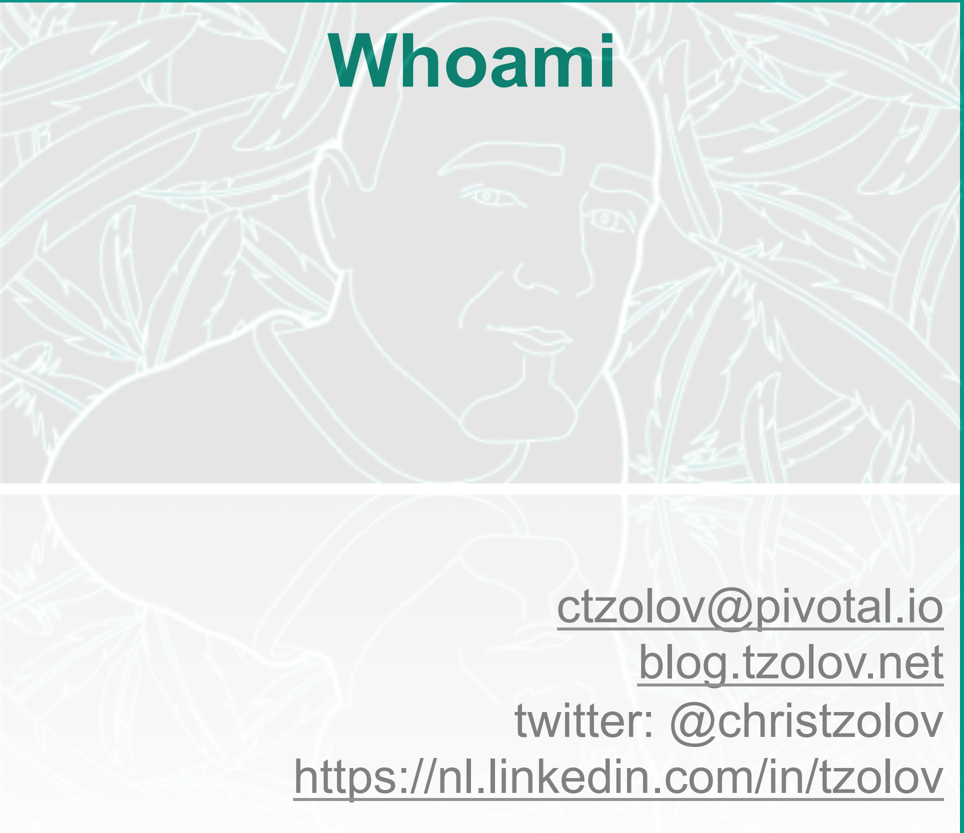


Pivotal

Apache Calcite for Enabling SQL Access to NoSQL Data Systems such as Apache Geode

Christian Tzolov

Whoami



Christian Tzolov

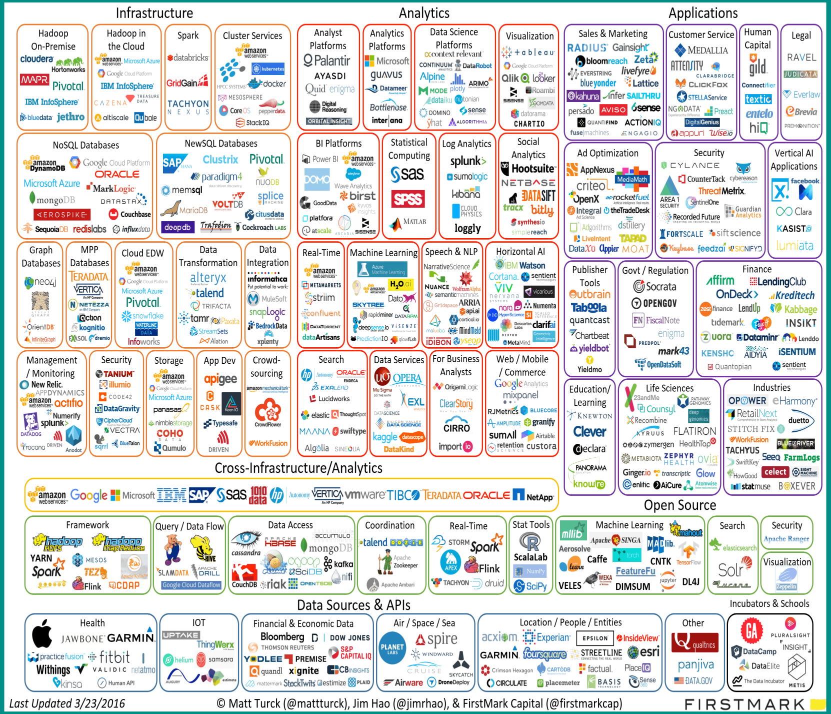
Engineer at Pivotal,
Big-Data, Hadoop,
Spring Cloud Dataflow,
Apache Geode, Apache HAWQ,
Apache Committer,
Apache Crunch PMC member

ctzolov@pivotal.io
blog.tzolov.net
twitter: @christzolov
https://nl.linkedin.com/in/tzolov

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.

Big Data Landscape 2016

- Volume
- Velocity
- Varity
- Scalability
- Latency
- Consistency vs. Availability (CAP)

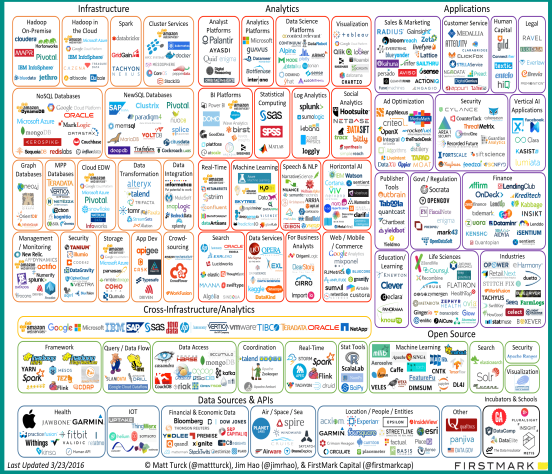


Last Updated 3/23/2016

© Matt Turck (@mattturck), Jim Hao (@jimrhao), & FirstMark Capital (@firstmarkcap)

Data Access

- {Old | New} SQL
- Custom APIs
  - Key / Value
  - Fluent APIs
  - REST APIs
- {My} Query Language



Unified Data Access?

At What Cost?

Apache Calcite APACHE GEODE

Pivotal

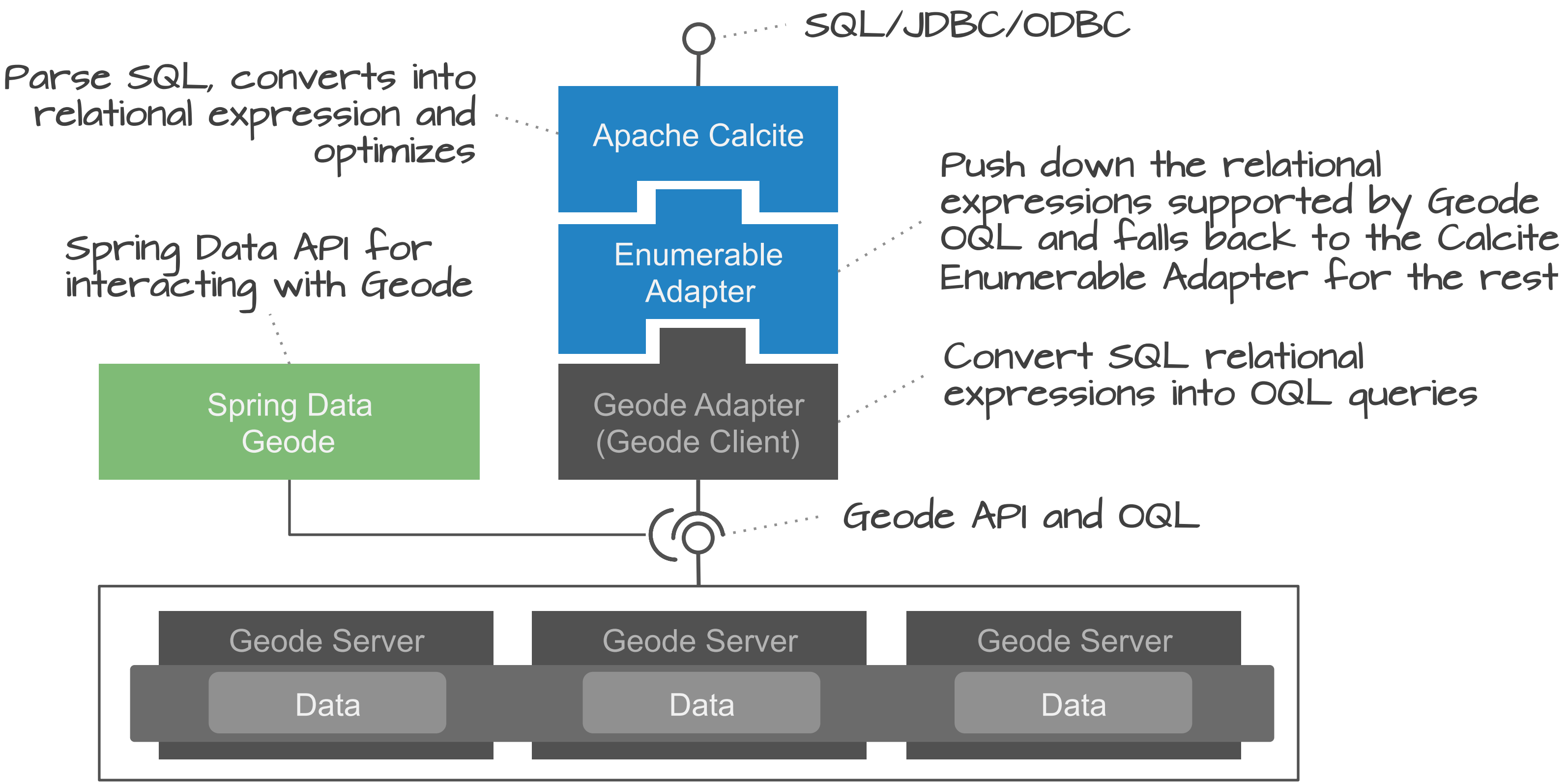
SQL?

- Apache Apex
- Apache Drill
- Apache Flink
- Apache Hive
- Apache Kylin
- Apache Phoenix
- Apache Samza
- Apache Storm
- Cascading
- Qubole Quark
- SQL-Gremlin

…

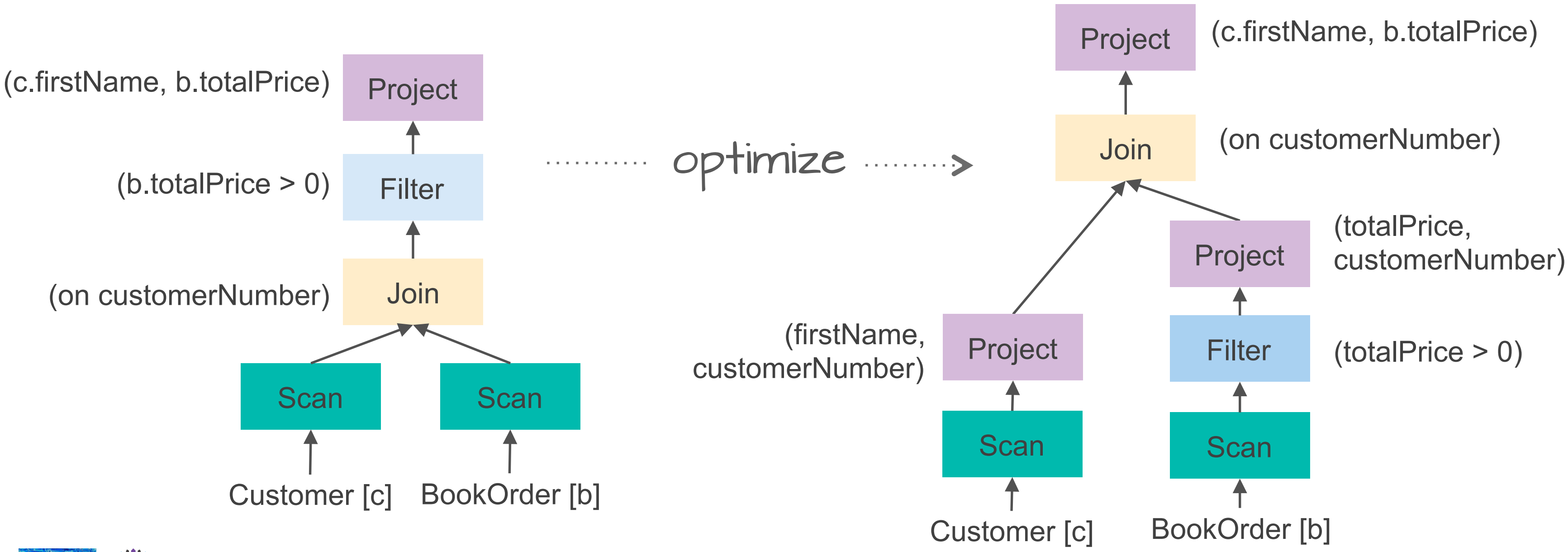
- **Apache Geode**

Geode Adapter - Overview


SQL/JDBC/ODBC
Parse SQL, converts into relational expression and optimizes
Apache Calcite
Enumerable Adapter
Push down the relational expressions supported by Geode OQL and falls back to the Calcite Enumerable Adapter for the rest
Spring Data API for interacting with Geode
Spring Data Geode
Geode Adapter (Geode Client)
Convert SQL relational expressions into OQL queries
Geode API and OQL
Geode Server
Data
Geode Server
Data
Geode Server
Data

SQL Relational Expressions

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



Geode Push Down Candidates

Relational Operator	Geode Support
LIMIT	YES (without OFFSET)
PROJECT	YES
FILTER	YES
JOIN	For collocated Regions only
AGGREGATE	YES for GROUP BY, DISTINCT, MAX, MIN, SUM, AVG, COUNT http://bit.ly/2eKApd0
SORT	YES

Apache Geode?

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

https://pivotal.io/big-data/case-study/distributed-in-memory-data-management-solution
https://pivotal.io/big-data/case-study/scaling-online-sales-for-the-largest-railway-in-the-world-china-railway-corporation

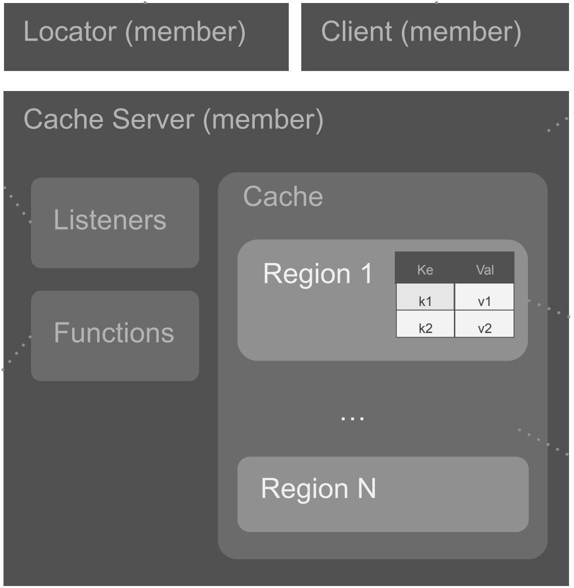
Apache Geode Features

- 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



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

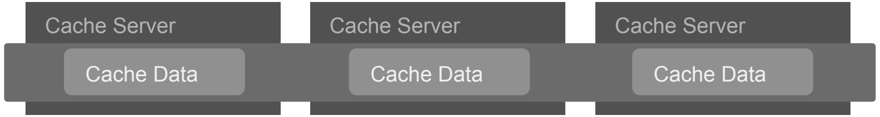
Functions - distributed, concurrent data processing

CacheServer - process connected to the distributed system with created **Cache**

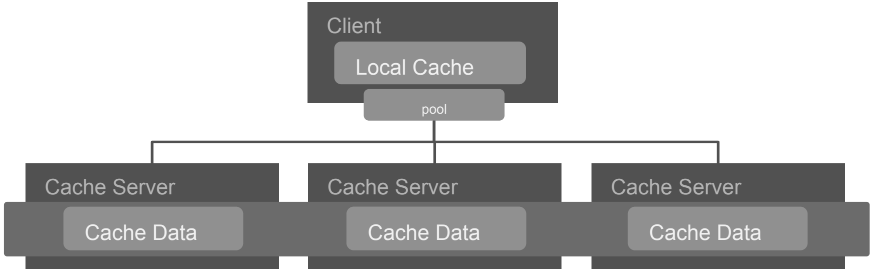
Region - consistent, **distributed Map** (key-value), Partitioned or Replicated

Cache - In-memory collection of **Regions**

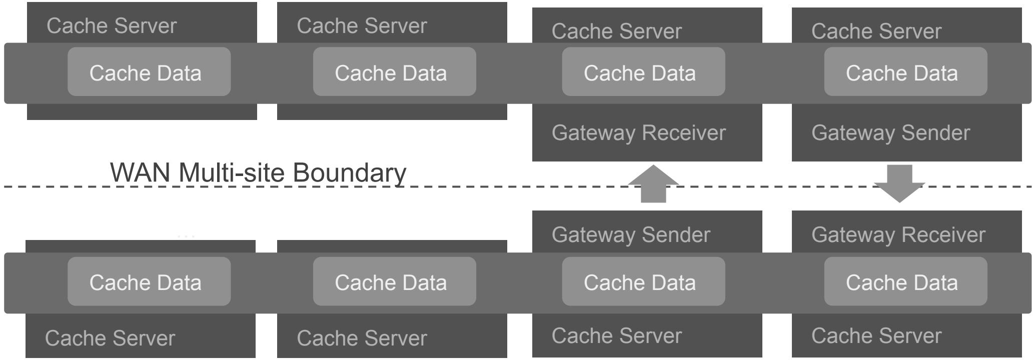
Geode Topology



Peer-to-Peer



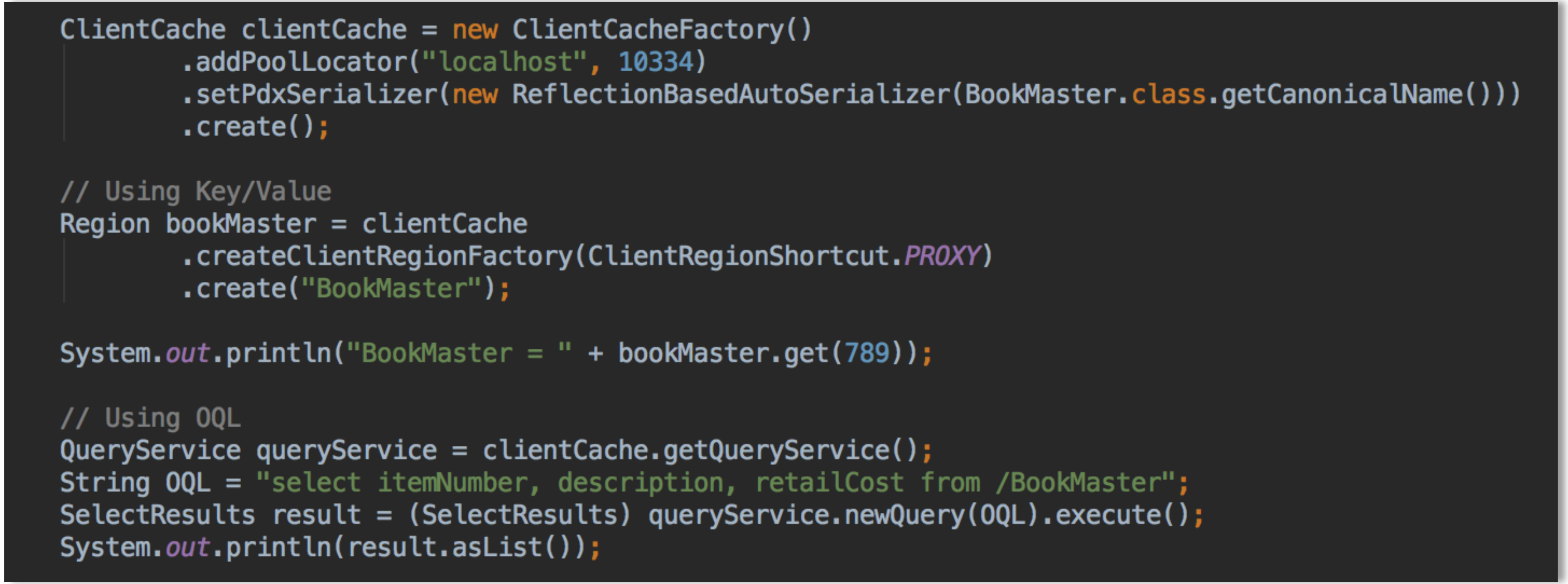
Client-Server



Multi-Site

Geode Client API

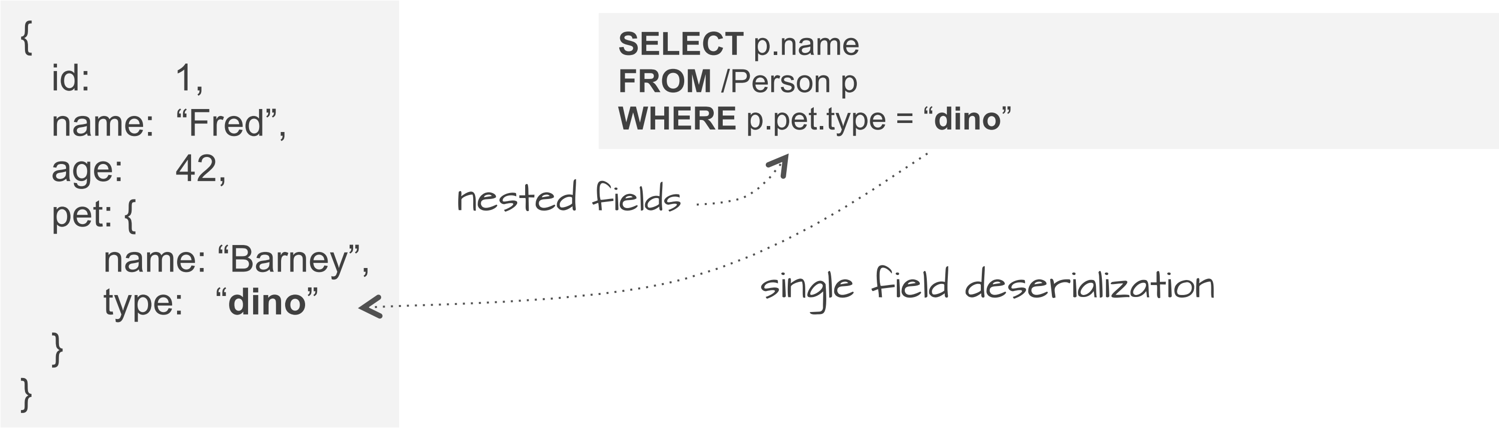
- Client Cache
- Key / Value - Region GET, PUT, REMOVE
- OQL – QueryService



```java
ClientCache clientCache = new ClientCacheFactory()
        .addPoolLocator("localhost", 10334)
        .setPdxSerializer(new ReflectionBasedAutoSerializer(BookMaster.class.getCanonicalName()))
        .create();

// Using Key/Value
Region bookMaster = clientCache
        .createClientRegionFactory(ClientRegionShortcut.PROXY)
        .create("BookMaster");

System.out.println("BookMaster = " + bookMaster.get(789));

// Using OQL
QueryService queryService = clientCache.getQueryService();
String OQL = "select itemNumber, description, retailCost from /BookMaster";
SelectResults result = (SelectResults) queryService.newQuery(OQL).execute();
System.out.println(result.asList());
```

Geode Data Types & Serialization

- Key-Value with complex value formats
- **P**ortable **D**ata e**X**change (PDX) Serialization – Delta propagation, schema evolution, polyglot support …
- **O**bject **Q**uery **L**anguage (OQL)



```
{
   id:       1,
   name:  "Fred",
   age:     42,
   pet: {
        name: "Barney",
        type:   "dino"
   }
}
```

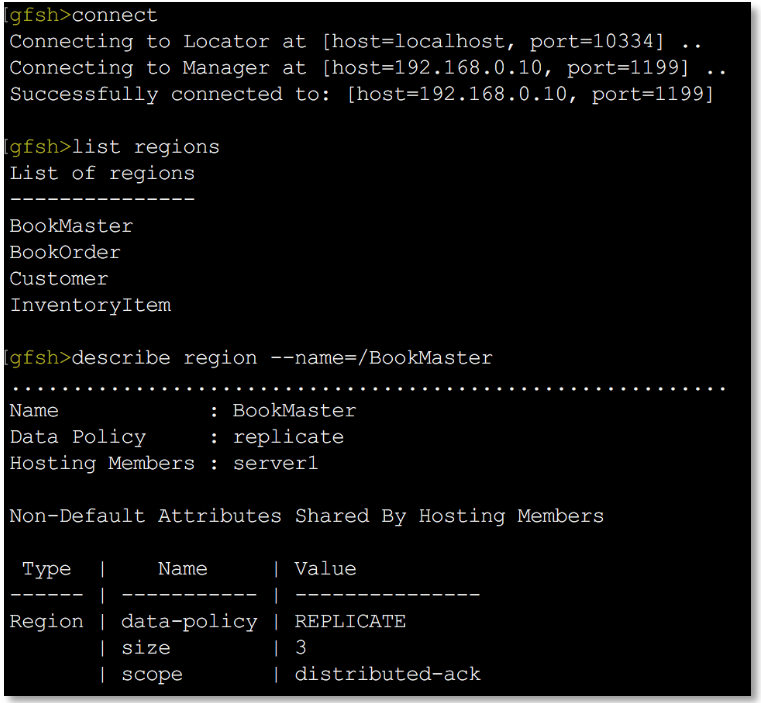
```
SELECT p.name
FROM /Person p
WHERE p.pet.type = "dino"
```

nested fields

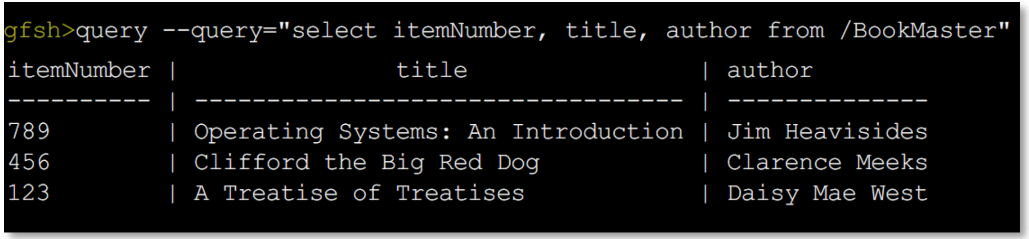
single field deserialization

Geode Demo (GFSH and OQL)

- Connect to Geode cluster,
- List available Regions
- Run OQL query



```
gfsh>connect
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=192.168.0.10, port=1199] ..
Successfully connected to: [host=192.168.0.10, port=1199]

gfsh>list regions
List of regions
---------------
BookMaster
BookOrder
Customer
InventoryItem

gfsh>describe region --name=/BookMaster
..........................................................
Name            : BookMaster
Data Policy     : replicate
Hosting Members : server1

Non-Default Attributes Shared By Hosting Members

 Type  |    Name     | Value
------ | ----------- | ---------------
Region | data-policy | REPLICATE
       | size        | 3
       | scope       | distributed-ack
```



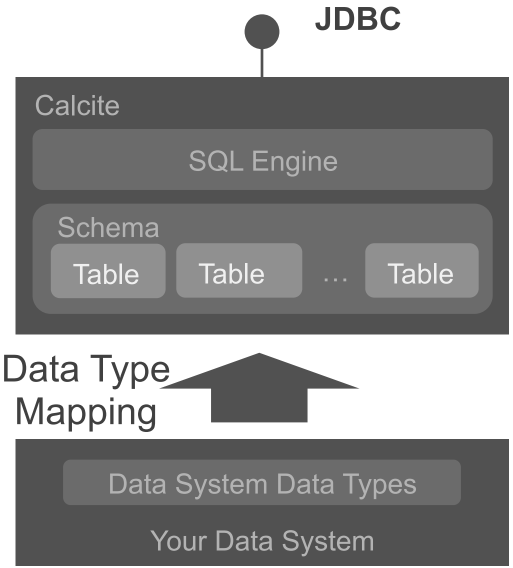
```
gfsh>query --query="select itemNumber, title, author from /BookMaster"
itemNumber |                title                 | author
---------- | ------------------------------------ | ---------------
789        | Operating Systems: An Introduction | Jim Heavisides
456        | Clifford the Big Red Dog           | Clarence Meeks
123        | A Treatise of Treatises            | Daisy Mae West
```

Apache Calcite?

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

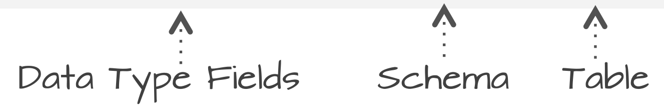
- Query Parser, Validator and Optimizer(s)
- JDBC drivers - local and remote
- SQL Streaming
- Agnostic to data storage and processing
- SQL completes vs. NoSQL integrity

Calcite Data Types



- 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

SELECT title, author **FROM** test.BookMaster



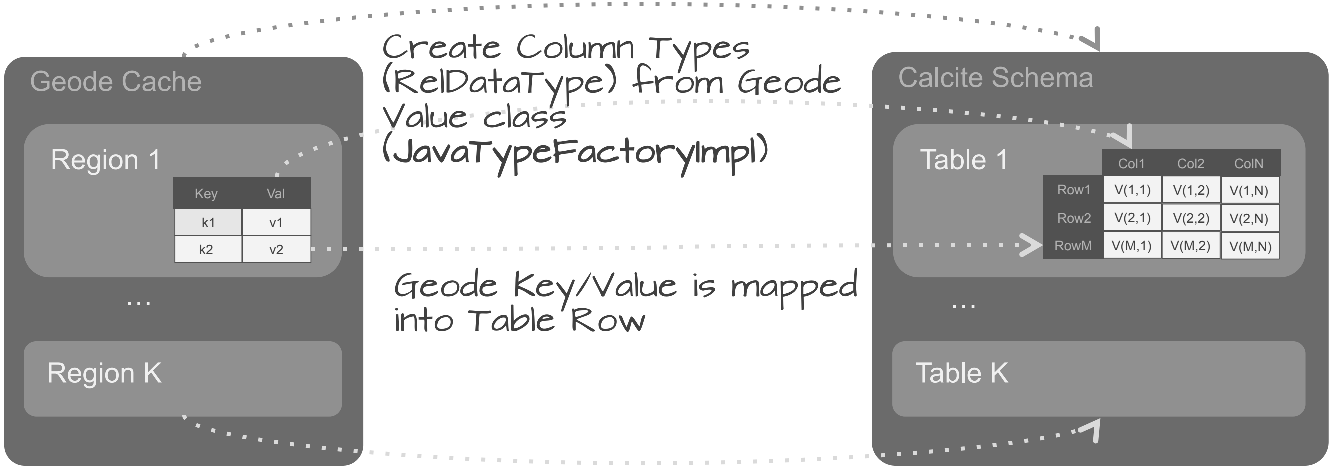
Calcite Data Types: RelDataType

Type of a scalar expression or row

- RelDataTypeFactory – RelDataType factory
- JavaTypeFactory - registers Java classes as record types
- **JavaTypeFactoryImpl** - Java Reflection to build RelDataTypes
- SqlTypeFactoryImpl - Default implementation with all SQL types

Geode to Calcite Data Types Mapping

Geode Cache is mapped into Calcite Schema



Key	Val
k1	v1
k2	v2

	Col1	Col2	ColN
Row1	V(1,1)	V(1,2)	V(1,N)
Row2	V(2,1)	V(2,2)	V(2,N)
RowM	V(M,1)	V(M,2)	V(M,N)

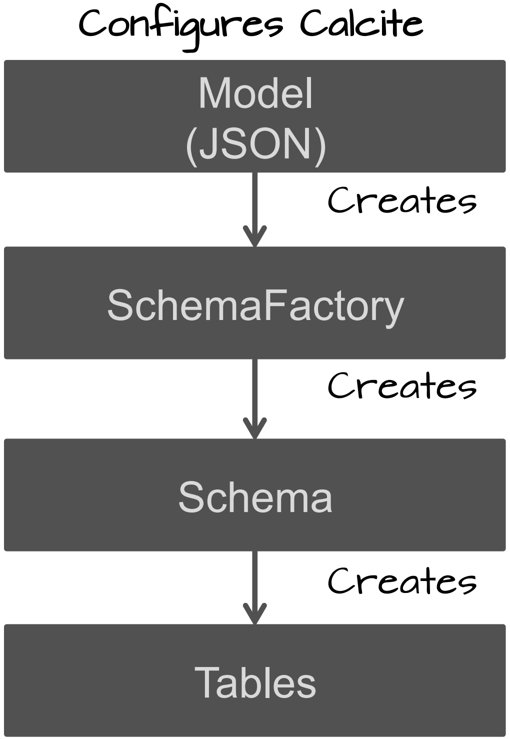
Create Column Types (RelDataType) from Geode Value class (JavaTypeFactoryImpl)

Geode Key/Value is mapped into Table Row

Regions are mapped into Tables

Calcite Bootstrap Flow

Typical calcite initialization flow



Configures Calcite

Model
(JSON)

Creates

SchemaFactory

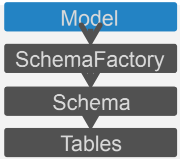
Creates

Schema

Creates

Tables

Calcite Model



The path to <my-model>.json is passed as JDBC connection argument:

```
!connect jdbc:calcite:model=target/test-classes/<my-model-path>.json
```

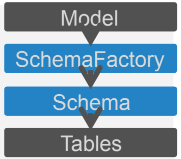
```
{
  version: '1.0',
  defaultSchema: 'TEST',
  schemas: [ {
     name: 'TEST',
     type: 'custom',

     factory: 'org.apache.calcite.adapter.geode.simple.GeodeSchemaFactory',

     operand: {
        locatorHost: 'localhost',
        locatorPort: '10334',
        regions: 'BookMaster',
        pdxSerializablePackagePath: 'net.tzolov.geode.bookstore.domain.*'
     }
   }]
}
```

Schema Name

Reference to your adapter schema factory implementation class

Parameters to be passed to your adapter schema factory implementation

Geode Calcite Schema and Schema Factory



```
public class GeodeSchemaFactory implements SchemaFactory {

  public Schema create(SchemaPlus parentSchema, String schemaName, Map<String,  Object> operand) {
    String locatorHost = (String) operand.get(“locatorHost”);
    int locatorPort = …
    String[] regionNames = …
    String pdxPackagePath = …

    return new GeodeSchema(locatorHost, locatorPort, regionNames, pdxPackagePath);
  }
}
```

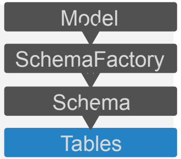
Retrieves the parameters set in the model.json

Create an Adapter Schema instance with the provided parameters.

```
public class GeodeSchema extends AbstractSchema {
   private String regionName = ..

   protected Map<String, Table> getTableMap() {
       final ImmutableMap.Builder<String, Table> builder = ImmutableMap.builder();
      Region region = … Get Geode Region by region name …
      Class valueClass= … Find region’s value type …

      builder.put(regionName, new GeodeScannableTable(regionName, valueClass, clientCache));
      return tableMap;
}
```

Create GeodeScannableTable instance for each Geode Region

Geode Scannable Table



```
public class GeodeScannableTable extends AbstractTable implements ScannableTable {
  public RelDataType getRowType(RelDataTypeFactory typeFactory) {
     return new JavaTypeFactoryImpl().createStructType(valueClass);
  }

  public Enumerable<Object[]> scan(DataContext root) {
    return new AbstractEnumerable<Object[]>() {
     public Enumerator<Object[]> enumerator() { return new GeodeEnumerator<Object[]>(clientCache, regionName); }
    }
```

Uses reflection (or pdx-instance) to builds RelDataType from value's class type

Returns an Enumeration over the entire target data store

```
public class GeodeEnumerator<E> implements Enumerator<E> {
  private E current;
  private SelectResults geodeIterator;
  public GeodeEnumerator(ClientCache clientCache, String regionName) {
     geodeterator = clientCache.getQueryService().newQuery("select * from /" + regionName).execute().iterator();
  }
  public boolean moveNext() { current = convert(geodeIterator.next()); return true;}
  public E current() {return current;}

  public abstract E convert(Object geodeValue) {
      Convert PDX value into  RelDataType row
  }
```

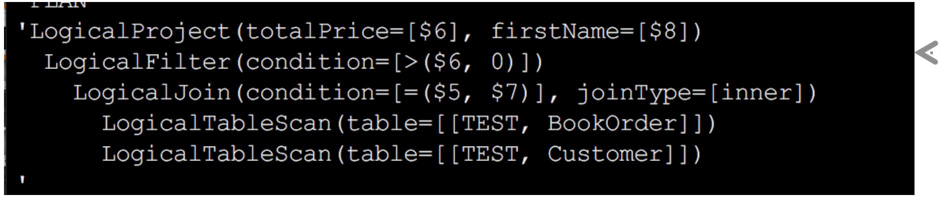
Defined in the Linq4j sub-project

Retrieves the entire Region!!

Converts Geode value response into Calcite row data

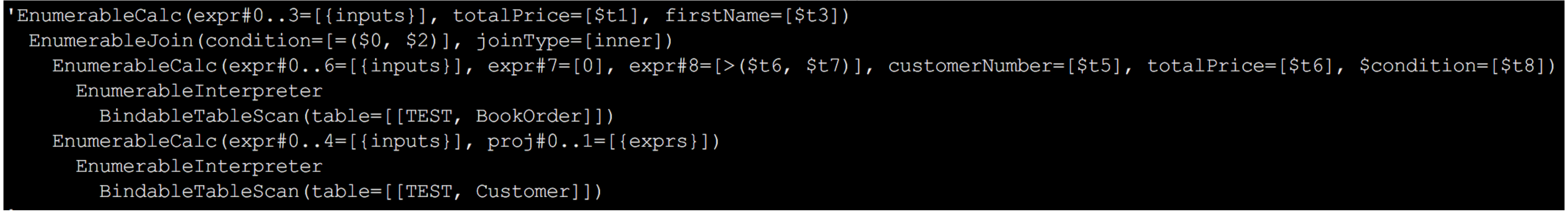
Geode Demo (Scannable Tables)

```
$ ./sqlline
sqlline> !connect jdbc:calcite:model=target/test-classes/model2.json admin admin

jdbc:calcite> !tables
jdbc:calcite> SELECT * FROM "BookMaster”;
jdbc:calcite> SELECT "yearPublished", AVG("retailCost") AS “AvgRetailCost” FROM "BookMaster" GROUP BY "yearPublished";
jdbc:calcite> 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]])
'
```

Without and With Implementation



```
'EnumerableCalc(expr#0..3=[{inputs}], totalPrice=[$t1], firstName=[$t3])
  EnumerableJoin(condition=[=($0, $2)], joinType=[inner])
    EnumerableCalc(expr#0..6=[{inputs}], expr#7=[0], expr#8=[>($t6, $t7)], customerNumber=[$t5], totalPrice=[$t6], $condition=[$t8])
      EnumerableInterpreter
        BindableTableScan(table=[[TEST, BookOrder]])
    EnumerableCalc(expr#0..4=[{inputs}], proj#0..1=[{exprs}])
      EnumerableInterpreter
        BindableTableScan(table=[[TEST, Customer]])
```

Non-Relational Tables

Scanned without intermediate relational expression.

- **ScannableTable** - can be scanned

```
Enumerable<Object[]> scan(DataContext root);
```

- **FilterableTable** - can be scanned, applying supplied filter expressions

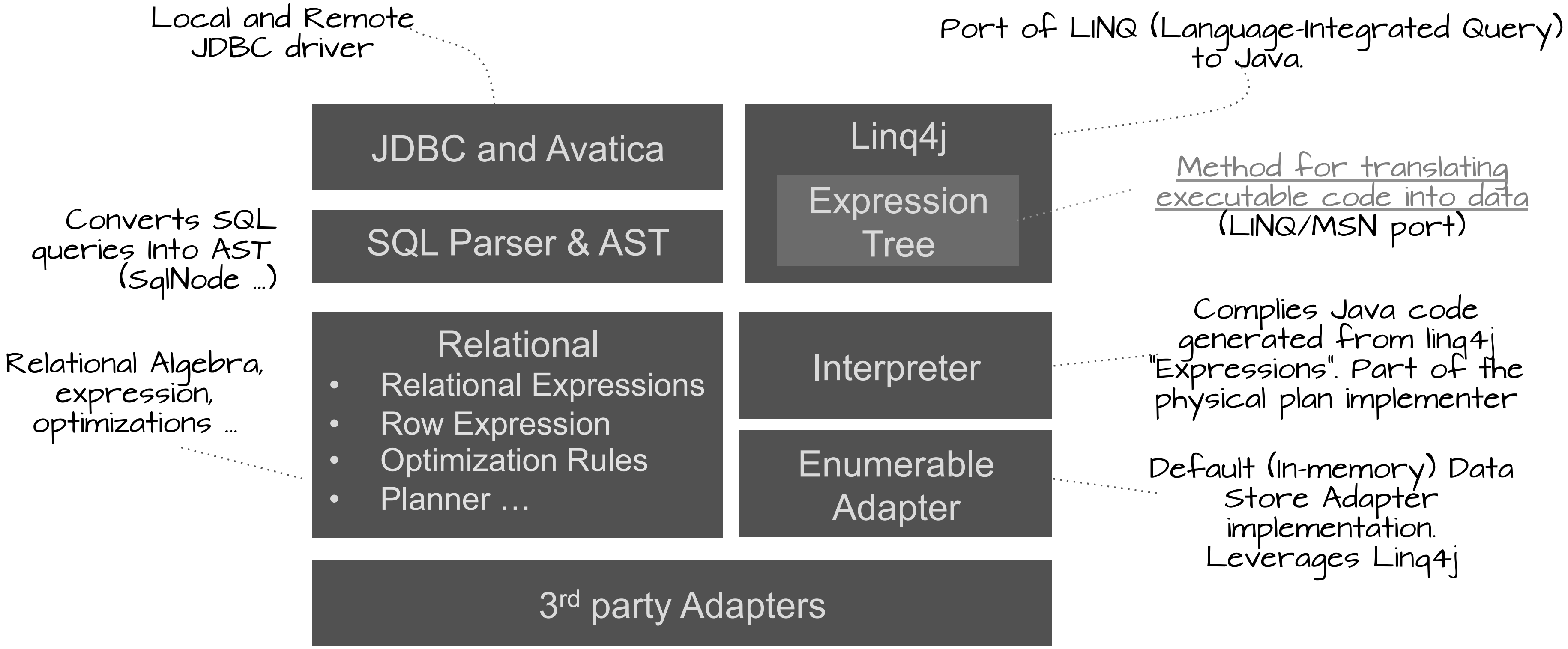
```
Enumerable<Object[]> scan(DataContext root, List<RexNode> filters);
```

- **ProjectableFilterableTable** - can be scanned, applying supplied filter expressions and projecting a given list of columns

```
Enumerable<Object[]> scan(DataContext root, List<RexNode> filters, int[] projects);
```

Calcite Ecosystem

Several “semi-independent” projects.



Local and Remote JDBC driver

JDBC and Avatica

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

Linq4j

Expression Tree

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

Converts SQL queries Into AST (SqlNode …)

SQL Parser & AST

Relational Algebra, expression, optimizations …

Relational
- Relational Expressions
- Row Expression
- Optimization Rules
- Planner …

Interpreter

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

Enumerable Adapter

Default (In-memory) Data Store Adapter implementation. Leverages Linq4j

3rd party Adapters

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

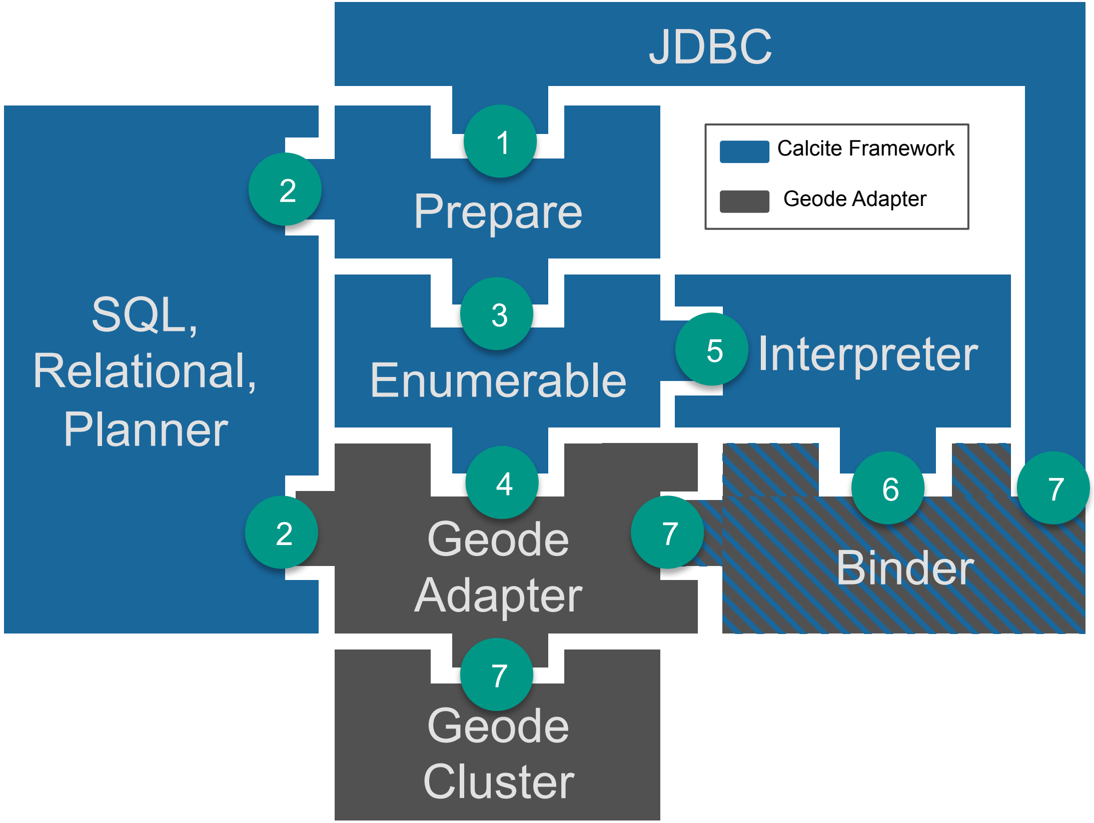
3. Start building a physical plan from the relation expressions

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

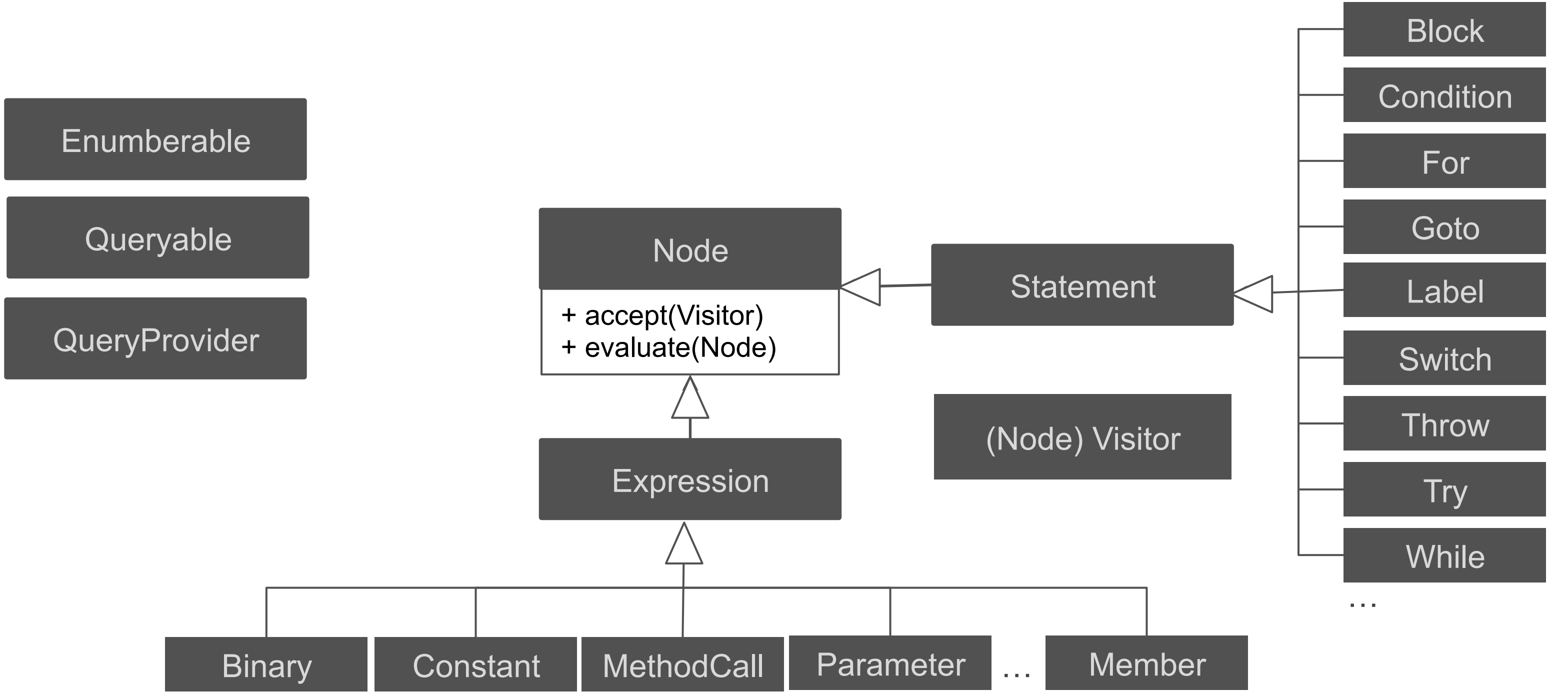
5. Pass the Expression tree to the Interpreter to generate Java code

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

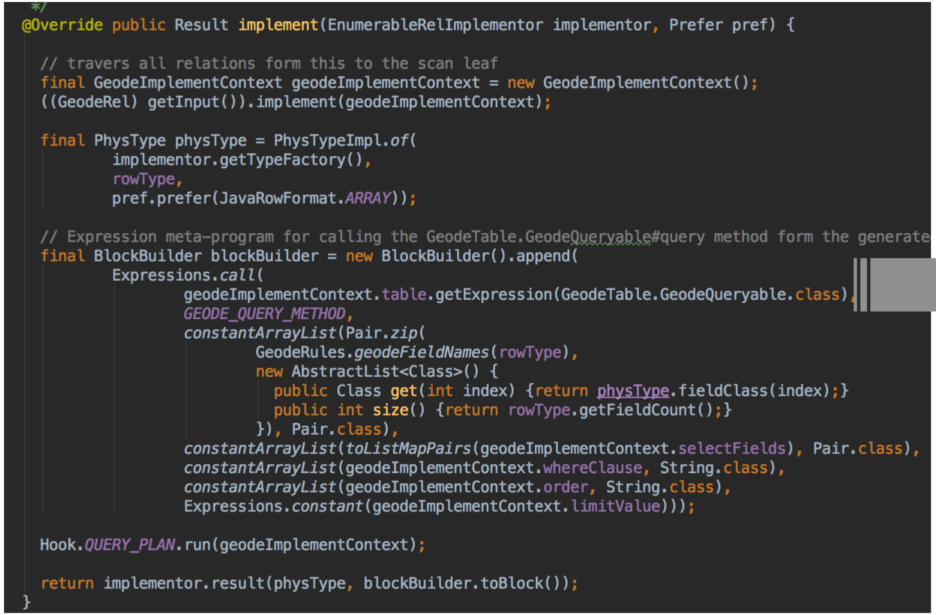


Linq4j and Expression Tree

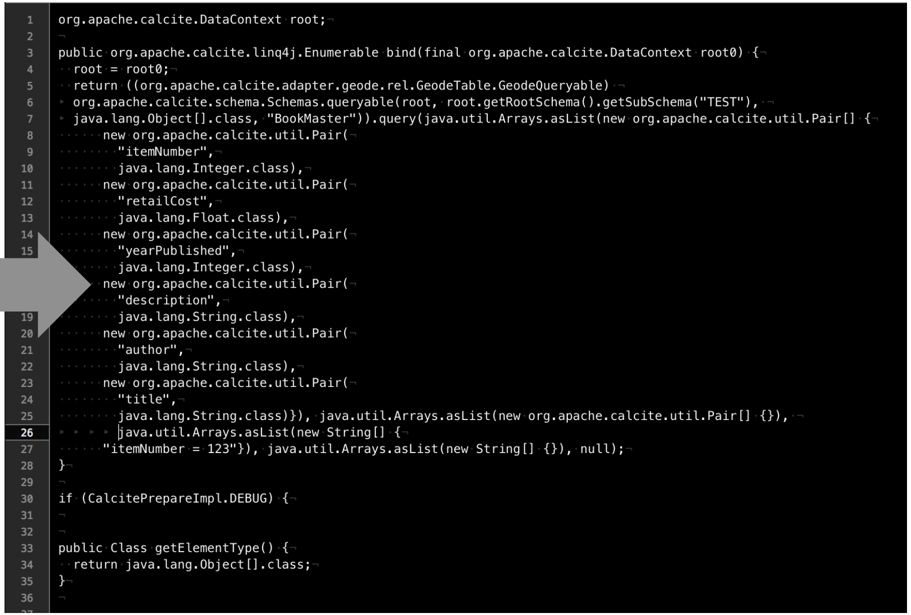


Bindable Generated Code

Calcite via Enumerable Converts Expressions into Java Code

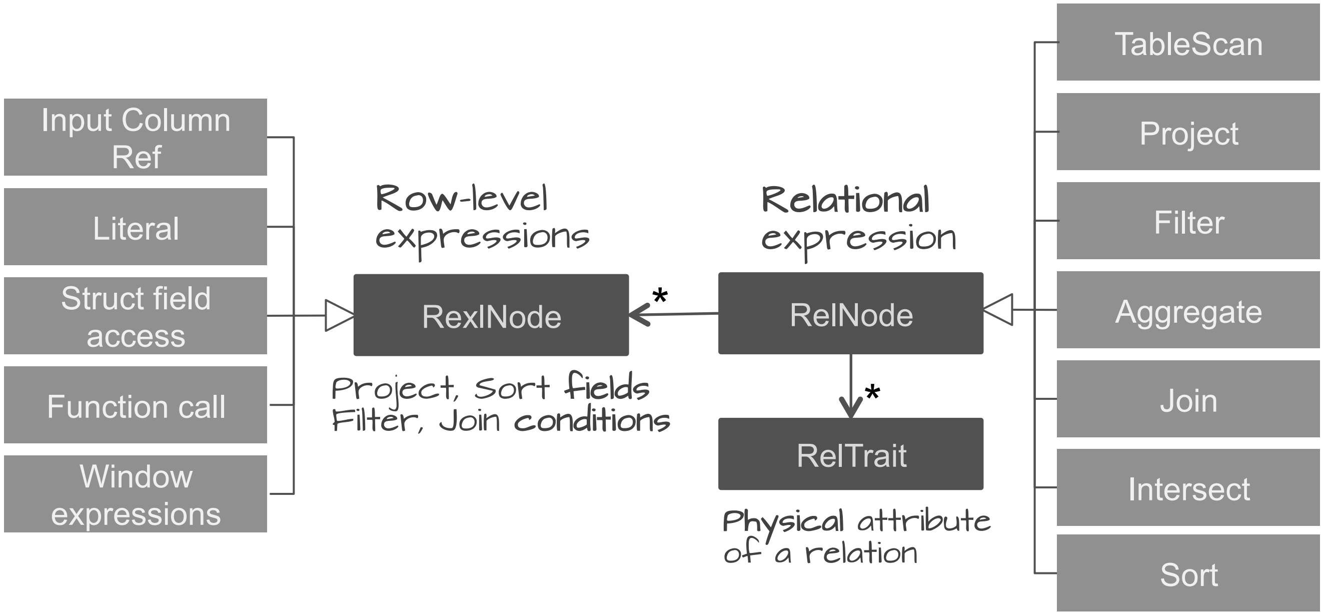


```java
  */
@Override public Result implement(EnumerableRelImplementor implementor, Prefer pref) {

  // travers all relations form this to the scan leaf
  final GeodeImplementContext geodeImplementContext = new GeodeImplementContext();
  ((GeodeRel) getInput()).implement(geodeImplementContext);

  final PhysType physType = PhysTypeImpl.of(
          implementor.getTypeFactory(),
          rowType,
          pref.prefer(JavaRowFormat.ARRAY));

  // Expression meta-program for calling the GeodeTable.GeodeQueryable#query method form the generate
  final BlockBuilder blockBuilder = new BlockBuilder().append(
          Expressions.call(
                  geodeImplementContext.table.getExpression(GeodeTable.GeodeQueryable.class),
                  GEODE_QUERY_METHOD,
                  constantArrayList(Pair.zip(
                          GeodeRules.geodeFieldNames(rowType),
                          new AbstractList<Class>() {
                            public Class get(int index) {return physType.fieldClass(index);}
                            public int size() {return rowType.getFieldCount();}
                          }), Pair.class),
                  constantArrayList(toListMapPairs(geodeImplementContext.selectFields), Pair.class),
                  constantArrayList(geodeImplementContext.whereClause, String.class),
                  constantArrayList(geodeImplementContext.order, String.class),
                  Expressions.constant(geodeImplementContext.limitValue)));

  Hook.QUERY_PLAN.run(geodeImplementContext);

  return implementor.result(physType, blockBuilder.toBlock());
}
```

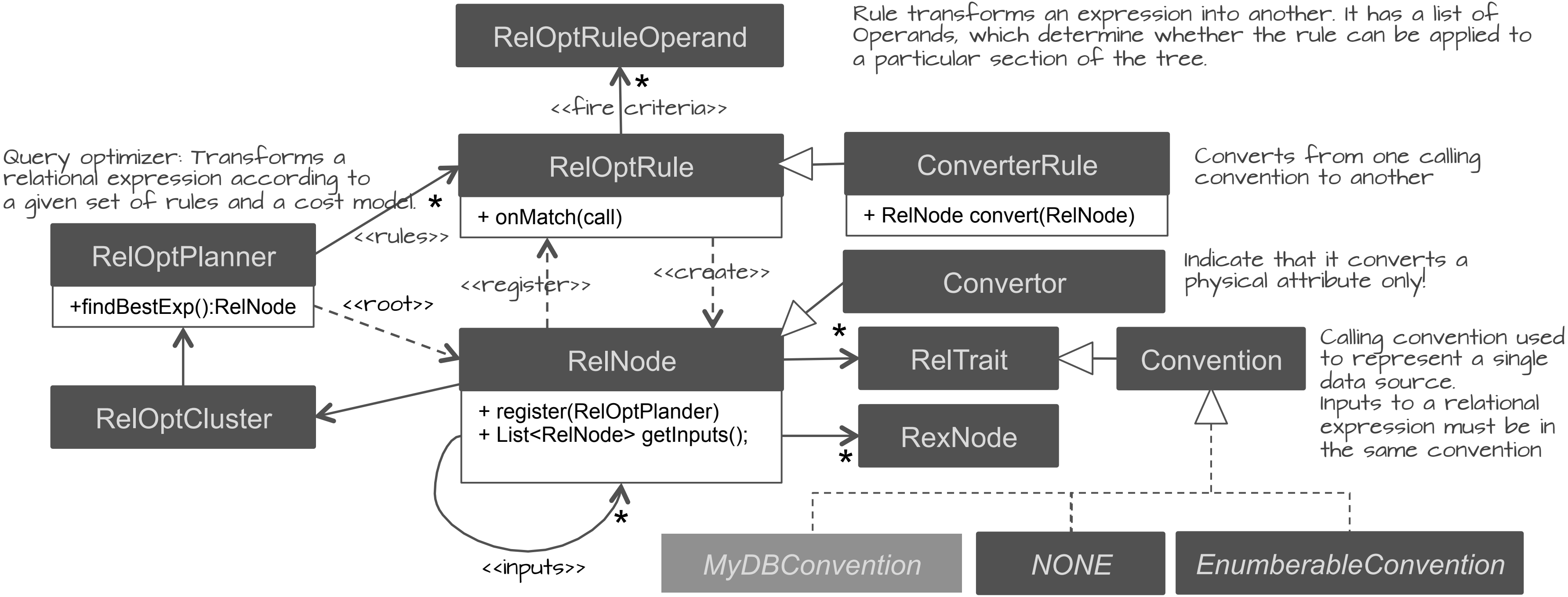


```java
org.apache.calcite.DataContext root;

public org.apache.calcite.linq4j.Enumerable bind(final org.apache.calcite.DataContext root0) {
  root = root0;
  return ((org.apache.calcite.adapter.geode.rel.GeodeTable.GeodeQueryable)
  org.apache.calcite.schema.Schemas.queryable(root, root.getRootSchema().getSubSchema("TEST"),
  java.lang.Object[].class, "BookMaster")).query(java.util.Arrays.asList(new org.apache.calcite.util.Pair[] {
      new org.apache.calcite.util.Pair(
        "itemNumber",
        java.lang.Integer.class),
      new org.apache.calcite.util.Pair(
        "retailCost",
        java.lang.Float.class),
      new org.apache.calcite.util.Pair(
        "yearPublished",
        java.lang.Integer.class),
      new org.apache.calcite.util.Pair(
        "description",
        java.lang.String.class),
      new org.apache.calcite.util.Pair(
        "author",
        java.lang.String.class),
      new org.apache.calcite.util.Pair(
        "title",
        java.lang.String.class)}), java.util.Arrays.asList(new org.apache.calcite.util.Pair[] {}),
        java.util.Arrays.asList(new String[] {
      "itemNumber = 123"}), java.util.Arrays.asList(new String[] {}), null);
}

if (CalcitePrepareImpl.DEBUG) {


public Class getElementType() {
  return java.lang.Object[].class;
}

```

Calcite Relational Expressions



Calcite Relational Expressions (2)



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.

Query optimizer: Transforms a relational expression according to a given set of rules and a cost model.

Converts from one calling convention to another

Indicate that it converts a physical attribute only!

Calling convention used to represent a single data source.
Inputs to a relational expression must be in the same convention

RelOptRuleOperand

RelOptRule
+ onMatch(call)

ConverterRule
+ RelNode convert(RelNode)

Convertor

RelOptPlanner
+findBestExp():RelNode

RelOptCluster

RelNode
+ register(RelOptPlander)
+ List<RelNode> getInputs();

RelTrait

Convention

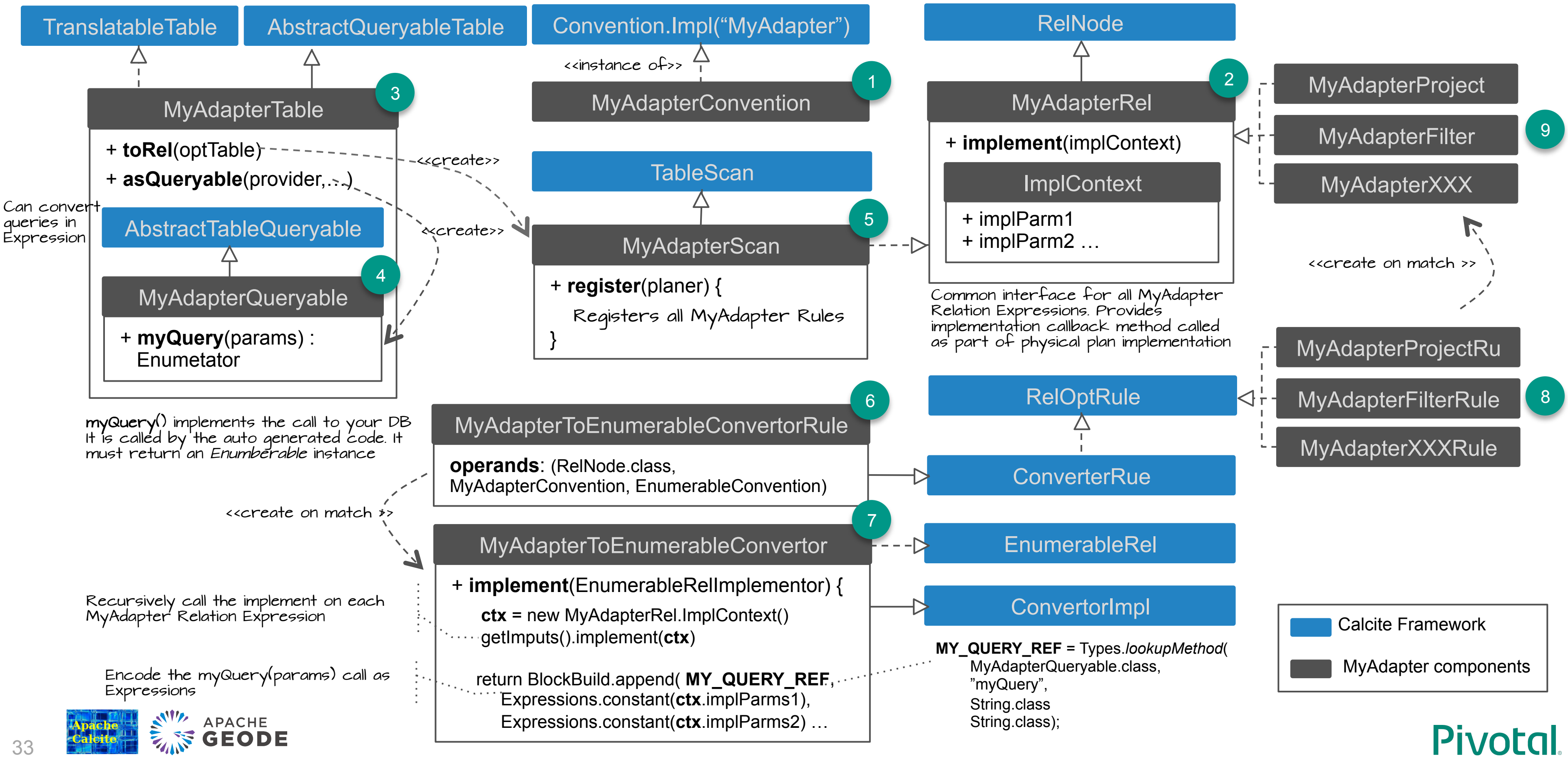
RexNode

MyDBConvention

NONE

EnumberableConvention

Calcite Adapter Patterns



TranslatableTable

AbstractQueryableTable

Convention.Impl("MyAdapter")

RelNode

<<instance of>>

3 MyAdapterTable

+ **toRel**(optTable)

+ **asQueryable**(provider,…)

<<create>>

<<create>>

Can convert queries in Expression

AbstractTableQueryable

4 MyAdapterQueryable

+ **myQuery**(params) : Enumetator

1 MyAdapterConvention

TableScan

5 MyAdapterScan

```
+ register(planer) {
   Registers all MyAdapter Rules
}
```

2 MyAdapterRel

+ **implement**(implContext)

ImplContext

+ implParm1

+ implParm2 …

Common interface for all MyAdapter Relation Expressions. Provides implementation callback method called as part of physical plan implementation

MyAdapterProject

9 MyAdapterFilter

MyAdapterXXX

<<create on match >>

MyAdapterProjectRu

RelOptRule

8 MyAdapterFilterRule

MyAdapterXXXRule

myQuery() implements the call to your DB It is called by the auto generated code. It must return an *Enumberable* instance

6 MyAdapterToEnumerableConvertorRule

operands: (RelNode.class, MyAdapterConvention, EnumerableConvention)

ConverterRue

<<create on match >>

7 MyAdapterToEnumerableConvertor

EnumerableRel

ConvertorImpl

Recursively call the implement on each MyAdapter Relation Expression

Encode the myQuery(params) call as Expressions

```
+ implement(EnumerableRelImplementor) {
   ctx = new MyAdapterRel.ImplContext()
   getImputs().implement(ctx)

  return BlockBuild.append( MY_QUERY_REF,
     Expressions.constant(ctx.implParms1),
     Expressions.constant(ctx.implParms2) …
```

```
MY_QUERY_REF = Types.lookupMethod(
     MyAdapterQueryable.class,
     "myQuery",
     String.class
     String.class);
```

🟦 Calcite Framework

⬛ MyAdapter components

Apache Calcite

APACHE GEODE

Pivotal

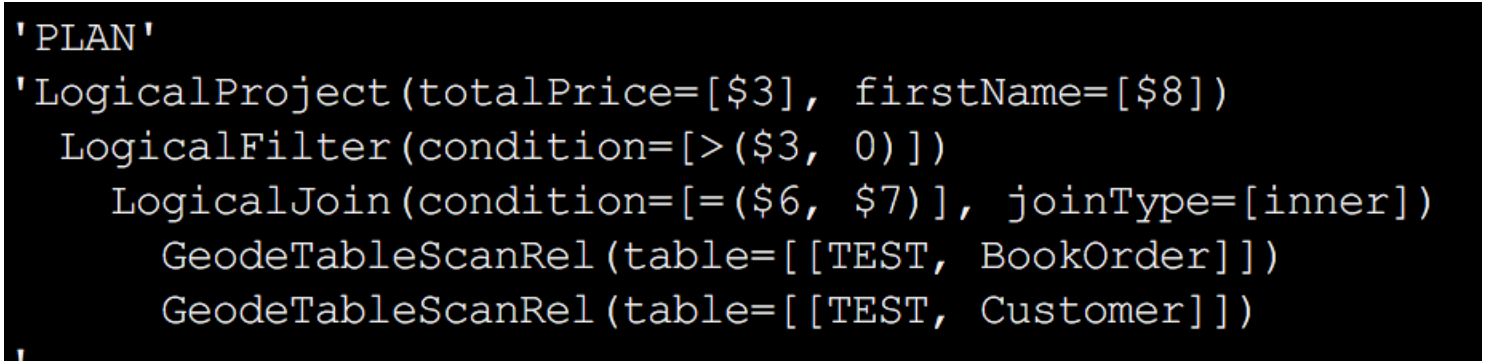
Calcite with Geode - Without Implementation

SELECT b."totalPrice", c."firstName"

FROM "BookOrder" as b

INNER JOIN "Customer" as c **ON** b."customerNumber" = c."customerNumber"

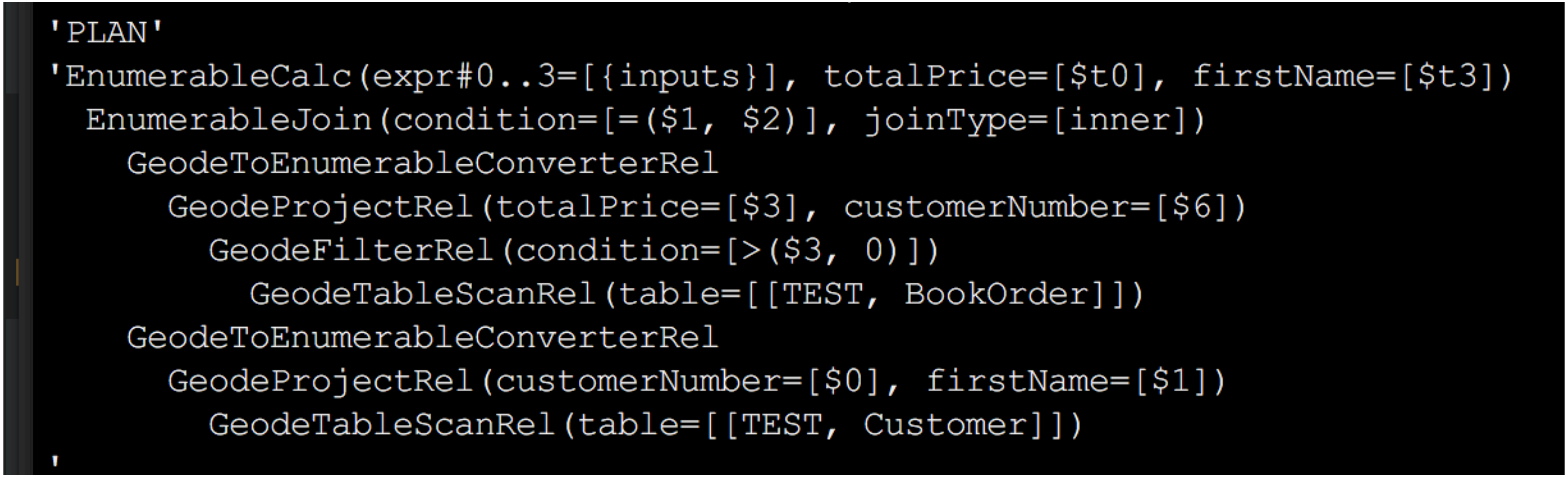
WHERE b."totalPrice" > 0;



```
'PLAN'
'LogicalProject(totalPrice=[$3], firstName=[$8])
  LogicalFilter(condition=[>($3, 0)])
    LogicalJoin(condition=[=($6, $7)], joinType=[inner])
      GeodeTableScanRel(table=[[TEST, BookOrder]])
      GeodeTableScanRel(table=[[TEST, Customer]])
'
```

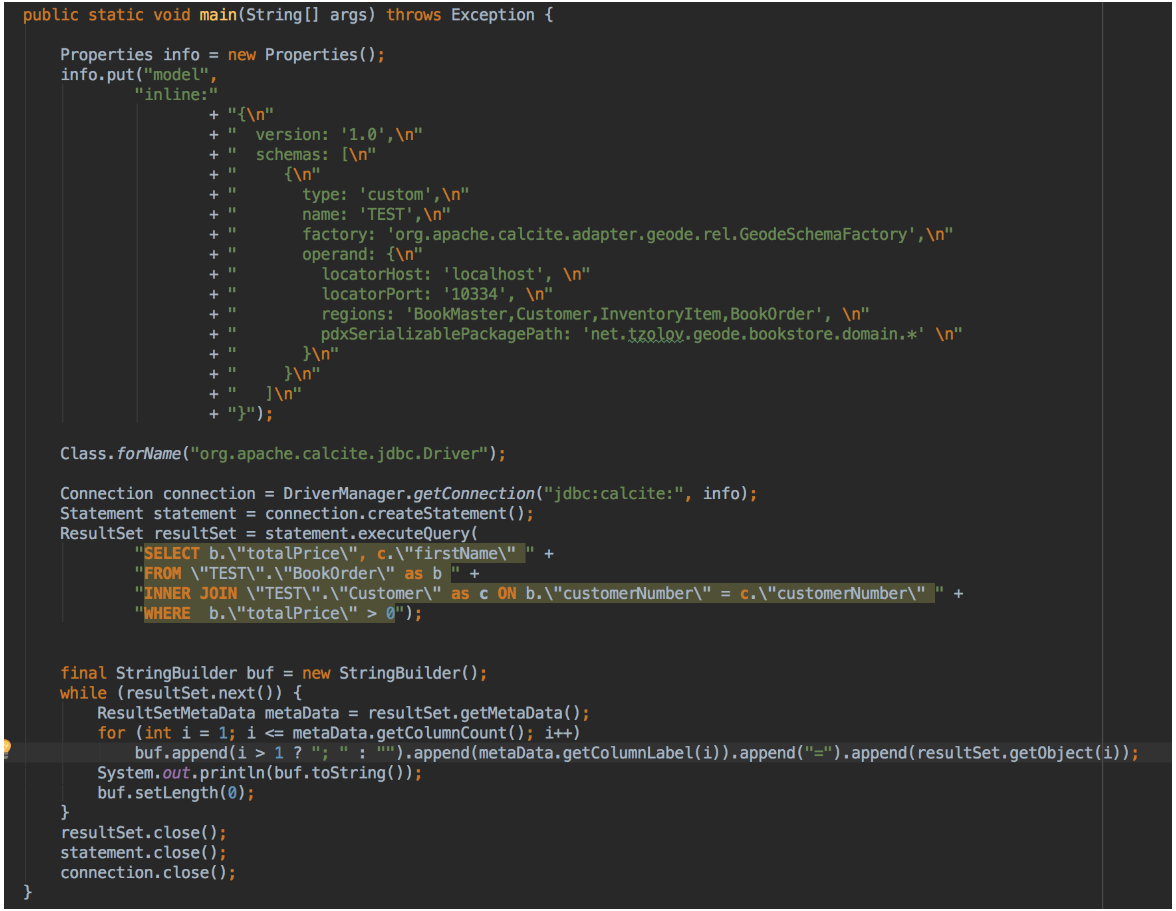
Calcite with Geode - With Implementation

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



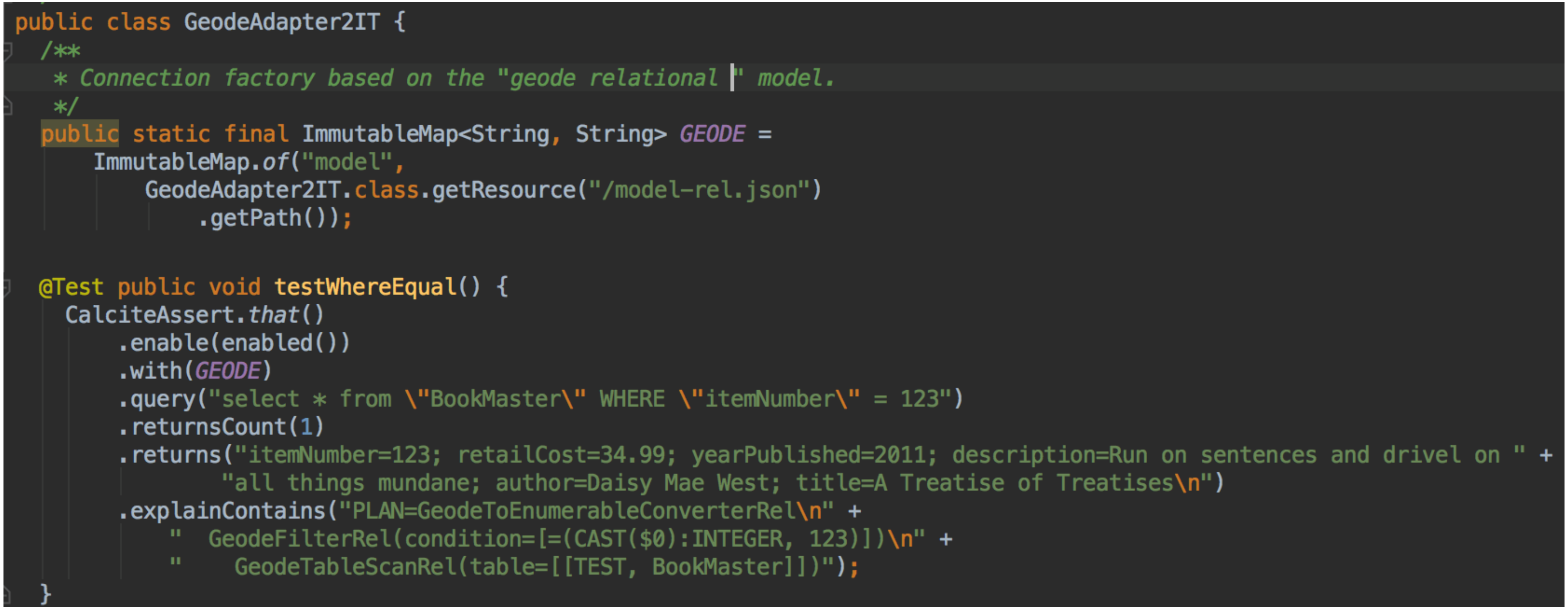
```
'PLAN'
'EnumerableCalc(expr#0..3=[{inputs}], totalPrice=[$t0], firstName=[$t3])
  EnumerableJoin(condition=[=($1, $2)], joinType=[inner])
    GeodeToEnumerableConverterRel
      GeodeProjectRel(totalPrice=[$3], customerNumber=[$6])
        GeodeFilterRel(condition=[>($3, 0)])
          GeodeTableScanRel(table=[[TEST,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

Calcite JDBC Connection



```java
public static void main(String[] args) throws Exception {

    Properties info = new Properties();
    info.put("model",
            "inline:"
                    + "{\n"
                    + "  version: '1.0',\n"
                    + "  schemas: [\n"
                    + "     {\n"
                    + "       type: 'custom',\n"
                    + "       name: 'TEST',\n"
                    + "       factory: 'org.apache.calcite.adapter.geode.rel.GeodeSchemaFactory',\n"
                    + "       operand: {\n"
                    + "         locatorHost: 'localhost', \n"
                    + "         locatorPort: '10334', \n"
                    + "         regions: 'BookMaster,Customer,InventoryItem,BookOrder', \n"
                    + "         pdxSerializablePackagePath: 'net.tzolov.geode.bookstore.domain.*' \n"
                    + "       }\n"
                    + "     }\n"
                    + "   ]\n"
                    + "}");

    Class.forName("org.apache.calcite.jdbc.Driver");

    Connection connection = DriverManager.getConnection("jdbc:calcite:", info);
    Statement statement = connection.createStatement();
    ResultSet resultSet = statement.executeQuery(
            "SELECT b.\"totalPrice\", c.\"firstName\" " +
            "FROM \"TEST\".\"BookOrder\" as b " +
            "INNER JOIN \"TEST\".\"Customer\" as c ON b.\"customerNumber\" = c.\"customerNumber\" " +
            "WHERE  b.\"totalPrice\" > 0");


    final StringBuilder buf = new StringBuilder();
    while (resultSet.next()) {
        ResultSetMetaData metaData = resultSet.getMetaData();
        for (int i = 1; i <= metaData.getColumnCount(); i++)
            buf.append(i > 1 ? "; " : "").append(metaData.getColumnLabel(i)).append("=").append(resultSet.getObject(i));
        System.out.println(buf.toString());
        buf.setLength(0);
    }
    resultSet.close();
    statement.close();
    connection.close();
}
```

What About Testing?



```
public class GeodeAdapter2IT {
  /**
   * Connection factory based on the "geode relational " model.
   */
  public static final ImmutableMap<String, String> GEODE =
      ImmutableMap.of("model",
          GeodeAdapter2IT.class.getResource("/model-rel.json")
              .getPath());

  @Test public void testWhereEqual() {
    CalciteAssert.that()
        .enable(enabled())
        .with(GEODE)
        .query("select * from \"BookMaster\" WHERE \"itemNumber\" = 123")
        .returnsCount(1)
        .returns("itemNumber=123; retailCost=34.99; yearPublished=2011; description=Run on sentences and drivel on " +
                "all things mundane; author=Daisy Mae West; title=A Treatise of Treatises\n")
        .explainContains("PLAN=GeodeToEnumerableConverterRel\n" +
            "  GeodeFilterRel(condition=[=(CAST($0):INTEGER, 123)])\n" +
            "    GeodeTableScanRel(table=[[TEST, BookMaster]])");
  }
```

TODO

- Improve nested data structures support
- Push down Join for collocated data sets
- Push down the COUNT expression
- Beyond OQL (e.g. implement Join, aggregations with custom functions)
- Leverage Calcite Streaming with Geode
- Transaction Support
- Table Statistics based on Region statistics
- Benchmarks and estimate the Calcite SQL overhead compared to pure OQL

References

- Apache Geode Adapter for Apache Calcite: https://github.com/tzolov/calcite
- Introduction to Apache Calcite (2016) : http://bit.ly/2fB1iBz
- Apache Calcite Overview (2014) : http://bit.ly/2fMJgbS
- Introduction to Apache Geode (2016) : http://bit.ly/1Rfztbd
- Apache Calcite Project (2016) : https://calcite.apache.org
- Apache Geode Project (2016) : http://geode.apache.org
- Geode Object Query Language (OQL) : http://bit.ly/2eKywgp
- Expression Tree Basic: http://bit.ly/2flBiXH

Credits

- To Dan Baskette for suggesting and motivating the project
- To Apache Geode and Apache Calcite for the inspiring projects and for the productive discussions
- To Pivotal for letting me work on projects like this

Pivotal®

Transforming How The World Builds Software