Apache AsterixDB

...

Eddie Coberly

Background

- Customer: Apache Software Foundation
  - Technical mentor: Dr. Preston Carman
- Description:
  - Fixed `interval_overlapping()` function for AsterixDB
    - Added functionality for dynamic hints
    - Implemented data deduplication
- Relevance
  - Didn’t work before my project

Goals at the beginning

- Project goals:
  - Complete last interval join function
  - Experiment with optimizing different parts of the code
- Criteria for success:
  - Working query
  - Pull request
  - Optimized query (stretch goal)

What was actually accomplished...

- Goals met:
  - Interval join function now deduplicates data before returning it
- Success criteria met:
  - Working query
  - Pull request (in progress)
    - Waiting for confirmation about specific commit process
    - Need affirmative code review rated +2
- Abandoned goals:
  - Optimized query
    - This was a stretch goal anyway

Schedule

	Projected	Actual
Winter	Have PR ready	Create framework for function
Spring	Optimize code Update documentation	Complete function Achieve passing tests PR (in progress)

Tools

- IntelliJ IDEA Java Editor (for coding)
- VMware Workstation 17 (for running AsterixDB on Windows)
  - Abandoned
  - Used Linux lab computer instead
- Gerrit (for submitting changes)
- Jenkins (for automated testing)
  - Not needed

Tools, cont'd

How IntelliJ helped:

- User-friendly
- Lots of keyboard shortcut tips
  - New one every time IDE was opened
- Easy for imports/dependencies
- Code formatting suggestions

Queries:
Static vs. Dynamic Hint

QUERY INPUT (5/5)

Default: TinyCollege | PLAN FORMAT: JSON | OUTPUT FORMAT: JSON | QUERY HISTORY: use TinyCollege; select f.name as staff, d.name as student from Staff as f, Students...

```
1 use TinyCollege;
2
3 select f.name as staff, d.name as student
4 from Staff as f, Students as d
5 where
6 /*+ range [date("1949-12-31"), date("1974-01-01"), date("2000-01-01"), date("2025-01-01")]*/
7 interval_overlapping(f.employment, d.attendance)
8 order by staff
```

SUCCESS: Execution time: 103.546268ms Elapsed time: 110.985102ms Size: 0.00 Kb

Objects Returned: 0

WARNINGS(0)

CLEAR EXPLAIN

QUERY OUTPUT

JSON JSONL TABLE TREE PLAN EXPORT

Items per page: 10 0 of 0

[]

METADATA INSPECTOR

DATAVERSES
- Default
- Metadata
- TinyCollege

REFRESH

DATASETS

DATATYPES

INDEX

USER DEFINED FUNCTIONS

QUERY INPUT (6/6)

Default: TinyCollege | PLAN FORMAT: JSON | OUTPUT FORMAT: JSON | QUERY HISTORY: use TinyCollege; select f.name as staff, d.name as student from Staff as f, Students...

```
1 use TinyCollege;
2
3 select f.name as staff, d.name as student
4 from Staff as f, Students as d
5 where
6 interval_overlapping(f.employment, d.attendance)
7 order by staff
```

ERROR: Code: 1 "java.lang.NullPointerException: Cannot invoke \"org.apache.hyracks.dataflow.common.data.partition.range.RangeMap.getSplitCount()\" because \"this.rangeMap\" is null"

WARNINGS(0)

CLEAR EXPLAIN

METADATA INSPECTOR

DATAVERSES
- Default
- Metadata
- TinyCollege

REFRESH

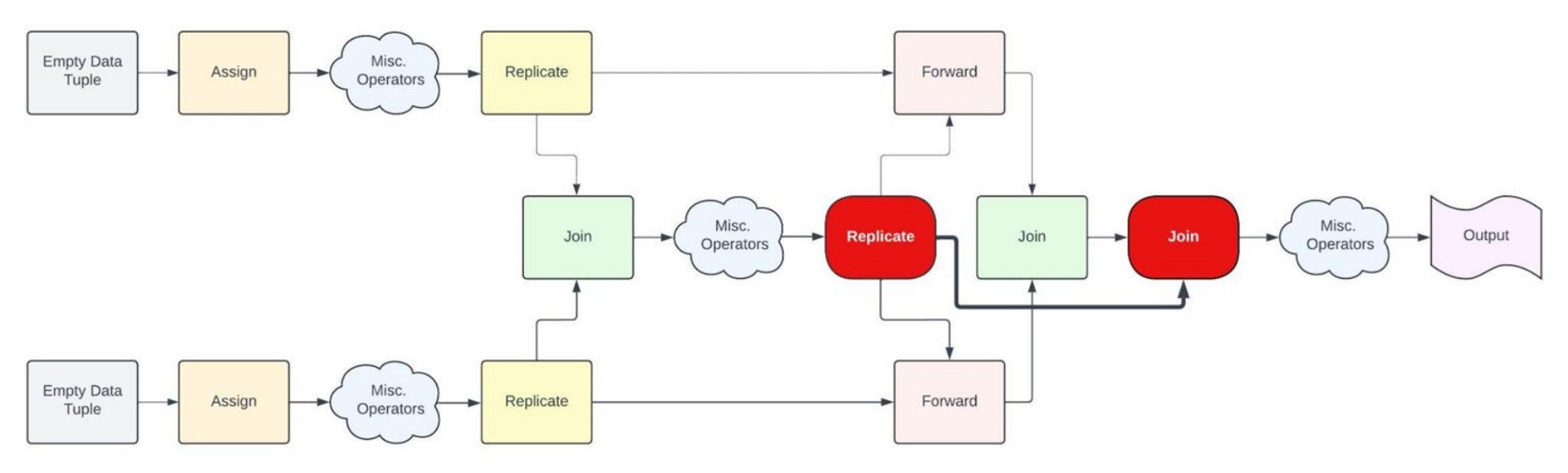
DATASETS

DATATYPES

INDEX

USER DEFINED FUNCTIONS

My solution:


Empty Data Tuple
Assign
Misc. Operators
Replicate
Forward
Join
Misc. Operators
Replicate
Join
Join
Misc. Operators
Output
Empty Data Tuple
Assign
Misc. Operators
Replicate
Forward

```
93          private final ArrayBackedValueStorage s1 = new ArrayBackedValueStorage();
94
            2 usages
95          private ISerializerDeserializer<ABoolean> booleanSerde =
96                  SerializerDeserializerProvider.INSTANCE.getSerializerDeserializer(BuiltinType.ABOOLEAN);
97
            cobeed
98          @Override
99          public void evaluate(IFrameTupleReference tuple, IPointable result) throws HyracksDataException {
100             resultStorage.reset();
101             if (partition == 0) {
102                 booleanSerde.serialize(ABoolean.TRUE, out);
103                 result.set(resultStorage);
104                 return;
105             }
106             eval0.evaluate(tuple, argPtr0);
107             eval1.evaluate(tuple, argPtr1);
108             eval2.evaluate(tuple, argPtr2);
109
110             if (PointableHelper.checkAndSetMissingOrNull(result, argPtr0, argPtr1, argPtr2)) {
111                 return;
112             }
113
114             byte type0 = argPtr0.getTag();
115             byte type1 = argPtr1.getTag();
116
117             if (type0 == ATypeTag.SERIALIZED_INTERVAL_TYPE_TAG && type0 == type1) {
118                 argPtr0.getValue(interval0);
119                 argPtr1.getValue(interval1);
120                 byte intervalType0 = interval0.getType();
121                 byte intervalType1 = interval1.getType();
122
123                 if (intervalType0 != intervalType1) {
124                     throw new IncompatibleTypeException(sourceLoc, getIdentifier(), intervalType0,
125                             intervalType1);
126                 }
127             } else if (type0 != ATypeTag.SERIALIZED_INTERVAL_TYPE_TAG) {
128                 throw new TypeMismatchException(sourceLoc, getIdentifier(), 0, type0,
129                         ATypeTag.SERIALIZED_INTERVAL_TYPE_TAG);
130             } else {
131                 throw new IncompatibleTypeException(sourceLoc, getIdentifier(), type0, type1);
132             }
```

```
    InnerJoinOperator intervalJoinOp =
            new InnerJoinOperator(new MutableObject<>(deduplicationTestCondition), leftInputOp, rightInputOp);
    intervalJoinOp.setSourceLocation(op.getSourceLocation());
    setSortMergeIntervalJoinOp(intervalJoinOp, fi, sideLeft, sideRight, context, intervalPartitions);
    intervalJoinOp.setSchema(op.getSchema());
    context.computeAndSetTypeEnvironmentForOperator(intervalJoinOp);

    Mutable<ILogicalOperator> opRef = new MutableObject<>(op);
    Mutable<ILogicalOperator> intervalJoinOpRef = new MutableObject<>(intervalJoinOp);

    InnerJoinOperator deduplicationTestOp =
            new InnerJoinOperator(new MutableObject<>(deduplicationTestCondition), intervalJoinOpRef,
                    exchMAPToDeduplicationJoinOpRef);
    deduplicationTestOp.setPhysicalOperator(new NestedLoopJoinPOperator(
            AbstractBinaryJoinOperator.JoinKind.INNER, AbstractJoinPOperator.JoinPartitioningType.BROADCAST));
    MutableObject<ILogicalOperator> referencePointTestJoinOpRef = new MutableObject<>(deduplicationTestOp);
    deduplicationTestOp.setSourceLocation(op.getSourceLocation());
    context.computeAndSetTypeEnvironmentForOperator(deduplicationTestOp);
    deduplicationTestOp.recomputeSchema();
    opRef.setValue(referencePointTestJoinOpRef.getValue());
    op.getInputs().clear();
    op.getInputs().addAll(deduplicationTestOp.getInputs());
    op.setPhysicalOperator(deduplicationTestOp.getPhysicalOperator());
    op.getCondition().setValue(deduplicationTestOp.getCondition().getValue());
    context.computeAndSetTypeEnvironmentForOperator(op);
    op.recomputeSchema();
}

1 usage  cobeed
private static ScalarFunctionCallExpression createIntervalDuplicateTest(AbstractBinaryJoinOperator op,
        LogicalVariable leftInputVar, LogicalVariable rightInputVar, LogicalVariable rangeMapVar) {
    // Compute reference interval ID
    ScalarFunctionCallExpression intervalId = new ScalarFunctionCallExpression(
            BuiltinFunctions.getBuiltinFunctionInfo(BuiltinFunctions.INTERVAL_JOIN_REMOVE_DUPLICATES),
            new MutableObject<>(new VariableReferenceExpression(leftInputVar)),
            new MutableObject<>(new VariableReferenceExpression(rightInputVar)),
            new MutableObject<>(new VariableReferenceExpression(rangeMapVar)));
    intervalId.setSourceLocation(op.getSourceLocation());

    return intervalId;
}
```

ARangeMapSerializerDeserializer.java

```java
 *
 * T can be of type date, time or datetime.
 */
7 usages  cobeed
public class ARangeMapSerializerDeserializer implements ISerializerDeserializer<RangeMap> {

    no usages
    private static final long serialVersionUID = 1L;

    public static final ARangeMapSerializerDeserializer INSTANCE = new ARangeMapSerializerDeserializer();

    1 usage  cobeed
    private ARangeMapSerializerDeserializer() {
    }

    cobeed
    @Override
    public RangeMap deserialize(DataInput in) throws HyracksDataException {
        try {
            int numFields = IntegerSerializerDeserializer.read(in);
            byte[] bytes = ByteArraySerializerDeserializer.read(in);
            int[] endOffsets = IntArraySerializerDeserializer.read(in);
            double[] percentages = DoubleArraySerializerDeserializer.read(in);

            return new RangeMap(numFields, bytes, endOffsets, percentages);
        } catch (IOException e) {
            throw HyracksDataException.create(e);
        }
    }

    cobeed
    @Override
    public void serialize(RangeMap instance, DataOutput out) throws HyracksDataException {
        try {
            IntegerSerializerDeserializer.write(instance.getNumFields(), out);
            ByteArraySerializerDeserializer.write(instance.getByteArray(), out);
            IntArraySerializerDeserializer.write(instance.getEndOffsets(), out);
            DoubleArraySerializerDeserializer.write(instance.getPercentages(), out);
        } catch (IOException e) {
            throw HyracksDataException.create(e);
        }
    }
}
```

RangeMapSerializerDeserializerTest.java

```
33
    cobeed
34  public class RangeMapSerializerDeserializerTest {
35
        cobeed
36      @Test
37      public void test() throws HyracksDataException {
38
39          ArrayBackedValueStorage abvs = new ArrayBackedValueStorage();
40          DataOutput out = abvs.getDataOutput();
41
42          AInt64SerializerDeserializer intSerde = AInt64SerializerDeserializer.INSTANCE;
43          int[] offsets = new int[3];
44          double[] percentages = new double[3];
45          try {
46              // Loop over list of integers
47              for (int i = 0; i < 3; i++) {
48                  intSerde.serialize(new AMutableInt64( value: (long) i + 10), out);
49                  offsets[i] = abvs.getLength();
50                  percentages[i] = i * 1.3;
51              }
52          } catch (IOException e) {
53              throw HyracksDataException.create(e);
54          }
55          RangeMap rangeMap = new RangeMap( numFields: 1, abvs.getByteArray(), offsets, percentages);
56
57          ArrayBackedValueStorage abvsResult = new ArrayBackedValueStorage();
58          DataOutput outResult = abvsResult.getDataOutput();
59
60          ARangeMapSerializerDeserializer rangeMapSerde = ARangeMapSerializerDeserializer.INSTANCE;
61          rangeMapSerde.serialize(rangeMap, outResult);
62
63          DataInput dataIn = new DataInputStream(new ByteArrayInputStream(abvsResult.getByteArray(),  offset: 0, abvsResult.getLength()));
64          // check output
65          RangeMap rangeMapResult = rangeMapSerde.deserialize(dataIn);
66          Assert.assertEquals(rangeMapResult.getNumFields(), rangeMap.getNumFields());
67          Assert.assertArrayEquals(rangeMapResult.getByteArray(), rangeMap.getByteArray());
68          Assert.assertArrayEquals(rangeMapResult.getEndOffsets(), rangeMap.getEndOffsets());
69          Assert.assertArrayEquals(rangeMapResult.getPercentages(), rangeMap.getPercentages(),  delta: 0.01);
70          Assert.assertEquals(rangeMapResult, rangeMap);
71      }
72  }
```

And then...

QUERY INPUT (1/1)

Default: TinyCollege | PLAN FORMAT: JSON | OUTPUT FORMAT: JSON | QUERY HISTORY: use TinyCollege; select f.name as staff, d.name as student from Staff as f, Students...

```
use TinyCollege;

select f.name as staff, d.name as student
from Staff as f, Students as d
where
interval_overlapping(f.employment, d.attendance)
order by staff
```

SUCCESS: Execution time: 517.933938ms Elapsed time: 576.303264ms Size: 175.84 Kb

Objects Returned: 4617

WARNINGS(0)

CLEAR | EXPLAIN

QUERY OUTPUT

JSON | JSONL | TABLE | TREE | PLAN | EXPORT

Items per page: 10 | 1 – 10 of 4617

{"staff":"AEJO","student":"jfby"}
{"staff":"AEJO","student":"jybx"}
{"staff":"AEJO","student":"qwpa"}
{"staff":"AEJO","student":"krgo"}
{"staff":"AEJO","student":"onya"}
{"staff":"AEJO","student":"egae"}
{"staff":"AEJO","student":"lntr"}
{"staff":"AEJO","student":"mfmo"}
{"staff":"AEJO","student":"ybsl"}

Challenges – Going In

- Expected:
  - Installing and configuring IDE and other tools
  - Reading and understanding legacy code
  - Learning Java and navigating the IDE
  - Tests run for a long time
- Unexpected:
  - Failed tests
  - Running software on Windows
  - Understanding architecture

Challenges – Going Out

- Expected:
  - Installing AsterixDB
  - Reading and understanding legacy code
  - Time-consuming build process
  - Learning Java and navigating the IDE
- Unexpected
  - Meeting with project mentor
    - Once or twice a week usually, but we had plenty of schedule conflicts
  - Confined to the CS lab
  - Staying awake
  - Keeping up with the hours
  - Submitting code
    - Documentation is sorely out-of-date

Capstone Experience

- Learn a new programming language
  - Java
- Use specialized IDE and suite of developer tools
  - Debugging
- Understand and contribute to legacy code
- Use existing framework to accomplish the goal
- Contact developers
- Regularly report to a customer

Resources

- https://asterixdb.apache.org/dev-setup.html
- https://asterixdb.apache.org/pushing.html
- https://asterixdb.apache.org/docs/0.9.9/sqlpp/builtins.html
- https://asterixdb.apache.org/docs/0.9.9/interval_join.html

Questions?