Blink’s Improvements to Flink SQL & Table API

Shaoxuan Wang,
Xiaowei Jiang
{xiaowei.jxw,shaoxuan.wang}@alibaba-inc.com

FlinkForward 2017.4
About Us

- **Xiaowei Jiang**
  - 2014-now Alibaba
  - 2010-2014 Facebook
  - 2002-2010 Microsoft
  - 2000-2002 Stratify

- **Shaoxuan Wang**
  - 2015-now Alibaba
  - 2014-2015 Facebook
  - 2010-2014 Broadcom
Agenda

- Background
- Why SQL & Table API
- Blink SQL & Table API (Selected Topics)
Background About Alibaba

- **Alibaba Group**
  - Operates the world’s largest e-commerce platform
  - Recorded GMV of $394 Billion in year 2015, $17.8 billion worth of GMV on Nov 11, 2016

- **Alibaba Search**
  - Personalized search and recommendation
  - Major driver for user’s traffic
What is Blink?

- Blink – A project to make Flink work well for large scale production at Alibaba
  - Run on Thousands of Nodes In Production
  - Support Key Production Scenarios, such as Search and Recommendation
  - Compatible with Flink API and Ecosystem
Documents Building for Search

Data Source
- Item
- Seller
- Category

Sync

HBase
- Item
- Seller

Join

HBase
- Result

Export

Search Engine
Why Flink SQL & Table API

- **Unify batch and streaming**
  - Flink currently offers DataSet API for batch and DataStream API for streaming
  - We want a single API that can run in both batch and streaming mode

- **Simplify user code**
  - Users only describe the semantics of their data processing
  - Leave hard optimization problems to the system
  - SQL is proven to be good at describing data processing
  - Table API makes multi-statement data processing easy to write
  - Table API also makes it possible/easy to extend standard SQL when necessary
### Stream-Table Duality

#### Stream

<table>
<thead>
<tr>
<th>word</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello</td>
<td>1</td>
</tr>
<tr>
<td>World</td>
<td>1</td>
</tr>
<tr>
<td>Hello</td>
<td>2</td>
</tr>
<tr>
<td>Bark</td>
<td>1</td>
</tr>
<tr>
<td>Hello</td>
<td>3</td>
</tr>
</tbody>
</table>

#### Dynamic Table

<table>
<thead>
<tr>
<th>word</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello</td>
<td>3</td>
</tr>
<tr>
<td>World</td>
<td>1</td>
</tr>
<tr>
<td>Bark</td>
<td>1</td>
</tr>
</tbody>
</table>

**Apply**

**Changelog**
Apply Changelog Stream to Dynamic Table

- **Append Mode**: each stream record is an insert modification to the dynamic table. Hence, all records of a stream are appended to the dynamic table.

- **Update Mode**: a stream record can represent an insert, update, or delete modification on the dynamic table (append mode is in fact a special case of update mode).

Derive Changelog Stream from Dynamic Table

- **REDO Mode**: where the stream records the new value of a modified element to redo lost changes of completed transactions.

- **REDO+UNDO Mode**: where the stream records the old and the new value of a changed element to undo incomplete transactions and redo lost changes of completed transactions.
Dynamic Tables generalize the concept of Static Tables
SQL serves as the unified way to describe data processing in both batch and streaming
There is no such thing as Stream SQL
Stream-Stream Inner Join
User Defined Function (UDF)
User Defined Table Function (UDTF)
User Defined Aggregate Function (UDAGG)
Retract (stream only)
Over Aggregates
A Simple Query: Select and Where

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>price</th>
<th>sales</th>
<th>stock</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Latte</td>
<td>6</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>8</td>
<td>Mocha</td>
<td>8</td>
<td>1</td>
<td>800</td>
</tr>
<tr>
<td>4</td>
<td>Breve</td>
<td>5</td>
<td>1</td>
<td>200</td>
</tr>
<tr>
<td>7</td>
<td>Tea</td>
<td>4</td>
<td>1</td>
<td>2000</td>
</tr>
<tr>
<td>1</td>
<td>Latte</td>
<td>6</td>
<td>2</td>
<td>998</td>
</tr>
</tbody>
</table>

```sql
SELECT id, name, price, sales, stock
FROM myTable WHERE name = 'Latte'
```
Stream-Stream Inner Join

<table>
<thead>
<tr>
<th>id1</th>
<th>name</th>
<th>stock</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Latte</td>
<td>1000</td>
</tr>
<tr>
<td>8</td>
<td>Mocha</td>
<td>800</td>
</tr>
<tr>
<td>4</td>
<td>Breve</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>Water</td>
<td>5000</td>
</tr>
<tr>
<td>7</td>
<td>Tea</td>
<td>2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id2</th>
<th>price</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

```
SELECT id1 AS id, name, price, sales, stock
FROM table1 INNER JOIN table2 ON id1 = id2
```

This is proposed and discussed in FLINK-5878
Blink SQL & Table API

- Stream-Stream Inner Join
- User Defined Function (UDF)
- User Defined Table Function (UDTF)
- User Defined Aggregate Function (UDAGG)
- Retract (stream only)
- Over Aggregates
Create and use a UDF is very simple and easy:

```scala
object AddFunc extends ScalarFunction {
  def eval(a: Long, b: Long): Long = a + b
  @varargs
def eval(a: Int*): Int = a.sum
}

tEnv.registerTable("MyTable", table)
tEnv.registerFunction("addFunc", AddFunc)
val sqlQuery = "SELECT addFunc(long1,long2), addFunc(int1,int2,int3) FROM MyTable"
```

We recently have enhanced UDF/UDTF to let them support variable types and variable arguments (FLINK-5826)
User Defined Table Function (UDTF)

We have shipped UDTF in flink 1.2 (FLINK-4469).

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>23</td>
</tr>
<tr>
<td>Jack</td>
<td>17</td>
</tr>
<tr>
<td>David</td>
<td>50</td>
</tr>
</tbody>
</table>

```
SELECT name, age
FROM myTable,
    LATERAL TABLE(splitFunc(line)) AS T(name, age)
```

```
case class User(name: String, age: Int)
class SplitFunc extends TableFunction[User] {
  def eval(str: String): Unit = {
    str.split(" ").foreach{ e =>
      val subSplits = e.split("#")
      collect(User(subSplits(0), subSplits(1).toInt))
    }
  }
}
```
User Defined Aggregate Function (UDAGG) - Motivation

Flink has built-in aggregates (count, sum, avg, min, max) for SQL and table API

```sql
SELECT name, SUM(price), COUNT(sales),
      MAX(price), MIN(price), AVG(price)
FROM myTable
GROUP BY name
```

What if user wants an aggregate that is not covered by built-in aggregates, say a weighted average aggregate?

We need an aggregate interface to support user defined aggregate function.
UDAGG – Accumulator (ACC)

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>price</th>
<th>sales</th>
<th>stock</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Latte</td>
<td>6</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>8</td>
<td>Mocha</td>
<td>8</td>
<td>1</td>
<td>800</td>
</tr>
<tr>
<td>4</td>
<td>Breve</td>
<td>5</td>
<td>1</td>
<td>200</td>
</tr>
<tr>
<td>7</td>
<td>Tea</td>
<td>4</td>
<td>1</td>
<td>2000</td>
</tr>
<tr>
<td>1</td>
<td>Latte</td>
<td>6</td>
<td>2</td>
<td>998</td>
</tr>
</tbody>
</table>

UDAGG represents its state using accumulator

```
SELECT name, SUM(price), COUNT(sales), MAX(price), MIN(price), AVG(price)
FROM myTable
GROUP BY name
```
UDAGG Interface

UDAGG example: a weighted average

```java
public static class WeightedAvgAccum {
    public long sum = 0; public int count = 0;
    public static class WeightedAvg extends AggregateFunction<Long, WeightedAvgAccum> {
        @Override
        public WeightedAvgAccum createAccumulator() {
            return new WeightedAvgAccum();
        }
        @Override
        public Long getValue(WeightedAvgAccum accumulator) {
            if (accumulator.count == 0) return null;
            else return accumulator.sum/accumulator.count;
        }
        public void accumulate(
            WeightedAvgAccum accumulator, long iValue, int iWeight) {
            accumulator.sum += iValue * iWeight;
            accumulator.count += iWeight;
        }
    }
    public static class AggregateFunction<T, ACC> extends UserDefinedFunction {
        def createAccumulator(): ACC
        def getValue(accumulator: ACC): T
    }
}
/* The implementations of accumulate must be declared publicly, not static and named exactly as "accumulate". accumulate method can be overloaded */
def Accumulate(accumulator: ACC, [user defined inputs]): Unit
```
Motivated by local & global aggregate (and session window merge etc.), we need a merge method which can merge the partial aggregated accumulator into one single accumulator.
SELECT cnt, COUNT(word) AS freq 
FROM ( 
    SELECT word, COUNT(num) AS cnt 
    FROM Table GROUP BY word 
) GROUP BY cnt 

**Incorrect! This value should be 2**
We need a retract method in UDAGG, which can retract the input values from the accumulator.
Retract – Solution

- Retraction is introduced to handle updates
- We use query optimizer to decide where the retraction is needed.

The design doc and the progress of retract implementation are tracked in FLINK-6047. A FLIP for retract is on the way. We aim to release it in flink-1.3
Master JIRA for UDAGG is FLINK-5564. We plan to ship it in release 1.3.
Blink SQL & Table API

- Stream-Stream Inner Join
- User Defined Function (UDF)
- User Defined Table Function (UDTF)
- User Defined Aggregate Function (UDAGG)
- Retract (stream only)
- Over Aggregates
Calculate moving average (in the past 5 seconds), and emit the result for each record.

```
SELECT time, itemID, MovingAverage(price) OVER (ORDER BY RowTime() RANGE BETWEEN INTERVAL '5' SECOND PRECEDING AND CURRENT ROW) AS avgPrice
FROM myTable
```
Group/Over Aggregates

- **Grouping methods:** Groupby / Over

- **Window types:**
  - Time/Count + TUMBLE/SESSION/SLIDE window;
  - OVER Range/Rows window

- **Time types:** Event time; Process time (only for stream)

We have been working closely with team dataArtisans, from the design to the implementation on FLIP11 (FLINK-4557). Upon now, except the unbounded group aggregate, all other group/over aggregates are fully supported via SQL query. We are working on the support for table API.
Current Status of Flink SQL & Table API

- UDF (several improvements will be released in 1.3)
- UDTF (FLINK-4469, released in 1.2)
- UDAGG (FLINK-5564, target for release 1.3)
- Group/Over Window Aggregate (FLINK-4557, target for release 1.3)
- Retract (FLINK-6047, target for release 1.3)
- Unbounded Stream Group Aggregate (FLINK-6216, bundled with retract design)
- Stream-Stream Inner Join (FLINK-5878, TBD)

We will keep merging Blink SQL & Table API back to Flink
Thank You!