# Blink’s Improvements to Flink SQL &Table API

FlinkForward 2017.4

**Shaoxuan Wang,**

**Xiaowei Jiang**

**{xiaowei.jxw,shaoxuan.wang}@alibaba-inc.com**

Alibaba Group

## 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)

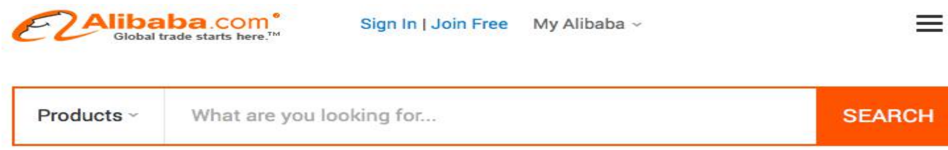
# Background About Alibaba

## 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

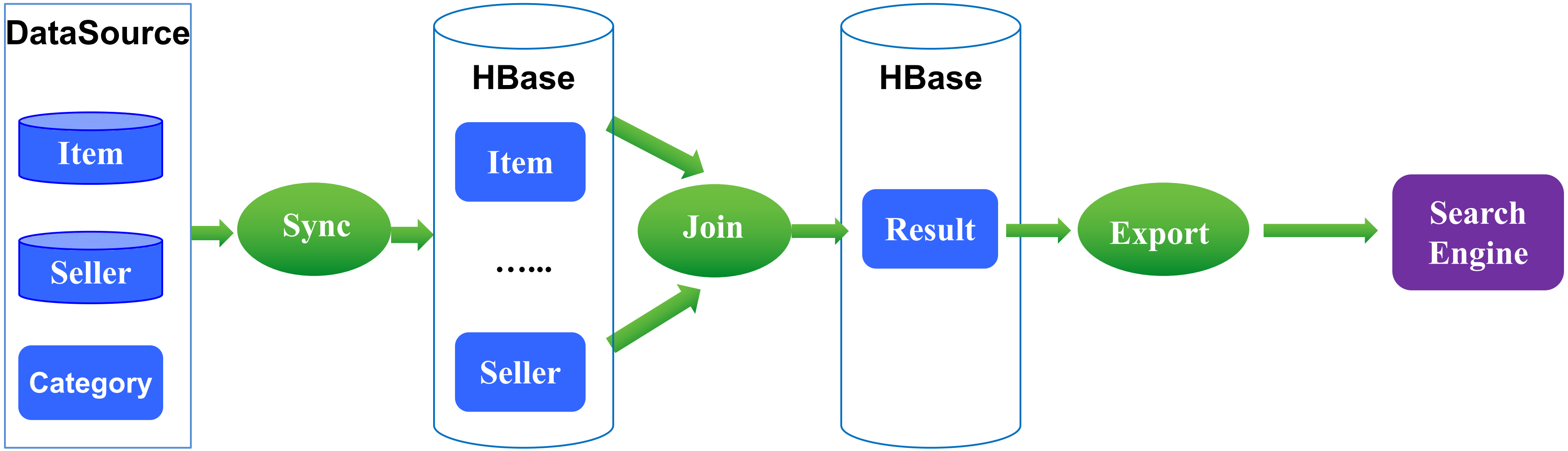


天猫
TMALL.COM
戴森颠覆你的生活
搜 索
连衣裙夏 | 智能手机 | t恤男 | 亲子装 | 凉鞋 | 夏凉被 | 睡衣女夏 | 防晒衣 | 电风扇

## 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



## Why Flink SQL & Table API

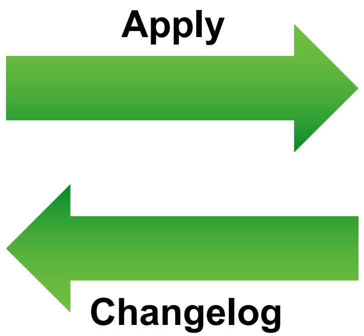
- **Unify batch and streaming**
  - **Flink currently offers DataSet API for batch and DataStream API for streaming**
  - **We want a single API that can run in both batch and streaming mode**

- **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**

| word | count |
|---|---|
| Hello | 1 |
| World | 1 |
| Hello | 2 |
| Bark | 1 |
| Hello | 3 |



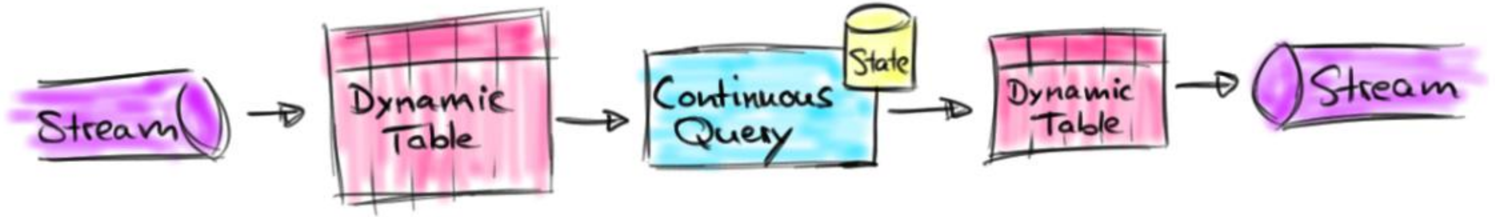
**Dynamic Table**

| word | count |
|---|---|
| Hello | 3 |
| World | 1 |
| Bark | 1 |

## Dynamic Tables

- **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

## Stream SQL



**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**

## 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

## A Simple Query: Select and Where

| id | name | price | sales | stock |
|---|---|---|---|---|
| 1 | Latte | 6 | 1 | 1000 |
| 8 | Mocha | 8 | 1 | 800 |
| 4 | Breve | 5 | 1 | 200 |
| 7 | Tea | 4 | 1 | 2000 |
| 1 | Latte | 6 | 2 | 998 |

```
SELECT id, name, price, sales, stock
FROM myTable WHERE name = 'Latte'
```

| id | name | price | sales | stock |
|---|---|---|---|---|
| 1 | Latte | 6 | 1 | 1000 |
| 1 | Latte | 6 | 2 | 998 |

# Stream-Stream Inner Join

| id1 | name | stock |
|---|---|---|
| 1 | Latte | 1000 |
| 8 | Mocha | 800 |
| 4 | Breve | 200 |
| 3 | Water | 5000 |
| 7 | Tea | 2000 |

| id2 | price | sales |
|---|---|---|
| 1 | 6 | 1 |
| 8 | 8 | 1 |
| 9 | 3 | 1 |
| 4 | 5 | 1 |
| 7 | 4 | 1 |

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

| id | name | price | sales | stock |
|---|---|---|---|---|
| 1 | Latte | 6 | 1 | 1000 |
| 8 | Mocha | 8 | 1 | 800 |
| 4 | Breve | 5 | 1 | 200 |
| 7 | Tea | 4 | 1 | 2000 |

*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

## User Defined Function (UDF)

Create and use a UDF is very simple and easy:

```
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)

| line |
|---|
| Tom#23 Jark#17 David#50 |

Scalar → Table (multi rows and columns)

| name | age |
|---|---|
| Tom | 23 |
| Jack | 17 |
| David | 50 |

```
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))
    }
  }
}
```

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

## User Defined Aggregate Function (UDAGG) - Motivation

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

```
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?

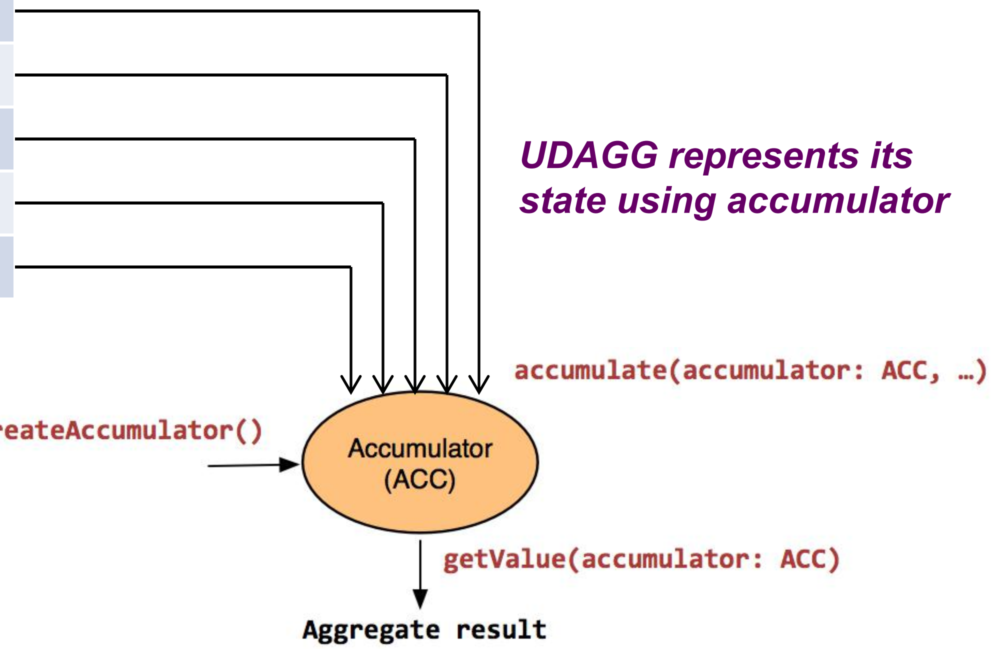
We need an aggregate interface to support user defined aggregate function.

# UDAGG – Accumulator (ACC)

| id | name | price | sales | stock |
|---|---|---|---|---|
| 1 | Latte | 6 | 1 | 1000 |
| 8 | Mocha | 8 | 1 | 800 |
| 4 | Breve | 5 | 1 | 200 |
| 7 | Tea | 4 | 1 | 2000 |
| 1 | Latte | 6 | 2 | 998 |

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

*UDAGG represents its state using accumulator*



# UDAGG – Interface

## UDAGG Interface

```
Abstract 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
```

## SQL Query

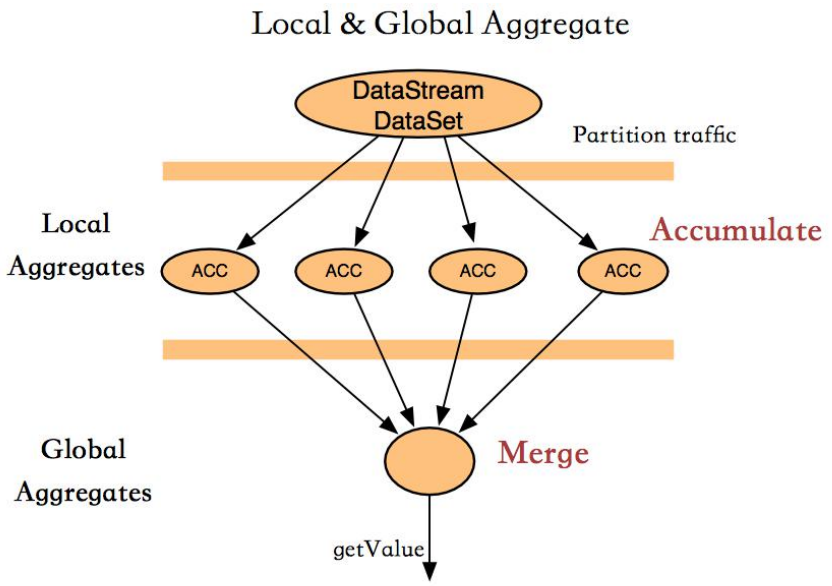
```
SELECT type, weightAvgFun(weight, cnt)
FROM myTable
GROUP BY type
```

## UDAGG example: a weighted average

```
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; }
}
```

## UDAGG – Merge

*How to count the total visits on TaoBao web pages in real time?*

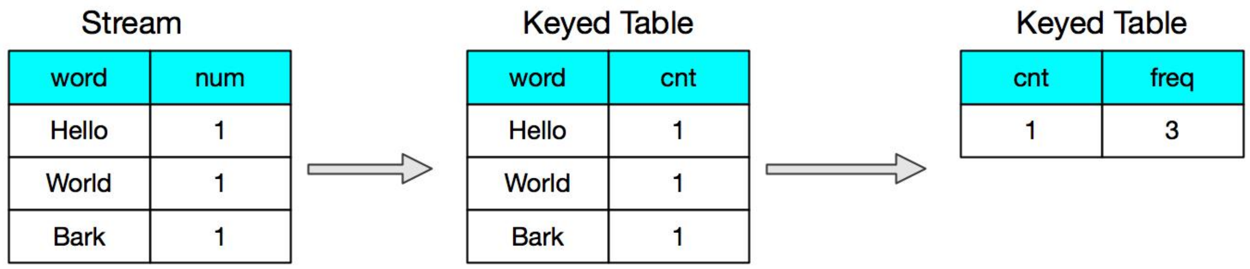


```
Abstract class AggregateFunction[T, ACC] extends UserDefinedFunction {
  def createAccumulator(): ACC
  def getValue(accumulator: ACC): T
}
/* The implementations of following methods: accumulate(MUST
have), merge(OPTIONAL) must be declared publicly, not static and
named exactly as "accumulate", “merge”*/
  def Accumulate(accumulator: ACC, [user defined inputs]): Unit
  def merge(it: java.util.iterator[ACC]): ACC
```

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

# UDAGG - Retract - Motivations

```
SELECT
  cnt,
  COUNT(word) AS freq
   FROM  (
       SELECT word,  COUNT(num) AS cnt
            FROM Table GROUP BY word
    ) GROUP BY cnt
```



Stream

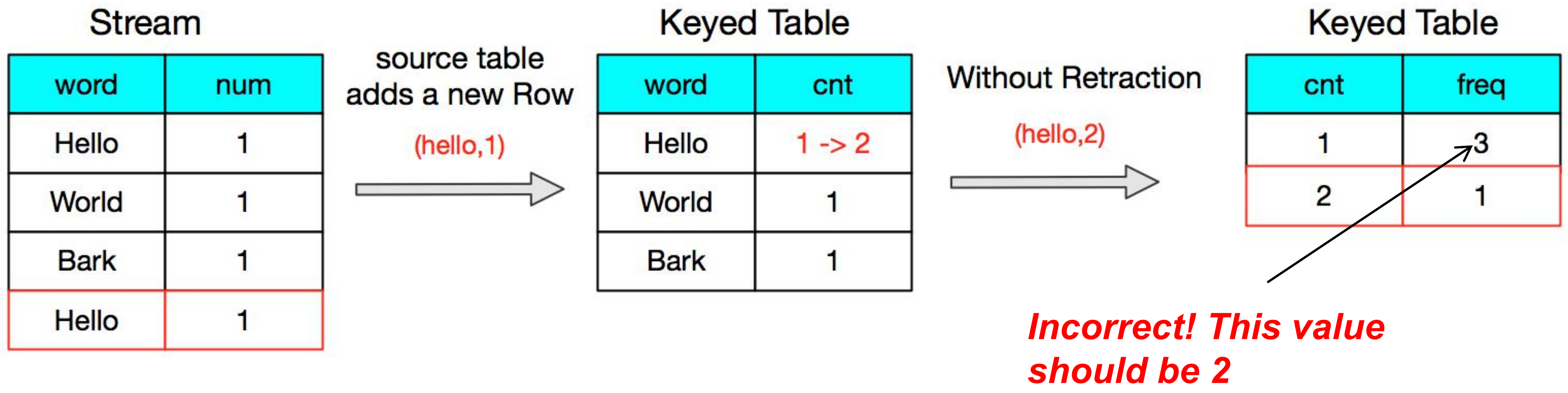
| word | num |
|---|---|
| Hello | 1 |
| World | 1 |
| Bark | 1 |

Keyed Table

| word | cnt |
|---|---|
| Hello | 1 |
| World | 1 |
| Bark | 1 |

Keyed Table

| cnt | freq |
|---|---|
| 1 | 3 |



Stream

| word | num |
|---|---|
| Hello | 1 |
| World | 1 |
| Bark | 1 |
| Hello | 1 |

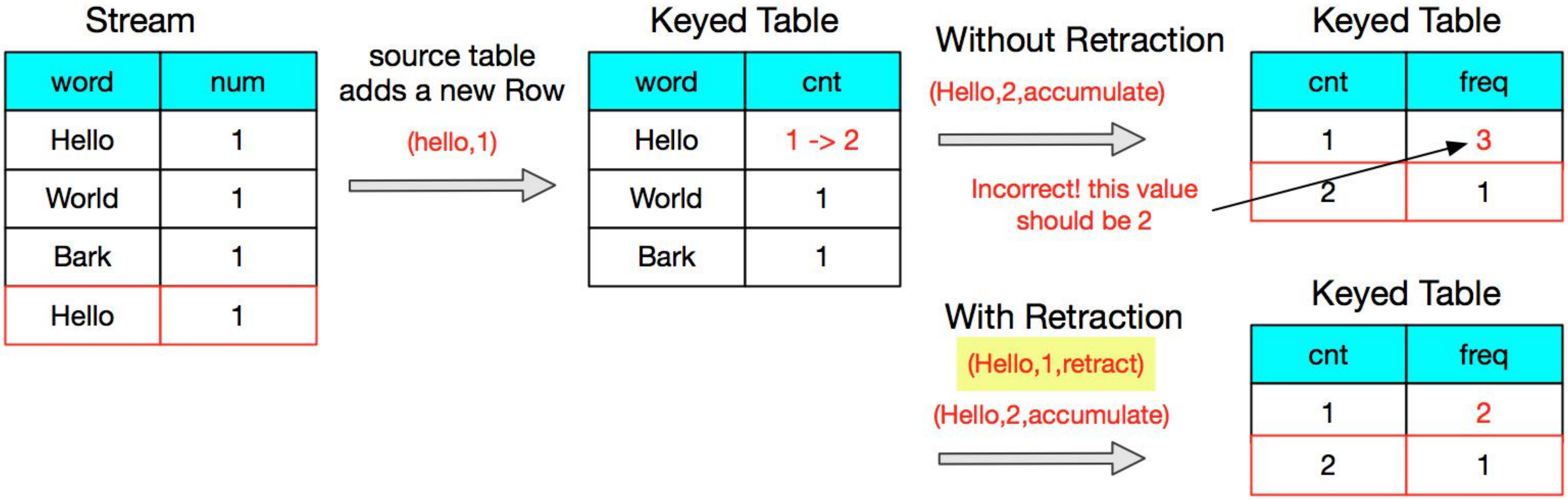
Keyed Table

| word | cnt |
|---|---|
| Hello | 1 -> 2 |
| World | 1 |
| Bark | 1 |

Keyed Table

| cnt | freq |
|---|---|
| 1 | 3 |
| 2 | 1 |

# UDAGG - Retract – Motivations



Stream

| word | num |
|---|---|
| Hello | 1 |
| World | 1 |
| Bark | 1 |
| Hello | 1 |

Keyed Table

| word | cnt |
|---|---|
| Hello | 1 -> 2 |
| World | 1 |
| Bark | 1 |

Keyed Table

| cnt | freq |
|---|---|
| 1 | 3 |
| 2 | 1 |

Keyed Table

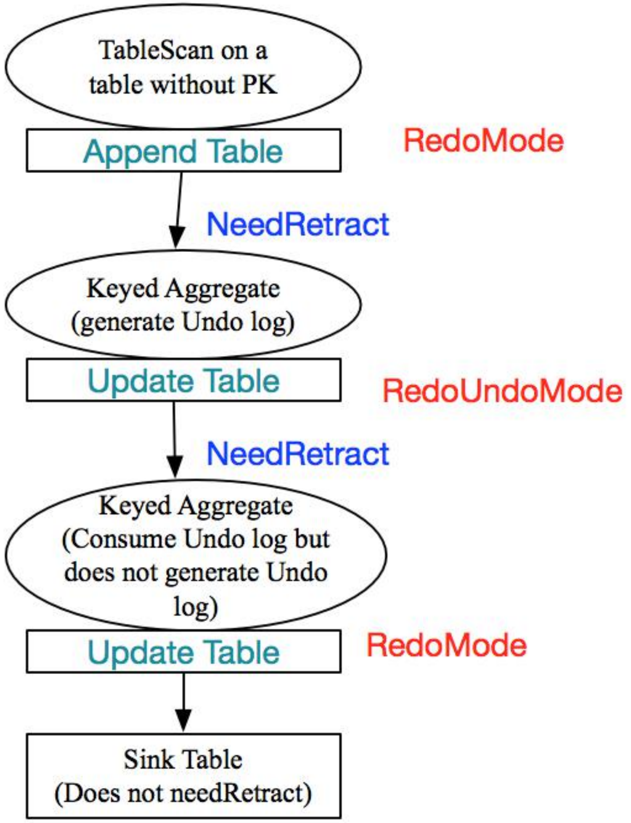
| cnt | freq |
|---|---|
| 1 | 2 |
| 2 | 1 |

*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*


TableScan on a table without PK
Append Table
RedoMode
NeedRetract
Keyed Aggregate (generate Undo log)
Update Table
RedoUndoMode
NeedRetract
Keyed Aggregate (Consume Undo log but does not generate Undo log)
Update Table
RedoMode
Sink Table (Does not needRetract)

## UDAGG – Summary

```
Abstract class AggregateFunction[T, ACC] extends UserDefinedFunction {
  def createAccumulator(): ACC
  def getValue(accumulator: ACC): T
}
/* The implementations of following methods: accumulate(MUST have),
merge(OPTIONAL), and retract(OPTIONAL) must be declared publicly, not
static and named exactly as "accumulate", “merge”, and “retract” */
  def accumulate(accumulator: ACC, [user defined inputs]): Unit
  def merge(it: java.util.iterator[ACC]): ACC
  def retract(accumulator: ACC, [user defined inputs]): Unit
```

*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

## Over Aggregates

*Calculate moving average (in the past 5 seconds), and emit the result for each record*

| time | itemID | price |
|---|---|---|
| 1000 | 101 | 1 |
| 3000 | 201 | 2 |
| 4000 | 301 | 3 |
| 5000 | 101 | 1 |
| 5000 | 401 | 4 |
| 7000 | 301 | 3 |
| 8000 | 501 | 5 |
| 10000 | 101 | 1 |

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

| time | itemID | avgPrice |
|---|---|---|
| 1000 | 101 | 1 |
| 3000 | 201 | 1.5 |
| 4000 | 301 | 2 |
| 5000 | 101 | 2.2 |
| 5000 | 401 | 2.2 |
| 7000 | 301 | 2.6 |
| 8000 | 401 | 3 |
| 10000 | 101 | 2.8 |

*time based Group Aggregate is not able to differentiate two records with the same row time.*

## 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

- Flink blog: “Continuous Queries on Dynamic Tables” *(posted at https://flink.apache.org/news/2017/04/04/dynamic-tables.html)*
- 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 release1.3)*
- 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*

## Q & A

Thank You!