Stream Analytics with SQL on Apache Flink®

Fabian Hueske
@fhueske
dataArtisans

Strata Data Conference, London
May, 24th 2017
About me

- Apache Flink PMC member
  - Contributing since day 1 at TU Berlin
  - Focusing on Flink’s relational APIs since 1.5 years

- Co-founder of data Artisans

- Co-author of “Stream Processing with Apache Flink”
  - Work in progress…

- PhD in Computer Science
Original creators of Apache Flink®

Providers of the dA Platform, a supported Flink distribution
Apache Flink

- Platform for scalable stream processing
- Fast
  - Low latency and high throughput
- Accurate
  - Stateful streaming processing in event time
  - Exactly-once state guarantees
- Reliable
  - Highly available cluster setup
  - Snapshot and restart applications
Powered by Flink
Flink’s DataStream API

- The DataStream API is very expressive
  - Application logic implemented as user-defined functions
  - Windows, triggers, evictors, state, timers, async calls, …

- Many applications follow similar patterns
  - Do not require the expressiveness of the DataStream API
  - Can be specified more concisely and easily with a DSL

- Q: What’s the most popular DSL for data processing?
Many good reasons to use SQL

- Declarative specification
- Effective optimization
- Efficient execution
- “Everybody” knows SQL

But does SQL work for streams as well?
SQL was not designed for streams

- Relations are bounded (multi-)sets. ↔ Streams are infinite sequences.
- DBMS can access all data. ↔ Streaming data arrives over time.
- SQL queries return a result and complete. ↔ Streaming queries continuously emit results and never complete.
Some RDBMS do it anyway 😊

- Materialized views (MV) are similar to regular views, but persisted to disk or memory
  - Used to speed-up analytical queries
  - MVs must be updated when the base tables change

- MV maintenance is very similar to SQL on streams
  - Base table updates are a changelog stream
  - MV definition is SQL query to evaluate
  - MV is query result
Apache Flink’s Relational APIs

- Standard SQL & LINQ-style Table API
- Unified APIs for batch & streaming data

A query specifies exactly the same result regardless whether its input is static batch data or streaming data.

- Common translation layers
  - Optimization based on Apache Calcite
  - Type system & code-generation
  - Table sources & sinks
Show me Code!

val tableApiResult: Table = tEnv
  .scan("sensors")
  .window(Tumble over 1.hour on 'mtime as 'w)
  .groupBy('w, 'room)
  .select('room, 'w.end, 'temp.avg as 'avgTemp)

val sqlResult: Table = tEnv.sql(""
| SELECT room, |
| TUMBLE_END(mtime, INTERVAL '1' HOUR), |
| AVG(temp) AS avgTemp |
| FROM sensors |
| GROUP BY TUMBLE(mtime, INTERVAL '1' HOUR), room |
""".stripMargin)
Continuous Queries

- Core concept is a “Dynamic Table”
  - Dynamic tables are changing over time
  - Insert, update, and delete changes

- Dynamic tables can be queries like static batch tables
  - Queries produce new dynamic tables
  - Queries do not terminate

- Stream $\leftrightarrow$ Dynamic table conversions
Stream → Dynamic Table

- Stream records are appended to dynamic table
  - Table grows as more data arrives
Querying a Dynamic Table

- Dynamic tables change over time
  - $A[t]$: Table A at time $t$

- Dynamic tables are queried with regular SQL
  - $q(A[t])$: Evaluate query $q$ on table A at time $t$

- As time progresses, the result is continuously updated
  - Similar to maintaining a materialized view
Querying a Dynamic Table

Table A

<table>
<thead>
<tr>
<th>time</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
</tr>
<tr>
<td>9</td>
<td>B</td>
</tr>
<tr>
<td>11</td>
<td>A</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
</tr>
<tr>
<td>14</td>
<td>C</td>
</tr>
<tr>
<td>15</td>
<td>A</td>
</tr>
</tbody>
</table>

q:

```
SELECT
  k,
  COUNT(k) AS cnt,
  TUMBLE_END(
    time,
    INTERVAL '5' SECONDS)
  AS endT
FROM A
GROUP BY
  k,
  TUMBLE(
    time,
    INTERVAL '5' SECONDS)
```

q(A)

<table>
<thead>
<tr>
<th>k</th>
<th>cnt</th>
<th>endT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>15</td>
</tr>
</tbody>
</table>
Querying a Dynamic Table

```
Table A
+---+---+
| time | k  |
|-----+---|
|  1  | A  |
|  2  | B  |
|  4  | A  |
|  5  | C  |
|  7  | B  |
|  8  | A  |
+---+---+

SELECT k, COUNT(k) as cnt
FROM A
GROUP BY k
```

```
| A[8] |
| A[9] |
| A[12] |
```

```sql
SELECT k, COUNT(k) as cnt
FROM A
GROUP BY k
```
Dynamic Table $\rightarrow$ Stream

- Converting a dynamic table into a stream
  - Dynamic tables can be updated
  - Updates must be encoded in outgoing stream

- Conversion of tables to streams inspired by DBMS logs
  - DBMS use logs to restore databases (and tables)
  - Logs are streams of records that encode updates
Dynamic Table $\rightarrow$ Stream: REDO/UNDO

Table A

<table>
<thead>
<tr>
<th>time</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

```
SELECT k, COUNT(k) AS cnt
FROM A
GROUP BY k
```

```
+ A, 3
- A, 2
+ B, 2
- B, 1
+ C, 1
+ A, 2
- A, 1
+ B, 1
+ A, 1
```

+ INSERT / - DELETE
Dynamic Table $\rightarrow$ Stream: REDO

```
Table A

<table>
<thead>
<tr>
<th>time</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
</tr>
</tbody>
</table>

SELECT k, COUNT(k) AS cnt FROM A GROUP BY k
```

- INSERT, * UPDATE (by KEY), - DELETE (by KEY)
Can We Run Any Query on Dynamic Tables?

- No, there are space and computation constraints 😞
- State size may not grow infinitely as more data arrives

```
SELECT sessionId, count(*) FROM clicks GROUP BY sessionId;
```

- A change of an input table may only trigger a partial re-computation of the result table

```
SELECT userId, RANK() OVER (ORDER BY highScore) FROM users;
```
Bounding the Size of Query State

- Adapt the semantics of the query

  ```sql
  SELECT sessionId, COUNT(*) AS clickCnt
  FROM clicks
  WHERE last(ctime, INTERVAL '1' DAY)
  GROUP BY sessionId
  ```

  - Aggregate data of last 24 hours. Discard older data.

- Trade the accuracy of the result for size of state
  - Remove state for keys that became inactive.
Current State of SQL & Table API

- Flink’s relational APIs are rapidly evolving
  - Lots of interest by community and many contributors
  - Used in production at large scale

- Features of the upcoming Flink 1.3.0 release
  - GroupBy & Over windowed aggregates
  - Non-windowed aggregates (with update changes)
  - User-defined aggregation functions
What can we build with this?

- Continuous ETL & Data Import
- Dashboards & Stateful Microservices
Conclusion

- Table API & SQL support many streaming use cases
  - High-level / declarative specification
  - Automatic optimization and translation
  - Efficient execution

- Updating results enables many exciting applications
  - Materialize a table that is updated by a stream in a KV store

- Check it out!
FLINK FORWARD

Berlin
11-13 Sep 2017
Flink Forward, the premier conference on Apache Flink®, is coming back to Berlin

Call for Submissions is open
Thank you!

@fhueske
@ApacheFlink
@dataArtisans

Available on O’Reilly Early Release!
dataArtisans

We are hiring!

data-artisans.com/careers