Achieve Results Faster with Apache Flink SQL

Apache Flink is one of the most widely used real-time data processing platforms due to its true streaming semantics, high throughput, and low latency capabilities. Some streaming solutions only emulate real-time streaming semantics, while others are too dependent on a specific streaming platform like Apache Kafka. With Apache Flink, developers have the option to use Java, Scala, or Python when building powerful real-time applications, all while keeping within true streaming semantics and being agnostic to which streaming platform they use.

Most application developers are used to building applications atop databases like PostgreSQL, where their data is at rest. Streaming applications are basically applications that are built using data that is in motion. Building your own Flink application using an imperative language like Java will require a deep and thorough understanding of streaming concepts. The semantics of streaming are very different from applications that use data at rest because they tend to follow batch processing semantics. For example, when you aggregate or join streaming data, you're required to provide a window. In data processing, a window is a time scope defined by a start and end time that is always moving forward). Windows are not required for batch processing. In batching, the window is technically assumed to be the end of the data set, so providing a window is not required. In streaming, there is no end because it's a continuous flow of data.

Not only do developers need to understand streaming semantics, they most likely will not be writing code that is optimized and efficient for Flink. It requires very skilled and experienced developers to know about these optimization techniques. If you happen to be one of these very skilled Flink developers, then chances are good that you're probably also a stream processing enthusiast and an Apache Flink committer.

So how can we make the power of Flink more accessible to developers without requiring them to be hyper-specialized in Flink and Java?

Relational Algebra

If you look up the term "relational algebra," you'll get a lot of definitions that are similar to, "it's a procedural query language that uses operators to query data." In simpler terms, it's a process that helps define how data should be queried. It does so by using operators that provide instructions on how to process data in a syntax that is easy to understand and write. Some of the operators used in relational algebra are:

- Select
- Union
- Join

As you can see, these operators are key terms used in the SQL language. SQL (structured query language) provides an abstract way to define how data should be processed. A SQL parser breaks apart and analyzes a SQL statement to identify the specific relational algebra operations and interpret the code.

Apache Flink provides two APIs which are based on relational algebra: a SQL interface (referred to as Flink SQL) and the Table API. We will only be covering SQL in this post. What's special about SQL is that it is easy to understand and can be written by many technologists. Most importantly, Flink SQL automatically translates and optimizes the SQL into relational algebra operations creating efficient runtime code. These optimization techniques are done by leveraging Apache Calcite, which is only available if you're using SQL. If you write a Flink application using a supported imperative language like Java, you will not have the optimization features provided by Apache Calcite. Trying to optimize complex data processing in Flink yourself will most likely not result in the level of performance and efficiency that Flink SQL will provide.

Flink SQL automatically translates and optimizes the SQL into relational algebra operations creating efficient runtime code.

Accessibility Advantage

SQL is used by many types of engineers, not just data engineers. It's used in both customer facing applications and to provide training data for data scientists when building machine learning models. One of the greatest strengths of SQL is that it makes data very accessible to many types of users.

Using SQL as a way to make data tools accessible is also very common. Let's take for example a tool called osquery, which is a tool for querying low-level operating system analytics using SQL. It usually takes a Linux administrator to know all the commands to retrieve low-level data from the operating system. In contrast, the SQL interface in osquery makes it easy for non-administrators to retrieve this information without knowing the Linux commands. Flink SQL provides this same accessibility.

Data Mesh

Likewise, using Flink SQL in a data mesh is a really easy way to enable domains to build their own data products. In a data mesh, domains are usually the source of data. They are the experts of the data they produce and therefore are the best personas to process it. A data mesh is a decentralized data platform architecture that pushes data ownership back to its domain for their application developers to process. These application developers don't necessarily know how to use the tools that data engineers use to process data. Having a tool like Flink SQL enables them to build data products much more quickly without having to learn a whole new technology.

Simplified Pattern Matching

With fewer lines of code, developers can build complicated pattern matching logic that would normally require 100x more lines of procedural code. Flink SQL comes with a complex event processing (CEP) library which allows for pattern detection in event streams. The example below shows a pattern matching library called MATCH_RECOGNIZE. It is able to find patterns of events in real-time streams with simple RegEx-style logic.

```
SELECT T.aid, T.bid, T.cid
FROM MyTable
  MATCH_RECOGNIZE (
    PARTITION BY userid
    ORDER BY proctime
    MEASURES A.id AS aid, B.id AS bid, C.id AS cid
    PATTERN (A B* C)
    DEFINE A AS name = 'a', B AS name = 'b', C AS name = 'c'
  ) AS T
```

The PATTERN clause in the SQL above defines the RegEx style pattern. PATTERN (A B* C) holds 3 variables A, B, and C. In this RegEx logic, A must occur once, followed by B which can have 0-N occurrences, and followed by at least one occurrence of C. A, B, and C are defined in the definitions as having a specific name value. All As will have a name = 'a', all Bs have name = 'b', and all Cs have name = 'c'. Writing such logic in Java is comparatively quite difficult. As a result, Flink SQL makes this capability accessible to a much broader range of generalist engineers.

Summary

Flink SQL provides the ability to process real-time data using SQL. It provides an efficient and optimized implementation of Flink that is hard to achieve with Java without being an expert. Apache Flink is the most powerful real-time processing framework today. It's agnostic to the streaming platforms currently in use across a wide variety of environments and is independently scalable. Flink SQL makes all the features of Flink accessible to generalist engineers. You can build real-time efficient applications and go faster to market with them. For more information about Flink SQL, please visit decodable.co and join our community.