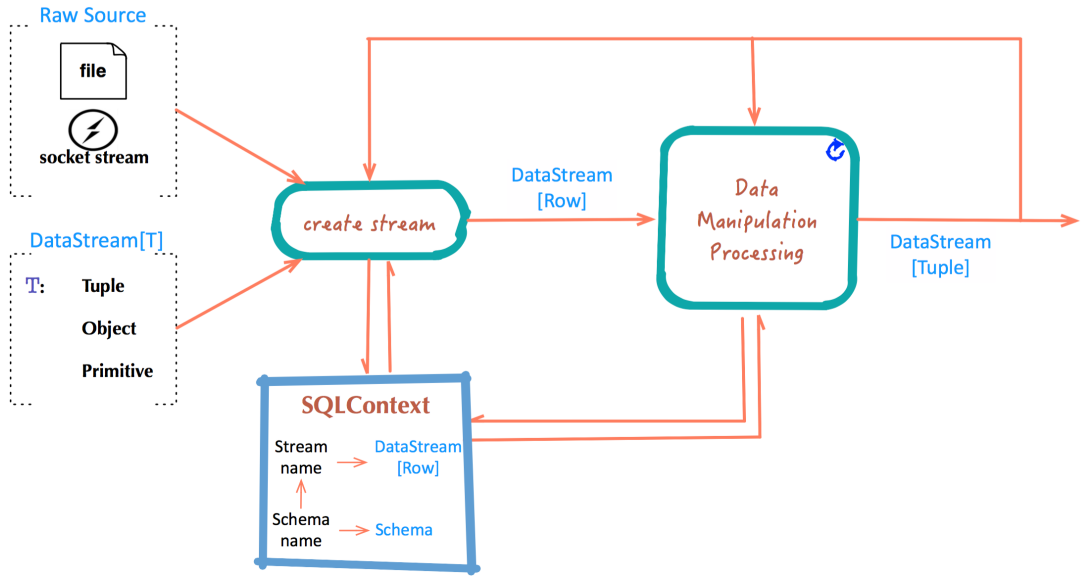
V. Implementation

1. Architecture



- **Source**: from Raw Source or pre-defined DataStream of type ***T*** in Flink. Type T could be a primitive data type (int, double, …) , a tuple, or an object type such as *Car(id: Int, speed: Int)*

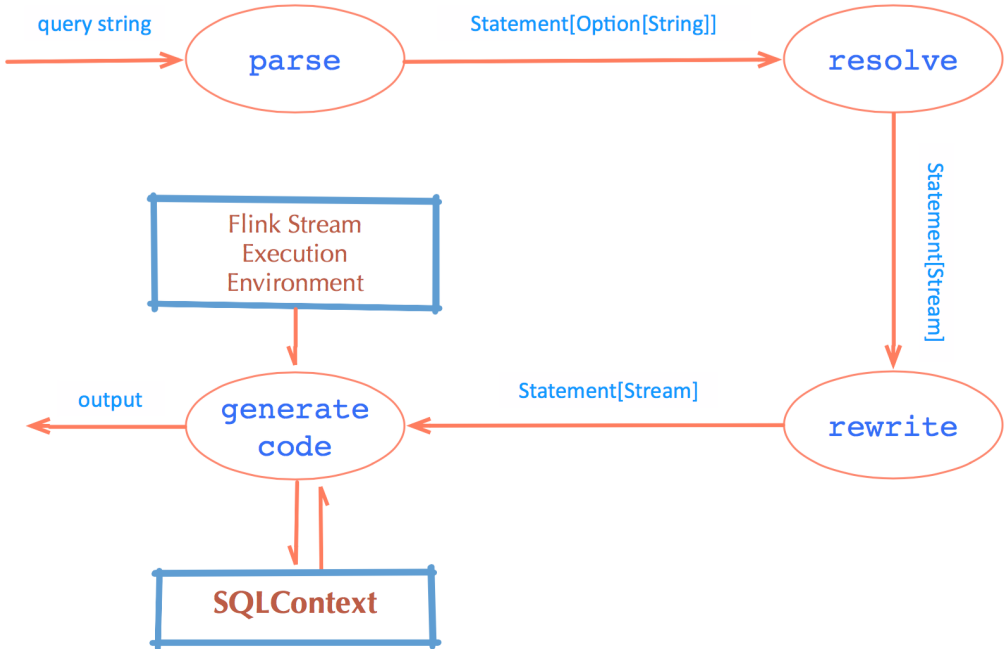
- **Create Stream :** the Source Stream will be transformed to a DataStream of a universal type *Row* . Row is simply a tuple containing an array of any data including Null. We do not specify the type of elements inside the Row. Those type will be derived from Schema.

All information about Schema and mapping between DataStream[Row] and Schema will be stored in *SQLContext*

- ***Data Manipulation Processing:*** DataStream[Row] will be fed into Data Manipulation Processing which consume a query string and then build up a chain of Operators. Those operators will process on the input DataStream[Row] and return a DataStream of tuple.

- **Iteration** : the output DataStream of tuple can be used to create another streams or feed into other queries.

2. Query Interpreter

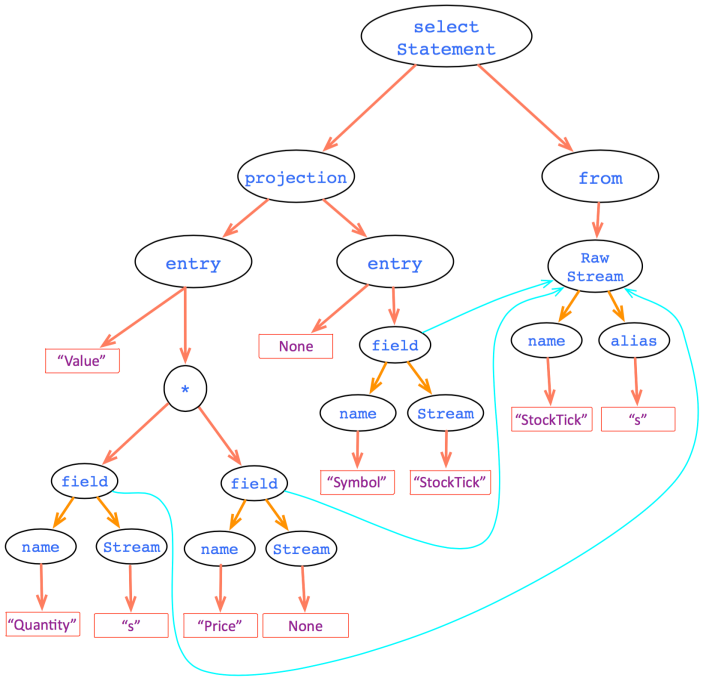


a. Step 1: Parsing

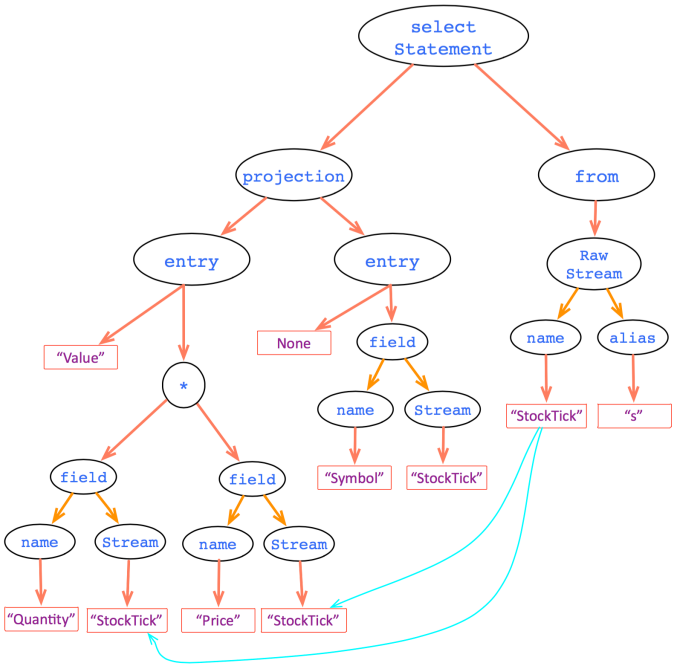
User issues a query string to system.
For example:

```
select StockTick.Symbol, (s.Quantity * price) as Value
From StockTick as s
```

We build up an Abstract Syntax Tree from input query string using a Parser



b. **Step 2: Resolving**



In the example, we are going to figure out which streams that "`StockTick.Symbol`", "`s.Quantity`", "`Price`" projections refer to. This step is important because perhaps many different streams have 2 fields which have an identical name. For example, 2 streams may own a field which is "Price".
This step helps to resolve which fields belongs to which streams so that we can generate the exact code for the query

c. **Step 3: Rewrite**

We apply one rewriting rule for subquery:

From (`select Symbol,Price From StockTick [Size 3]) as s`

=

From (`select Symbol,Price From StockTick) [Size 3] as s`

In the first query , its subquery will return a Windowed Stream which is not supported now. Therefore the query is invalid

In the second query, its subquery return a concrete DataStream which is acceptable.

However, the 2 queries are identical in term of semantics. Thus, we transform the first to the second.

d. **Step 4: Code generation**

Using Scala compile-time and run-time reflection to build a Scala Abstract Syntax Tree

3. Evaluation

Compare average runtime of

- 10 queries without subquery
- 10 queries with 1-layer nested query
- 10 queries with 2-layer nested query
- 10 queries with 3-layer nested query

4. Future Work

- Pattern matching
- Random Query Generator for Testing