Cloudera DataFlow for Data Hub 7.2.11

SQL Stream Builder

Date published: 2019-12-16
Date modified: 2021-09-09

https://docs.cloudera.com/

Legal Notice

Contents

Running a simple SQL job

You can use this Getting Started use case to get familiar with the most simple form of running a SQL Stream job.

About this task

The Getting Started contains the basic steps of running a SQL Stream job. When executing the job, you do not need to select a sink as the results are displayed in the browser. The SQL Stream Builder provisions a job on your cluster to run the SQL queries. You can select the Logs tab to review the status of the SQL job. As data is returned, it shows up in the Results tab.

Note: When adding SQL statements to the SQL window, you do not need to add the semicolon (;) at the end of the statement as SSB can run the command without the semicolons.

Before you begin

As the Getting Started is using the Stateful Tutorial as an example, you need to set up a Kafka topic as transaction.log.1 in Streams Messaging Manager, and submit the Kafka Data Generator job to generate data to the source topic. For more information, see the Stateful Tutorial.

Procedure

1. Navigate to the Streaming SQL Console.
   a) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
   b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
   c) Select Streaming SQL Console from the list of services.

   The **Streaming SQL Console** opens in a new window.
2. Click on Data Providers from the main menu.
3. Register a Kafka Provider.
4. Click on Console from the main menu.
5. Click on Tables tab.
6. Add a Kafka table.
   a) Name the Table to transactions
   b) Select the registered Kafka cluster.
   c) Select transaction.log.1 as the Kafka topic.
   d) Select JSON as Data Format.
   e) Click Detect Schema.
   SSB detects the schema and displays it to the Schema Definition field.
   f) Click Save Changes.
7. Click on Compose tab.
8. Provide a name for the SQL job in the SQL Job Name text box.

   Note: You can also use the random name button to generate a name for your job.

   Important: Do not add any sink to the SQL Job as the output is generated to the browser.
9. Add the following SQL statement to the SQL window:

```
SELECT * FROM transactions
```

10. Click on Execute.
    You can see the generated output in the Results tab.
11. Click on Stop to stop the previous query.
12. Add the following SQL statement to the SQL window:

```
SELECT itemId, quantity
FROM (
   SELECT itemId, quantity,
     ROW_NUMBER() OVER (
       ORDER BY '', quantity) AS rownum
   FROM transactions)
WHERE rownum <= 4
```

Related Information

Using Streaming SQL Console
Data Sources in SSB
Using Tables in SQL Stream jobs
Monitoring SQL Stream jobs

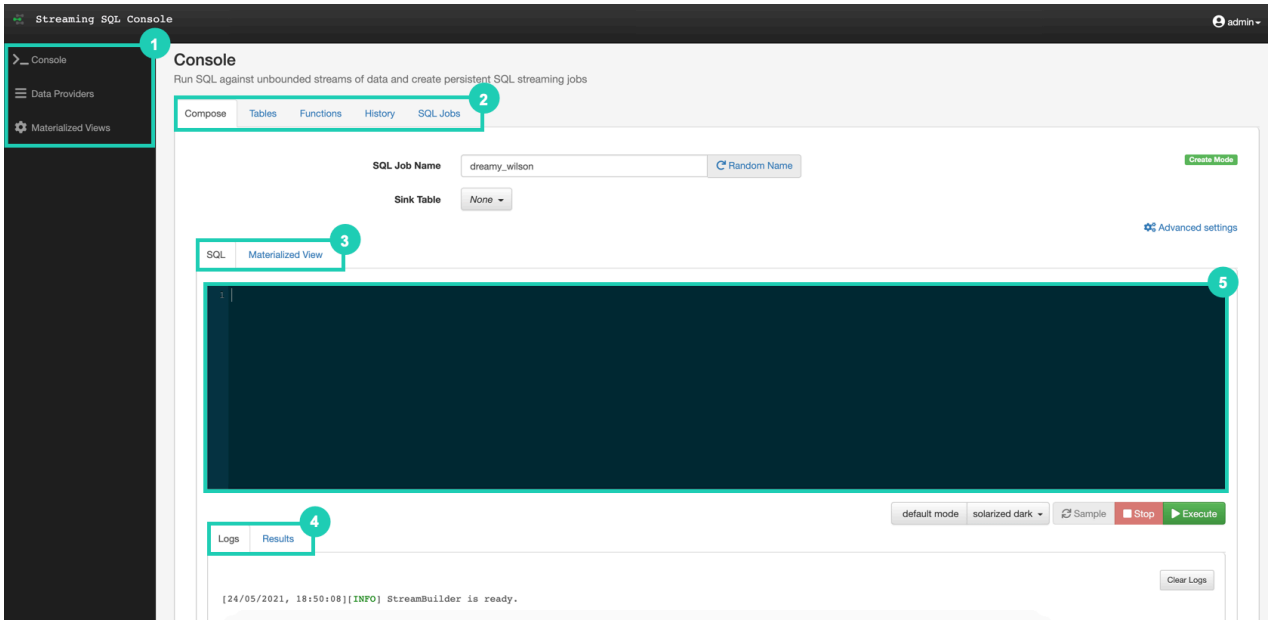
Using the Streaming SQL Console

The Streaming SQL Console is the user interface for the SQL Stream Builder. You can manage your queries, tables, functions and monitor the history of the SQL jobs using the SQL Stream Console.

1. Navigate to Management Console > Environments , and select the environment where you have created your cluster.
2. Select the Streaming Analytics cluster.
3. Click Streaming SQL Console from the services.

   The Streaming SQL Console opens in a new window.

The following illustration details the main menu and the tabs of the user interface.



1. The main menu consist of the following:
   - Console - The homepage of the Console where you can find the SQL window, other tabs and the Log window.
   - Data Providers - Adding and managing data providers for Tables.
   - Materialized Views - Setting and managing API keys.
2. The main tabs consist of the following:
   - Compose - By default, the Compose tab is selected on the Console. You can create your SQL queries on the Compose tab.
   - Tables - You can view and register Tables on this tab.
   - Functions - You can add the Javascript Functions to your SQL query.
   - History - You can review the history of submitted SQL queries.
   - SQL Jobs - You can review the history of submitted SQL jobs.
3. The alternate windows consist of the following:
   - SQL - By default, the SQL Window is displayed on the Console. You can add your SQL Queries to the window to compose queries.
   - Materialized View - Displays settings to create Materialized Views.
4. The audit tabs consist of the following:
   - Logs - Displays the status of the SQL Console.
   - Results - Displays the results of the executed SQL queries.
5. SQL Window - SQL statement editor of the Console.

Managing teams in Streaming SQL Console

You can manage your team, team members and invite new team members under the Teams menu on the SQL Stream Builder console.

About this task

By default, a new user is assigned to the SQLStreamBuilder team which is a default team for the administrator within SSB. Every user who can access the Streaming SQL Console on the same cluster, is automatically added and listed as Team Members in the SQLStreamBuilder team. Only the administrator has the privilege to change the access level for a team member, and inactivate-activate a team member from the SQLStreamBuilder team. Team members can create their own team. In this case, only the Team Owner can delete their team. A team can be deleted by the Team Owner of that certain team. A team cannot be deleted if it is a primary team of a user or it is the default team (SQLStreamBuilder).

Every team member in a team (SQLStreamBuilder team or user created team) can access the created Virtual Tables, User Defined Functions, Materialized Views and API keys within a team. A team member can also view the jobs submitted in a team they are a member of. A user can be a part of multiple teams, and can switch between them. On the Streaming SQL Console, the currently selected team is shown under the Active Team header.

A team member can invite other members to join their team. The invitation can be accepted or ignored. To view the invitation within a team click the View Team Invitations button.

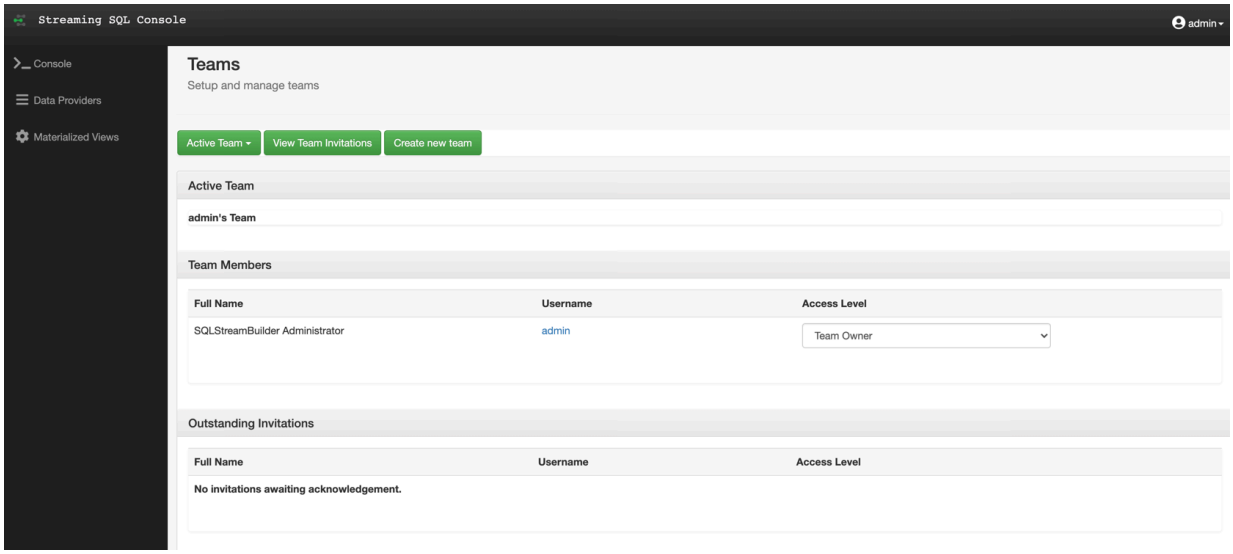
Procedure

1. Navigate to the Streaming SQL Console.
   a) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
   b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
   c) Select Streaming SQL Console from the list of services.

   The **Streaming SQL Console** opens in a new window.

2. Select your username.
3. Click Teams.

   You are redirected to the **Teams** page.



Registering Data Providers in SSB

Data Providers are a set of data endpoints to be used as sources, sinks and catalogs. Data Providers allow you to connect to an already installed component on your cluster, then use that provider for adding tables in SQL Stream Builder.

You can access the **Data Providers** page through the Streaming SQL Console:

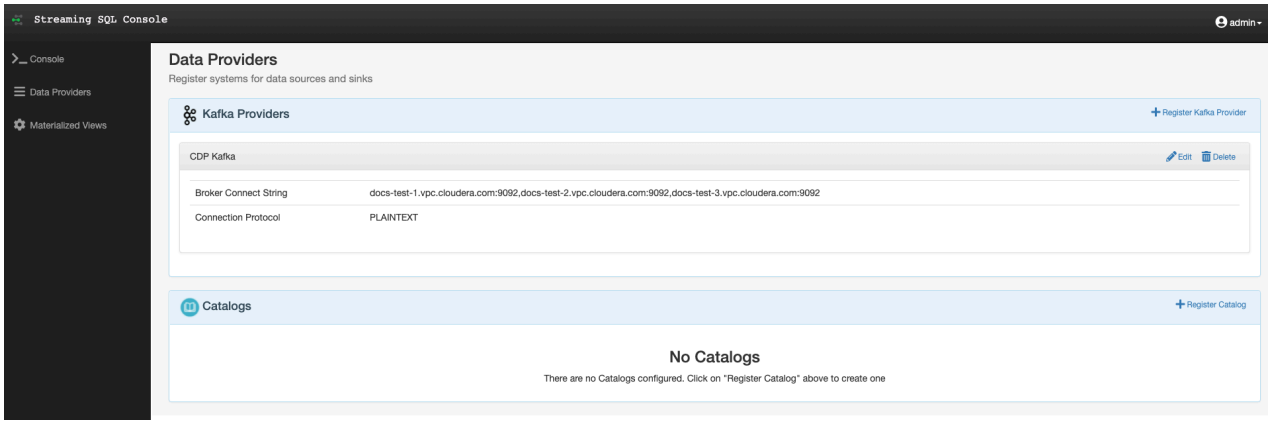
1. Navigate to Management Console > Environments , and select the environment where you have created your cluster.
2. Select the Streaming Analytics cluster.
3. Click Streaming SQL Console from the services.

   The Streaming SQL Console opens in a new window.
4. Click Data Providers on the main menu.

   You are redirected to the Data Providers page.

You can register Kafka as a data provider, or Kudu, Hive and Schema Registry as a catalog. When registering the components, SSB can access the already existing topics from Kafka, tables from Kudu and Hive, and the schema in Schema Registry. This also means that when you update a data provider, for example add new topics, tables and schemas, SSB automatically detects the changes.

You can also manage your data providers after registering them. You can Edit the providers if there is any change in the connection. You can also Delete them when you no longer need the specific provider.



Related Information

Adding Kafka as Data Provider
Adding catalogs as Data Provider

Managing registered Data Providers

You can edit or delete the registered Data Providers if you need to change their configurations or if you no longer need them.

Editing registered Data Providers

1. Click Data Providers from the main menu.
2. Search for the Kafka provider or catalog you want to modify.
3. Click Edit.

   The Edit Provider or Catalog window appears.
4. Change the settings as required.

   Note: You must validate the modified catalog before saving the changes.
5. Click Save Changes.

Deleting registered Data Providers

1. Click Data Providers from the main menu.
2. Search for the Kafka provider or catalog you want to modify.
3. Click Delete.
4. Click Confirm to delete the provider or catalog.

Integration with Kafka

The Kafka and SQL Stream Builder integration enables you to use the Kafka-specific syntax to customize your SQL queries based on your deployment and use case.

Performance & Scalability

You can achieve high performance and scalability with SQL Stream Builder, but the proper configuration and design of the source Kafka topic is critical. SQL Stream Builder can read a maximum of one thread per Kafka partition. You can achieve the highest performance configuration when setting the SQL Stream Builder threads equal to or higher than the number of Kafka partitions.

If the number of partitions is less than the number of SQL Stream Builder threads, then SQL Stream Builder has idle threads and messages show up in the logs indicating as such. For more information about Kafka partitioning, see the Kafka partitions documentation.

Kafka record metadata access

There are cases when it is required to access additional metadata from the Kafka record to implement the correct processing logic. SQL Stream Builder has access to this information using the Input Transforms functionality. For more information about Input Transforms, see the Input Transforms section.

The following attributes are supported in the headers:

```
record.topic
record.key
record.value
record.headers
record.offset
record.partition
```

For example, an input transformation can be expressed as the following:

```
var out = JSON.parse(record);
out['topic'] = message.topic;
out['partition'] = message.partition;
JSON.stringify(out);
```

For which you define a schema manually, or use the Detect Schema feature:

```
{
  "name": "myschema",
  "type": "record",
  "namespace": "com.cloudera.test",
  "fields": [
    {
      "name": "id",
      "type": "int"
    },
    {
      "name": "topic",
      "type": "string"
    },
    {
      "name": "partition",
      "type": "string"
    }
  ]
}
```

The attribute record.headers is an array that can be iterated over:

```
var out = JSON.parse(record);
var header = JSON.parse(record.headers);
var interested_keys = ['DC'];               // should match schema defini
tion

out['topic'] = record.topic;
out['partition'] = record.partition;
Object.keys(header).forEach(function(key) {
    if (interested_keys.indexOf(key) > -1){  // if match found for schema,
set value
        out[key] = header[key];
    }
```

```
});
JSON.stringify(out);
```

For which you define a schema as follows:

```
{
  "name": "myschema",
  "type": "record",
  "namespace": "com.cloudera.test",
  "fields": [
    {
      "name": "id",
      "type": "int"
    },
    {
      "name": "topic",
      "type": "string"
    },
    {
      "name": "partition",
      "type": "string"
    },
    {
      "name": "DC",
      "type": "string"
    }
  ]
}
```

Related Information

Creating Input Transforms

Using Tables in SQL Stream jobs

The core abstraction for Streaming SQL is a Table which represents both inputs and outputs of the queries. SQL Stream Builder tables are an extension of the tables used in Flink SQL to allow a bit more flexibility to the users.

A Table is a logical definition of the data source that includes the location and connection parameters, a schema, and any required, context specific configuration parameters. Tables can be used for both reading and writing data in most cases. You can create and manage tables either manually or they can be automatically loaded from one of the catalogs as specified using the Data Providers section.

In SELECT queries the FROM clause defines the table sources which can be multiple tables at the same time in case of JOIN or more complex queries.

For example:

```
SELECT
lat,lon
FROM
airplanes -- the name of the virtual table source
WHERE
icao <> 0;
```

When you execute a query, the results go to the Sink Table that you selected in the SQL window. This allows you to create aggregations, filters, joins, and so on, and then route the results to another table. The schema for the results is the schema that you created when you ran the query.

Supported tables in SSB

SSB supports different table types to ease development and access to all kinds of data sources. There are two main categories of tables, user defined tables and catalog tables. User defined tables need to be added manually and catalog tables are accessible automatically after registering a catalog provider.

User defined tables

The user defined tables need to be added manually using the Add Tables wizard of the Streaming SQL Console.

- Kafka Table

  Apache Kafka Tables represent data contained in a single Kafka topic in JSON or AVRO format. It can be defined using the Streaming SQL Console wizard. For more advanced use cases Kafka tables can be substituted by Flink DDL tables.
- Flink DDL Table

  Flink DDL tables represent tables created by using the standard Flink SQL CREATE TABLE/ CREATE VIEW syntax. This supports full flexibility in defining new or derived tables and views. You can either provide the syntax by directly adding it to the Flink DDL window or use one of the predefined DDL templates.
- Webhooks

  Webhooks can only be used as sink tables for SQL queries. The result of your SQL query is sent to a specified webhook.

Catalog tables

The catalog tables are automatically imported based on the catalog type you have registered on the Data Providers page. The following catalogs are supported in SSB:

- Schema Registry
- Kudu
- Hive
- Custom catalogs

Note: You cannot edit the properties of the already existing tables that are automatically imported from the catalogs. To distinguish between editable and not editable tables, in other words, user defined and catalog tables, the Edit and Delete table options are not available on the Tables page as the illustration shows below:

Related Information

Creating Kafka tables

Creating tables with Flink DDL

Managing time in SSB

Time attributes define how streams behave for time based operations such as window aggregations or joins. For Kafka tables you can use the Event Time tab to create source provided or user provided timestamp and watermarks. For other tables you can define time related attributes in the Flink DDL or directly in the SQL query. You can use timestamps that are already provided in the source or you can use custom timestamps.

Source-provided timestamps

Source-provided timestamps are inserted directly into the data stream by the source connector. This query uses the source-provided order_time field to perform a temporal join on multiple Kafka topics:

```
-- Table of orders
CREATE TABLE orders (
```

```
    order_id    STRING,
    price       DECIMAL(32,2),
    currency    STRING,
    order_time  TIMESTAMP(3),
    WATERMARK FOR order_time AS order_time
) WITH (/* ... */);

-- Table of currency rates
CREATE TABLE currency_rates (
    currency STRING,
    conversion_rate DECIMAL(32, 2),
    update_time TIMESTAMP(3),
    WATERMARK FOR update_time AS update_time
) WITH (/* ... */);
-- Event time temporal join to enrich orders with currencies
SELECT
     order_id,
     price,
     currency,
     conversion_rate,
     order_time,
FROM orders
LEFT JOIN currency_rates FOR SYSTEM TIME AS OF orders.order_time
ON orders.currency = currency_rates.currency
```

User-provided timestamps

You can also specify timestamps contained in the data stream itself. For example, if your schema includes a field called "order_time", it is possible to construct a query such as:

```
-- Table of orders
-- Converts order_time_string field to timestamp
CREATE TABLE orders (
    order_id    STRING,
    price       DECIMAL(32,2),
    currency    STRING,
    order_time_string STRING,
    order_time as to_timestamp(order_time_string),
    WATERMARK FOR order_time AS order_time
) WITH (/* ... */);
```

When an invalid timestamp is found in the stream (for example, NaN), the timestamp of the message is going to be 0. This way the message is excluded from the current window.

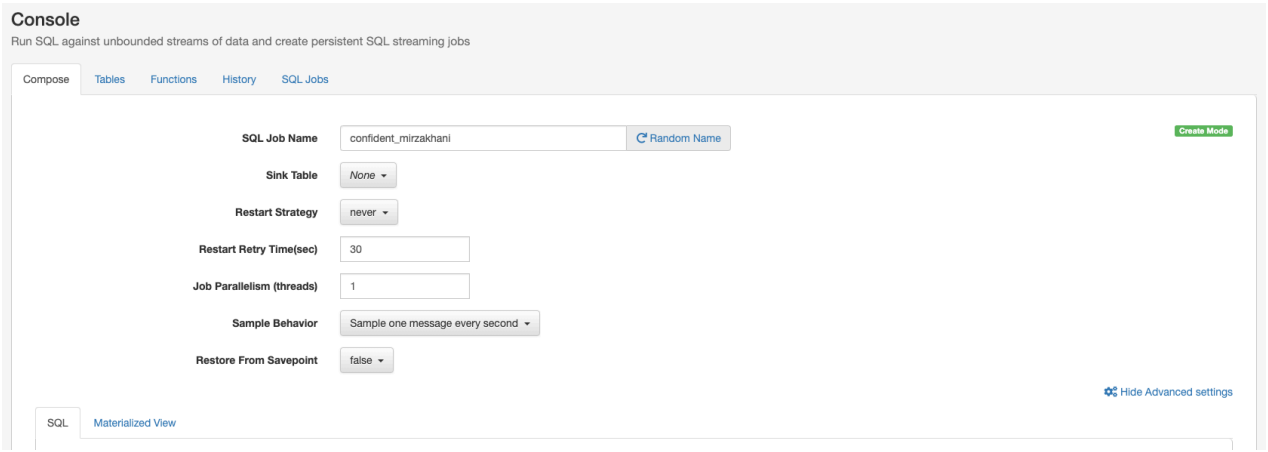
When your data does not include a timestamp in a suitable format, it is possible to compute a new timestamp column from another existing column using the Input Transform feature of SSB.

Job Lifecycle

Configuring advanced job management

If you need to further customize your SQL Stream job, you can add more advanced features to configure the job restarting method and time, threads for parallelism, sample behavior, exatly once processing and restoring from savepoint.

Before running a SQL query, you can configure advanced features by clicking on the Show Advanced Settings button on the Compose tab.



Restart strategy and restart retry time

The job is restarted after Restart Retry Time seconds, if set to Always. It is not restarted if you select Stop from the SQL Jobs tab. If set to Never, the job does not restart unless you select Restart from the Compose tab.

Job parallelism

The number of threads to start to process the job. Each thread consumes a slot on the cluster. When the Job Parallelism is set to 1, the job consumes the least resources. If the data provired supports parallel reads, increasing the parallelism can raise the maximum throughput. For example, when using Kafka as a data provider, setting the parallelism to the equal number as the partitions of the topic can be a starting point for performance tuning.

Sample behavior

How often to sample data (in milliseconds) from the output stream. 1000ms is common and recommended.

Restore From Savepoint

Enabling restoring from savepoint for the SQL job, using the state in Flink.

Stopping, restarting and editing SQL jobs

As a SQL Stream job processes streaming data, you need to stop the job to finish the process. You can restart a SQL Stream job after stopping it. In case you need to update or change the configurations that you have set for a SQL Stream job, you can restart it. You can navigate through the job life cycle using the Streaming SQL Console.

Stopping a SQL Stream job

You can stop a running job either on the **Compose** or the **SQL Jobs** page.
Stopping job from Compose page

1. Select Console from the main menu.

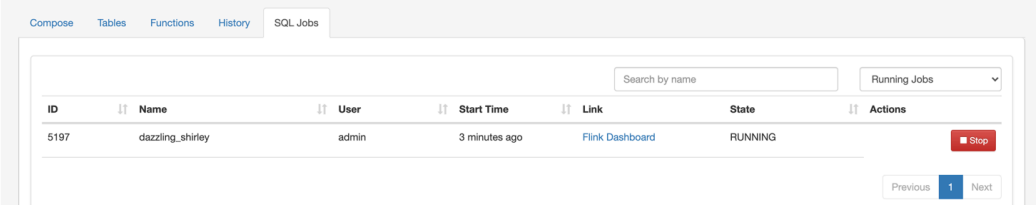
   By default, you are on the Compose page when selecting Console from the main menu.
2. Click Stop under the SQL window.

Stopping job from SQL Jobs page

1. Click Console on the main menu.

2. Select SQL Jobs tab.

   By default, Running Jobs are displayed on the **SQL Jobs** page.



3. Click on the job you want to stop.
   a. You can further filter down the results, by directly searching for the job name in the Search field.
4. Click Stop under Actions.

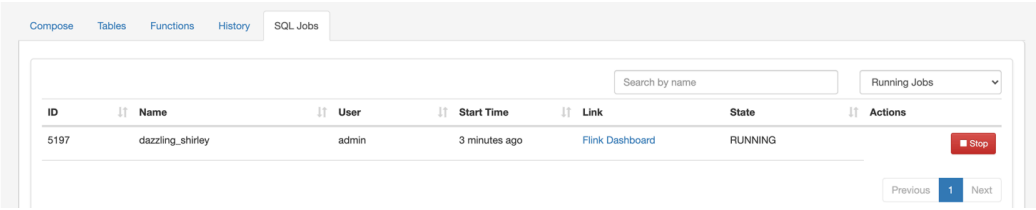
Restarting a running SQL Stream job

You can restart a running SQL job using the Restart button under the SQL window.

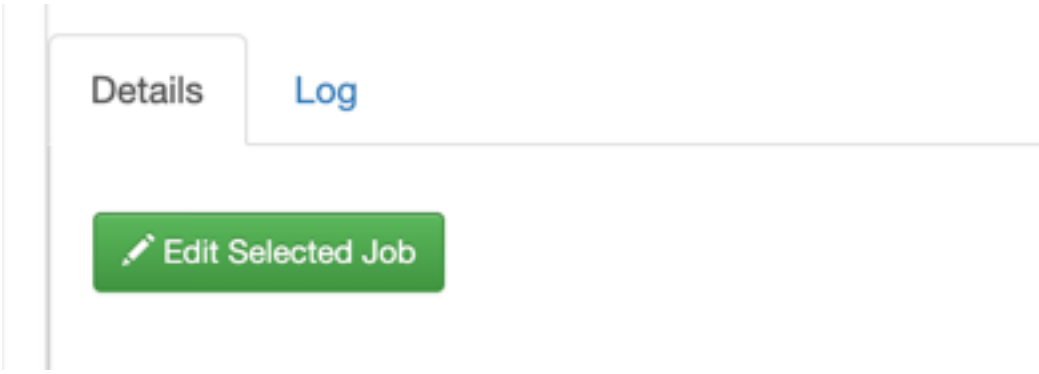
In case the job is running in the background, you can load it with its properties from the SQL Jobs tab.
Restarting job from SQL Jobs page

1. Click Console on the main menu.
2. Select SQL Jobs tab.

   By default, Running Jobs are displayed on the **SQL Jobs** page.



3. Click on the job you want to restart.
   a. You can further filter down the results, by directly searching for the job name in the Search field.
4. Click Edit Selected Job under the **Details** tab.



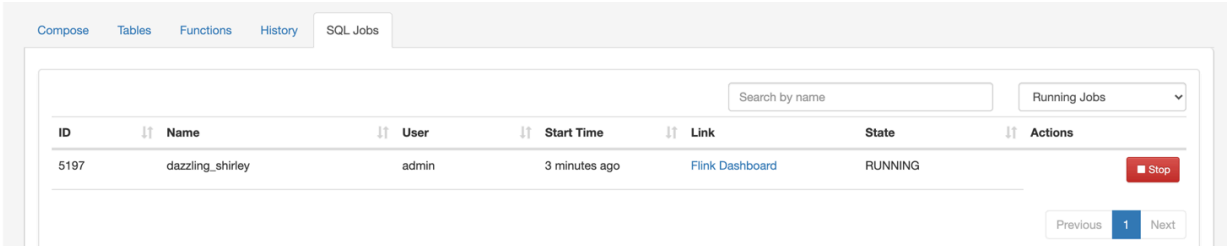
   The SQL job and its configuration is loaded in the SQL window on the **Compose** page.
5. Click Restart.

Restarting a stopped SQL Stream job

You can restart a SQL job that was previously stopped by locating it in the SQL Jobs page.

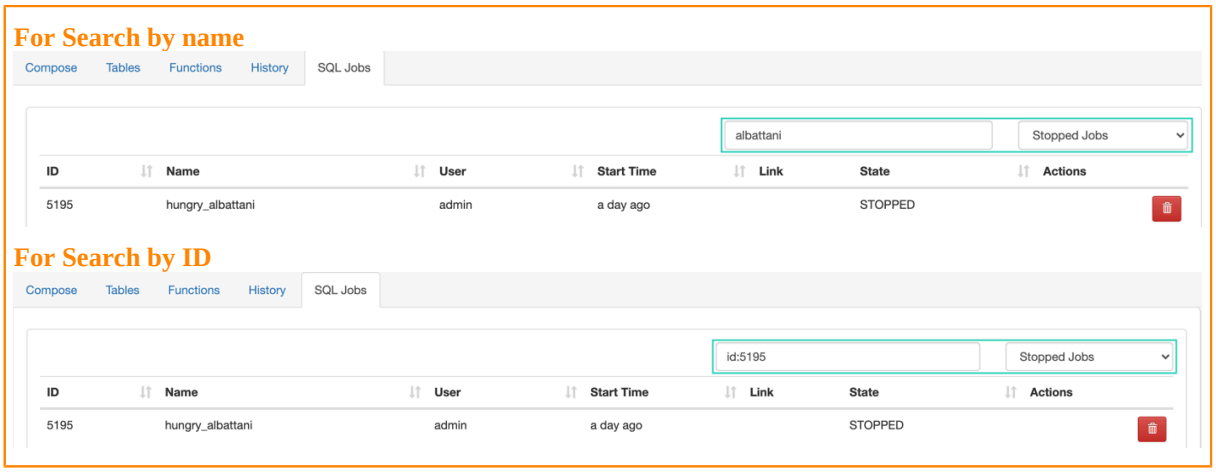
1. Click Console on the main menu.
2. Select SQL Jobs tab.

   By default, Running Jobs are displayed on the **SQL Jobs** page.

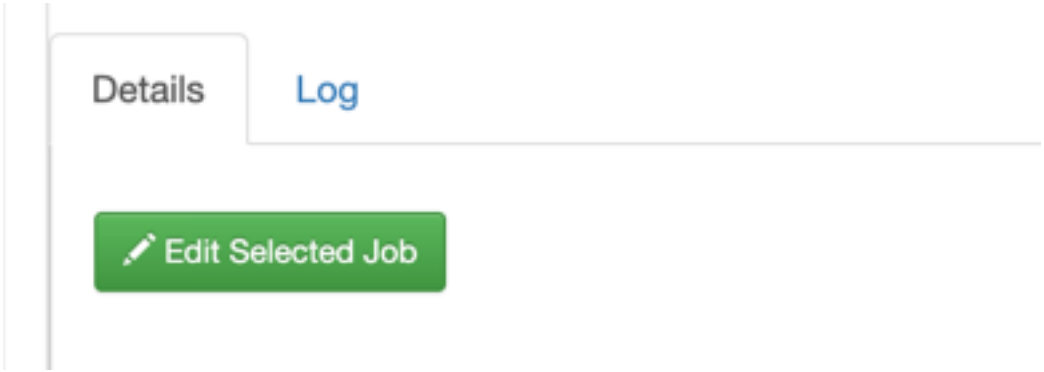


3. Select Stopped Jobs from the drop-down menu.
4. Click on the job you want to restart.

   a. You can further filter down the job list by searching for the job name, or locate a specific job ID by prefixing your search with id.



5. Click Edit Selected Job under the **Details** tab.



   The SQL job and its configuration is loaded in the SQL window on the **Compose** page.
6. Click on Execute.

Note: If you are using Materialized Views, the same Materialized View parameters will be selected. Make sure that the configured parameters are still appropriate for the job if the source schemas have been modified.

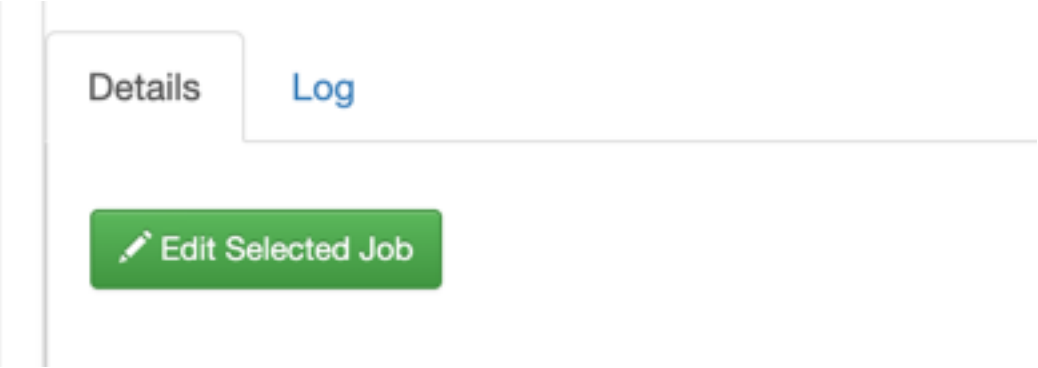
Editing a SQL Stream job

Before editing, you must stop the running SQL job. After modifying the properties of the job, you need to execute them again to apply the changes.

1. Click Console on the main menu.
2. Select SQL Jobs tab.

   By default, Running Jobs are displayed on the **SQL Jobs** page.
3. Select Stopped Jobs from the drop-down menu.
4. Click on the job you want to restart.

   a. You can further filter down the results, by directly searching for the job name in the Search field.
5. Click Edit Selected Job under the **Details** tab.



   The SQL job and its configuration is loaded in the SQL window on the **Compose** page.
6. Edit any configuration of the selected SQL job.

   You need to click on Advanced Settings to display more configuration of the SQL job.
7. Click on Execute.

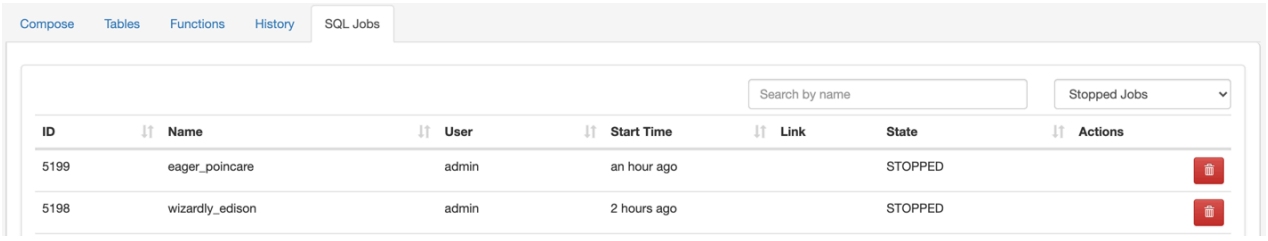
Deleting a SQL Stream job

You can delete a stopped SQL job from SQL Stream Builder on the SQL Jobs tab. By deleting the SQL job, you remove them from the list of stopped jobs.

1. Click Console on the main menu.
2. Select SQL Jobs tab.

   By default, Running Jobs are displayed on the **SQL Jobs** page.
3. Select Stopped Jobs from the drop-down menu.

   The list of stopped jobs is diplayed on the **SQL Jobs** page.
4. Click on Delete under Actions.



Sampling data for a running job

You can sample data from a running job. This is useful if you want to inspect the data to make sure the job is running correctly and producing the results you expect.

About this task

Sampling the results to your browser allows you to inspect the queried data and iterate on your query. You can sample 100 rows in the Results tab by clicking on the Sample button in the Console. In case you do not add any sink to the SQL job, the results automatically appear in the Results tab.

Procedure

1. Select Console on the main menu.
2. Go to the SQL Jobs tab.
3. Select the job you want to edit.
4. Go to the Details tab at the bottom.
5. Click Edit Selected Job.

   The SQL window in Edit Mode appears.
6. Click Sample.

Results

Sample results are displayed in the results window. If there is no data meeting the SQL query, sampling stops after a few attempts.

Creating a JavaScript function

With SQL Stream Builder, you can create user functions to write powerful functions in JavaScript, that you can use to enhance the functionality of SQL.

About this task

User functions can be simple translation functions like Celsius to Fahrenheit, more complex business logic, or even looking up data from external sources. User functions are written in JavaScript. When you write them, you create a library of useful functions.

Procedure

1. Navigate to the Streaming SQL Console.
   a) Navigate to Management Console > Environments , and select the environment where you have created your cluster.
   b) Select the Streaming Analytics cluster from the list of Data Hub clusters.
   c) Select Streaming SQL Console from the list of services.

   The **Streaming SQL Console** opens in a new window.
2. Select Console from the left-hand menu.
3. Select Functions > Add .
4. Name the function HELLO_WORLD, give it a short description, select JavaScript as the language.
5. Select STRING as output type and add STRING to the input type by selecting it from the list and clicking the plus button.
6. Paste this code into the JavaScript editor:

```
// check to see if the card is VISA
```

```
function HELLO_WORLD(card){
    var cardType = "Other";
    if (card.charAt(0) == 4){
      cardType = "Visa";
    }
    return cardType;
}
HELLO_WORLD($p0); // this line must exist
```

7. Click Save.
8. Once created, you can use a User Function in your SQL statement:

```
-- simple usage
SELECT HELLO_WORLD(card) AS IS_VISA
FROM ev_sample_fraud;

-- in the predicate
SELECT amount, card
FROM ev_sample_fraud
WHERE HELLO_WORLD(card) = "Visa";
```

Note: Valid inputs can be a field in the source virtual table or any other valid input. Functions must be in upper case.

Note: User Functions have access to the Java 8 API, this increases the overall usefulness and power. For example:

```
function GETPLANE(icao) {
  try {
    var c = new java.net.URL('http://yyyyyy.io' + icao).openConnec
tion();
    c.requestMethod='GET';
    var reader = new java.io.BufferedReader(new java.io.InputStreamRe
ader(c.inputStream));
    return reader.readLine();
  } catch(err) {
    return "Unknown: " + err;
  }
}
GETPLANE($p0);
```

Developing JavaScript functions

When developing JavaScript functions that are more complicated than just simple logic, it is recommended to use the jjs command-line utility to create and iterate while writing functions.

About this task

After the function performs the required task, migrate it to the console. Additionally these files/functions can be saved in a source code control system like git/Github.

Procedure

1. Create a file for your function.

Note: It is recommended to name the file with the same name as that of the function.

2. Create some sample input when calling the function.

3. Call jjs on the command line to test the function.

```
$>cat TO_EPOCH.js
function TO_EPOCH(strDate) {
   var strFmt = "yyyy-MM-dd HH:ss:mm";
   var c = new java.text.SimpleDateFormat(strFmt).parse(strDate).getTime()
/1000;
   return c.toString();
}

print(TO_EPOCH("2019-02-02 22:23:13"));

then
$>jjs TO_EPOCH.js
1549167203
```

What to do next

After you have successfully developed the JavaScript code, copy and paste only the function to your code window when creating the JavaScript function in SQL Stream Builder.

Monitoring SQL Stream jobs

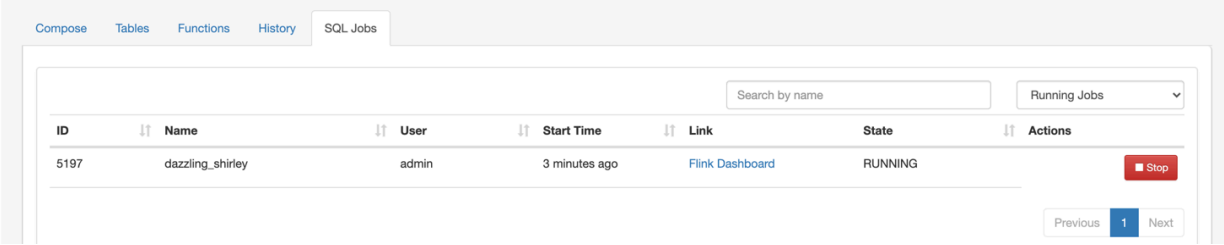
You can use the Streaming SQL Console to review the status, properties and log of your SQL Stream jobs executed in SQL Stream Builder. Using the Flink Dashboard, you can also monitor the Flink job that is submitted when you execute a SQL query.

Using the Streaming SQL Console

When using the **SQL Jobs** page in Streaming SQL Console to monitor your SQL jobs, you can review the ID, the Name, the Start time, the State of the submitted jobs, and the User who submitted the SQL jobs. When monitoring running jobs, you are also able to open the Flink Dashboard, and stop the job using the Stop button under Actions. The Flink Dashboard link and Stop button are not available for Stopped Jobs.

1. Click Console on the main menu.
2. Select SQL Jobs tab.

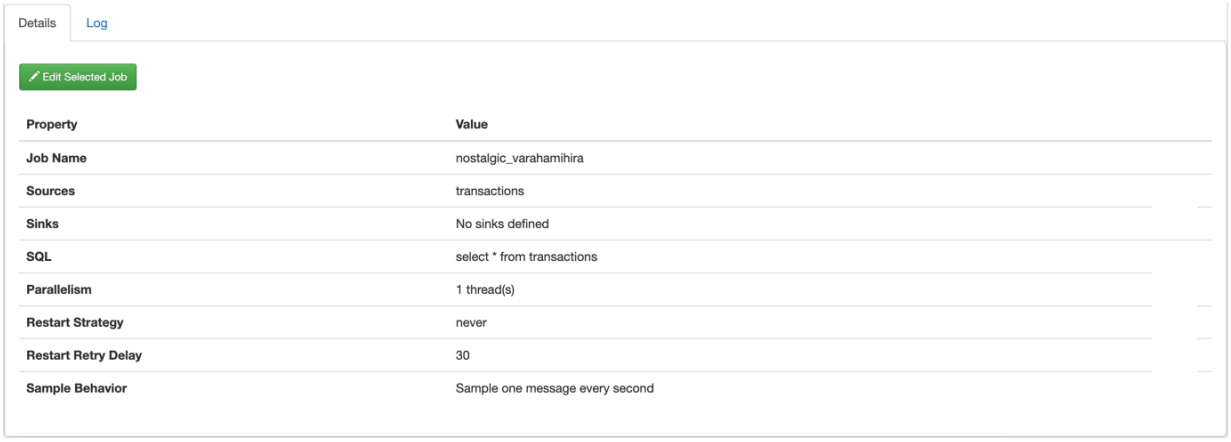
   By default, Running Jobs are displayed on the **SQL Jobs** page.



3. Select Running Jobs or Stopped Jobs from the drop-down menu.
4. Click on the job you want to monitor.

   a. You can further filter down the results, by directly searching for the job name in the Search field.

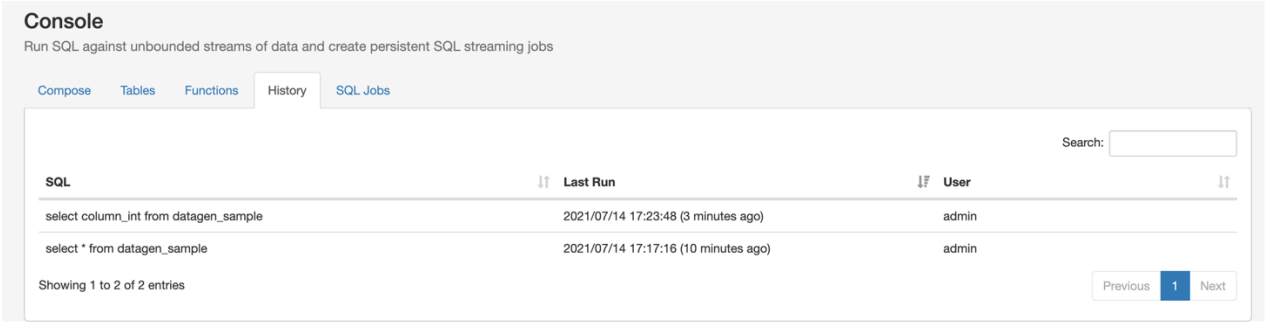
5. Select Details tab on the bottom panel to display additional details and configurations about the SQL job.



6. Select Log tab on the bottom panel to review the log information of the job submission.

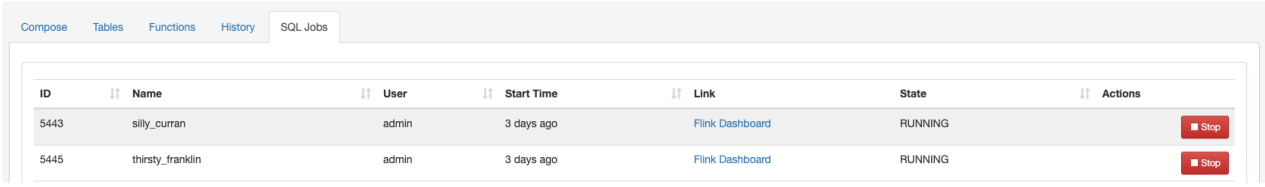
History of SQL queries

You can review and reuse the SQL queries that were previosuly executed on the **History** page. When you click on one of the SQL queries, it is automatically imported to the SQL window for execution. You can filter the SQL queries by the time they were last run or by the user who run them. You can also search for a type of SQL query using the Search field.



Using the Flink Dashboard

You can also monitor your running SQL jobs using the Flink Dashboard. You can easily reach the Flink Dashboard on the SQL jobs tab below the Console main menu. After clicking on the Flink Dashboard, you are redirected to the Flink Dashboard interface.



SQL Syntax Guide

The SQL Syntax Guide provides a collection of SQL syntaxes that is supported in SQL Stream Builder.

SQL Syntax

- Filtering WHERE
- Projections FOO || 'baz'
- Tumble, Hopping window expressions TUMBLE(), HOP()
- Grouping GROUP BY
- Filter after aggregation HAVING
- Join JOIN ON..
- Comparison operators
- Logical operators
- Arithmetic operators and functions
- Character string operators and functions
- Binary string operators and functions
- Date/Time functions
- Type conversion
- Aggregate functions
- Grouped window functions
- Grouped Auxiliary functions

Data types

- Datatypes

Metadata commands

- show tables - list virtual tables.
- desc <vtable> - describe the specified virtual table, showing columns and types.
- show jobs - list current running SQL jobs.
- show history - show SQL query history (only successfully parsed/executed).
- help - show help.

SQL Examples

You can use the SQL examples for frequently used functions, syntax and techniques in SQL Stream Builder (SSB). SSB uses Calcite Compatible SQL, but to include the functionality of Flink you need to customize certain SQL commands.

Metadata commands

```
-- show all tables
SHOW tables;
SHOW vtables;

-- describe or show schema for table
DESCRIBE payments;
DESC payments;
```

Timestamps, intervals and time

```
-- eventTimestamp is the Kafka timestamp
-- as unix timestamp. Magically added to every schema.
SELECT max(eventTimestamp) FROM solar_inputs;

-- make it human readable
SELECT CAST(max(eventTimestamp) AS varchar) as TS FROM solar_inputs;

-- date math with interval
SELECT * FROM payments
WHERE eventTimestamp > CURRENT_TIMESTAMP-interval '10' second;
```

Aggregation

```
-- hourly payment volume

SELECT SUM(CAST(amount AS numeric)) AS payment_volume,
CAST(TUMBLE_END(eventTimestamp, interval '1' hour) AS varchar) AS ts
FROM payments
GROUP BY TUMBLE(eventTimestamp, interval '1' hour);
```

```
-- detect multiple auths in a short window and
-- send to lock account topic/microservice

SELECT card,
MAX(amount) as theamount,
TUMBLE_END(eventTimestamp, interval '5' minute) as ts
FROM payments
WHERE lat IS NOT NULL
AND lon IS NOT NULL
GROUP BY card, TUMBLE(eventTimestamp, interval '5' minute)
HAVING COUNT(*) > 4 -- >4==fraud
```

Working with arrays

```
-- unnest each array element as separate row
SELECT b.*, u.*
 FROM bgp_avro b,
 UNNEST(b.path) AS u(pathitem)
```

Note: Arrays start at 1 not 0.

Union ALL

```
-- union two different tables
SELECT * FROM clickstream
WHERE useragent = 'Chrome/62.0.3202.84 Mobile Safari/537.36'
UNION ALL
SELECT * FROM clickstream
WHERE useragent = 'Version/4.0 Chrome/58.0.3029.83 Mobile Safari/537.36'
```

Math

```
-- simple math
SELECT 42+1 FROM mylogs;
```

```
-- inline
SELECT (amount+10)*upcharge AS total_amount
FROM payments
WHERE account_type = 'merchant'
```

```
-- convert C to F
SELECT (temp-32)/1.8 AS temp_fahrenheit
FROM reactor_core_sensors;
```

```
-- daily miles accumulator, 100:1
-- send to persistent storage microservice
-- for upsert of miles tally
SELECT card,
SUM(amount)/100 AS miles,
TUMBLE_END(eventTimestamp, interval '1' day)
FROM payments
GROUP BY card, TUMBLE(eventTimestamp, interval '1' day);
```

Joins

```
-- join multiple streams
SELECT o.name,
       sum(d.clicks),
       hop_end(r.eventTimestamp, interval '20' second, interval '40' second)
 FROM click_stream o join orgs r on o.org_id = r.org_id
                join models d on d.org_id = r.org_id
 GROUP BY o.name,
        hop(r.eventTimestamp, interval '20' second, interval '40' second)
```

```
-- join with temporal table where LatestRates is a temporal table
SELECT
  o.amount, o.currency, r.rate, o.amount * r.rate
FROM
  Orders AS o
  JOIN LatestRates FOR SYSTEM_TIME AS OF o.proctime AS r
  ON r.currency = o.currency
```

Hyperjoins

Joins are considered "hyperjoins" because SQLStreamBuilder has the ability to join multiple tables in a single query, and because a table is created from a data provider, these joins can span multiple clusters/connect strings, but also multiple types of sources (join Kafka and a database for instance).

```
SELECT us_west.user_score+ap_south.user_score
FROM kafka_in_zone_us_west us_west
FULL OUTER JOIN kafka_in_zone_ap_south ap_south
ON us_west.user_id = ap_south.user_id;
```

Misc SQL tricks

```
-- concatenation
SELECT 'testme_'||name FROM logs;
```

```
-- select the datatype of the field
SELECT eventTimestamp, TYPEOF(eventTimestamp) as mytype FROM airplanes;
```

Escaping and quoting

Typical escaping and quoting is supported.

- Nested columns

```
SELECT foo.`bar` FROM table; -- must quote nested column
```

- Literals

```
SELECT "some string literal" FROM mytable; -- a literal
```

Built In Functions

```
-- convert EPOCH time to timestamp
select EPOCH_TO_TIMESTAMP(1593718981) from ev_sample_fraud;

-- convert EPOCH milliseconds to timestamp
select EPOCHMILLIS_TO_TIMESTAMP(1593718838150) from ev_sample_fraud;
```