Cloudera DataFlow for Data Hub 7.2.14

Querying data with SQL Stream Builder

Date published: 2022-02-24
Date modified: 2022-02-24

https://docs.cloudera.com/

Legal Notice

Contents

Prepare your environment

Before you start querying your data using SQL Stream Builder (SSB) in CDP Public Cloud, you need to register and prepare your environment so that a chosen user or group can use the clusters and services in the environment securely.

As a first step, an admin needs to register an AWS, Azure or GCP environment. In CDP Public Cloud, this environment is based on the virtual private network in your cloud provider account. This means that CDP can access and identify the resources in your cloud provider account which is used for CDP services, and shared between clusters in the environment. After registering your AWS, Azure or GCP environment, you can provision clusters, and set users or groups to access your environment and services.

For more information about registering an AWS, Azure or GCP environment, see the Management Console documentation.

In CDP Public Cloud, the Streaming Analytics clusters use Apache Knox to authenticate users. Knox is automatically set up for your environment. To access the Streaming SQL Console, you need to upload your keytab to your cluster, and unlock the keytab in Streaming SQL Console. You also need to set Ranger policies for the SSB user when using Schema Registry, Kudu or Hive. You can further authorize the users using Team Management feature of SSB in the Streaming SQL Console

Make sure that you have completed the following prerequisites:

- You have a CDP Public Cloud environment.
- You have a CDP username (it can be your own CDP user or a CDP machine user) and a workload password set to access Data Hub clusters.

  The predefined resource role of this user is at least EnvironmentUser. This resource role provides the ability to view Data Hub clusters and set the FreeIPA password for the environment.
- Your user is synchronized to the CDP Public Cloud environment.
- You have a Streaming Analytics cluster.
- You have uploaded your keytab file to the Streaming Analytics cluster.
- You have unlock your keytab in the Streaming SQL Console.
- Your CDP user has the correct permissions set up in Ranger allowing access to SQL Stream Builder.

Assigning resource roles

As an administrator, you need to give permissions to users or groups to be able to access and perform tasks in your Data Hub environment.

Procedure

1. Navigate to Management Console > Environments and select your environment.
2. Click Actions > Manage Access .
3. Search for a user or group that needs access to the environment.
4. Select EnvironmentUser role from the list of Resource Roles.
5. Click Update Roles.
   The Resource Role for the selected user or group will be updated.
6. Navigate to Management Console > Environments , and select the environment where you want to create a cluster.
7. Click Actions > Synchronize Users to FreeIPA .

8. Click Synchronize Users.

Note: There might be cases where the status of the environment is synchornized with warnings and has failed status. This does not indicate that the synchronization has failed.

Creating IDBroker mapping

As an administrator, you must create IDBroker mapping for a user or group to access cloud storage. As a part of Knox, the IDBroker allows a user to exchange cluster authentication for temporary cloud credentials.

Procedure

1. Navigate to Management Console > Environments and select your environment.
2. Click Actions > Manage Access .
3. Click on the IDBroker Mappings tab.
4. Click Edit to add a new user or group and assign roles to have writing access for the cloud storage.
5. Search for the user or group you need to map.
6. Go to the IAM Summary page where you can find information about your cloud storage account.
7. Copy the Role ARN.
8. Go back to the IDBroker Mapping interface on the Cloudera Management Console page.
9. Paste the Role ARN to your selected user or group.
10. Click Save and Sync.

Setting workload password

As a user, you need to set a workload password for your EnvironmentUser account to be able to access the SQL Stream Builder nodes through SSH connection.

Procedure

1. Navigate to Management Console > Environments and select your environment.
2. Click Actions > Manage Access .
3. Click Workload Password.
4. Give a chosen workload password for your user.
5. Confirm the given password by typing it again.
6. Click Set Workload Password.

Creating Streaming Analytics cluster

As a user, you need to create the Streaming Analytics cluster in your environment to use SQL Stream Builder in Data Hub. You also need to add Streams Messaging, Real-time Data Mart, Data Engineering or Operational Database cluster based on your connector choice.

Procedure

1. Navigate to Management Console > Environments , and select the environment where you want to create a cluster.
2. Click Create Data Hub.
3. Select Streaming Analytics Light Duty cluster from Cluster Definition.
4. Provide a chosen cluster name.

5. Click Provision Cluster.

Configuring Ranger policies for SSB

You must add SQL Stream Builder (SSB) service user to the Ranger policies that are needed for Schema Registry, Hive and Kudu to provide access to topics, schemas and tables used by the components.

Before you begin

- Add the required Ranger policies for Flink. For more information, see Configuring Ranger policies for Flink documentation.

Procedure

1. Navigate to Management Console > Environments , and select the environment where you have created your cluster.
2. Click on the Data Lake tab.
3. Select Ranger from the services.
   You are redirected to the Ranger user interface.
4. Select the Streaming Analytics cluster under the needed service where you need to add the SSB user.

   Based on the connector type you want to use with SSB, you need to add the SSB user to the Schema Registry, Kudu or Hive service. The following minimal permissions are required for the SSB user:

Service	Policy name
Schema Registry	• all-schema-group, schema-metadata, schema-branch, schema-version
Kudu	• Database • Table • Column
Hive	• all-global • all-database, table, column • all-database, table • all-database • all-hiveservice • all-database, udf • all-url

Retrieving keytab file

As a user, you need to retrieve the keytab file of your profile and upload it to the Streaming SQL Console to be able to run SQL jobs.

Procedure

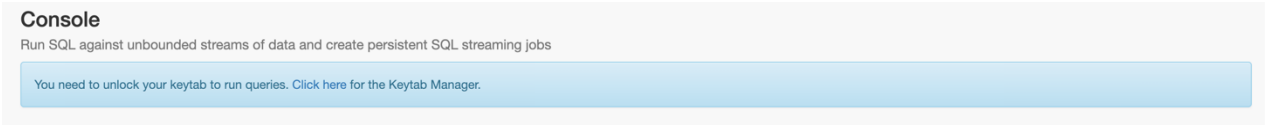
1. Navigate to Management Console > Environments , and select the environment where you have created your cluster.
2. Click on your profile name.
3. Click Profile.
4. Click Actions > Get Keytab.
5. Choose the environment where your Data Hub cluster is running.
6. Click Download.
7. Save the keytab file in a chosen location.

Uploading and unlocking your keytab in SSB

When accessing the Streaming SQL Console for the first time in Data Hub, you must upload and unlock the keytab file corresponding with your profile before you can use SQL Stream Builder (SSB).

About this task

The following message is displayed on the Console in case the keytabs are still locked:



Before you begin

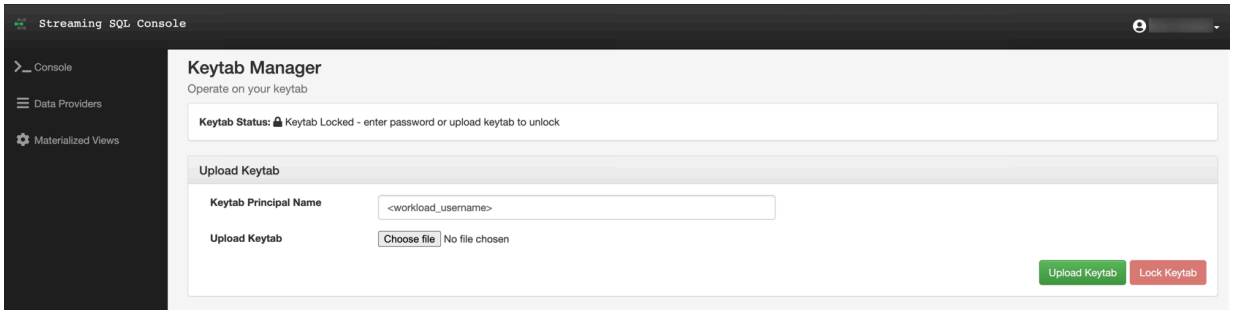
- Retrieve the keytab file for your user profile. For more information, see the *Retrieving keytab file* section.

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 your username at the right top corner of the Streaming SQL Console.
3. Click Manage keytab.

   You are redirected to the **Keytab Manager** page.



4. Click Choose file to upload your keytab file.
5. Search and select your keytab file.
6. Click Upload Keytab.
7. Click Unlock Keytab.

   In case there is an error when unlocking your keytab, you can get more information about the issue with the following steps:

   a. Manually upload your keytab to the Streaming Analytics cluster:

   ```
   scp <location>/<your_keytab_file> <workload_username>@<manager_node_FQDN
   >:.
                        Password:<your_workload_password>
   ```

   b. Access the manager node of your Streaming Analytics cluster:

   ```
   ssh <workload_username>@<manager_node_FQDN>
   ```

```
Password: <workload_password>
```

c. Use kinit command to authenticate your user:

```
kinit -kt <keytab_filename>.keytab <workload_username>
```

d. Use the flink-yarn-session command to see if the authentication works properly:

```
flink-yarn-session -d \
-D security.kerberos.login.keytab=<keytab_filename>.keytab \
-D security.kerberos.login.principal=<workload_username>
```

In case the command fails, you can review the log file for further information about the issue.

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. 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: Before using SQL Stream Builder, review the *Streaming Analytics* sections in the Release Notes as limitations may apply to the current version of Streaming Analytics.

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. Provide a name for the SQL job in the SQL Job Name text box.

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

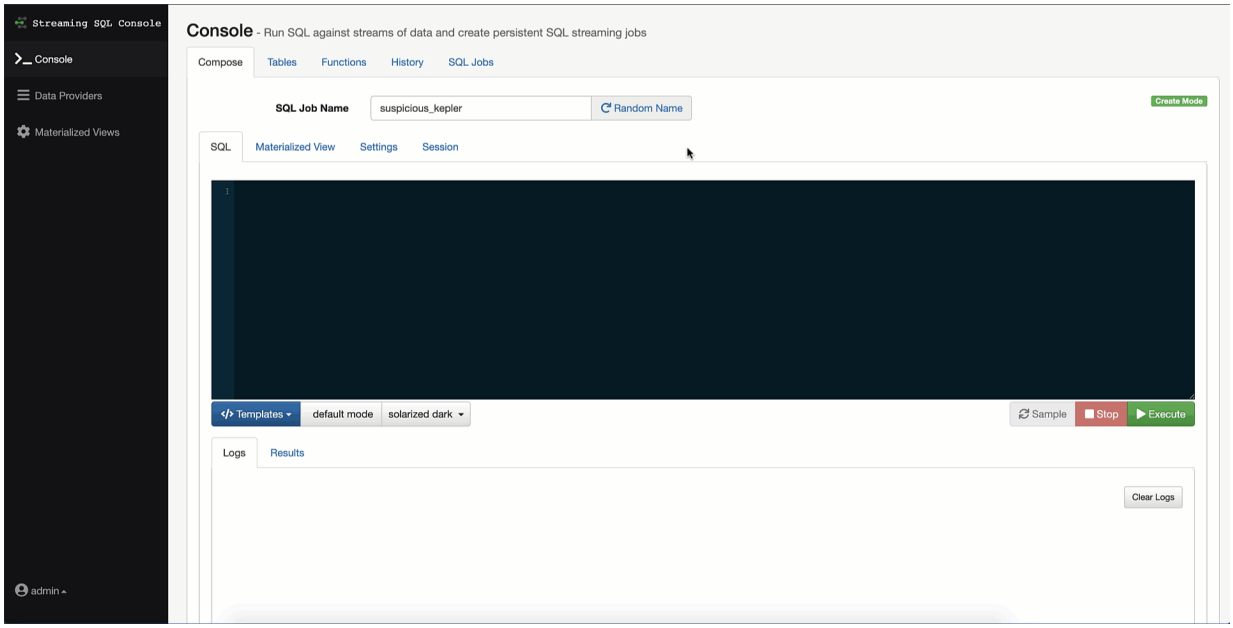
   Important: Do not write the data out to any table using the INSERT INTO statement, as the output is generated to the browser.
3. Select *datagen* from the Templates.

   The CREATE TABLE statement is imported to the SQL window.
4. Add a SELECT query to the SQL window after the datagen template.

   ```
   SELECT * FROM <table_name>
   ```

   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.
5. Click Execute.
   You can see the generated output on the Results tab.

6. Click on the Stop button to stop the job.



Creating Kafka tables in SSB

You can query your streaming data using Kafka tables in SQL Stream Builder (SSB). You have the option to use the Kafka service in your environment, or connect to an external Kafka service. When creating Kafka tables you can use the Console wizard, the predefined templates or you can directly add a custom CREATE TABLE statement with the required properties in the SQL window.

Before you begin

- You have prepared your environment based on the checklist in the *Prepare your environment* section.
- You have set Ranger policies for Flink and SSB.
- You have created a Streams Messaging cluster in your environment.
- You have created a Kafka topic from which you can read messages from in SSB.

Adding Kafka as Data Provider

You need to register Kafka as a Data Provider using the Streaming SQL Console to create Kafka tables in SQL Stream Builder (SSB).

Before you begin

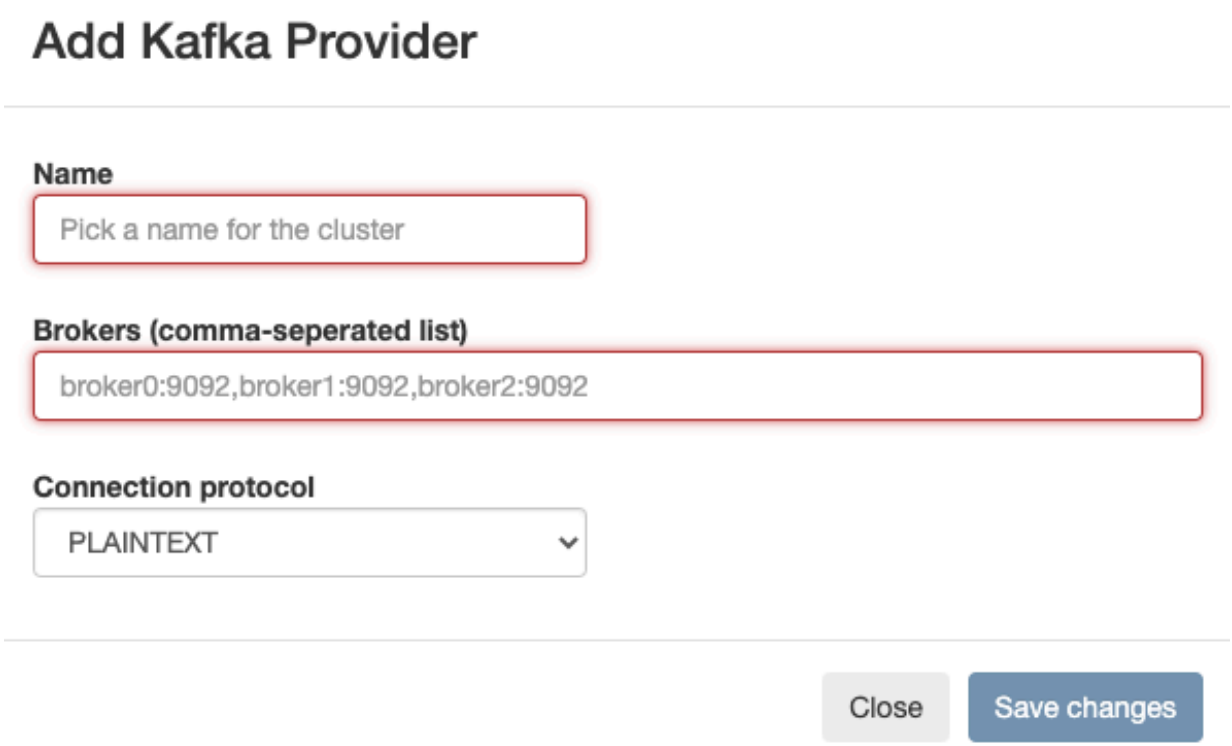
- Make sure that you have Kafka service on your cluster.
- Make sure that you have the right permissions set in Ranger.

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 Data Providers from the main menu.
3. Click Register Kafka Provider.

   The Add Kafka Provider window appears.



4. Add a Name to your Kafka provider.
5. Add the broker host name(s) to Brokers.

   You need to copy the Kafka broker name(s) from Cloudera Manager.
   a) Go to your cluster in Cloudera Manager.
   b) Click Kafka from the list of services.
   c) Click Instances.
   d) Copy the hostname of the Kafka broker(s) you want to use.
   e) Go back to the Add Kafka Provider page.
   f) Paste the broker hostname to the Brokers field.

   Note: You can add more than one broker hostname by separating them by commas.

   g) Add the default Kafka port after the hostname(s).

   Example:

```
docs-test-1.vpc.cloudera.com:9092,
docs-test-2.vpc.cloudera.com:9092
```

6. Select the Connection Protocol.

   The connection protocol must be the same as it is configured for the Kafka cluster in Cloudera Manager.

   You can choose from the following protocols:

   a) Select Plaintext, and click Save Changes.
   b) Select SSL, and click Save Changes.
   c) Select SASL/SSL, and choose an SASL Mechanism.

      1. Select Kerberos, and provide the Kafka Truststore location. Click Save Changes.
      2. Select Plain, and provide the SASL username and password. Click Save Changes.

Results

You have registered Kafka as a data provider to be able to add Kafka as a table in your SQL query. The already existing Kafka topics can be selected when adding Kafka as a table.

Creating Kafka tables

You have the option to create Kafka tables using the Console wizard, the built-in templates or directly adding a custom CREATE TABLE statement with the required properties in the SQL window.

Creating Kafka tables using Console wizard

After registering a Kafka data provider, you can use the Add table wizard in Streaming SQL Console to create a Kafka table.

Before you begin

- Make sure that you have registered Kafka as a Data Provider.
- Make sure that you have created topics in Kafka.

  Important: When creating the topic for the Kafka sink, make sure to not use log compaction as it can cause the SQL job to fail.

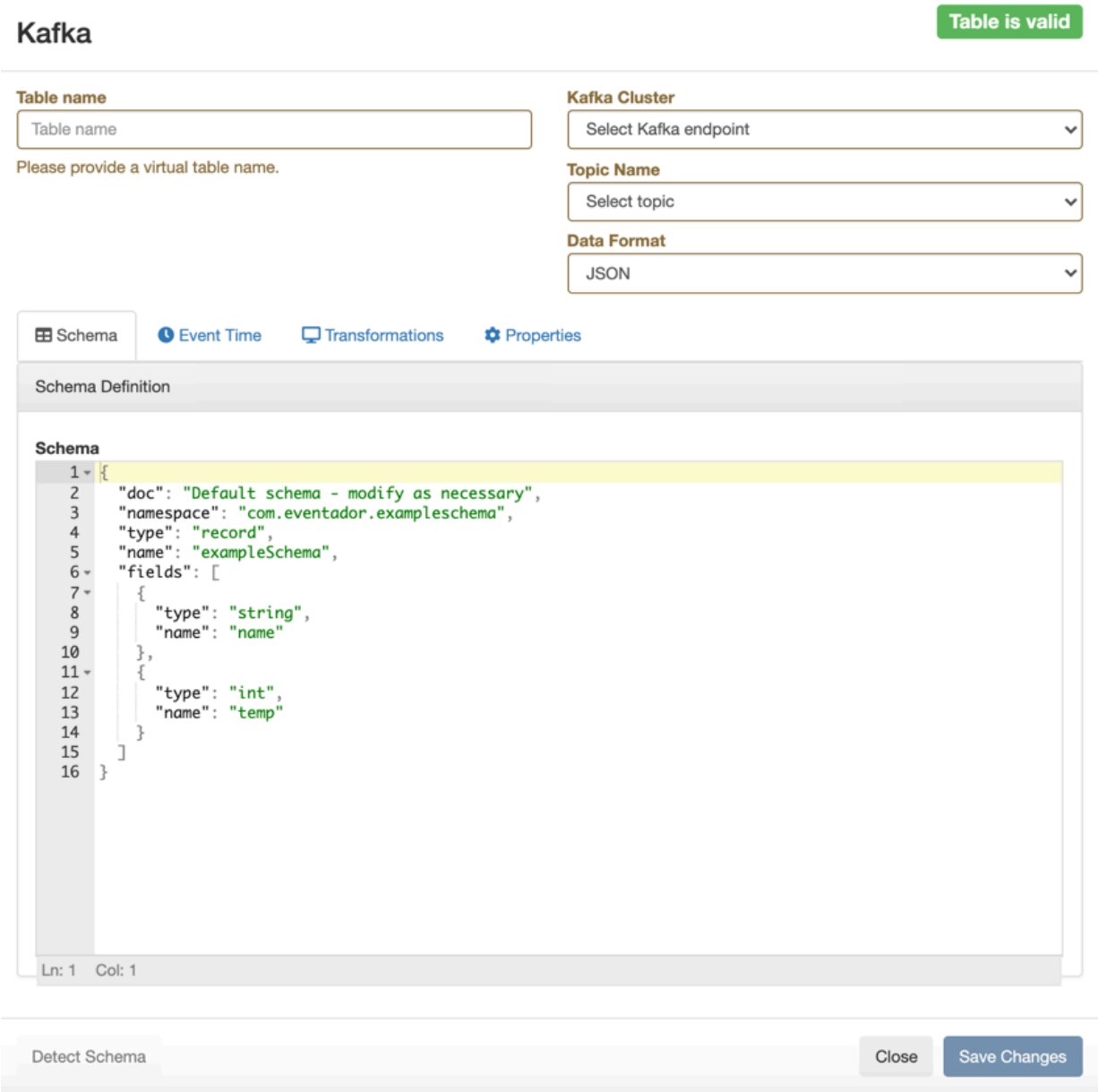
- Make sure there is generated data in the Kafka topic.
- Make sure that you have the right permissions set in Ranger.

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- side menu.
3. Go to the Tables tab.

4. Select Add table > Apache Kafka .
   The Kafka Table window appears.



5. Provide a Name for the Table.

   Note: You will use this name in the FORM clause when running the SQL statement.

6. Select a registered Kafka provider as Kafka cluster.
7. Select a Kafka topic from the list.

   Note: The automatically created topics for the websocket output is also listed here. Select the topic you want to use for the SQL job.

8. Select the Data format.
   - You can select JSON as data format.
   - You can select AVRO as data format.

   Note: You can only select the AVRO format when Schema Registry is used.

9. Determine the Schema for the Kafka table.
   a) Add a customized schema to the Schema Definition field.
   b) Click Detect Schema to read a sample of the JSON messages and automatically infer the schema.

   Note: If there are no messages in the topic, then no schema will be inferred.

10. Customize your Kafka Table with the following options:
    a) Configure the Event Time if you do not want to use the Kafka Timestamps.
       1. Unselect the checkbox of Use Kafka Timestamps.
       2. Provide the name of the Input Timestamp Column.
       3. Add a name for the Event Time Column.
       4. Add a value to the Watermark Seconds.
    b) Configure an Input Transform, add the code using the Transformations tab.
    c) Configure any Kafka properties required using the Properties tab.

    For more information about how to configure the Kafka table, see the Configuring Kafka tables section.
11. Select Save Changes.

Results
The Kafka Table is ready to be used for the SQL job either at the FROM or at the INSERT INTO statements.

Configuring Kafka tables
The user defined Kafka table can be configured based on the schema, event time, input transformations and other Kafka specific properties.

Schema tab

Schema is defined for a given Kafka source when the source is created. The data contained in the Kafka topic can either be in JSON or AVRO format.

When specifying a schema you can either paste it to the Schema Definition field or click the Detect schema button to identify the schema used on the generated data. The Detect Schema functionality is only available for JSON data.

If the schema of the Kafka table where the output data is queried is not known at the time of adding the table, you can select the Dynamic Schema option. This is useful when you want to insert data to the table, and there is no guarantee that the input schema matches with the output schema. If you select the Dynamic Schema option when adding a table, you can only use that table as a sink.

Note: If your schema contains a field named timestamp, this causes a schema validation error as timestamp is a reserved word used for Kafka internal timestamps.

Event Time tab

You can specify Watermark Definitions when adding a Kafka table. Watermarks use an event time attribute and have a watermark strategy, and can be used for various time-based operations. The Event Time tab provides the following properties to configure the event time field and watermark for the Kafka stream:

- Input Timestamp Column: name of the timestamp column in the Kafka topic from where the watermarks are mapped to the Event Time Column of the Kafka table

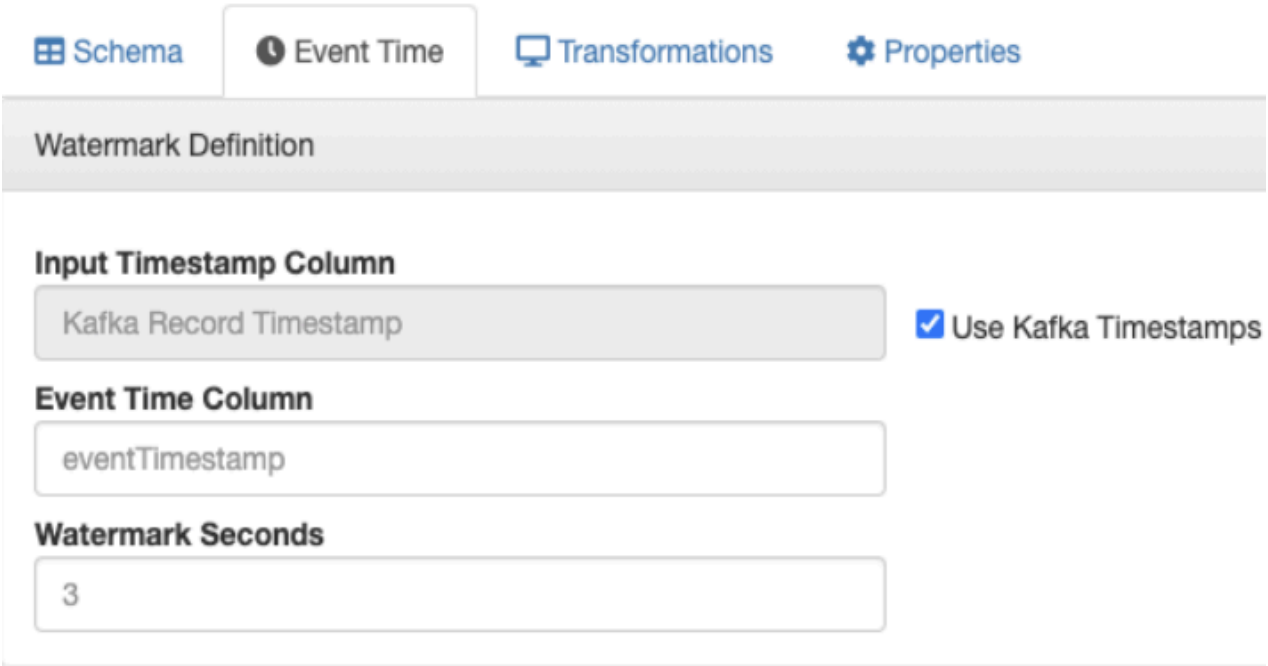
- Event Time Column: default or custom name of the resulting timestamp column where the watermarks are going to be mapped in the created Kafka table
- Watermark seconds: number of seconds used in the watermark strategy. The watermark is defined by the current event timestamp minus this value.

You have the following options to configure the watermark strategy for the Kafka tables:

- Using the default Kafka Timestamps setting
- Using the default Kafka Timestamps setting, but providing custom name for the Event Time Column
- Not using the default Kafka Timestamps setting, and providing all of the Kafka timestamp information manually
- Not using watermark strategy for the Kafka table

Using the default Kafka Timestamp setting

By default, the Use Kafka Timestamps checkbox is selected when you create the Kafka table. In the Event Time Column, the new event time field is extracted from the Kafka message header with the *'eventTimestamp'* predefined column name.



After saving your changes, you can view the created DDL syntax for the Table on the right side under the DDL tab. You can review the generated watermark strategy for your table that was set on the Watermark Definition tab.

The following DDL example shows the default setting of the Event Time Column and Watermark Seconds where the corresponding fields were not modified.

```
'eventTimestamp' TIMESTAMPS(3) METADATA FROM 'timestamp',
WATERMARK FOR 'eventTimestamp' AS 'eventTimestamp' - INTERVAL '3' SECOND
```

Using the default Kafka Timestamp setting with custom Event Time Column name

When you want to modify the timestamp field of the DDL from the stream itself, you must provide a custom name of the Event Time Column. You can also add a custom value to the Watermark Seconds. The following example shows that *'ets'* is the custom name for the Event Time Column, and *'4'* is the custom value for the Watermark Seconds.

The Event Time Column can only be modified if the following requirements are met for the timestamp field of the Input Timestamp Column:

- The column type must be "long".
- The format must be in epoch (in milliseconds since January 1, 1970).

The DDL syntax should reflect the changes made for the watermark strategy as shown in the following example:

```
'ets' TIMESTAMP(3) METADATA FROM 'timestamp',
WATERMARK FOR 'ets' - INTERVAL '4' SECOND
```

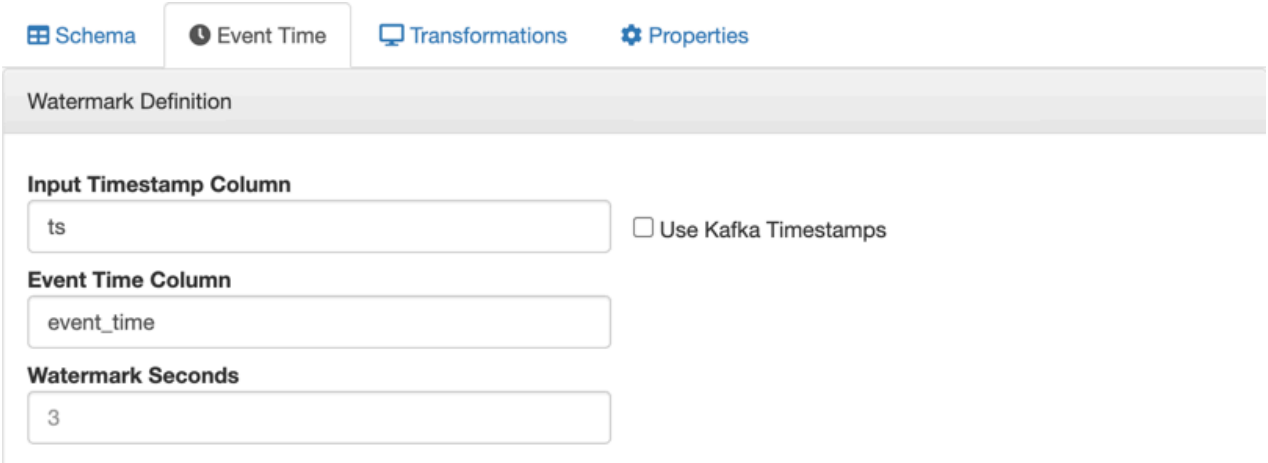
Manually providing the Kafka timestamp information

When you want to manually configure the watermark strategy of the Kafka table, you can provide the timestamp column name from the Kafka source, and add a custom column name for the resulting Kafka table. Make sure that you provide the correct column name for the Input Timestamp Column that exactly matches the column name in the Kafka source data.

To manually provide information for the watermark strategy, unselect the Use Kafka Timestamps checkbox, and provide the following information to the column name fields:

- Input Timestamp Column: name of the timestamp field in the Kafka source
- Event Time Column: predefined *'eventTimestamp'* name or custome column name of the timestamp field in the created Kafka table

As an example, you have a timestamp column in the source Kafka topic named as *'ts'*, and want to add a new timestamp column in your Kafka table as *'event_time'*. You provide the original timestamp column name in the Input Timestamp Column as *'ts'*, and add the custom *'event_time'* name to the Event Time Column.



This results in that the watermarks from the *'ts'* column is going to be mapped to the *'event_time'* column of the created Kafka table. As *'event_time'* will become the timestamp column name in the Kafka table, you must use the custom name (in this example the *'event_time'*) when querying the Kafka stream. This configuration of the timestamp columns are optional.

The Event Time Column can only be modified if the following requirements are met for the timestamp field of the Input Timestamp Column:

- The column type must be "long".
- The format must be in epoch (in milliseconds since January 1, 1970).

Not using watermark strategy for Kafka table

In case you do not need any watermark strategies, unselect the Use Kafka Timestamps checkbox, and leave the column and seconds field empty.

Note: Flink validates the input and output schemas of the data. You can only insert data into a Kafka topic, if the input and output data matches based on the schema defined for the topic. When customizing the timestamp column, make sure that the output data has the same schema.

Note: When configuring the timestamp for the Kafka tables, you must consider the timezone setting of your environment as it can effect the results of your query. For more information, see the Known Issues in the Release Notes.

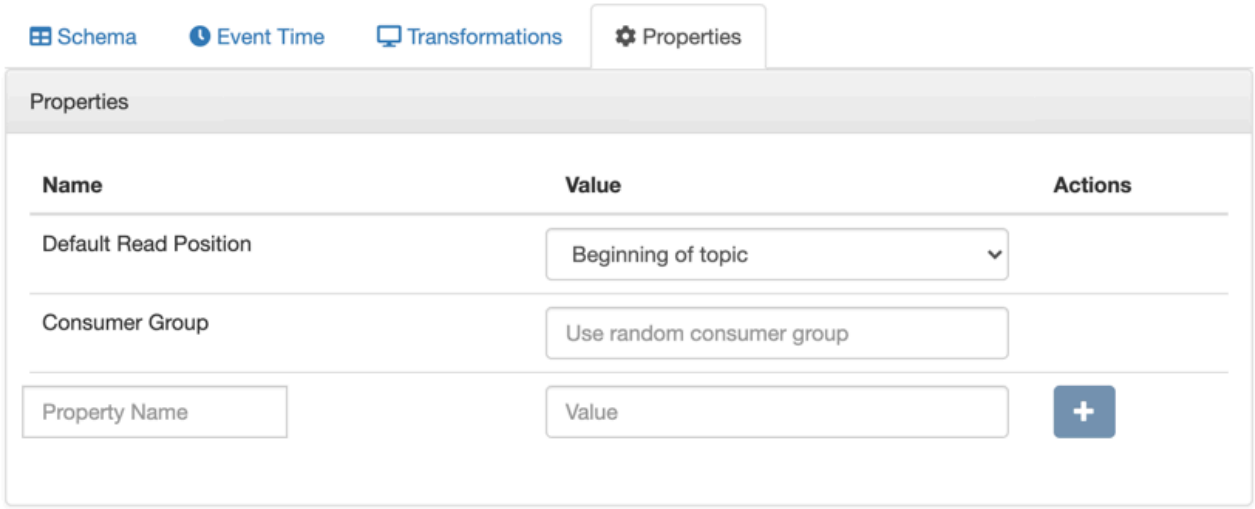
Input Transform tab

You can apply input transformations on your data when adding a Kafka table as a source to your queries. Input transformations can be used to clean or arrange the incoming data from the source using javascript functions.

For more information about Input Transform, see the Creating input transformations document.

Properties tab

You can specify certain properties to define your Kafka source in detail. You can also add customized properties additionally to the default ones. To create properties, you need to give a name to the property and provide a value for it, then click Actions.



Assigning Kafka keys in streaming queries

Based on the Sticky Partitioning strategy of Kafka, when null keyed events are sent to a topic, they are randomly distributed in smaller batches within the partitions. As the results of the SQL Stream queries by default do not include a key, when written to a Kafka table, the Sticky Partitioning strategy is used. In many cases, it is useful to have more fine-grained control over how events are distributed within the partitions. You can achieve this in SSB by configuring a custom key based on your specific workload.

For example:

```
SELECT sensor_name AS _eventKey --sensor_name becomes the key in the output
kafka topic
FROM sensors
WHERE eventTimestamp > current_timestamp;
```

To configure keys in DDL-defined tables (those that are configured using the Templates), refer to the official Flink Kafka SQL Connector documentation for more information (specifically the key.format and key.fields options).

Related Information

Creating Input Transforms

Creating Input Transforms

Input Transforms are a powerful way to clean, modify, and arrange data that is poorly organized, has changing format, has data that is not needed or otherwise hard to use. With the Input Transfrom feature of SQL Stream Builder, you can create a javascript function to transform the data after it has been consumed from a Kafk topic, and before you run SQL queries on the data.

About this task

You can use Input Transforms in the following situations:

- The source is not in your control, for example, data feed from a third-party provider
- The format is hard to change, for example, a legacy feed, other teams of feeds within your organization
- The messages are inconsistent
- The data from the sources do not have uniform keys, or without keys (like nested arrays), but are still in a valid JSON format
- The schema you want does not match the incoming topic

Note: When using Input Transforms the schema you define for the Kafka table is applied on the output of the transformed data.

You can use the Input Transforms on Kafka tables that have the following characteristics:

- Allows one transformation per source.
- Takes record as a JSON-formatted string input variable. The input is always named record.
- Emits the output of the last line to the calling JVM. It could be any variable name. In the following example, out and emit is used as a JSON-formatted string.

A basic input transformation looks like this:

```
var out = JSON.parse(record.value);     // record is input, parse JSON forma
tted string to object
                                                    // add more transformations
 if needed
JSON.stringify(out);                                // emit JSON formatted str
ing of object
```

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 Tables.

   You can add the Input Transform to the Kafka table when you create the Kafka table:

   a) Choose Apache Kafka from the Add table drop-down.

   You can add the Input Transform to an already existing Kafka table:

   a) Select the edit button for the Kafka table you want to add a transformation.

   The Kafka table wizard appears.
4. Click Transformations.

   You have the following options to insert your Input Transform:

   a) Add your javascript transformation code to the Data Transformation box.

      Make sure the output of your transform matches the Schema definition detected or defined for the Kafka table.
   b) Click Install default template and schema.

      The Install Default template and schema option fills out the Data Transformation box with a template that you can use to create the Input Transform, and matches the schema with the format.
5. Click Save changes.

Creating Kafka tables using Templates

The built-in templates allow you to simply and easily create tables by filling out the imported CREATE TABLE statement in the SQL window with detailed description of the properties.

You can create tables directly from the SQL window on the Console page by using the pre-defined connector templates.

When using the predefined templates, you have the following options for the Kafka table:

CDP Kafka

Automatically using the Kafka service that is registered in the Data Providers, and runs on the same cluster as the SQL Stream Builder service. You can choose between JSON, Avro and CSV data types.

Kafka

When connecting to a Kafka service that is not hosted in your cluster. You can choose between JSON, Avro, CSV and raw data types.

Upsert Kafka

Connecting to a Kafka service in the upsert mode. This means that when using it as a source, the connector produces a changelog stream, where each data record represents an update or delete event. The value in the data records is interpreted as an update of the last value for the same key. When using the table as a sink, the connector can consume a changelog stream, and write insert/update_after data as normal Kafka message valuea. Null values are represented as delete.

You can access and import the Templates from Streaming SQL Console:

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 main menu.
3. Click Templates under the SQL window.
4. Select the template you want to use.

   The template is imported to the SQL window.
5. Customize the fields of the template.
6. Click Execute.

   The table is created based on the selected template. You can review the table using the Table tab.

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"
    }
  ]
}
```

Creating tables with Flink SQL in SSB

You can create tables in SQL Stream Builder (SSB) using Flink DDL. When using Flink DDL, you can use Kudu, Hive, Schema Registry as catalogs. When using Kudu, Hive and Schema Registry catalog, you need to create a Streams Messaging, Real-time Data Mart or Data Engineering cluster within your environment. Aditionally to catalogs, there are predefined templates based on the supported connectors that you only need to fill out to use as a CREATE TABLE statement for your SQL query.

Before you begin

- You have prepared your environment based on the checklist in the *Prepare your environment* section.
- You have set Ranger policies for Flink and SSB.
- You have created a Streams Messaging, Real-time Data Mart or Data Engineering cluster in your environment.
- You have tables in Kudu, Hive and added a schema to Schema Registry.

Adding catalogs as Data Provider

When using Flink DDL in SQL Stream Builder (SSB), you need to add Schema Registry, Kudu or Hive as a Catalog in the Streaming SQL Console. You can also add File systems and JDBC as Custom Catalogs from the supported connectors.

Adding Schema Registry as Catalog

You need to add Schema Registry as a Catalog using the Streaming SQL Console in SQL Stream Builder (SSB) to use Schema Registry with Flink DDL.

Before you begin

- Make sure that you have Schema Registry service on your cluster.
- Make sure that you have the right permissions set in Ranger.

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 Data Providers from the main menu.
3. Click Register Catalog.

   The Add Catalog window appears.
4. Add a Name to your catalog.
5. Select Schema Registry from the Catalog Type drop-down.
6. Select the Kafka cluster you registered as Data Provider.
7. Add the Schema Registry URL.
   a) Go to your cluster in Cloudera Manager.
   b) Select Schema Registry from the list of services.
   c) Click on Instances.
   d) Copy the Hostname of Schema Registry.
   e) Add the default port of Schema Registry after the hostname.

   Example:

   ```
   http://docs-test-1.vpc.cloudera.com:7788/api/v1
   ```
8. Select if you wan to enable TLS for Schema Registry.
   a) If yes, provide the Schema Registry Truststore location and password.
9. Click on Add Filter.
   a) Provide a Database and Table filter if you want to select specific tables to use from the catalog.
10. Click on Validate.
11. If the validation is successful, click Add Tables.

Results

Schema Registry is added as a Catalog and ready to be used in Flink DDL. The already existing schemas in Schema Registry are automatically imported to SSB.

Adding Kudu as Catalog

You need to add Kudu as a Catalog using the Streaming SQL Console in SQL Stream Builder (SSB) to create Kudu tables with Flink DDL.

Before you begin

- Make sure that you have Kudu service on your cluster.
- Make sure that you have the right permissions set in Ranger.

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 Data Providers from the main menu.
3. Click Register Catalog.

   The Add Catalog window appears.
4. Add a Name to your catalog.
5. Select Kudu from the Catalog Type drop-down.
6. Add the Host URL of Kudu Masters.
   a) Go to your cluster in Cloudera Manager.
   b) Select Kudu from the list of services.
   c) Click on Instances.
   d) Copy the Hostname of the Master Default Group.
   e) Add the default port of Kudu after the hostname.

   Example:

   ```
   http://docs-test-1.vpc.cloudera.com:7051
   ```
7. Click on Add Filter.
   a) Provide a Database and Table filter if you want to select specific tables to use from the catalog.
8. Click on Validate.
9. If the validation is successful, click Add Tables.

Results

Kudu is added as a Catalog and ready to be used in Flink DDL. The already existing tables in Kudu are automatically imported to SSB.

Adding Hive as Catalog

You need to add Hive as a Catalog using the Streaming SQL Console in SQL Stream Builder (SSB) to create Hive tables with Flink DDL.

Before you begin

- Make sure that you have Hive service on your cluster.
- Make sure that you have the right permissions set in Ranger.

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 Data Providers from the main menu.

3. Click Register Catalog.
   The Add Catalog window appears.
4. Add a Name to your catalog.
5. Select Hive from the Catalog Type drop-down.
6. Click on Add Filter.
   a) Provide a Database and Table filter if you want to select specific tables to use from the catalog.
7. Click on Validate.
8. If the validation is successful, click Add Tables.

Results

Hive is added as a Catalog and ready to be used in Flink DDL. The already existing tables in Hive are automatically imported to SSB.

Adding Custom Catalogs

You can use custom catalogs in cases when you need to add a catalog that is not predefined in the Streaming SQL Console, and when more advanced settings need to be specified for the catalogs. This case you need to provide the configuration directly to Flink with the custom catalog option.

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 Data Providers from the main menu.
3. Click Register Catalog.
   The Add Catalog window appears.
4. Add a Name to your catalog.
5. Select Custom from the Catalog Type drop-down.
6. Provide a Property Name and Value for the Catalog.
7. Click on Add Filter.
   a) Provide a Database and Table filter if you want to select specific tables to use from the catalog.
8. Click on Validate.
9. If the validation is successful, click Add Tables.

Results

The custom catalog is added and ready to be used in Flink DDL.

Creating Flink tables using Templates

You can use the predefined templates to create tables by choosing one of the connector templates on the Console page of Streaming SQL Console. The Flink SQL templates are predefined examples of CREATE TABLE statements which you can fill out with your job specific values.

You can create tables using the predefined templates in SQL Stream Builder. The predefined templates consist of the CREATE TABLE statement, and every connection property that is needed for the given connector. You only need to fill out the templates and execute them in the Streaming SQL Console to create the table. When filling out

the templates, you can add configuration based on what is specified in the given connector template. You can also customize the table name, column names and timestamp information for a template.

You can choose from the following templates based on the connector type:

Blackhole

The BlackHole connector can be used to write all input records into. It is designed for high performance testing and UDF to output, not a substantive sink.

Datagen

The Data generator connector can be used to sample randomly generated data in SSB. You can use this connector to try out and test SQL queries as records are generated until the job is running.

Faker

The Faker connector can be used to generate fake data based on the Java faker expression. You can use this connector to try out and test SQL queries as records are generated until the job is running.

Filesystem

The filesystem connector can be used to access partitioned files in file systems, and enables reading and writing from a local or distributed file system such as local, HDFS, S3 and so on. You only need to specify the data format that is used in the file system. You can choose from the following formats:

- Avro
- JSON
- CSV
- ORC
- Parquet

JDBC

The JDBC connector enables reading data from and writing data into any relational databases with a JDBC driver. You can use PostgreSQL, MySQL and Hive as databases for the connector.

Debezium CDC

You can use the Debezium CDC connector to stream changes in real-time from MySQL, PostgreSQL, Oracle into Kafka. Debezium provides a unified format schema for changelog and supports serializing messages using JSON and Avro.

You can access and import the Templates from Streaming SQL Console:

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 main menu.
3. Click Templates under the SQL window.
4. Select the template you want to use.

   The template is imported to the SQL window.
5. Customize the fields of the template.
6. Click Execute.

   The table is created based on the selected template. You can review the table using the Table tab.

For full reference on the Flink SQL DDL functionality, see the official Apache Flink documentation.

Creating Webhook tables

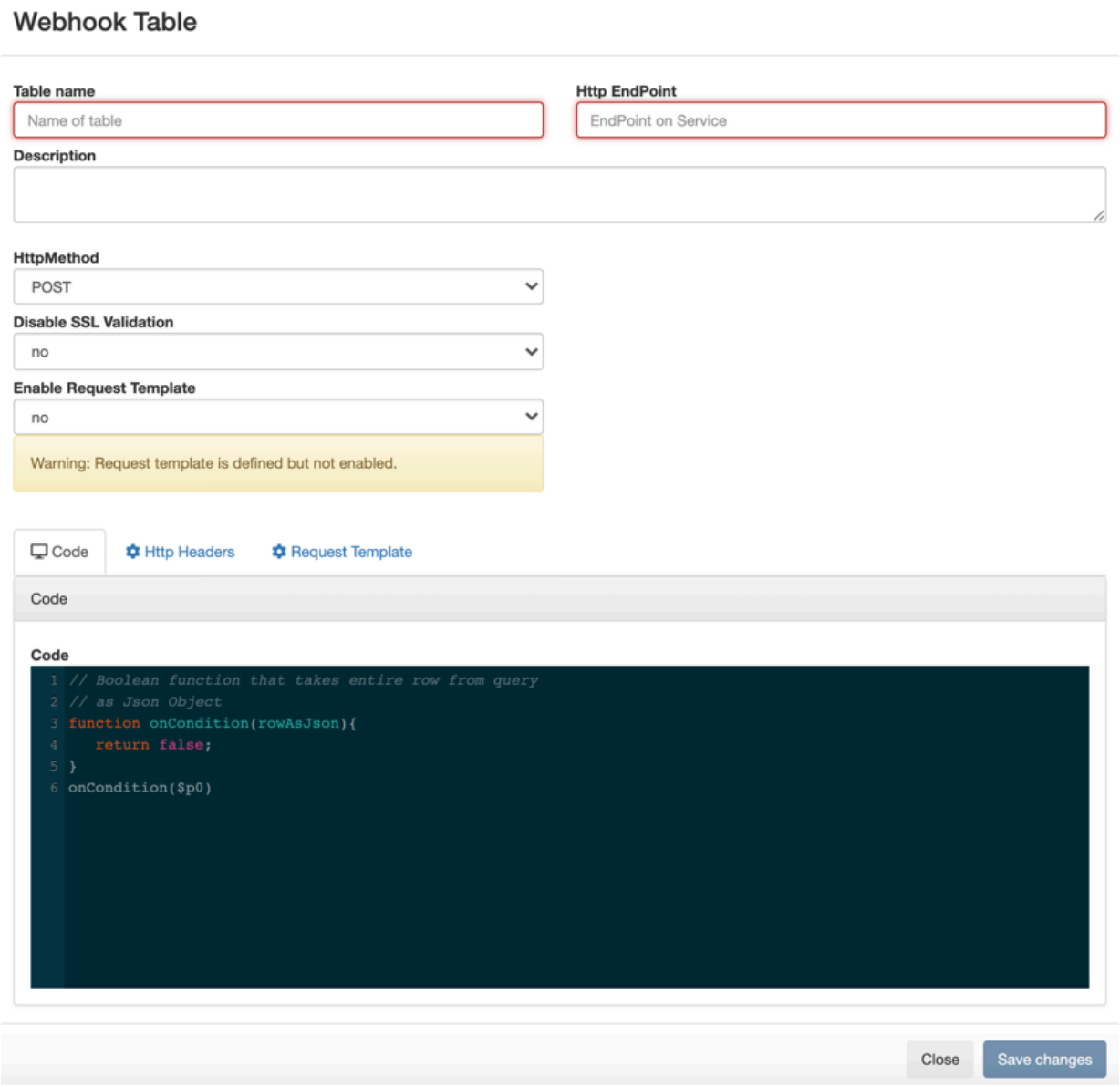
You can configure the webhook table to perform an HTTP action per message (default) or to create code that controls the frequency (for instance, every N messages). When developing webhook sinks, it is recommended to check your webhook before pointing at your true destination.

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 main menu.
3. Select the Tables tab.

4. Select Add table > Webhook .
   The Webhook Table window appears.



5. Provide a name to the Table.
6. Enter an HTTP endpoint. The endpoint must start with http:// or https://.

   Note: You can use hookbin for testing of the webhook sink. Paste the hookbin endpoint into the text field, and inspect the output on the hookbin site. Once you have the right output result, then point it at your final endpoint.
7. Add a Description about the webhook sink.
8. Select POST or PUT in the HTTP Method select box.
9. Choose to Disable SSL Validation, if needed.

10. Enable Request Template, if needed.
    a) If you selected Yes, then the template defined in the Request Template tab is used for output.

    This is useful if the service you are posting requires a particular data output format. The data format must be a valid JSON format, and use "${columnname}" to represent fields. For example, a template for use with Pagerduty looks like this:

```
{
   "incident":{
      "type":"incident",
      "title":"${icao} is too high!",
        "body":{
   "type":"incident_body",
         "details":"Airplane with id ${icao} has reached an altitude of
 ${altitude} meters."
 }
    }
}
```

11. In the Code editor, you can specify a code block that controls how the webhook displays the data.

    For a webhook that is called for each message the following code is used:

```
// Boolean function that takes entire row from query as Json Object
function onCondition(rowAsJson)
{return true;   // return false here for no-op, or plug in custom
        logic}
onCondition($p0)
```

Note: The rowAsJson is the result of the SQL Stream query being run in the {"name":"value"} format.

12. Add HTTP headers using the HTTP Headers tab, if needed.

    Headers are name:value header elements. For instance, Content-Type:application/json, etc.

13. Click Save Changes.

Results

The Webhook table is ready to be used after the INSERT INTO statement in your SQL query.

Running SQL Stream jobs

Every time you run an SQL statement in the SQL Stream console, it becomes a job and runs on the deployment as a Flink job. You can manage the running jobs using the Jobs tab on the UI.

About this task

There are two logical phases to run a job:

1. Parse: The SQL is parsed and checked for validity and then compared against the virtual table schema(s) for correct typing and key/columns.
2. Execution: If the parse phase is successful, a job is dynamically created, and runs on an open slot on your cluster. The job is a valid Flink job.

Before you begin

- Make sure that you have registered a Data Provider if you use the Kafka service on your cluster.
- Make sure that you have added Kudu, Hive or Schema Registry as a catalog if you use them for your SQL job.

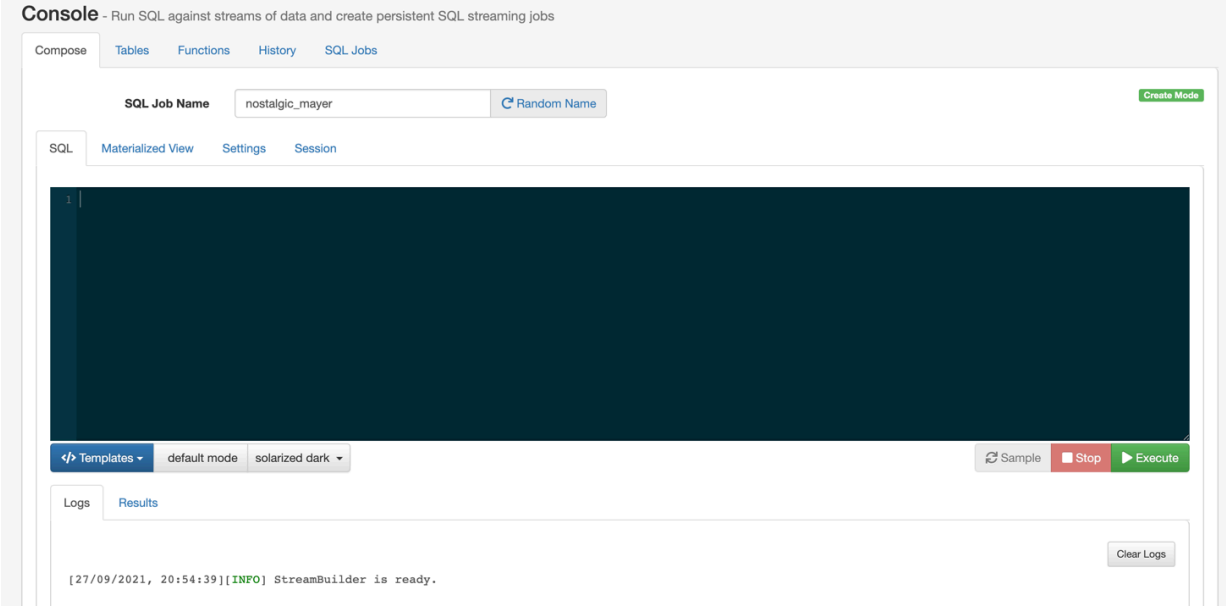
Procedure

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. Provide a name for the SQL job.
   a) Optionally, you can click Random Name to generate a name for the SQL job.
3. Create a table in SQL window.

   You have the option to create a table in the following ways:

   - Using the Console wizard under Tables tab to add Kafka and Webhook tables.
   - Using the Templates under the SQL window.
   - Add your custom CREATE TABLE statement to the SQL window.
4. Add a SQL query to the SQL window.

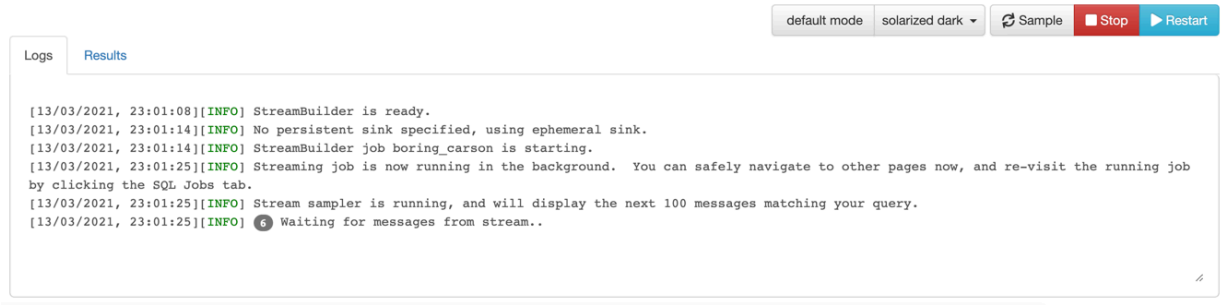


Note: You cannot start a job without adding a SQL statement in the SQL editor window. In case, there are no SQL statements provided and you click Execute , the following error message is displayed: You must provide a SQL query.

When starting a job, the number of slots consumed on the specified cluster is equal to the parallelism setting. The default is one slot. To change the parallelism setting and more job related configurations, click Settings.

5. Click Execute.

   The Logs window updates the status of SSB.



6. Click Results to check the sampled data.

   These results are only samples, not the entire result of the new stream being created from the output of the query. The entire result set is written out based on what you define at the INSERT INTO statement. You can also output the result to a Materialized View.

   Note: While SSB is querying unbounded data streams, you can click Stop to stop the job execution.

Results

A job is generated that runs the SQL continuously on the stream of data from a table, and pushes the results to a table, to the Console under the Results tab or to a Materialized View.

Related Information

Registering Data Providers in SSB

Using Tables in SQL Stream jobs

SQL Job Lifecycle SQL Stream jobs

Monitoring SQL Stream jobs

Flink SQL Overview

As Flink SQL is used in SQL Stream Builder, you can execute the supported Flink DDL, DML and query statements directly from the SQL window in Streaming SQL Console.

Flink DDL

Flink SQL supports Data Definition Language (DDL) statements to create, modify and remove objects within a data structure.

The following table summarizes the supported DDL statements in SQL Stream Builder:

DDL	Description	Option
CREATE TABLE	Creating table for the SQL query. You can create tables based on the supported connectors.	• Adding WATERMARK and PRIMARY KEY information • Creating table with LIKE clause
CREATE VIEW	Creating custom views using columns from tables. There is no physical data behind a view.	• Adding queries, expressions and joins
CREATE FUNCTION	Creating User Defined Function (UDF) for a query.	• Using Javascript or Java language • Using Functions tab on Streaming SQL Console
DROP TABLE	Deleting a table, view or function.	• Using Streaming SQL Console functionality to delete table, view or function
DROP VIEW		
DROP FUNCTIONS		
ALTER TABLE	Modifying table, view or function properties.	• Using RENAME TO or SET for table • Using RENAME TO or AS for view and function
ALTER VIEW		
ALTER FUNCTION		

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.

Flink DML

Flink SQL supports Data Manipulation Language (DML) statements to manipulate the data itself with adding, deleting or modyfing.

The following table summarizes the supported DML statements in SQL Stream Builder:

DML	Description	Option
INSERT INTO	Inserting query results into a specified table.	• Inserting columns or values into a table • Adding OVERWRITE to overwrite the existing data in the table

Flink Queries

Flink SQL supports queriyng data with SELECT statement and using different types of operations.

You can query data from a table using the SELECT statement.

Query	Description	Option
SELECT	Querying data from a table using different operations.	• Selecting all data in a table • Selecting data using different types of operations

The following table summarizes the supported operations for SELECT statement in SQL Stream Builder:

Operation	Description	Option
WHERE	Querying data based adding filters.	• Using boolien expression
JOIN	Joining data from tables based on a equivalent column.	• Using temporal joins based on event time or processing time • Using lookup join
GROUP BY	Grouping results using built-in or user defined functions.	• Using with streaming table procudes updated results
ORDER BY	Ordering results to be sort based on a specified expression.	• Using with streaming table the primary sorting key needs to be time

Other supported statements

Beside the supported DDL, DML and SELECT, the supported statements also include DESCRIBE, SHOW and SET that you can use in SQL Stream Builder.

The following table summarizes the supported statements for in SQL Stream Builder:

Operation	Description
DESCRIBE TABLES	Describing the schema of a table
SHOW TABLES	Showing a list of existing tables, views, functions, databases or catalogs
SHOW VIEWS	
SHOW FUNCTIONS	
SHOW DATABASES	
SHOW CATALOGS	
SET	Setting properties of session

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;
-- 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 SQL Stream Builder has the ability to join multiple tables in a single query, and because the Kafka 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;
```

Enriching streaming data with join

In SQL Stream builder, you can enrich your streaming data with values from a slowly changing dataset using join statements.

Regular join

Join statements in SQL serves to combine columns and rows from two or more tables based on a shared column. When you join tables from a slowly changing source such as HDFS, Kudu, Hive and so on, you can simply use the regular JOIN syntax of SQL. The following example shows a regular INNER JOIN where the *Orders* table is joined with *Product* table based on the *productId*:

```
SELECT * FROM Orders
INNER JOIN Product
ON Orders.productId = Product.id
```

A regular join can only be used with bounded tables. In a streaming context data is produced continuously, and with a regular join both sides of the join would need to be buffered indefinitely to store all of the events that would match with the result of the SQL query. To get results from a given amount of time and to join streaming tables, a time boundary needs to be specified. This means the tables not only need to be joined by a key or column, but also on a time attribute.

Interval join

When joining streaming tables, the time attribute can be defined in the SQL syntax using BETWEEN and an interval value:

```
SELECT *
FROM Orders o, Shipments s
WHERE o.id = s.order_id
AND o.order_time BETWEEN s.ship_time - INTERVAL '4' HOUR AND s.ship_time
```

In this case, *Orders* table is joined with the *Shipments* table and the results are going to be generated based on the *id* column as long as the order time and shipment time is within four hours of each other. The condition of this scenario is that the events in the streams happen almost at the same time with minimal delay, so the time boundary can be defined between an approximate interval.

When you want to join a streaming table with a slowly changing table, time attributes can differ as one of the tables stores data over a long period of time, while the streaming table receives new data continuiously. To join these types of tables, a time needs to be defined that can serve as a reference point for both types of tables.

Joining streaming and bounded tables

Beside regular join and interval join, in Flink SQL you are able to join a streaming table and a slowly changing dimension table for enrichment. In this case, you need to use a temporal join where the streaming table is joined with a versioned table based on a key, and the processing or event time.

A versioned table is a table that contains a time attribute, and reflects the records from a table at a specific point of time. When you use append-only or regularly updated sources, the values related to a key are updated over a long period of time. For example, a table can contain the currency rates since last month. At every change of the currency rate, a new value is added to the stream, therefore to the table. With creating a versioned table of the currency rate, you can specify which rate you need in an exact point of time: use the currency rates from 12:00.

For more information of Version Tables, see the official Apache Flink documentation.

Temporal join

After determining the version of the bounded table, you also need to define a time for the streaming table. In Flink, event time and processing time can be specified. When using event time, you need to create a temporal join.

Event time is the time that each individual event occurred on its producing device. To have an event time attribute in your SQL query, you need to define a timestamp column with a watermark definition column when creating the table:

```
CREATE TABLE orders (
    order_id    STRING,
    price       DECIMAL(32,2),
    currency    STRING,
    order_time  TIMESTAMP(3),
    WATERMARK FOR order_time AS order_time
) WITH (
  ...
)
```

When you want to use event time in a JOIN, you need to refer to the event time column as defined for the table in the *FOR SYSTEM_TIME AS OF* part of the SQL query:

```
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
```

Lookup join

As a special case of temporal join, you can use the processing time as a time attribute. In Flink, processing time is the system time of the machine, also known as “wall-clock time”. When you use the processing time in a JOIN SQL syntax, Flink translates into a lookup join and uses the latest version of the bounded table. The following example shows the join syntax that needs to be used for enriching streaming data:

```
SELECT o.order_id, o.total, c.country, c.zip
FROM Orders AS o
  JOIN Customers FOR SYSTEM_TIME AS OF PROCTIME()
    ON o.customer_id = c.id
```

In the above example, *Customers* serves as the lookup table. The FOR SYSTEM_TIME AS OF PROCTIME() syntax indicates that you always want to look up in the latest version of the table. With including the processing time in the SQL syntax, you can query the latest version of a lookup table, and enrich your streaming data with the corresponding value.

Note: When using SQL Stream Builder, you can simply use the PROCTIME() function as the version of the lookup table when performing a lookup join. This means that you do not need to create and reference a processing time column in your probe stream anymore.

In SQL Stream Builder, the following connectors are supported as lookup tables:

- Kudu
- Hive
- JDBC

Example: joining Kafka and Kudu tables

Using lookup join, you can join Kafka and Kudu tables to enrich the streaming data of Kafka with information from the Kudu tables. In the this example, Orders of a Kafka streaming table are enriched with metadata information from a Kudu table.

As a prerequisite for the example, the following steps were completed in SQL Stream Builder:

- Registering Kafka as a Data Provider.
- Registering Kudu as a Catalog.
- Creating Orders Kafka table.
- Creating ItemMeta Kudu table.
- Generating data for Kafka and Kudu tables.

The tables in the example contain the following information:

Tables	Columns
Kafka - Orders	• order_number • price • order_time • item_id
Kudu - ItemMeta	• id • info

In the scope of this example, the *Orders* table will be joined with latest version of the *ItemMeta* table based on the item *id*, and the selected information is sampled under the **Results** tab of the Streaming SQL Console:

```
SELECT order_time, item_id, info, price
FROM Orders
JOIN kudu.default_database.ItemMeta FOR SYSTEM_TIME AS OF PROCTIME()
ON item_id = id
```

After running the SQL query, the results are continuously sampled to the Streaming SQL Console, the rows of *order_time*, *item_id* and *price* are enriched with *info* column from the *ItemMeta* Kudu table: