Incremental Processing

Incremental processing is a processing method which involves processing only a data partition newly added to a dataset when the existing data is already processed, instead of re-processing the complete dataset. This processing method improves efficiency by eliminating the system overhead of re-processing already processed data.

e.g., The data in an Analytics table needs to be summarized per day. The following table illustrates how the dataset is processed when the summarization script is run.

<table>
<thead>
<tr>
<th></th>
<th>Without Incremental Processing</th>
<th>With Incremental Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Run</td>
<td>The complete dataset is processed to summarize the data.</td>
<td>The complete dataset is processed to summarize the data.</td>
</tr>
<tr>
<td>2nd Day</td>
<td>The complete dataset is processed to summarize the data.</td>
<td>Only the updates made to the dataset during the previous day are processed to summarize the data.</td>
</tr>
</tbody>
</table>

Once the summarization is complete, you need to commit the status to indicate that the data was successfully processed. This is done via the following command.

INCREMENTAL_TABLE_COMMIT orders;

Publishing events

Incremental analytics uses the timestamps of the events sent when retrieving the data for processing.

When defining event streams for incremental analytics, you can add an extra attribute to the event payload named `_timestamp` of the `LONG` attribute type. This allows a specific timestamp to be specified for each event. For more information, see Understanding Event Streams and Event Tables - Adding an event stream.

If the `_timestamp` attribute is not specified in the event stream definition, the timestamp of each event is derived from the system date at the time the event was persisted in the database.

Syntax

```
create temporary table orders using CarbonAnalytics options (tableName "ORDERS", schema "customerID STRING, phoneType STIRNG, OrderID STRING, cost DOUBLE, _timestamp LONG -i", incrementalParams "orders, DAY");
```

In order to apply incremental processing to an Analytics table, the `incrementalParams` attribute should be added to the table definition as shown in the extract above. If this parameter is not added, the table is considered a typical analytics table, and the complete table is processed for each query.

Parameters

The following parameters are configured to support incremental processing as shown in the above sample syntax.

Unique ID

<table>
<thead>
<tr>
<th>Name</th>
<th>uniqueID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required /Optional</td>
<td>Required</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Description</td>
<td>This is the unique ID of the incremental analytics definition. This ID should be used in the incremental table commit command as shown in the sample syntax.</td>
</tr>
</tbody>
</table>

**Time Period**

<table>
<thead>
<tr>
<th>Name</th>
<th>timePeriod</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required /Optional</td>
<td>Required</td>
</tr>
<tr>
<td>Description</td>
<td>The duration of the time period that you are processing. This can be MINUTE, HOUR, DAY, MONTH or YEAR. DAS has the ability to process the timestamp of each event and identify the unit of time during which it was sent. Note that time periods must be specified in the sample syntax. For example, if you specify HOUR as the time period, an event sent at 10.20PM is identified as an event sent during the 22.00 - 23.00 hour.</td>
</tr>
</tbody>
</table>
Example

The following is a list of events, and it is required to calculate the number of orders placed during each day.

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>Phone Type</th>
<th>Order ID</th>
<th>Cost</th>
<th>_timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nexus 5x</td>
<td>33s1sa2s</td>
<td>400</td>
<td>26th May 2016 12:00:01</td>
</tr>
<tr>
<td>12</td>
<td>Galaxy S7</td>
<td>kskds221</td>
<td>600</td>
<td>27th May 2016 02:00:02</td>
</tr>
<tr>
<td>43</td>
<td>iPhone 6s</td>
<td>sadl3122</td>
<td>700</td>
<td>27th May 2016 15:32:04</td>
</tr>
<tr>
<td>2</td>
<td>MoTo X</td>
<td>sdda221s</td>
<td>350</td>
<td>27th May 2016 16:22:10</td>
</tr>
<tr>
<td>32</td>
<td>LG G5</td>
<td>lka2s24dkQ</td>
<td>550</td>
<td>27th May 2016 19:42:42</td>
</tr>
</tbody>
</table>

The summarization script is run on 27th May 2016, at 12.00 noon. The last processed order ID kskds221.

Then the summarization table is as shown below.

<table>
<thead>
<tr>
<th>Day</th>
<th>Event Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>26th May 2016</td>
<td>1</td>
</tr>
<tr>
<td>27th May 2016</td>
<td>1</td>
</tr>
</tbody>
</table>

As a result, the summarized table for 27th May 2016 should have one event (because the other events that arrived on 27th May 2016 were received after 12.00 noon).

The next time the script is run, WSO2 DAS checks the timestamp of the event that was processed last. In this example, the last processed event is order ID kskds221 with timestamp 27th May 2016 02:00:02. Then DAS retrieves the data from the time period starting at 27th May 2016 00:00:00 to process all the data that has a timestamp greater than 27th May 2016 00:00:00.

This updates the value for the entry 27th May 2016 with the value 4 in the summarization table. The summarization table is not affected by any data received before 26th May 2016 because no data is retrieved from that time period. The resulting summarization table is shown below.

<table>
<thead>
<tr>
<th>Day</th>
<th>Event Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>26th May 2016</td>
<td>1</td>
</tr>
<tr>
<td>27th May 2016</td>
<td>4</td>
</tr>
</tbody>
</table>

The following sample further demonstrates how the incremental processing is carried out.

An event stream is defined with the following configuration. For more information about event streams, see Understanding Event Streams and Event Tables.
The Spark script written to process the data received by this event stream is as follows.

```json
{
    "streamId": "APIStats:1.0.0",
    "name": "APIStats",
    "version": "1.0.0",
    "nickName": "",
    "description": "",
    "metaData": [],
    "correlationData": [],
    "payloadData": [
        {
            "name": "name",
            "type": "STRING"
        },
        {
            "name": "count",
            "type": "INT"
        },
        {
            "name": "_timestamp",
            "type": "LONG"
        }
    ]
}
```
create temporary table APIStats using CarbonAnalytics options (tableName "APIStats", schema "name STRING, count INT, _timestamp LONG", incrementalParams "api_stats_1, MINUTE");

create temporary table APIStatsMinuteSummary using CarbonAnalytics options (tableName "APIStatsMinSummary", schema "name STRING, count INT, _timestamp LONG, year INT, month INT, day INT, hour INT, min INT", primaryKeys "name, min, hour, day, month, year", incrementalParams "api_stats_min_1, HOUR");

create temporary table APIStatsHourSummary using CarbonAnalytics options (tableName "APIStatsHourSummary", schema "name STRING, count INT, _timestamp LONG, year INT, month INT, day INT", primaryKeys "name, hour, day, year", incrementalParams "api_stats_hour_1, DAY");

create temporary table APIStatsDaySummary using CarbonAnalytics options (tableName "APIStatsDaySummary", schema "name STRING, count INT, _timestamp LONG, year INT, month INT, day INT", primaryKeys "name, day, year");

insert into table APIStatsMinuteSummary select name, sum(count) as count, getMinuteStartingTime(getYear(first(_timestamp)), getMonth(first(_timestamp)), getDay(first(_timestamp)), getHour(first(_timestamp)), getMinute(first(_timestamp))) as _timestamp, getYear(first(_timestamp)) as year, getMonth(first(_timestamp)) as month, getDay(first(_timestamp)) as day, getHour(first(_timestamp)) as hour, getMinute(first(_timestamp)) as min from APIStats group by name, getYear(_timestamp), getMonth(_timestamp), getDay(_timestamp), getHour(_timestamp), getMinute(_timestamp);

INCREMENTAL_TABLE_COMMIT api_stats_1;

insert into table APIStatsHourSummary select name, sum(count) as count, getHourStartingTime(year, month, day, hour) as _timestamp, year, month, day, hour from APIStatsMinuteSummary group by name, year, month, day, hour;

INCREMENTAL_TABLE_COMMIT api_stats_min_1;

insert into table APIStatsDaySummary select name, sum(count) as count, getDateStartingTime(year, month, day) as _timestamp, year, month, day from APIStatsHourSummary group by name, year, month, day;

INCREMENTAL_TABLE_COMMIT api_stats_hour_1;

The incrementalParams "api_stats_1, MINUTE" parameter specifies incremental processing to be applied. When the script is run, the system identifies the last minute during which the summarization was done, and processes all events with a timestamp that is greater than the timestamp of the start of that minute.

To use the above sample, you need to add this UDF (download). The associated fully qualified class name entry for spark-udf-config.xml is org.wso2.das.sample.udf.DateTimeUDF.

For detailed instructions on how to add a UDF, see Creating Spark User Defined Functions.