

PATTERN MATCHING CAPABILITIES IN ORACLE BAM 12.1.3



Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Table of Contents

Disclaimer	1
Introduction	3
Pattern Matching Use Case Scenario:	3
Preparing the Project	4
Importing the Project	5
Running a PatternMatch Dashboard	9
To run a PatternMatch dashboard, you must run through the following steps:	9
Creating Continuous Queries Using CQL Templates	10
Data Objects	10
Suggested Best Practice	12
Creating the PatternMatch Project	13
Continuous queries	14
Designing a KPI Alert Template	15
Designing a Trending Design Template	18
Designing a Missing Event Template	21
Designing a Monitor Count Template	25
Designing a Moving Aggregation Template	28
Designing a Top N Template	30
Designing a Duplicate Detection Template	33
Continuous Queries Monitoring	36
Designing a Fired KPI Alert Events Query	36
Designing Fired KPI Alert Output Events	37
Understanding Dashboards	38
PatternMatch Dashboard	38



StrPatternMatch Dashboard

40

You can create a strPatternMatch Dashboard using the procedure outlined for creating the PatternMatch Dashboard, and add the following views to it.

40

Troubleshooting

41

Best Practices

42

Introduction

This whitepaper provides information on defining continuous queries and building dashboards to actively monitor outputs using advanced Pattern Matching capabilities provided by Oracle BAM 12c. Continuous Queries differ from traditional queries in that once a query registers, it runs until it is de-registered. This makes continuous queries ideal for addressing real time monitoring with temporal (the last moving 10 minutes, for example) and PatternMatch (e.g. trending, missing event, etc.) capabilities.

Oracle BAM has seven most commonly used Out Of the Box (OOTB) Continuous Query templates, which can be used to define complex PatternMatch scenarios without writing any CQL statements. These templates are parameters-driven and CQL are generated and registered to the BAM server, based on specific parameters. When a pattern is detected by the CQ, you can associate an alert with it, which triggers a list of actions. For example, the outputs from these queries can be daisy-chained to drive the dashboards so you can continuously monitor the number of alerts generated by each of these queries.

For more information on Oracle CQL, see: [Understanding Oracle CQL](#).

Oracle BAM uses OOTB templates for creating CQs, and supports both stream and relational data objects. Stream Data is a type of data which always generates a 'plus' event; for example, GPS data or stock price data that shows the current value of the event. This means, you can insert data into a DO, but you cannot modify or delete them. Archived Relation Data is like traditional relational data where CRUD operations can be performed. Archived Relation obtains the initial state of the query from the database and then performs incremental computations on top of it. Oracle BAM supports the following templates. Note that the Duplicate Detection Template' can only support a stream DO.

1. KPI Alert Template.
2. Duplicate Detection Template (supports only Stream DOs)
3. Trending Detection Template
4. Monitor Count Template
5. Moving Aggregation Template
6. Missing Event Template
7. Top N Template

As an example, you can use the KPI Alert Template to generate a query which fires each time the KPI values cross a certain threshold. You can choose to display this using a line chart which shows alert counts for each minute, or any other time value, for the last 10 minutes, or any other time range.

Pattern Matching Use Case Scenario:

Consider a situation where you are monitoring product sales efficiency for a company, using metadata from their in-house call center to detect emerging patterns or trends that may hamper or enhance your sales process. This requires that you:

- ✓ Detect when the average call wait time for all the calls 'closed' in the last 2 minutes is greater than a predefined threshold.
- ✓ Detect if the callProcessingTime is increasing by more than 10% for two consecutive intervals.
- ✓ Detect when a call "Suspend" action is not directly followed by "Resume".
- ✓ Identify if more than one event is detected in the past 2 minutes for the same customerLocationId and productId pair.
- ✓ Continuously monitor the moving average of callProcessingtime in the last 2 minutes for each productId.
- ✓ Continuously update the top 5 products with maximum call Processing Time in the last 2 minutes.
- ✓ Detect if more than one event arrives within 2 minutes of each other with the same CustomerLocationId, productId and callStatus.

Apart from this, you must continuously monitor the count of these anomalies within the past 10 minutes. This document explains how the OOTB templates available in BAM 12c are used to detect these patterns or trends and how this information is displayed on a dashboard so you can track patterns in real time.

Preparing the Project

To start, you must ensure that the project environment is set up correctly as follows.

1. Copy the "samples" directory to ORACLE_HOME/soa/bam
2. Update ../common/setEnv.sh with proper info.
3. Update ../bin/BAMCommandConfig.xml and add the following parameters (make sure to replace **password** with valid password):
<password>**password**</password>
4. Set environment variable JAVA_HOME.
5. If there's a previous installed project, clear all continuous queries. If you are importing a PatternMatch project for the first time, you can ignore these steps:
 - I. Open BAM composer: <http://<hostname>:<port>/bam/composer>
 - II. Choose 'Administrator'
 - III. Click on 'Continuous Queries Monitoring'
 - IV. Under the 'Project' dropdown list, select 'PatternMatch'
 - V. Check on the 'select all' checkbox
 - VI. Click on 'Deactivate Query'
 - VII. Click on 'Drop Query'



Importing the Project

To import the project, execute: `importPatternMatch.sh` from `ORACLE_HOME/soa/bam/samples/bam-103-pattern-match`. Then, check if all project artifacts are ordered correctly, as shown in [Figure 1](#).

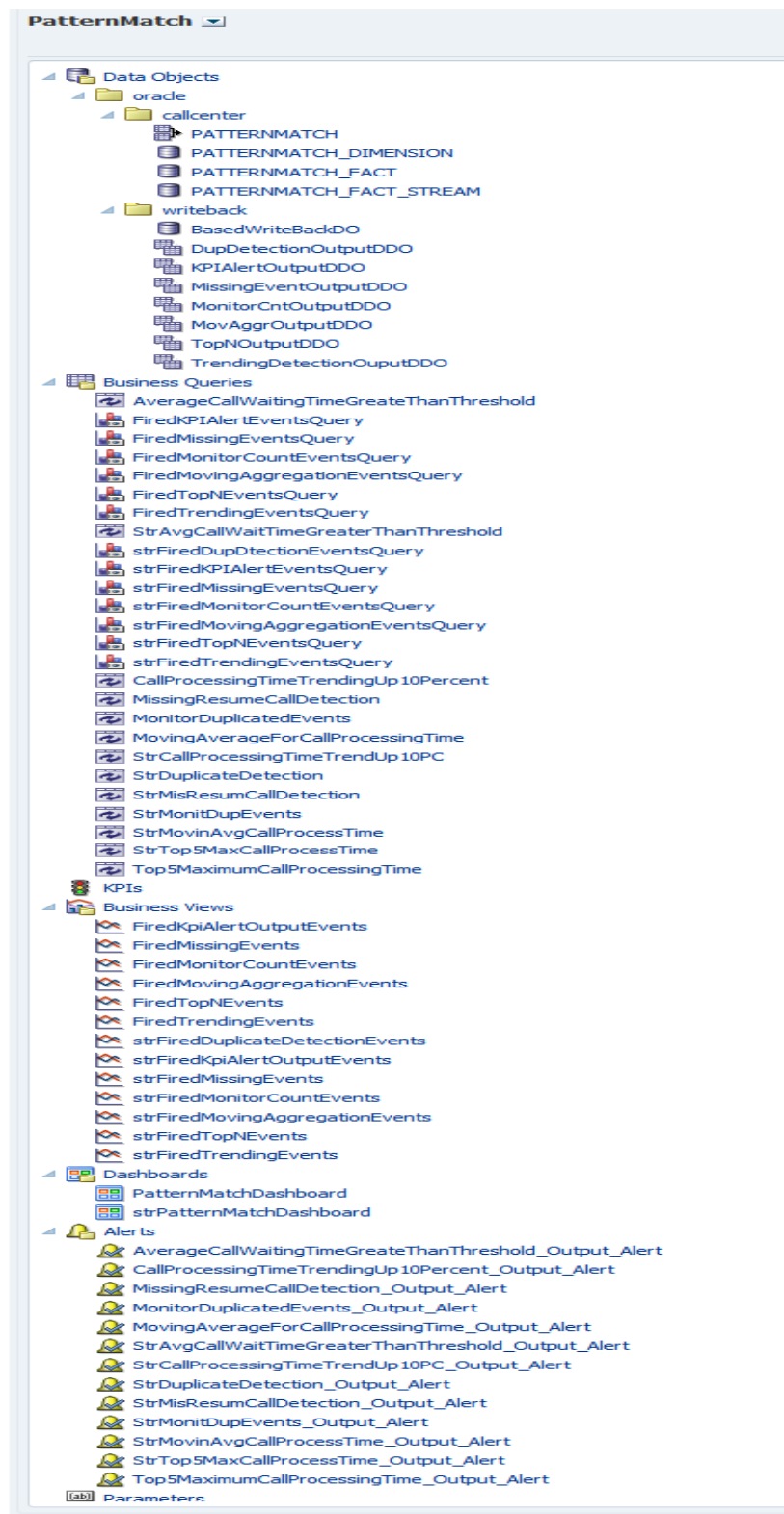



Figure 1 – Example PatternMatch Project



The following Data objects must be present under Data Objects Oracle → callcenter

1. PATTERNMATCH_FACT (Archived Relation)
2. PATTERNMATCH_DIMENSION (Archived Relation)
3. PATTERNMATCH (Logical DO - a combination of PATTERNMATCH_FACT and PATTERNMATCH_DIMENSION)
4. PATTERNMATCH_FACT_STREAM (Stream DO)

The project also contains the following pre-seeded DOs under oracle → writeback

1. BasedWriteBackDO
2. DupDetectionOutputDDO
3. KPIAlertOutputDDO
4. MissingEventOutputDDO
5. MonitorCntOutputDDO
6. MovAggrOutputDDO
7. TopNOutputDDO
8. TrendingDetectionOuputDDO

In the PatternMatch project, there are six continuous queries over logical DOs and seven continuous queries over stream DOs.

1. KPI Alert Template
 1. AverageCallWaitingTimeGreateThanThreshold over the logical DO PATTERNMATCH
 2. StrAvgCallWaitTimeGreaterThenThreshold over stream DO PATTERNMATCH_FACT_STREAMBoth these queries have an alert action configured to write back the output to KPIAlertOutputDDO.
2. Duplicate Detection Template (supports only Stream DOs)
 1. StrDuplicateDetection over stream DO PATTERNMATCH_FACT_STREAMThis query has an alert action configured to write back the output to DupDetectionOutputDDO
3. Trending Detection Template
 1. CallProcessingTimeTrendingUp10Percent over the logical DO PATTERNMATCH
 2. StrCallProcessingTimeTrendUp10PC over stream DO PATTERNMATCH_FACT_STREAMBoth these queries have an alert action configured to write back the output to TrendingDetectionOuputDDO
4. Monitor Count Template
 1. MonitorDuplicatedEvents over the logical DO PATTERNMATCH
 2. StrMonitDupEvents over stream DO PATTERNMATCH_FACT_STREAMBoth these queries have an alert action configured to write back the output to MonitorCntOutputDDO

5. Moving Aggregation Template
 1. MovingAverageForCallProcessingTime over the logical DO PATTERNMATCH
 2. StrMovinAvgCallProcessTime over stream DO PATTERNMATCH_FACT_STREAMBoth these queries have an alert action configured to write back the output to MovAggrOutputDDO

6. Missing Event Template
 1. MissingResumeCallDetection over the logical DO PATTERNMATCH
 2. StrMisResumCallDetection over stream DO PATTERNMATCH_FACT_STREAMBoth these queries have an alert action configured to write back the output to MissingEventOutputDDO


7. Top N template
 1. Top5MaximumCallProcessingTime over the logical DO PATTERNMATCH
 2. StrTop5MaxCallProcessTime over stream DO PATTERNMATCH_FACT_STREAMBoth these queries have an alert action configured to write back the output to TopNOutputDDO

The following Group Queries are defined over writeback DOs so they can use the output of the continuous queries as event sources:

1. FiredKPIAlertEventsQuery & strFiredKPIAlertEventsQuery over KPIAlertOutputDDO
2. strFiredDupDtctionEventsQuery over DupDetectionOutputDDO over DupDetectionOutputDDO
3. FiredTrendingEventsQuery and strFiredTrendingEventsQuery over TrendingDetectionOuputDDO
4. FiredMonitorCountEventsQuery and strFiredMonitorCountEventsQuery over MonitorCntOutputDDO
5. FiredMovingAggregationEventsQuery and strFiredMovingAggregationEventsQuery over MovAggrOutputDDO
6. FiredMissingEventsQuery and strFiredMissingEventsQuery over MissingEventOutputDDO
7. FiredTopNEventsQuery and strFiredTopNEventsQuery over TopNOutputDDO

Ensure that the following business views are present:

1. FiredKpiAlertOutputEvents & strFiredKpiAlertOutputEvents
1. strFiredDuplicateDetectionEvents
2. FiredTrendingEvents& strFiredTrendingEvents
3. FiredMonitorCountEvents & strFiredMonitorCountEvents

- 
4. FiredMovingAggregationEvents & strFiredMovingAggregationEvents
 5. FiredMissingEvents & strFiredMissingEvents
 6. FiredTopNEvents & strFiredTopNEvents

Two dashboards PatternMatch and strPatternMatch should be present with their corresponding business views.

For the Patternmatch dashboard, the following views must be present:

1. FiredKpiAlertOutputEvents
2. FiredTrendingEvents
3. FiredMonitorCountEvents
4. FiredMovingAggregationEvents
5. FiredMissingEvents
6. FiredTopNEvents

For the strPatternMatch dashboard, the following views must be present:

1. strFiredKpiAlertOutputEvents
2. strFiredDuplicateDetectionEvents
3. strFiredTrendingEvents
4. strFiredMonitorCountEvents
5. strFiredMovingAggregationEvents
6. strFiredMissingEvents
7. strFiredTopNEvents

Running a PatternMatch Dashboard

To run a PatternMatch dashboard, you must run through the following steps:

1. Populate data: Go to `ORACLE_HOME/soa/bam/samples/bam-103-pattern-match` and execute:
`startPatternMatch.sh <wls password>`

This script will start populating both PATTERNMATCH (logical DO) and PATTERNMATCH_FACT_STREAM (stream DO) which will in turn drive all the views in PatternMatch strPatternMatch dashboards.

- Open the Patternmatch dashboard using the following url:
<http://<host>:<port>/bam/composer/faces/proxy?page?project=PatternMatch&dashboard=PatternMatchDashboard>
- Open the strPatternmatch dashboard using the following URL:
<http://<host>:<port>/bam/composer/faces/proxy?page?project=PatternMatch&dashboard=strPatternMatchDashboard>
- Once you are done, to stop data population, go to ORACLE_HOME/soa/bam/samples/bam-103-pattern-match and execute:
stopPatternMatch.sh

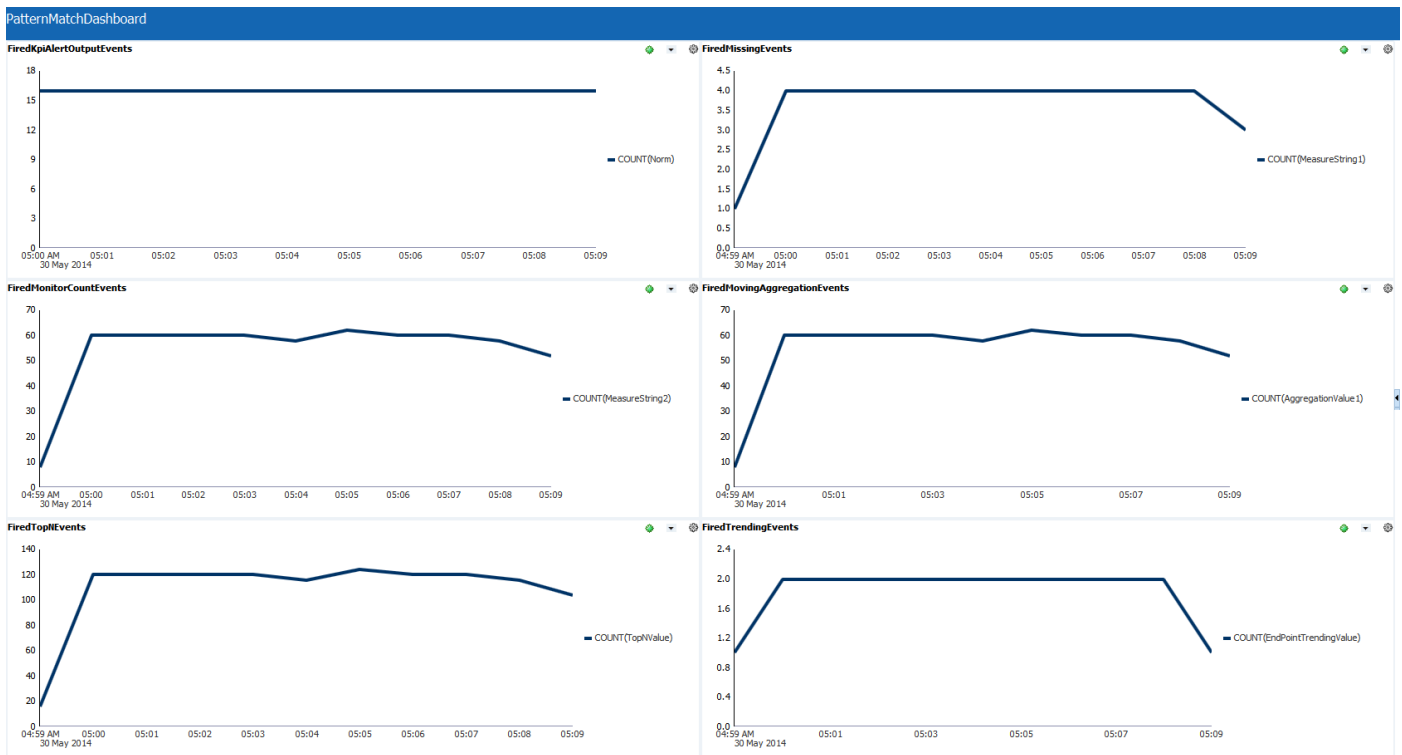


Figure 2 – Sample PatternMatch dashboard

Creating Continuous Queries Using CQL Templates

This section outlines how each of the CQL templates is used to create continuous queries and how outputs from the queries are used to drive real time dashboards.

Data Objects

PATTERNMATCH_FACT is an archived relation DO with the following columns. The contents of an archived relation DO are saved in the database and CRUD operations are allowed.

Column	Column Type	Data Type
customerLocationId	DIMENSION	VARCHAR
productId	DIMENSION	VARCHAR
customerStatus	DIMENSION	VARCHAR
callPriority	ATTRIBUTE	INT
callProcessingTime	MEASURE	INT
callWaitTime	MEASURE	INT
callStatus	ATTRIBUTE	VARCHAR
callClosedTime	DIMENSION	DATETIME
callCreatedTime	DIMENSION	DATETIME

PATTERNMATCH_DIMENSION is also an archived relation DO with the following columns:

Column	Column Type	Data Type
customerLocationId	ATTRIBUTE	VARCHAR
customerLocationName	ATTRIBUTE	VARCHAR

Patternmatch is a Logical DO Logical DOs do not have underlying persistent stores. They are always merges between existing DataObjects. PatternMatch is a merge between PATTERNMATCH_FACT and PATTERNMATCH_DIMENSION using matching 'customerLocationId' as the join condition. It has the following columns.

Column	Source DO	Column Type	Data Type
customerLocationId	PATTERNMATCH_FACT	DIMENSION	VARCHAR
productId	PATTERNMATCH_FACT	DIMENSION	VARCHAR
customerStatus	PATTERNMATCH_FACT	DIMENSION	VARCHAR
callPriority	PATTERNMATCH_FACT	MEASURE	INT
callProcessingTime	PATTERNMATCH_FACT	MEASURE	INT
callWaitTime	PATTERNMATCH_FACT	MEASURE	INT
callStatus	PATTERNMATCH_FACT	DIMENSION	VARCHAR
callClosedTime	PATTERNMATCH_FACT	ATTRIBUTE	DATETIME
callCreatedTime	PATTERNMATCH_FACT	ATTRIBUTE	DATETIME
customerLocationName	PATTERNMATCH_DIMENSION	DIMENSION	VARCHAR

Oracle BAM also supports the concept of Derived Data Objects. You can use an existing DataObject and extend it by incorporating additional fields. Any data operation done on the derived DO will be applied to the base DO as well. PATTERNMATCH_FACT_STREAM is a stream DO. You can create it with either the 'Archived'=true or the Archived=false condition. If Archive=true, then the data is saved in the database. In this case, PATTERNMATCH_FACT_STREAM has 'Archived'=false enabled. It has the following columns:

Column	Column Type	Data Type
customerLocationId	DIMENSION	VARCHAR
productId	DIMENSION	VARCHAR
customerStatus	DIMENSION	VARCHAR
callPriority	ATTRIBUTE	INT
callProcessingTime	MEASURE	INT
callWaitTime	MEASURE	INT
callStatus	ATTRIBUTE	VARCHAR
callClosedTime	DIMENSION	DATETIME
callCreatedTime	DIMENSION	DATETIME
customerLocationName	ATTRIBUTE	VARCHAR

Suggested Best Practice

If you have Dimension DO where you expect the data to change rarely, then consider marking it a 'Slow Changing Dimension'. If a DO is marked as a Slow Changing Dimension, the CQL query will be optimized to not keep the fact DO tuples in memory. This can give you a considerable performance advantage and reduce memory use. The trade off is that CQ Service restarts the query each time it detects any event on the Slow Changing Dimension DO. Note that this optimization kicks in only when a logical DO based on the Slow Changing Dimension DO is used in the query, and not when the same DO is used directly, outside of a query. This option should ideally be exercised only when you are sure that the changes to your Dimension DO will be very infrequent, because frequent changes to the Dimension DO will result in frequent query restarts and it can worsen performance.

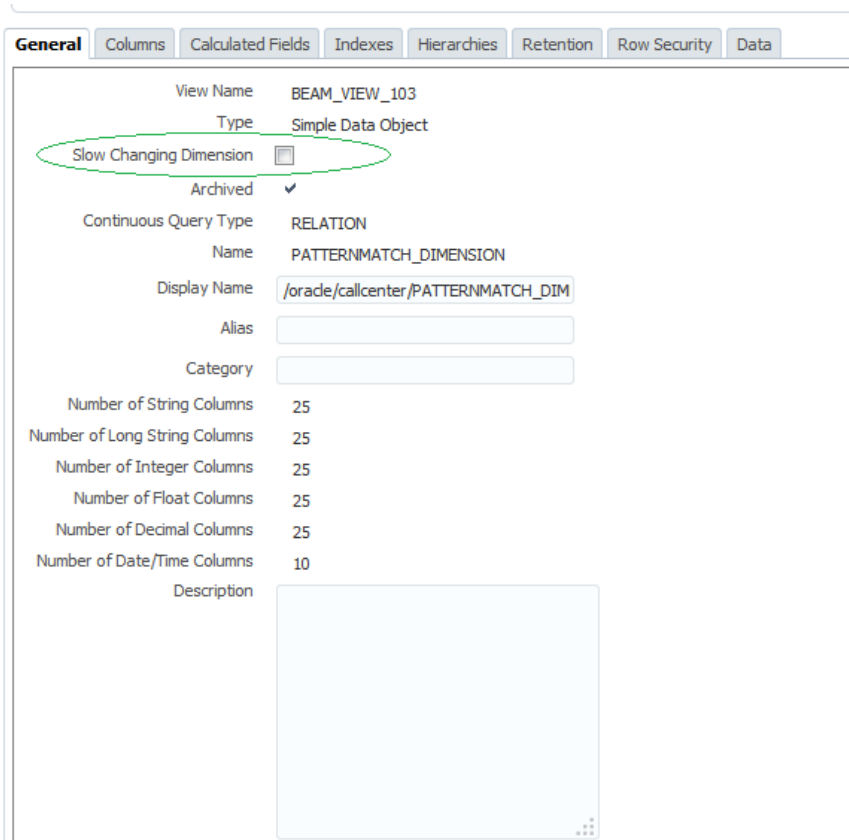


Figure 3 – Enabling the Slow Changing Dimension option

Creating the PatternMatch Project

The project is a logical organization of work. You can create the project from Oracle BAM Composer and add all the required Data Objects to the project.

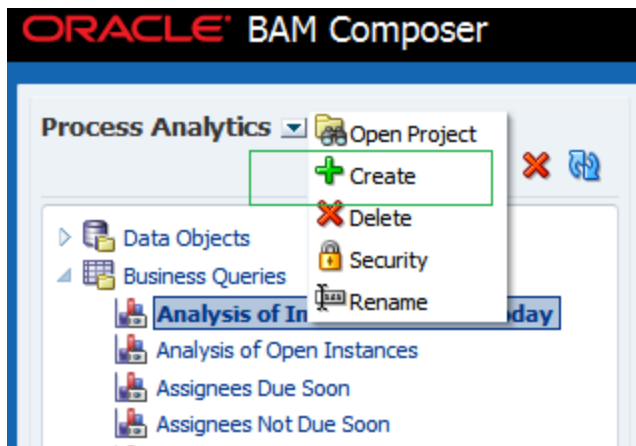


Figure 4 – The 'Create Project' list item in BAM Composer

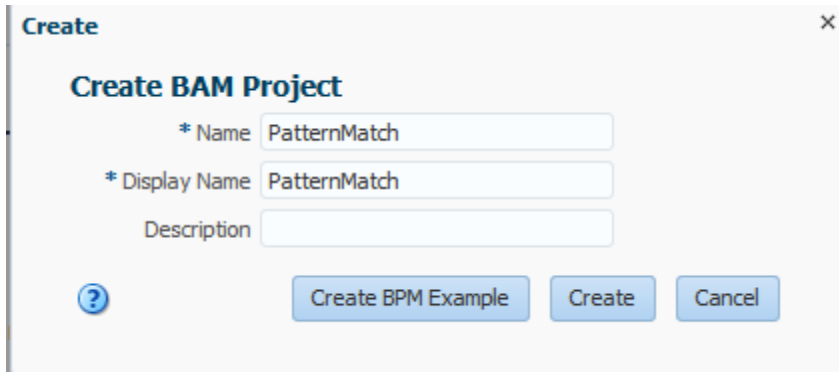


Figure 5 – The 'Create BAM Project' start screen

Once the project is created, click on the Data Objects link and add the required Data Objects into the project.

Continuous queries

This section outlines each OOTB Template in detail. Here are some CQL terms used in this document with their respective explanations:

1. Stream DO: Stream represents an event stream, which only supports an insert operation. Events arriving on a stream DO are processed by any Continuous Queries over the stream DO currently registered with BAM, but the data is not saved.
2. Archived Stream: Archived Stream DO is essentially a stream DO but the data is saved in the database. The amount of data replayed to a newly registering or restarting continuous query is based on settings in the DO.
3. Archived Relation: An Archived Relation obtains the initial state of the query from the database and allows incremental computation on top of it.
4. Rolling Window: A Rolling Window of N minutes implies that the query will consider only events that have arrived in the last N minutes. When time progresses, events that arrived before N minutes will 'fall off' the window, i.e., they will no longer affect the output of the query. The "Use rolling window" option is made mandatory for all CQL Template queries over Archived Relations. For more details, see Best Practices.
5. Update Interval: An update interval can be used to throttle the output of the query. If you do not specify one, the query will give you an output each time there is a change. If you specify an update interval, the output is provided only at those intervals.

To create a continuous Query using a CQL Template, open the project and click on the "Business Queries" link. Enter a name and display name; make sure "Continuous Query" is selected by default and click "Create".

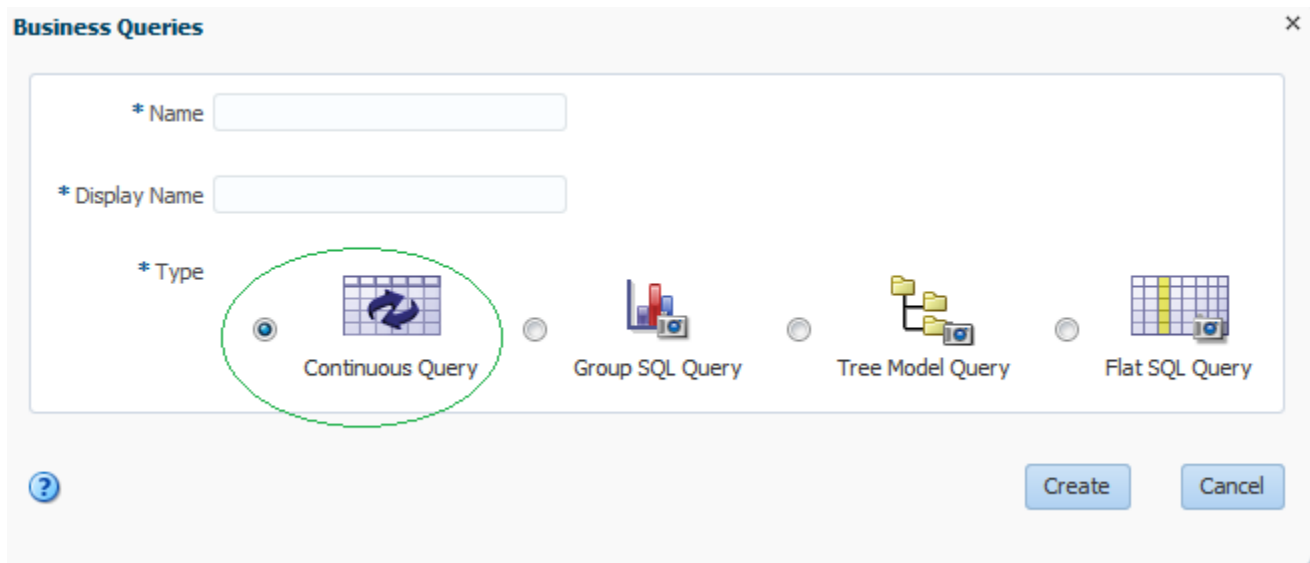


Figure 6 – Creating a continuous query

Steps specific to each template are detailed below. Ensure that the check box “Activate Continuous Queries :<Query_Name>” is checked before you save the query.

Designing a KPI Alert Template

The goal is to design a query which will, for each product ID, fire an alert when the average call wait time for all the calls ‘closed’ in the last two minutes is greater than $100(\text{Norm}) + 10(\text{Allowed deviation})$. This is achieved using a KPI Alert Template. This template supports all types of DOs.

Over Logical DO (Query: AverageCallWaitingTimeGreateThanThreshold)

Data Object: PatternMatch

- **Filter**
 - ✓ Branch: all are true - callStatus is equal to "CLOSED"
- **Measure**
 - ✓ Measure field: callWaitTime
 - ✓ Aggregation Function: Average
 - ✓ Group By productId
 - ✓ Use rolling window = true with Range Length = 2 minute based on callCreatedTime
- **Threshold**
 - Criteria: Measure is greater than (Norm + Deviation)
 - Norm Definition: Constant, 100
 - Deviation definition: Scalar, 10
- **Alert Event**
 - Output all fields
- **Action**

Insert Event output into KPIAlertOutputDDO, and use the following mapping:

DataObject column	Event Output Field
ProjectName	PROJECT_NAME
QueryName	QUERY_NAME
GroupFieldString1	productId
Measure	MEASURE
Norm	NORM
AllowedDeviation	ALLOWED_DEVIATION
ActualDeviation	ACTUAL_DEVIATION

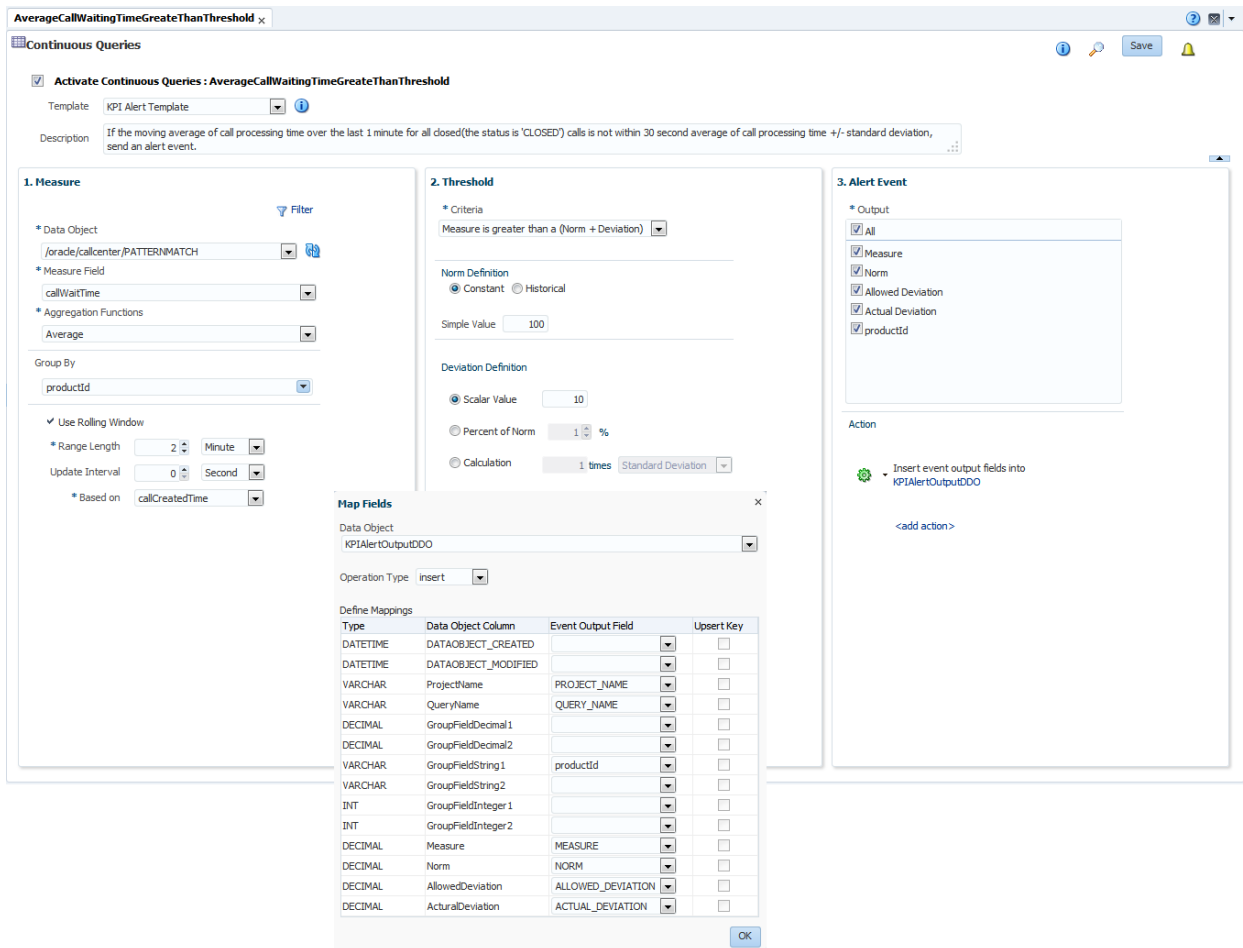


Figure 7 – Creating a continuous query using the KPI Alert Template.

Click the preview icon to see what the final query generated by the template looks like. In this case, the CQL statement is as follows:

```
CREATE QUERY PatternMatch.AverageCallWaitingTimeGreateThanThreshold as
ISTREAM(SELECT AVG(callWaitTime) AS MEASURE , 100.0 AS NORM , 10.0 AS
ALLOWED_DEVIATION , AVG(callWaitTime) - 100.0 AS ACTUAL_DEVIATION , productId AS
productId , 'PatternMatch' AS PROJECT_NAME ,
```

'AverageCallWaitingTimeGreateThanThreshold' AS QUERY_NAME FROM
 PatternMatch.PATTERNMATCH[RANGE 2 minute ON callCreatedTime] AS T WHERE
 (callStatus="CLOSED") GROUP BY T.productId HAVING AVG(T.callWaitTime) - 100.0 > 10.0)
 destination
 "combined:jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.m
 db.alertengine;jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqserv
 ice.mdb.reportcache?batch=true"
 Over Stream DO (StrAvgCallWaitTimeGreaterThanThreshold)

Data Object: PATTERNMATCH_FACT_STREAM

To continue, follow the same procedure as the one outlined for the KPI Alert template query over
 Logical DO.

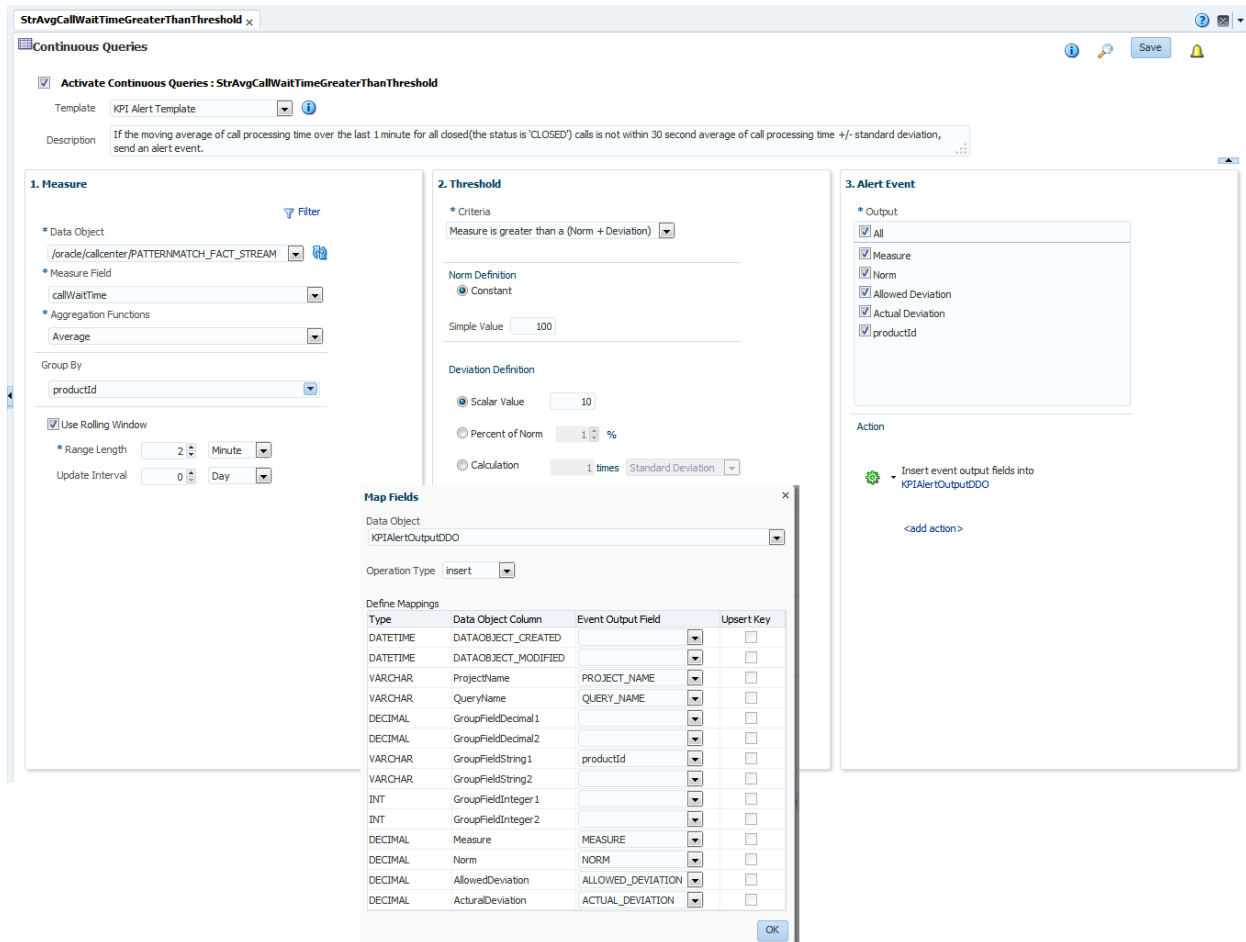


Figure 8 – Creating a continuous query using the KIP Alert Template, over a stream DO.

Click the preview icon to see what the final query generated by the template looks like. In this case, the CQL statement is as follows:

```
CREATE QUERY PatternMatch.StrAvgCallWaitTimeGreaterThanThreshold as  

  ISTREAM(SELECT AVG(callWaitTime) AS MEASURE , 100.0 AS NORM , 10.0 AS
```

ALLOWED_DEVIATION , AVG(callWaitTime) - 100.0 AS ACTUAL_DEVIATION , productId AS
productId , 'PatternMatch' AS PROJECT_NAME , 'StrAvgCallWaitTimeGreaterThanThreshold'
AS QUERY_NAME FROM PatternMatch.PATTERNMATCH_FACT_STREAM[RANGE 2 minute]
AS T WHERE (callStatus="CLOSED") GROUP BY T.productId HAVING AVG(T.callWaitTime) -
100.0 > 10.0) destination
"combined:jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.m
db.alertengine:jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqserv
ice.mdb.reportcache?batch=true"

Designing a Trending Design Template

The goal is to detect if the callProcessingTime is increasing by more than 10% for two consecutive intervals. The trending detection template is used to achieve this. This template works with all types of Dos.

Over Logical DO (Query: CallProcessingTimeTrendingUp10Percent)

Data Object: PatternMatch

- **Filter**

- ✓ None: No filters specified for this query.

- **Measure**

- ✓ Measure Field: callProcessingTime
- ✓ Aggregation Function: None
The callProcessingTime field is observed for changes, so an aggregation function need not be specified. On the other hand, if you want to detect, say, trends in Sum of callProcessingTime or Average of callProcessingTime, then you must specify appropriate aggregation functions.
- ✓ Partition By: productId
Trends are identified for each productId independently.
- ✓ Use rolling window = true with Range Length = 2 minute based on callCreatedTime. In this example, the 'update interval' field isn't used. Generally, if an aggregation function is applied in trend detection, the 'update interval' field can help.

- **Trending**

- ✓ Change greater than 10%
- ✓ Consecutive interval 2

- **Output**

- ✓ Select all available fields

- **Action**

Insert Event output into TrendingDetectionOutputDDO, and use the following mapping:

DataObject column	Event Output Field
ProjectName	PROJECT_NAME
QueryName	QUERY_NAME
GroupFieldString1	T.productID
EndPointTrendingValue	T.endCallProcessingTime
StartPointTrendingDecimalValue	T.callProcessingTime

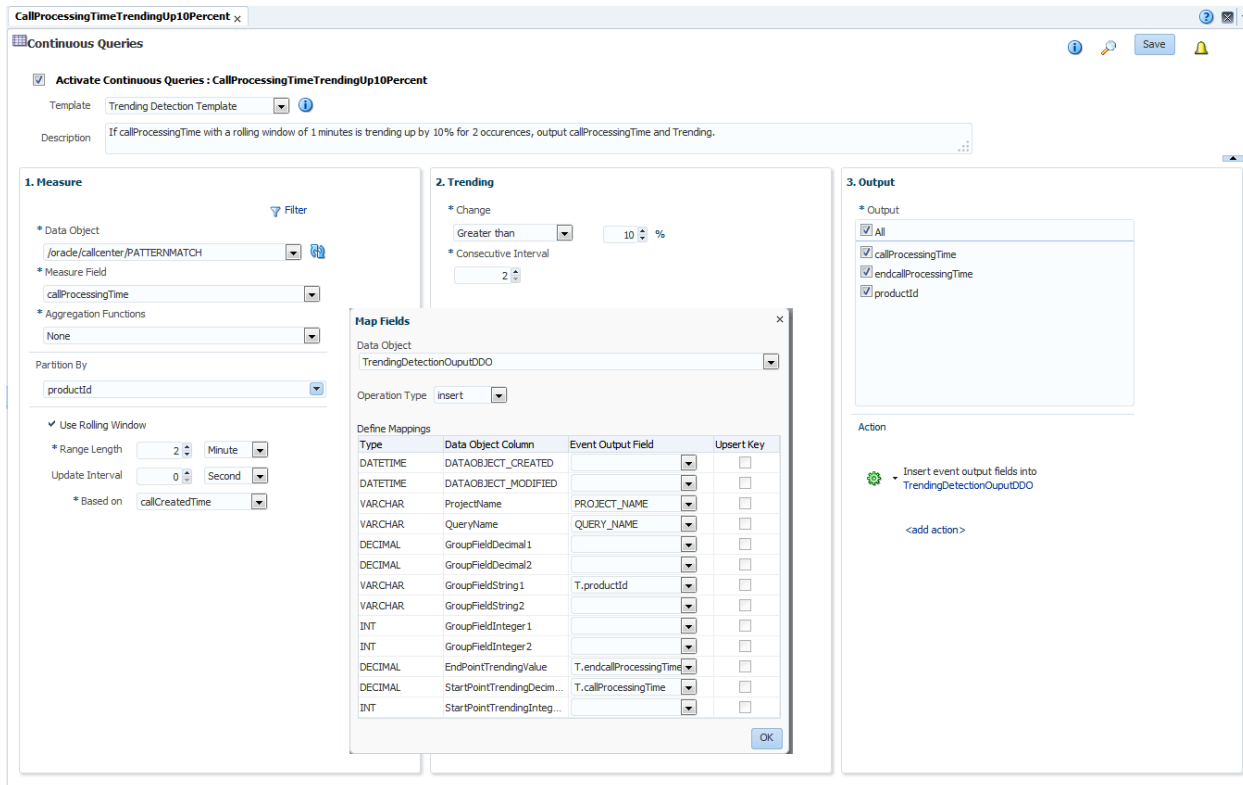


Figure 9 – Create a continuous query using the Trending Detection Template

Click the preview icon to see what the final query generated by the template looks like. In this case, the CQL statement is as follows:

```
CREATE QUERY PatternMatch.CallProcessingTimeTrendingUp10Percent as SELECT
T.callProcessingTime , T.endcallProcessingTime , T.productId , 'PatternMatch' AS
PROJECT_NAME , 'CallProcessingTimeTrendingUp10Percent' AS QUERY_NAME FROM (
SELECT productId , callCreatedTime , callProcessingTime , endcallProcessingTime FROM (
ISTREAM(SELECT productId , callCreatedTime AS callCreatedTime , callProcessingTime AS
```

```

callProcessingTime , callProcessingTime AS endcallProcessingTime FROM
PatternMatch.PATTERNMATCH[RANGE 2 minute ON callCreatedTime] AS S ) AS P ORDER
BY callCreatedTime ) AS Q MATCH_RECOGNIZE ( PARTITION BY productId MEASURES
C.callProcessingTime AS endcallProcessingTime,A.callProcessingTime AS
callProcessingTime,A.productId AS productId ALL MATCHES PATTERN (A B+ C) DEFINE B AS
B.callProcessingTime>1.1*prev(B.callProcessingTime) and count(*) <= 2, C AS
C.callProcessingTime>1.1*last(B.callProcessingTime) and count(*) = 3 ) AS T destination
"combined:jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.m
db.alertengine;jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqserv
ice.mdb.reportcache?batch=true"

```

Over Stream DO (Query: StrCallProcessingTimeTrendUp10PC)

Data Object: PATTERNMATCH_FACT_STREAM

To continue, follow the same procedure as outlined in the Trending Detection Template Query over Logical DO.

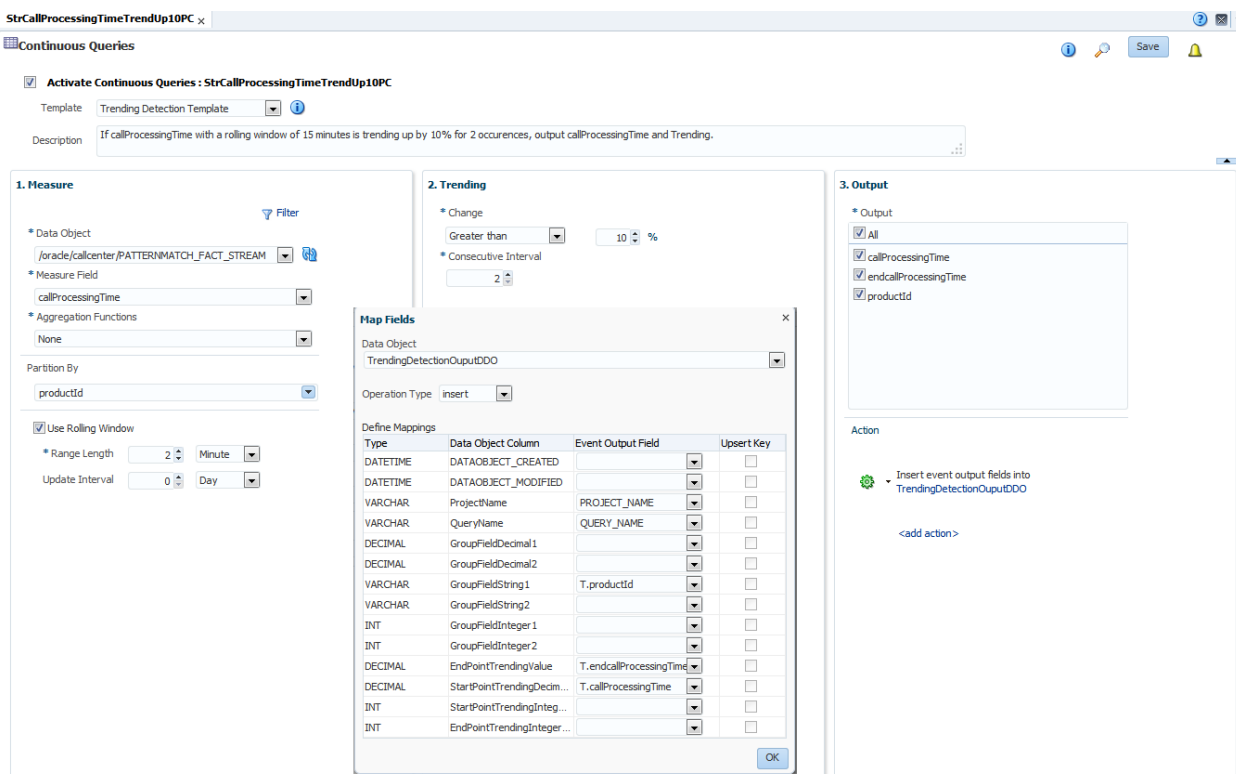


Figure 10 – Creating the StrCallProcessingTimeTrendUp10PC query

Click the preview icon to see what the final query generated by the template looks like. In this case the CQL statement is as follows:

```

CREATE QUERY PatternMatch.StrCallProcessingTimeTrendUp10PC as SELECT
T.callProcessingTime , T.endcallProcessingTime , T.productId , 'PatternMatch' AS

```

```
PROJECT_NAME , 'StrCallProcessingTimeTrendUp10PC' AS QUERY_NAME FROM (
ISTREAM(SELECT productId , callProcessingTime AS callProcessingTime , callProcessingTime
AS endcallProcessingTime FROM PatternMatch.PATTERNMATCH_FACT_STREAM[RANGE 2
minute ] AS S) ) AS Q MATCH_RECOGNIZE ( PARTITION BY productId MEASURES
C.callProcessingTime AS endcallProcessingTime,A.callProcessingTime AS
callProcessingTime,A.productId AS productId ALL MATCHES PATTERN (A B+ C) DEFINE B AS
B.callProcessingTime>1.1*prev(B.callProcessingTime) and count(*) <= 2, C AS
C.callProcessingTime>1.1*last(B.callProcessingTime) and count(*) = 3 ) AS T destination
"combined;jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.m
db.alertengine;jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqserv
ice.mdb.reportcache?batch=true"
```

Designing a Missing Event Template

The goal is to design a query that can detect when a call “Suspend” action is not directly followed by “Resume”. This is achieved using the Missing Event Template which supports all types of DOs.

Over Logical DO (Query: MissingResumeCallDetection)

Data Object: PatternMatch

- **Filter**

- ✓ None No filter specified.

- **Measure**

- ✓ productId
You can select any additional fields that you wish to expose in the query output.
- ✓ Partition by: productId
This implies that that the PatternMatching is done independently for each productId. Consider the following order of events
productId, callStatus
1, Suspend
2, Open -> This will not trigger an output because productId one and two are tracked separately
1, Resume
2, Suspend
1, Closed->This will not trigger an output because productId one and two are tracked separately
2, On Hold → This will trigger an output because for productId=2, ‘Suspend’ was followed by ‘On Hold’ instead of ‘Resume’
- ✓ Use rolling windows = true, Range Length two minutes, based on callCreatedTime.

- **Events**

- ✓ Pattern: Event A is not followed directly by event B
The other options available are “Event C is missing between event A and Event B” and “Event A is not followed by Event B in specific duration”

- **Output**

Output all fields

- **Action**

Insert Event output into MissingEventOutputDDO, and use the following mapping

DataObject column	Event Output Field
ProjectName	PROJECT_NAME
QueryName	QUERY_NAME
MeasureString1	T. B_productId
MeasureString2	T.B_callStatus

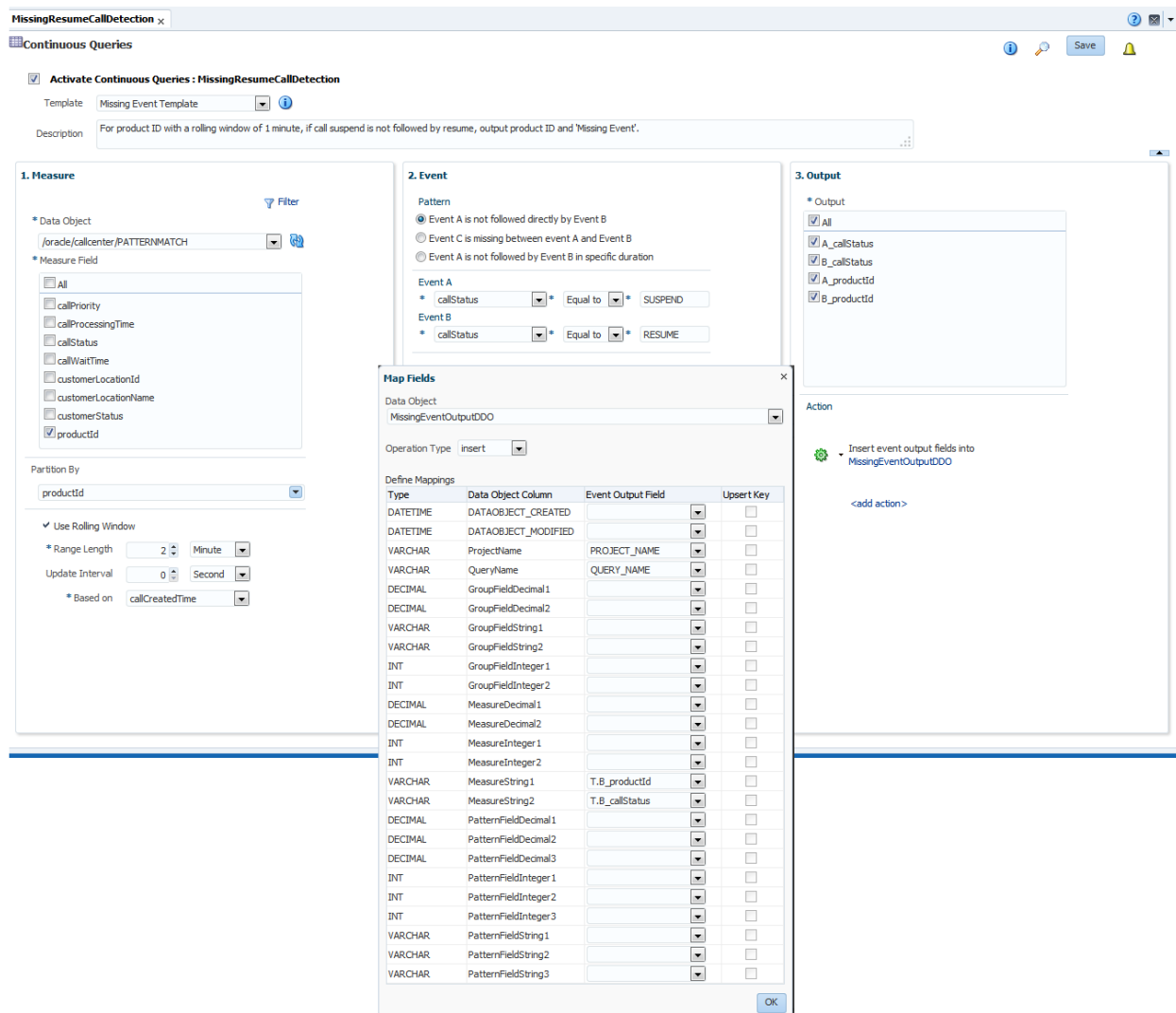


Figure 11 – Setting up the MissingResumeCallDetection Query

Click the preview icon to see what the final query generated by the template looks like. In this case, the CQL statement is as follows:

```
CREATE QUERY PatternMatch.MissingResumeCallDetection as SELECT T.A_callStatus ,
T.B_callStatus , T.A_productId , T.B_productId , 'PatternMatch' AS PROJECT_NAME ,
'MissingResumeCallDetection' AS QUERY_NAME FROM ( SELECT callCreatedTime , callStatus
, productId FROM ( ISTREAM(SELECT callCreatedTime , callStatus , productId FROM
PatternMatch.PATTERNMATCH[RANGE 2 minute ON callCreatedTime] AS S ) ) AS P ORDER
BY callCreatedTime ) AS Q MATCH_RECOGNIZE ( PARTITION BY productId MEASURES
A.callStatus AS A_callStatus,NOTB.callStatus AS B_callStatus,A.productId AS
A_productId,NOTB.productId AS B_productId ALL MATCHES PATTERN (A NOTB) DEFINE A
AS (callStatus='SUSPEND'),NOTB AS (NOT(callStatus='RESUME')) ) AS T destination
"combined:jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.m
```

db.alertengine;jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqservice.mdb.reportcache?batch=true"

Over Stream DO (StrMisResumCallDetection)

Data Object: PATTERNMATCH_FACT_STREAM

To continue, use the same procedure as the one outlined for the Missing Event Template Query over Logical DO.

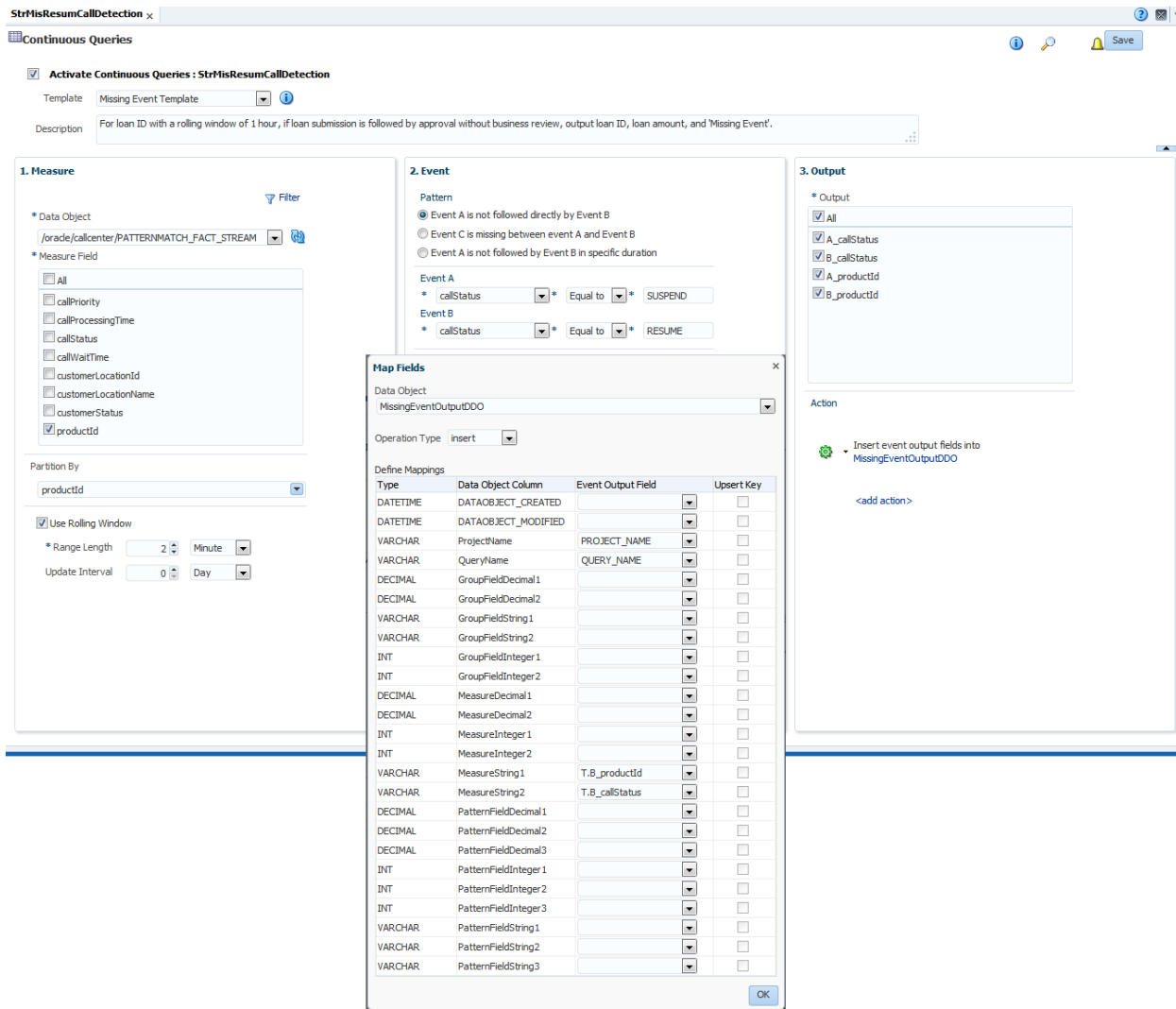


Figure 12 – Setting up the StrMisResumCallDetection Query

Click the preview icon to see what the final query generated by the template looks like. In this case, the CQL statement is as follows.

```
CREATE QUERY PatternMatch.StrMisResumCallDetection as SELECT T.A_callStatus ,
T.B_callStatus , T.A_productId , T.B_productId , 'PatternMatch' AS PROJECT_NAME ,
```

```
'StrMisResumCallDetection' AS QUERY_NAME FROM ( ISTREAM(SELECT callStatus ,
productld FROM PatternMatch.PATTERNMATCH_FACT_STREAM[RANGE 2 minute ] AS S )
AS Q MATCH_RECOGNIZE ( PARTITION BY productld MEASURES A.callStatus AS
A_callStatus,NOTB.callStatus AS B_callStatus,A.productld AS A_productld,NOTB.productld AS
B_productld ALL MATCHES PATTERN (A NOTB) DEFINE A AS (callStatus='SUSPEND'),NOTB
AS (NOT(callStatus='RESUME')) ) AS T destination
"combined:jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.m
db.alertengine;jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqserv
ice.mdb.reportcache?batch=true"
```

Designing a Monitor Count Template

The goal is to design a query that will indicate if more than one event is detected in the past 2 minutes for the same customerLocationId and productld pair. This is achieved using Monitor Count Template which supports all types of Dos.

Over Logical DO (Query: MonitorDuplicatedEvents)

Data Object: Patternmatch

- **Filter**

- ✓ None. No filter specified.

- ✓ **Measure**

- ✓ Measure Field: Callstatus
You will get a comma separated list of all call statuses in the output field AGGcallStatus.
 - ✓ Group By: customerLocationId, productld
The count of each of these pairs is monitored separately.
 - ✓ Count: Greater than 1
You have the option to specify a variety of options like greater than, less than, equal to, and so on.
 - ✓ Use rolling window: yes, Range length two minutes, based on callCreatedTime.

- **Output**

Output all fields

- **Action**

Insert Event output into MonitorCntOutputDDO, and use the following mapping.

DataObject column	Event Output Field
ProjectName	PROJECT_NAME
QueryName	QUERY_NAME

GroupFieldString1	T. customerLocationId
GroupFieldString2	T. productId
MeasureString2	T. AGGcallStatus

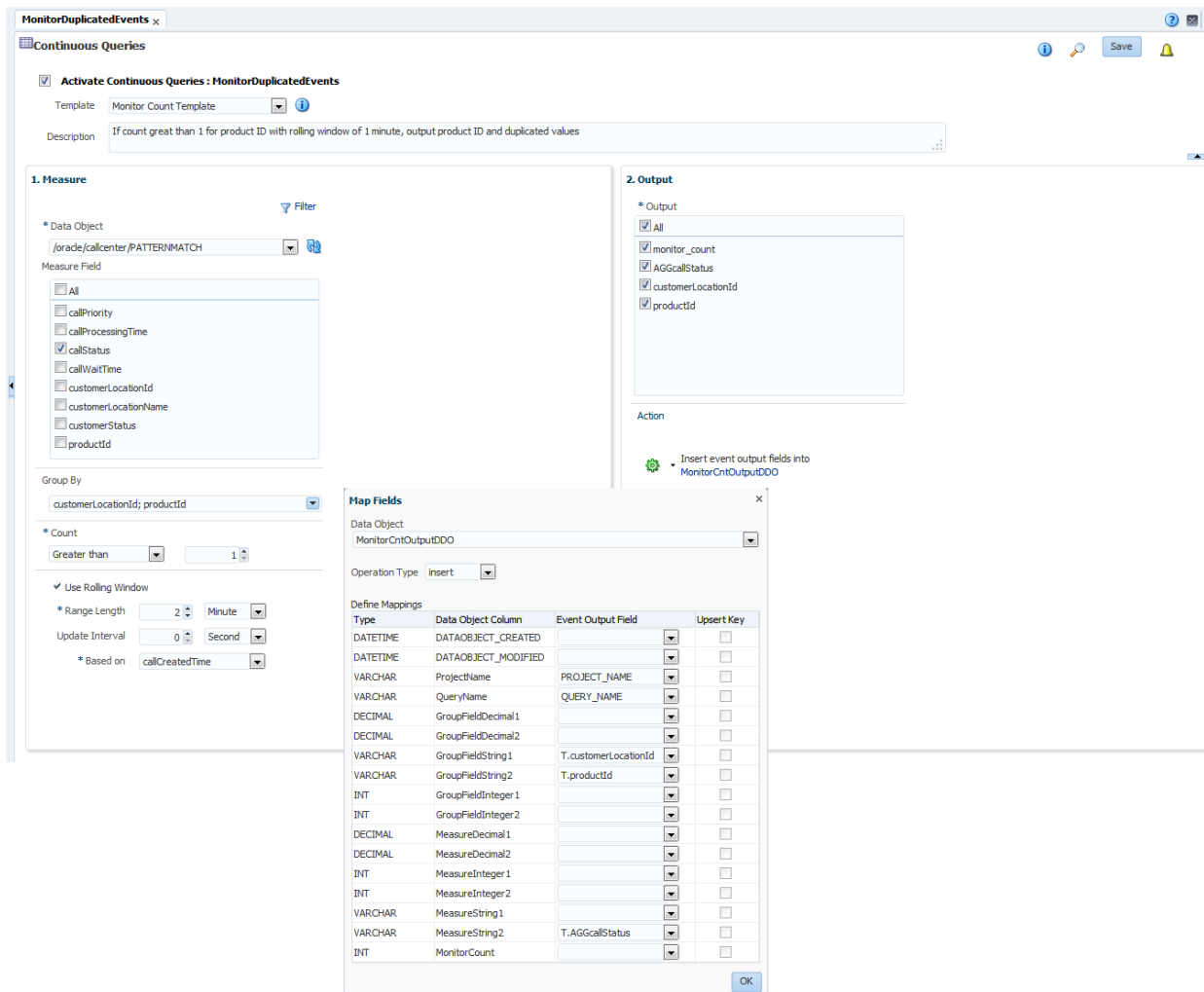


Figure 13 – Setting up the MonitorDuplicatedEvents Query

Click the preview icon to see what the final query generated by the template looks like. In this case, the CQL statement is as follows.

```
CREATE QUERY PatternMatch.MonitorDuplicatedEvents as SELECT T.monitor_count ,
T.AGGcallStatus , T.customerLocationId , T.productId , 'PatternMatch' AS PROJECT_NAME ,
'MonitorDuplicatedEvents' AS QUERY_NAME FROM ( ISTREAM(SELECT count(*) AS
monitor_count , listAggWrapper(listAgg(callStatus)," ") AS AGGcallStatus , customerLocationId
AS customerLocationId , productId AS productId FROM PatternMatch.PATTERNMATCH[RANGE
2 minute ON callCreatedTime] AS S GROUP BY S.customerLocationId,S.productId HAVING
COUNT(*) > 1) ) AS T destination
"combined:jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.m
```

db.alertengine;jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqservice.mdb.reportcache?batch=true"

Over Stream DO (StrAvgCallWaitTimeGreaterThanThreshold)

Data Object: PATTERNMATCH_FACT_STREAM

To continue, follow the same procedure as outlined for the Monitor Count Template Query over Logical DO.

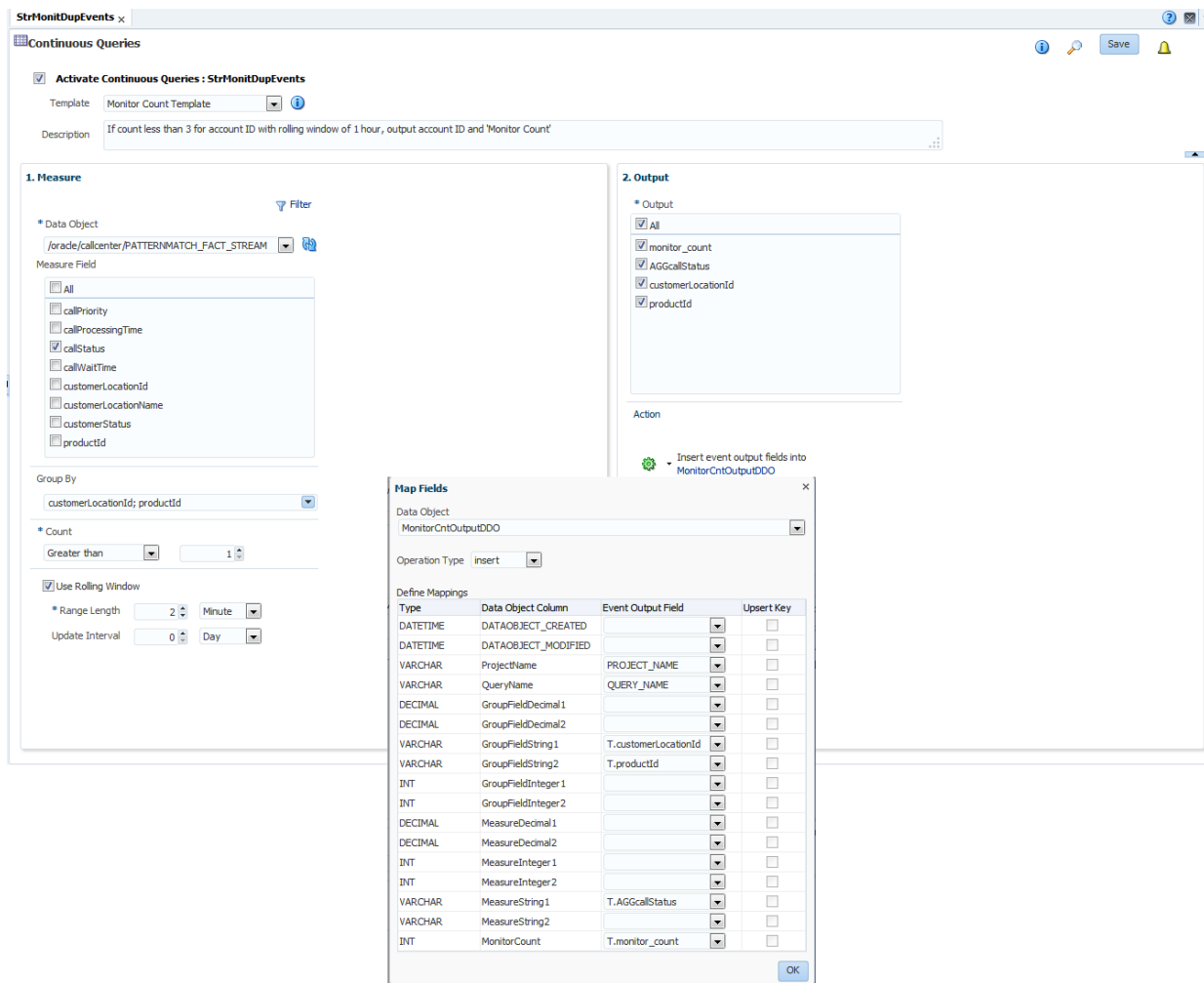


Figure 14 – Setting up the StrMonitDupEvents Query

Click the preview icon to see what the final query generated by the template looks like. In this case the CQL statement is as follows.

```
CREATE QUERY PatternMatch.StrMonitDupEvents as SELECT T.monitor_count ,
T.AGGcallStatus , T.customerLocationId , T.productId , 'PatternMatch' AS PROJECT_NAME ,
'StrMonitDupEvents' AS QUERY_NAME FROM ( ISTREAM(SELECT count(*) AS monitor_count
, listAggWrapper(listAgg(callStatus)," ") AS AGGcallStatus , customerLocationId AS
customerLocationId , productId AS productId FROM
```

```
PatternMatch.PATTERNMATCH_FACT_STREAM[RANGE 2 minute ] AS S GROUP BY
S.customerLocationId,S.productId HAVING COUNT(*) > 1 ) AS T destination
"combined:jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.m
db.alertengine;jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqserv
ice.mdb.reportcache?batch=true"
```

Designing a Moving Aggregation Template

The goal is to design a query that will output the moving average of callProcessing time in the last two minutes for each productId. This is achieved using the Moving Aggregation Template, which supports all types of DOs.

Over Logical DO (Query: MovingAverageForCallProcessingTime)

Data Object: PatternMatch

- **Filter**

- ✓ None. No filter specified.

- **Measure**

- ✓ Measure field: callProcessingTime (Average)
The other options are max, min, count and sum. You can select multiple aggregation functions for the same field.
 - ✓ Group By: productId
 - ✓ Use rolling window = True; Range length -two minutes, based on callCreatedTime.

- **Output**

- ✓ All fields.

- **Action**

Insert Event output into MovAggrOutputDDO, and use the following mapping.

DataObject column	Event Output Field
ProjectName	PROJECT_NAME
QueryName	QUERY_NAME
GroupFieldString1	T.productId
AggregationValue1	T.AVGcallProcessingTime

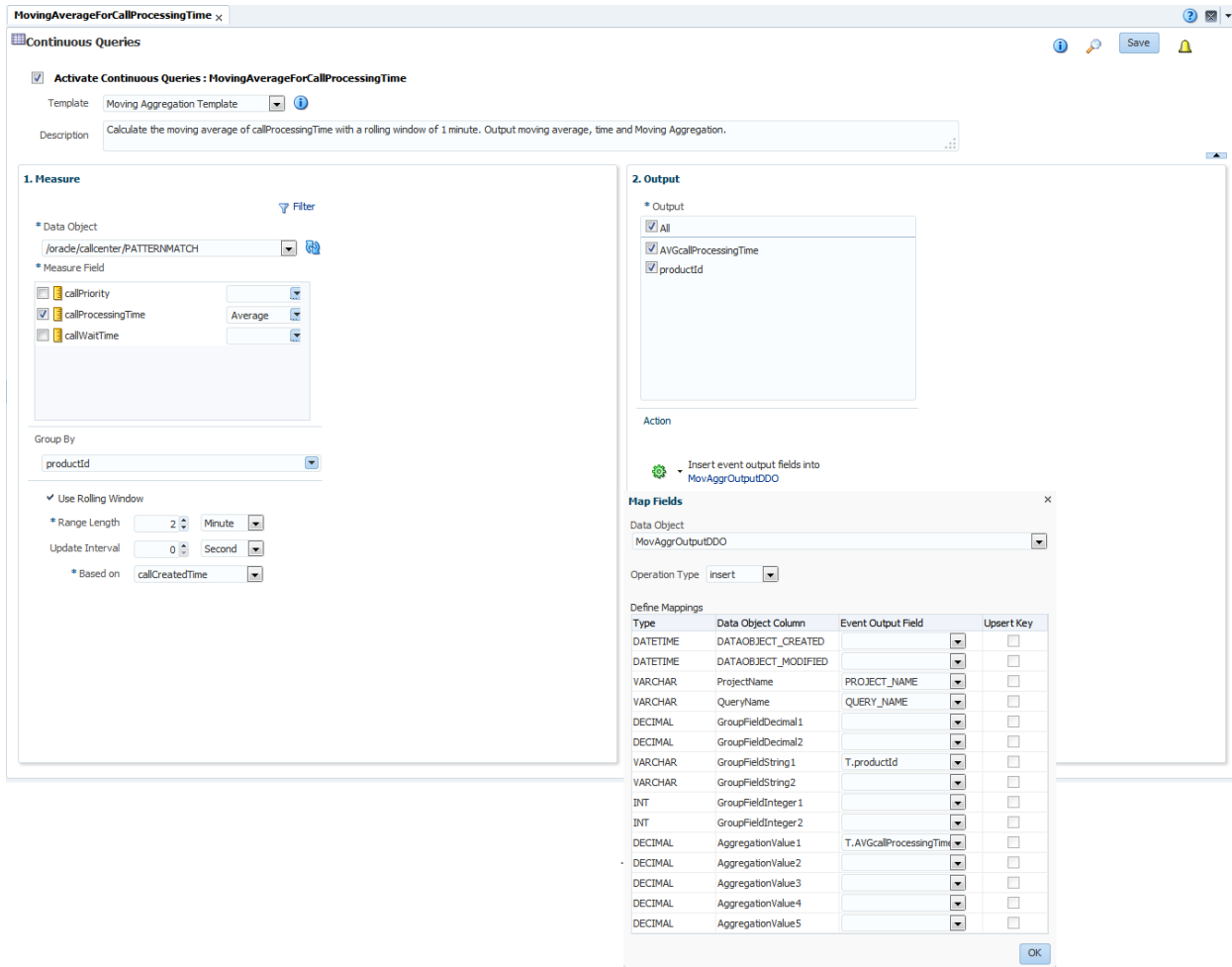


Figure 15 – Setting up the MovingAverageforCallProcessingTime Query

Click the preview icon to see what the final query generated by the template looks like. In this case, the CQL statement is as follows.

```
CREATE QUERY PatternMatch.MovingAverageForCallProcessingTime as SELECT
T.AVGcallProcessingTime , T.productId , 'PatternMatch' AS PROJECT_NAME ,
'MovingAverageForCallProcessingTime' AS QUERY_NAME FROM ( ISTREAM(SELECT
AVG(callProcessingTime) AS AVGcallProcessingTime , productId AS productId FROM
PatternMatch.PATTERNMATCH[RANGE 2 minute ON callCreatedTime] AS S GROUP BY
S.productId ) AS T destination
```

```
"combined:jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.m
db.alertengine;jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqserv
ice.mdb.reportcache?batch=true"
```

Over Stream DO (Query Name: StrMovinAvgCallProcessTime)

Data Object: PATTERNMATCH_FACT_STREAM

To continue, follow the same procedure as outlined for the Moving Aggregation Template Query over Logical DO.

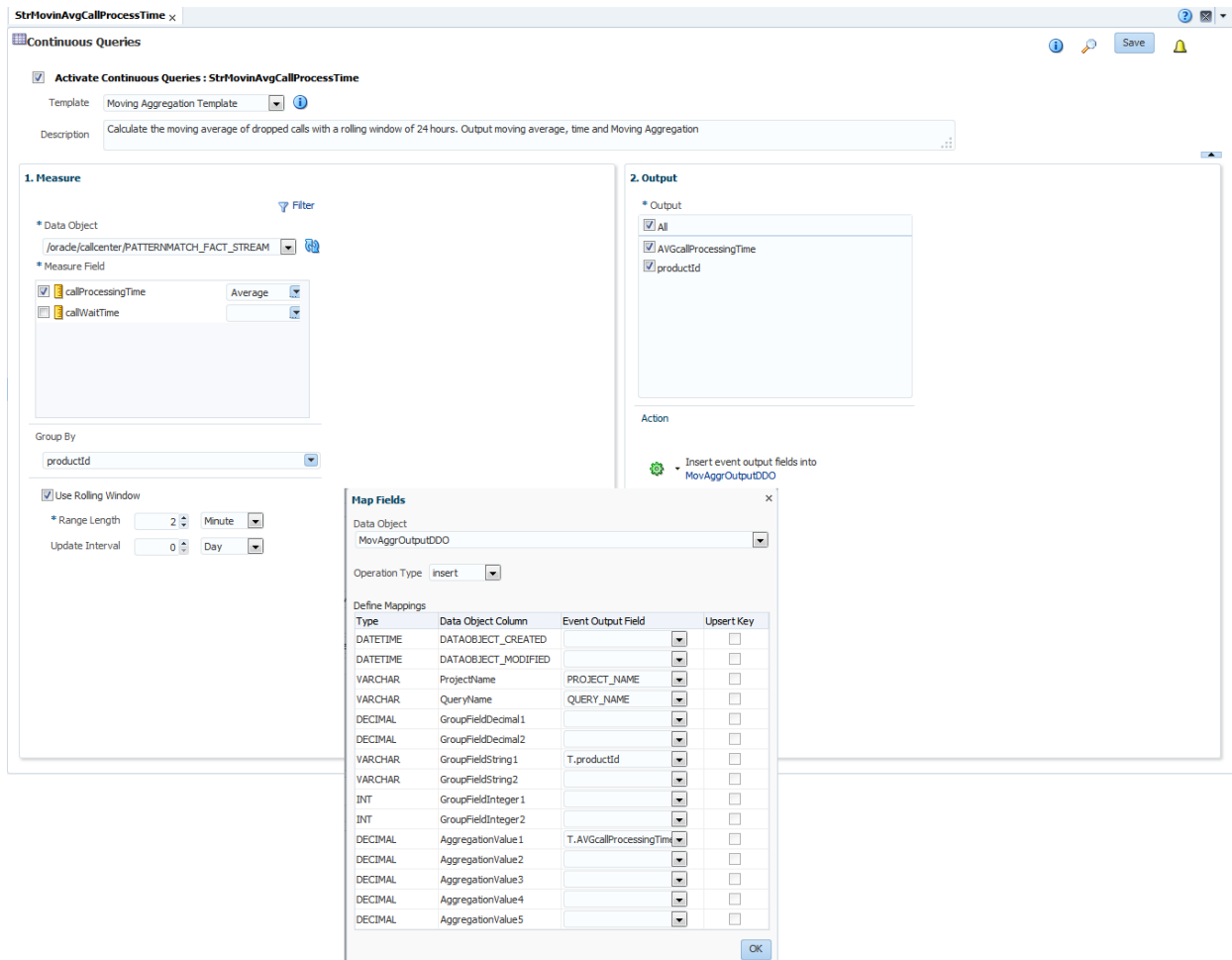


Figure 16 – Setting up the StrMovinAvgCallProcessTime Query

Click the preview icon to see what the final query generated by the template looks like. In this case the CQL statement is as follows.

```
CREATE QUERY PatternMatch.StrMovinAvgCallProcessTime as SELECT
T.AVGcallProcessingTime , T.productId , 'PatternMatch' AS PROJECT_NAME ,
'StrMovinAvgCallProcessTime' AS QUERY_NAME FROM ( ISTREAM(SELECT
AVG(callProcessingTime) AS AVGcallProcessingTime , productId AS productId FROM
PatternMatch.PATTERNMATCH_FACT_STREAM[RANGE 2 minute ] AS S GROUP BY
S.productId) ) AS T destination
```

```
"combined;jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.m
db.alertengine;jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqserv
ice.mdb.reportcache?batch=true"
```

Designing a Top N Template

The goal is to design a query that will indicate the top five products with the maximum callProcessingTime in the last two minutes. This is achieved using Top N Template, which supports all types of DOs.

Over Logical DO (Query: Top5MaximumCallProcessingTime)

Data Object: Patternmatch

- **Filter**

- ✓ None. No filter specified.

- **Measure**

- ✓ Measure Field: callProcessingTime
 - ✓ Aggregation Function: Max
 - ✓ Group By: productId
 - ✓ Top N: 5
Here, you can specify the number of top rows you want in the output.
 - ✓ Use rolling window: True; Range length 2 Minute, based on callCreatedTime

- **Output**

Output all fields.

- **Action**

Insert Event output into TopNOutputDDO, and use the following mapping.

DataObject column	Event Output Field
ProjectName	PROJECT_NAME
QueryName	QUERY_NAME
GroupFieldString1	T. productId
TopNValue	T. MAXcallProcessingTime

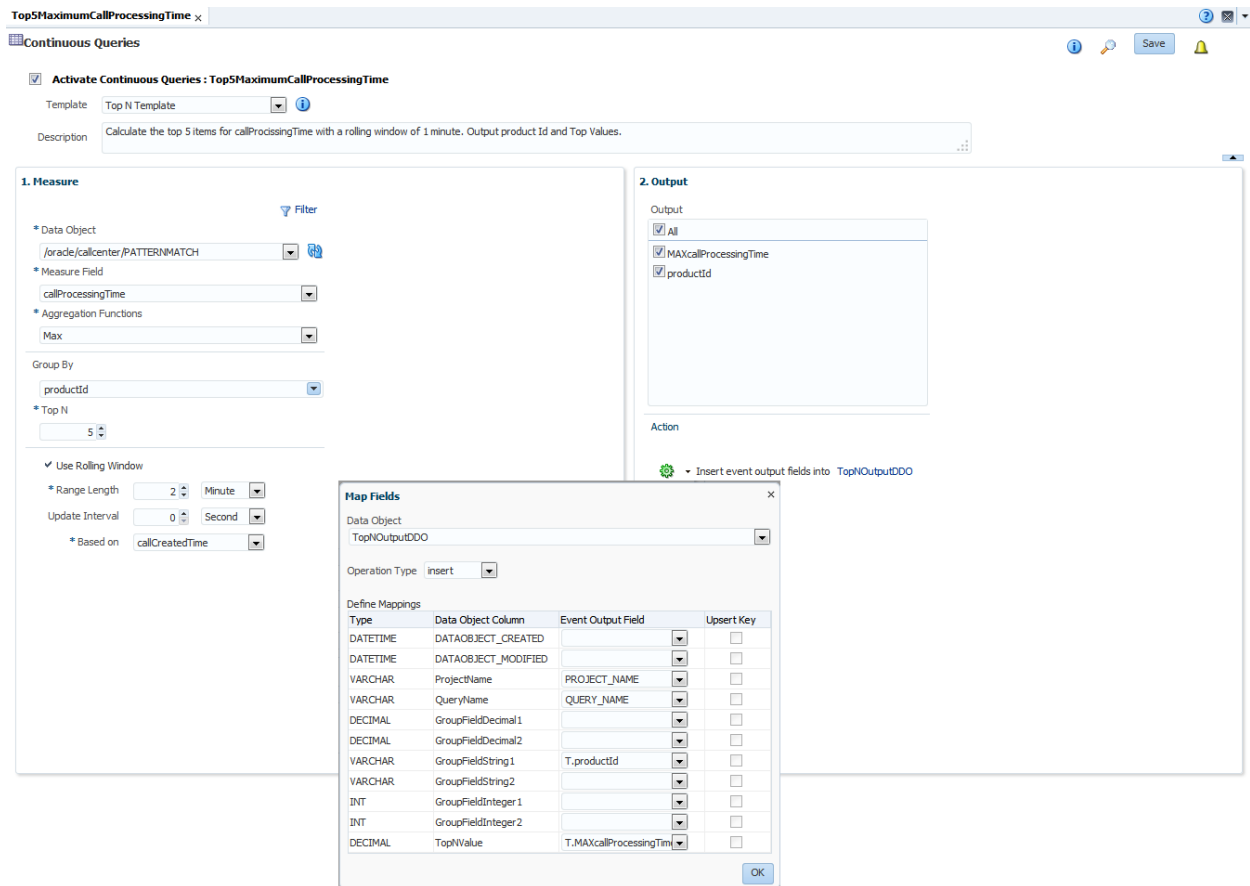


Figure 17 – Setting up the Top5MaximumCallProcessingTime Query

Click the preview icon to see what the final query generated by the template looks like. In this case the CQL statement is as follows.

```
CREATE QUERY PatternMatch.Top5MaximumCallProcessingTime as SELECT
T.MAXcallProcessingTime , T.productId , 'PatternMatch' AS PROJECT_NAME ,
'Top5MaximumCallProcessingTime' AS QUERY_NAME FROM ( RSTREAM(SELECT
S.productId , S.MAXcallProcessingTime FROM ( SELECT productId , MAX(callProcessingTime)
AS MAXcallProcessingTime FROM PatternMatch.PATTERNMATCH[RANGE 2 minute ON
callCreatedTime] AS Q GROUP BY Q.productId ) AS S ORDER BY S.MAXcallProcessingTime
DESC ROWS 5) ) AS T destination
"combined:jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.m
db.alertengine;jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqserv
ice.mdb.reportcache?batch=true"
```

Over Stream DO (Query Name: StrTop5MaxCallProcessTime)

Data Object: PATTERNMATCH_FACT_STREAM

To continue, follow the same procedure as the one outlined for the Top N template Query over Logical DO.

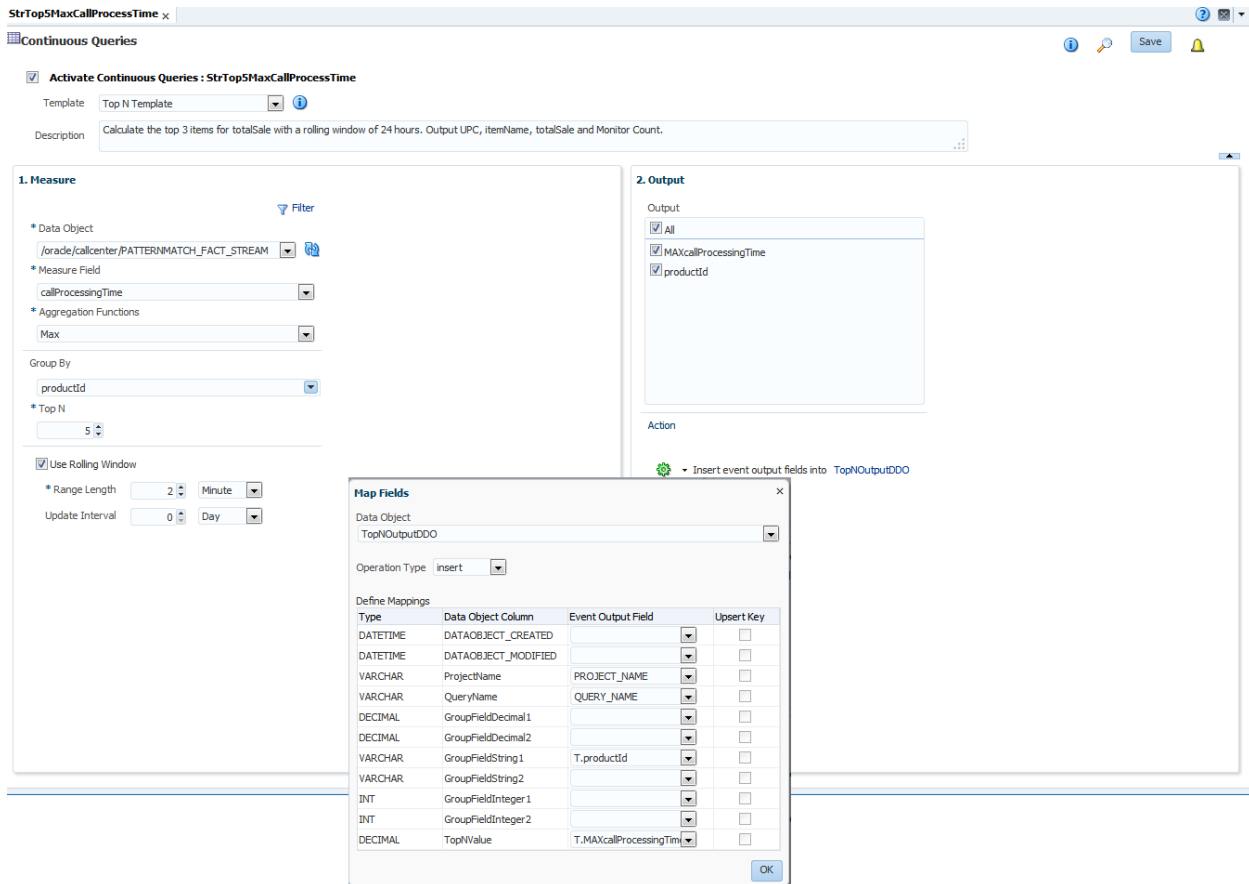


Figure 18 – Setting up the StrTop5MaxCallProcessTime Query

Click the preview icon to see what the final query generated by the template looks like. In this case the CQL statement is as follows.

```
CREATE QUERY PatternMatch.StrTop5MaxCallProcessTime as SELECT
T.MAXcallProcessingTime , T.productId , 'PatternMatch' AS PROJECT_NAME ,
'StrTop5MaxCallProcessTime' AS QUERY_NAME FROM ( RSTREAM(SELECT S.productId ,
S.MAXcallProcessingTime FROM ( SELECT productId , MAX(callProcessingTime) AS
MAXcallProcessingTime FROM PatternMatch.PATTERNMATCH_FACT_STREAM[RANGE 2
minute ] AS Q GROUP BY Q.productId ) AS S ORDER BY S.MAXcallProcessingTime DESC
ROWS 5) ) AS T destination
```

```
"combined:jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.m
db.alertengine;jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqserv
ice.mdb.reportcache?batch=true"
```

Designing a Duplicate Detection Template

The goal is to design a query that will provide an output if more than one event arrives within two minutes of each other with the same CustomerLocationId, productId and callStatus. This is achieved using the Duplicate Detection Template. This template supports only Stream DOs and

does not currently support filters. The use of the rolling window is mandatory and the update interval field is not supported.

Over Stream DO (StrDuplicateDetection)

Data Object: PATTERNMATCH_FACT_STREAM

- **Measure**

- ✓ Measure fields: customerLocationId, productId, callStatus
- ✓ Use Rolling window: True; Range length: two minutes

- **Output**

- ✓ Output All fields

- **Action**

Insert Event output into DupDetectionOutputDDO, and use the following mapping.

DataObject column	Event Output Field
ProjectName	PROJECT_NAME
QueryName	QUERY_NAME
GroupFieldString1	T.customerLocationId
GroupFieldString2	T.productId
MeasureString1	T.callStatus

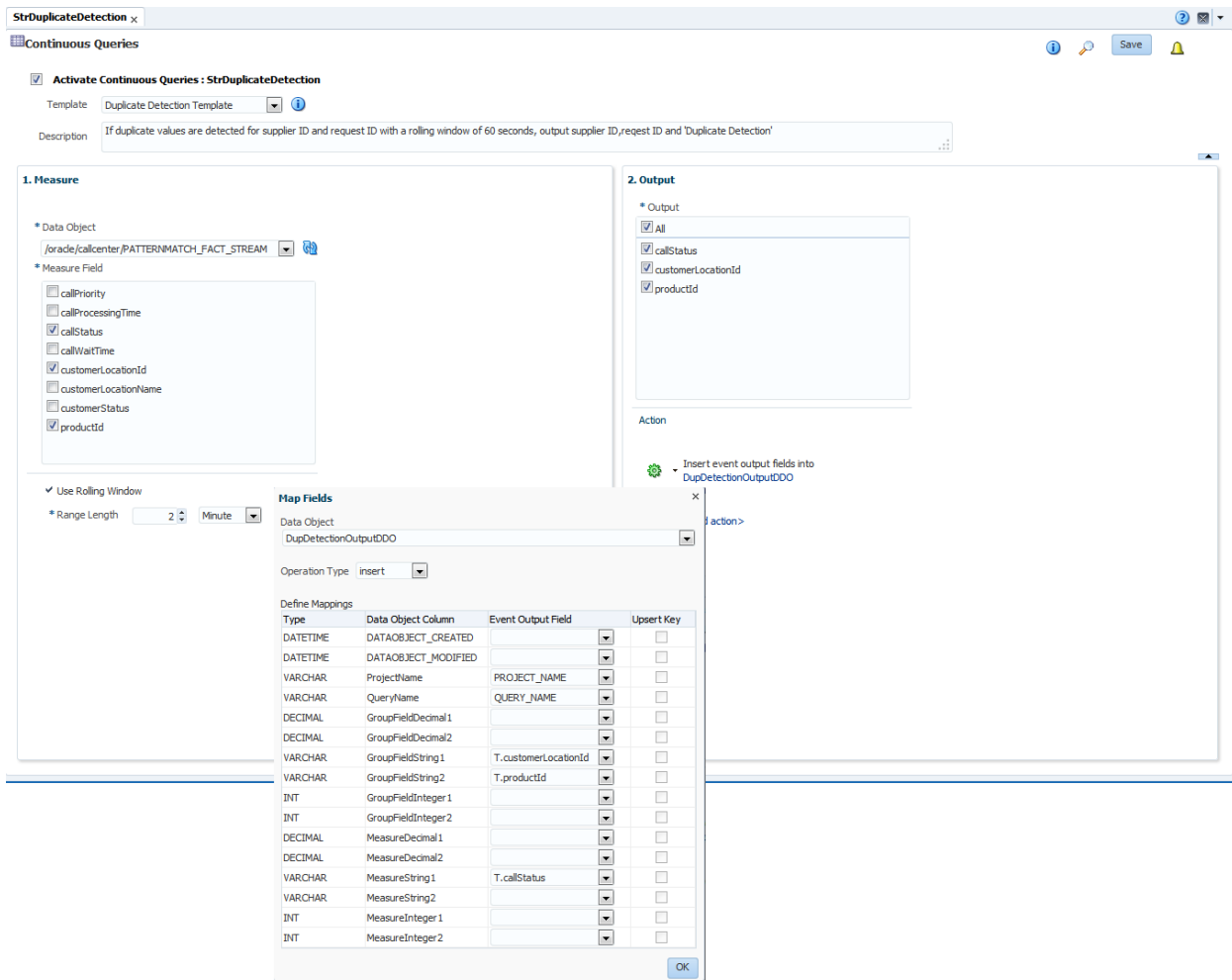


Figure 19 – Setting up the StrDuplicateDetection Query

Click the preview icon to see what the final query generated by the template looks like. In this case the CQL statement is as follows.

```
CREATE QUERY PatternMatch.StrDuplicateDetection as SELECT T.callStatus ,
T.customerLocationId , T.productId , 'PatternMatch' AS PROJECT_NAME ,
'StrDuplicateDetection' AS QUERY_NAME FROM ( ISTREAM(SELECT S.callStatus ,
S.customerLocationId , S.productId FROM
PatternMatch.PATTERNMATCH_FACT_STREAM[NOW] AS
D,PatternMatch.PATTERNMATCH_FACT_STREAM[RANGE 2 minute ] AS S WHERE
D.callStatus=S.callStatus AND D.customerLocationId=S.customerLocationId AND
D.productId=S.productId GROUP BY S.callStatus,S.customerLocationId,S.productId HAVING
COUNT(*)>1) ) AS T destination
```

"combined;jms:queue/oracle.beam.cqservice.mdb.alertengine:queuecf/oracle.beam.cqservice.mdb.alertengine;jms:queue/oracle.beam.cqservice.mdb.reportcache:queuecf/oracle.beam.cqservice.mdb.reportcache?batch=true"

Continuous Queries Monitoring

You can use the Continuous Queries Monitoring screen to see Active queries. This is accessed from the Continuous Queries Monitoring link on the Administrator panel. This page gives you the following details:

- ✓ Name of the query
- ✓ Status
- ✓ Server on which the query is currently running and the complete CQL Statement.

Project	Query	Status	Server	Statement
PatternMatch	StrAvgCallWaitTimeGreaterThanThresh...	Active	bam_server1	CREATE QUERY StrAvgCallWaitTimeGreaterThanThreshold as ISTREAM(SELECT AVG(callWaitTime) AS MEASURE, 100.0 AS NORM, 10.0 AS ALLOWED_DEVIATION, AVG...
PatternMatch	StrTop5MaxCallProcessTime	Active	bam_server1	CREATE QUERY StrTop5MaxCallProcessTime as SELECT T.MAXcallProcessingTime, T.productId, 'PatternMatch' AS PROJECT_NAME, 'StrTop5MaxCallProcessTime' AS QU...
PatternMatch	Top5MaximumCallProcessingTime	Active	bam_server1	CREATE QUERY Top5MaximumCallProcessingTime as SELECT T.MAXcallProcessingTime, T.productId, 'PatternMatch' AS PROJECT_NAME, 'Top5MaximumCallProcessingTi...
PatternMatch	StrMsResumCallDetection	Active	bam_server1	CREATE QUERY StrMsResumCallDetection as SELECT T.A_callStatus, T.B_callStatus, T.A_productId, T.B_productId, 'PatternMatch' AS PROJECT_NAME, 'StrMsResum...
PatternMatch	CallProcessingTimeTrendingUp10Percent	Active	bam_server1	CREATE QUERY CallProcessingTimeTrendingUp10Percent as SELECT T.callProcessingTime, T.endcallProcessingTime, T.productId, 'PatternMatch' AS PROJECT_NAME, 'C...
PatternMatch	MissingResumeCallDetection	Active	bam_server1	CREATE QUERY MissingResumeCallDetection as SELECT T.A_callStatus, T.B_callStatus, T.A_productId, T.B_productId, 'PatternMatch' AS PROJECT_NAME, 'MissingRe...
PatternMatch	AverageCallWaitingTimeGreateThanTh...	Active	bam_server1	CREATE QUERY AverageCallWaitingTimeGreateThanThreshold as ISTREAM(SELECT AVG(callWaitTime) AS MEASURE, 100.0 AS NORM, 10.0 AS ALLOWED_DEVIATION, ...
PatternMatch	StrMoniDupEvents	Active	bam_server1	CREATE QUERY StrMoniDupEvents as SELECT T.monitor_count, T.AGGcallStatus, T.customerLocationId, T.productId, 'PatternMatch' AS PROJECT_NAME, 'StrMoniDu...
PatternMatch	StrMovinAvgCallProcessTime	Active	bam_server1	CREATE QUERY StrMovinAvgCallProcessTime as SELECT T.AVGcallProcessingTime, T.productId, 'PatternMatch' AS PROJECT_NAME, 'StrMovinAvgCallProcessTime' AS Q...
PatternMatch	MovingAverageForCallProcessingTime	Active	bam_server1	CREATE QUERY MovingAverageForCallProcessingTime as SELECT T.AVGcallProcessingTime, T.productId, 'PatternMatch' AS PROJECT_NAME, 'MovingAverageForCallPro...
PatternMatch	MonitorDuplicatedEvents	Active	bam_server1	CREATE QUERY MonitorDuplicatedEvents as SELECT T.monitor_count, T.AGGcallStatus, T.customerLocationId, T.productId, 'PatternMatch' AS PROJECT_NAME, 'Monit...
PatternMatch	StrCallProcessingTimeTrendUp10PC	Active	bam_server1	CREATE QUERY StrCallProcessingTimeTrendUp10PC as SELECT T.callProcessingTime, T.endcallProcessingTime, T.productId, 'PatternMatch' AS PROJECT_NAME, 'StrCall...
PatternMatch	StrDuplicateDetection	Active	bam_server1	CREATE QUERY StrDuplicateDetection as SELECT T.callStatus, T.customerLocationId, T.productId, 'PatternMatch' AS PROJECT_NAME, 'StrDuplicateDetection' AS QUE...

Statement
 CREATE QUERY StrAvgCallWaitTimeGreaterThanThreshold as ISTREAM(SELECT AVG(callWaitTime) AS MEASURE, 100.0 AS NORM, 10.0 AS ALLOWED_DEVIATION, AVG(callWaitTime) - 100.0 AS ACTUAL_DEVIATION, productId AS productId, 'PatternMatch' AS PROJECT_NAME, 'StrAvgCallWaitTimeGreaterThanThreshold' AS QUERY_NAME FROM PATTERNMATCH_FACT_STREAM[RANGE 2 minute] AS T WHERE (callStatus='CLOSED')) GROUP BY T.productId HAVING AVG(T.callWaitTime) - 100.0 > 10.0) destination "combinedjms:queue/oracle.beam.cqservice.mdb.alertengine:queue/oracle.beam.cqservice.mdb.alertengine:jms:queue/oracle.beam.cqservice.mdb.reportcache:queue/oracle.beam.cqservice.mdb.reportcache?batch=true"

Figure 20 – Continuous Queries Monitoring Screen.

Designing a Fired KPI Alert Events Query

The goal is to design a group query which will give the count of events fired by the CQL 'AverageCallWaitingTimeGreateThanThreshold' for each minute. The continuous query's output is constantly written back to KPIAlertOutputDDO, so this is achieved by creating a group query on KPIAlertOutputDDO with adequate filters on ProjectName and QueryName. The auto populated field 'DATAOBJECT_CREATED' is present in every DO and it will always contain the time at which the record was created.

Data Object: /oracle/writeback/KPIAlertOutputDDO

Measure (Y-axis)

- Norm(Count)

Dimensions (X-axis)

- DATAOBJECT_CREATED
 - ✓ Time Grouping: Use Time Series
 - ✓ Time Unit: Minute
 - ✓ Continuous time series: true
 - ✓ Quantity: 1

Filters

- Branch :All Are True
 - ✓ ProjectName is equal to 'PatternMatch'
 - ✓ QueryName is equal to 'AverageCallWaitingTimeGreateThanThreshold'

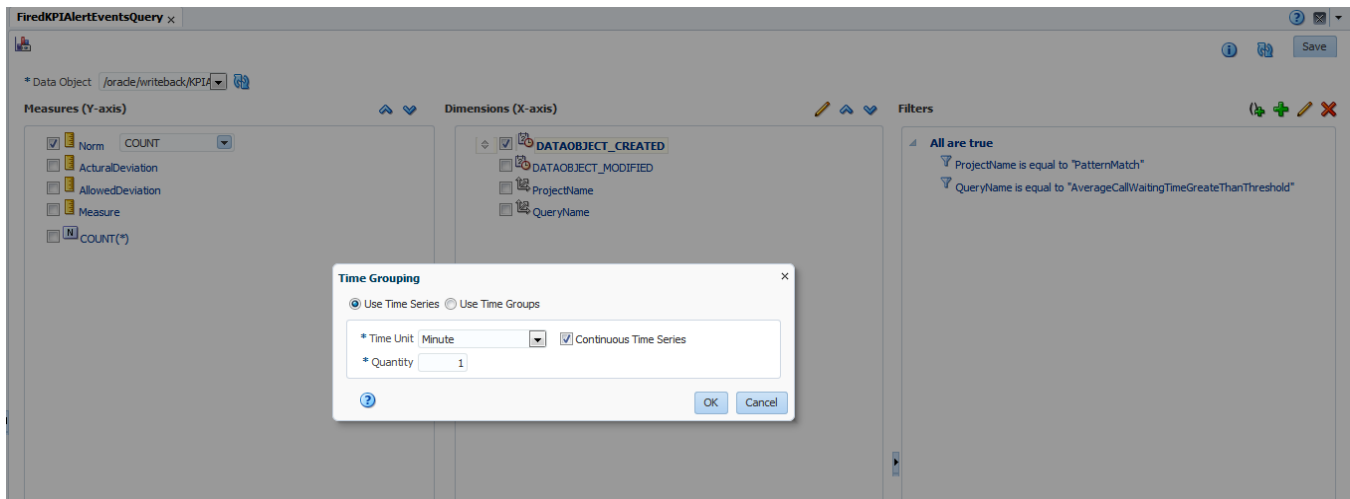


Figure 21 – Time Grouping Feature

You can design the other group queries by repeating the procedure outlined for the previous queries. Ensure that you select the appropriate Data Object and set the correct Query name in the filter condition “QueryName is equal to”.

Designing Fired KPI Alert Output Events

The goal is to create an Active business view using a line chart based on the query FiredKPIAlertEventsQuery. The View must be an active view, which means it must show the data for the last 10 minutes, or any other specified time value, and keep updating automatically.

Business Query: FiredKpiAlertOutputEvents

Category: Line, **View Type:** Line

- **Run Time-interaction**

- ✓ Click on “Run time interaction” and click on Active Data tab.
- ✓ Check “Turn this query into a continuous query”
- ✓ Check “Use time window”
- ✓ Sliding Range based on DATAOBJECT_CREATED
- ✓ Range Length: 10 minutes
- ✓ Update interval: 1 second

This converts the Group SQL query into a continuous query with the range being 10 minutes and slide being one second.

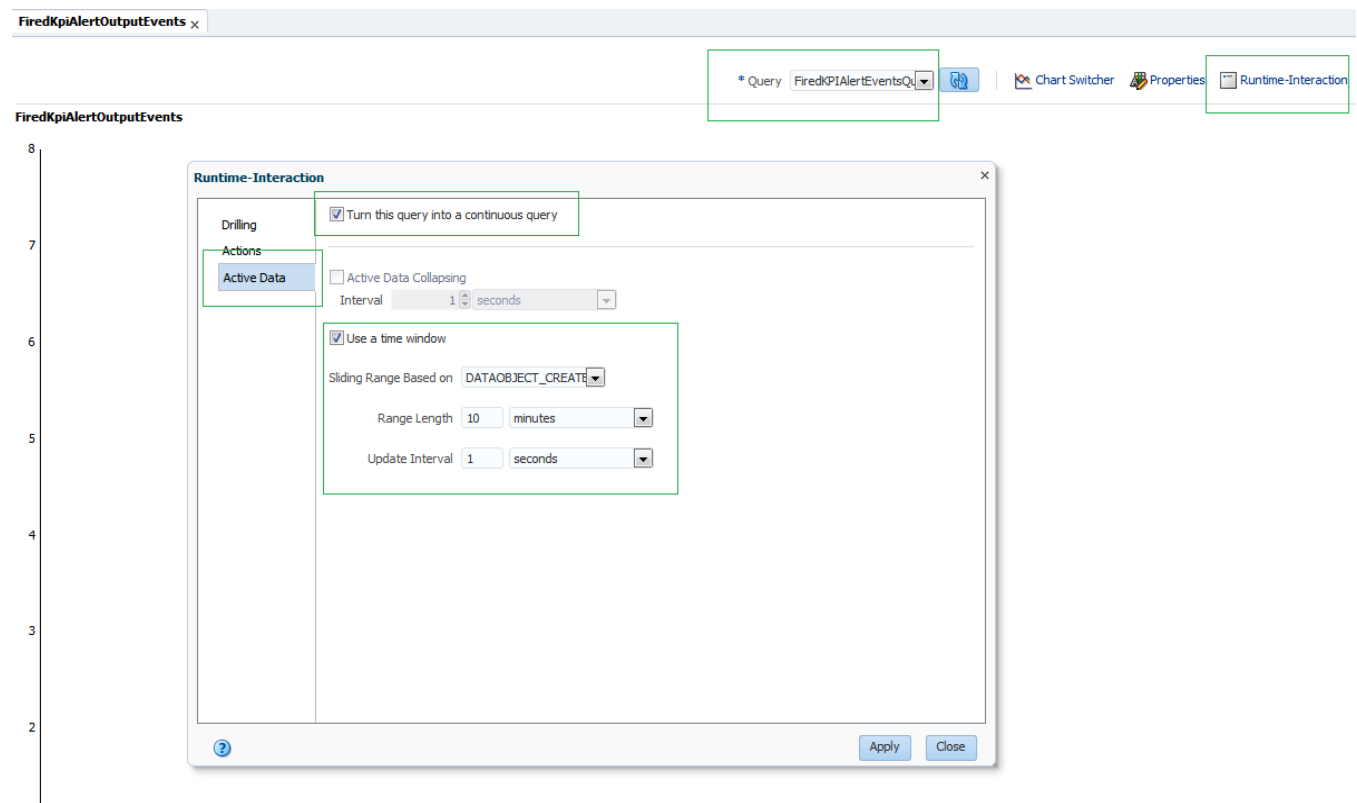


Figure 22 – Setting up the FiredKpiAlertOutputEvents Query

You can design all business views using the aforementioned procedure. Ensure that you select appropriate query names from the drop down.

Understanding Dashboards

PatternMatch Dashboard

The goal is to create a dashboard with six views to show the output rate of all Continuous Queries based on the PatternMatch Logical DO. From the PatternMatch Project, create a Dashboard using the “Type 8” Style template as shown in [Figure 23](#).

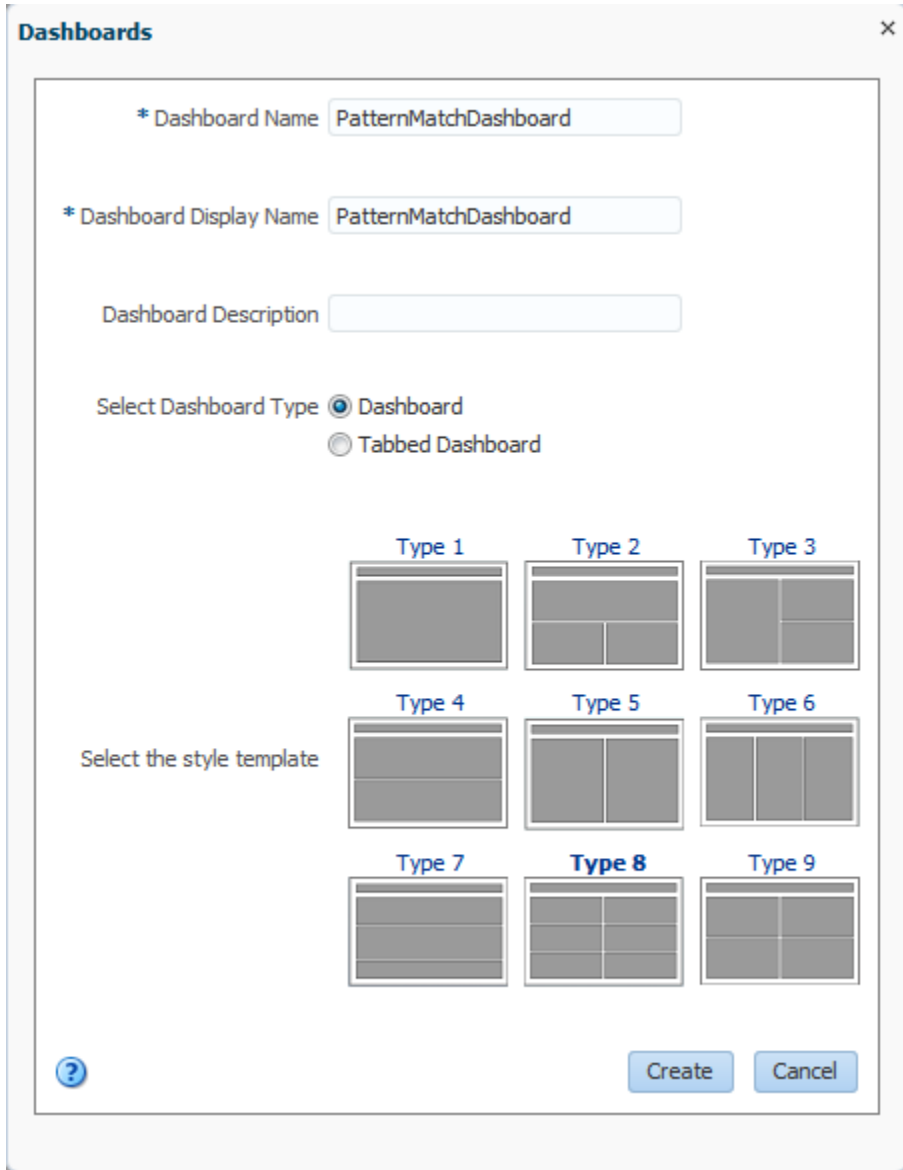


Figure 23 – Creating a PatternMatch Dashboard

Then, drag and drop the following views into one of the cells in the dashboard as shown in [Figure 24](#).

- FiredKpiAlertOutputEvents
- FiredMissingEvents
- FiredMonitorCountEvents
- FiredMovingAggregationEvents
- FiredTrendingEvents
- FiredTopNEvents

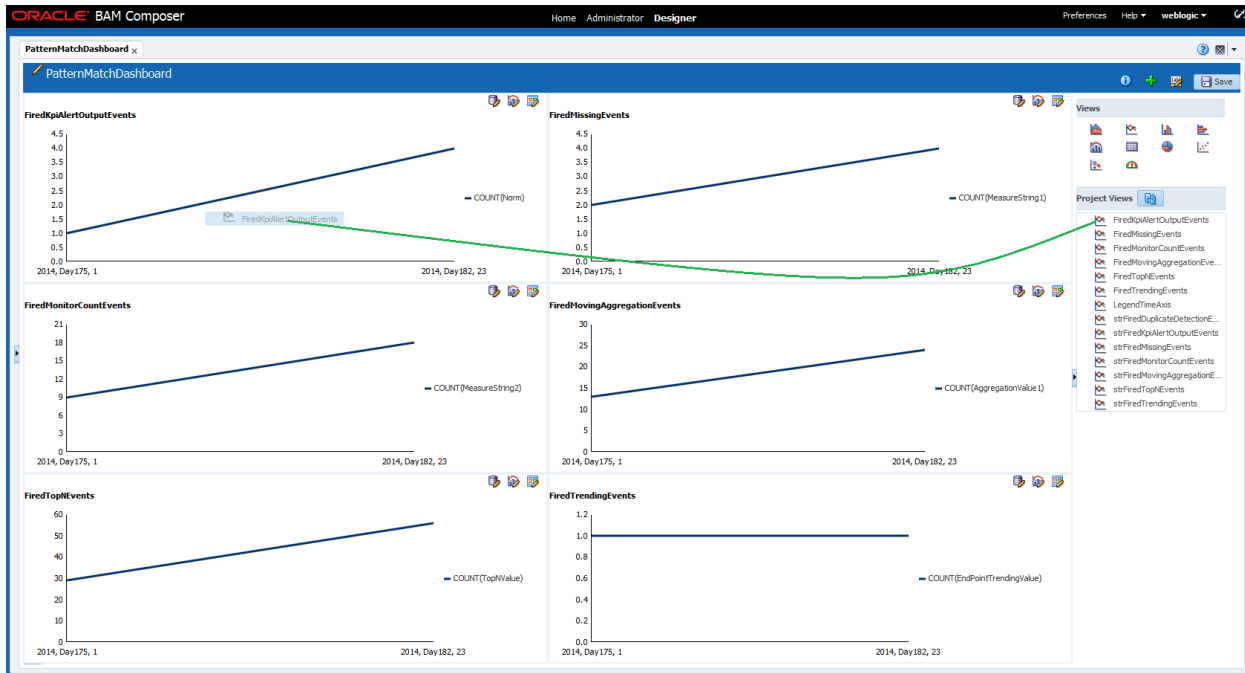


Figure 24 – PatternMatch Dashboard with Various Business Views

The business views are now added to PatternMatch Dashboard.

StrPatternMatch Dashboard

You can create a strPatternMatch Dashboard using the procedure outlined for creating the PatternMatch Dashboard, and add the following views to it.

- ✓ strFiredKpiAlertOutputEvents
- ✓ strFiredMissingEvents
- ✓ strFiredMonitorCountEvents
- ✓ strFiredMovingAggregationEvents
- ✓ strFiredTopNEvents
- ✓ strFiredTrendingEvents

Add an extra cell in the bottom row to accommodate the additional strFiredDuplicateDetectionEvents

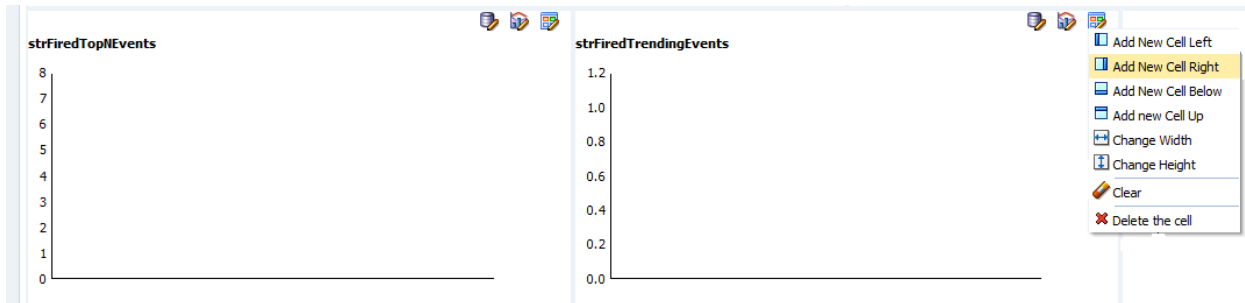


Figure 25 – Adding a New Cell to accommodate the additional strFiredDuplicateDetectionEvents.

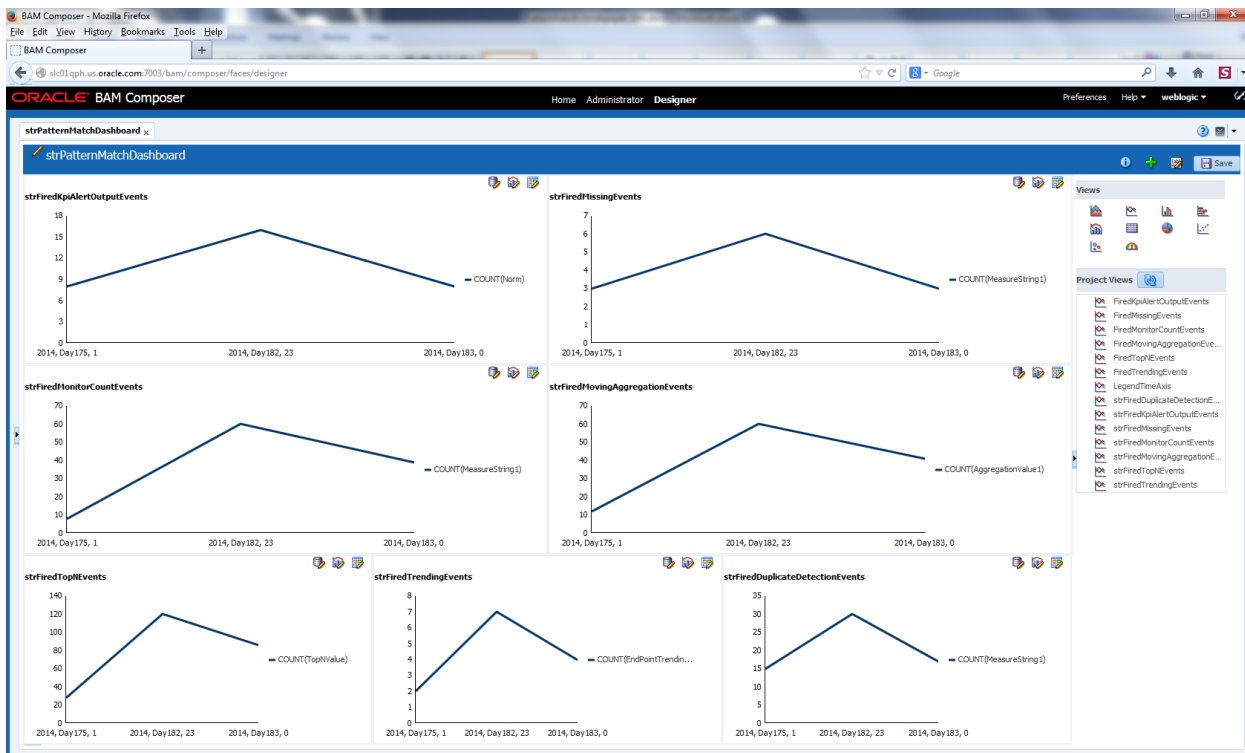


Figure 26 – The strPatternMatch Dashboard

Troubleshooting

1. If your artifacts are not imported:
 - Verify that samples/common/setEnv.sh has correct values for all parameters
 - Verify that the correct password is specified in <password></password> tag of bin/ BAMCommandConfig.xml
 - Check for any specific errors in console or in log files that are generated in the bin directory, for example, bamcommand.log.*

2. If you get errors while populating data:

- Verify that samples/common/setEnv.sh has correct values for all parameters
3. If the dashboard is not showing any data:

- In case of PatternMatch Dashboard not showing any data, check if the PATTERNMATCH_DIMENSION DO has the following data:
- customerlocationId,customerLocationName
US, United States
CA, CANADA

The PATTERNMATCH_FACT DO is populated with data to ensure that the dimension side is not empty. This ensures that the PatternMatch Logical DO is populated with data, thereby satisfying the join condition. If the two aforementioned rows are not present in the dimension DO, you must add them manually and try again.

- Ensure that the writeback DOs are populated with new data every minute. If not, open the Continuous Queries Monitoring screen and check if all the 13 CQL Template queries are Active. If not, navigate to the designer, open each query individually, and save it again with the “Activate Continuous Query” checkbox checked

Best Practices

For queries over Archived Relation DOs (or Derived or Logical DOs built on top of Archived Relations), depending on your expected input event rates, ensure that you specify an appropriate “Range Length”. If you specify a large value, those many events must be fetched from the database which can result in serious performance implications. In such cases, the “use rolling window” option is checked by default and you cannot change it. This avoids an overload in fetching records from the database while starting the query.

blogs.oracle.com/oracle

facebook.com/oracle

twitter.com/oracle

oracle.com



Oracle Corporation, World Headquarters

500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries

Phone: +1.650.506.7000
Fax: +1.650.506.7200

Hardware and Software, Engineered to Work Together

CONNECT WITH US

 blogs.oracle.com/oracle

 facebook.com/oracle

 twitter.com/oracle

 oracle.com

Copyright © 2014, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0914

 | Oracle is committed to developing practices and products that help protect the environment