

Oracle® Fusion Middleware

Understanding Stream Analytics



F18224-01
August 2019



Oracle Fusion Middleware Understanding Stream Analytics,

F18224-01

Copyright © 2018, 2019, Oracle and/or its affiliates. All rights reserved.

Primary Author: Oracle Corporation

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

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.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

| | |
|-----------------------------|-----|
| Audience | vi |
| Documentation Accessibility | vi |
| Related Documents | vi |
| Conventions | vii |

1 Overview of Oracle Stream Analytics

| | |
|--|-----|
| 1.1 About Oracle Stream Analytics | 1-1 |
| 1.2 Why Oracle Stream Analytics? | 1-2 |
| 1.3 How Does Oracle Stream Analytics Work? | 1-2 |

2 Getting to Know Artifacts in Oracle Stream Analytics

| | |
|---|-----|
| 2.1 Understanding Different Types of Connections | 2-1 |
| 2.2 Understanding Streams | 2-2 |
| 2.3 Understanding References | 2-3 |
| 2.4 Understanding Geo Fences | 2-3 |
| 2.4.1 What is a Manual Geo Fence? | 2-4 |
| 2.4.2 What is a Database-based Geo Fence? | 2-4 |
| 2.5 Understanding Pipelines | 2-4 |
| 2.6 Understanding Dashboards | 2-5 |
| 2.7 Understanding Cubes | 2-5 |
| 2.8 Understanding Stream Analytics Patterns | 2-6 |
| 2.8.1 What is the Spatial: Speed Pattern? | 2-7 |
| 2.8.2 What is the Geo Code Pattern? | 2-7 |
| 2.8.3 What is the Interaction: Single Stream Pattern? | 2-7 |
| 2.8.4 What is the Interaction: Two Stream Pattern? | 2-7 |
| 2.8.5 What is the Spatial: Point to Polygon Pattern? | 2-7 |
| 2.8.6 What is the Proximity: Single Stream Pattern? | 2-8 |
| 2.8.7 What is the Proximity: Two Stream Pattern? | 2-8 |
| 2.8.8 What is the Proximity: Stream with Geo Fence Pattern? | 2-8 |
| 2.8.9 What is the Direction Pattern? | 2-8 |

| | | |
|--------|--|------|
| 2.8.10 | What is the Geo Fence Pattern? | 2-8 |
| 2.8.11 | What is the Geo Fence Filter: Inside Pattern? | 2-9 |
| 2.8.12 | What is the Reverse Geo Code: Near By Pattern? | 2-9 |
| 2.8.13 | What is the Reverse Geo Code: Near By Place Pattern? | 2-9 |
| 2.8.14 | What is the Correlation Pattern? | 2-9 |
| 2.8.15 | What is the Quantile Pattern? | 2-9 |
| 2.8.16 | What is the Detect Duplicates Pattern? | 2-9 |
| 2.8.17 | What is the Change Detector Pattern? | 2-9 |
| 2.8.18 | What is the W Pattern? | 2-10 |
| 2.8.19 | What is the 'A' Followed by 'B' Pattern? | 2-10 |
| 2.8.20 | What is the Top N Pattern? | 2-10 |
| 2.8.21 | What is the Bottom N Pattern? | 2-10 |
| 2.8.22 | What is the Up Trend Pattern? | 2-10 |
| 2.8.23 | What is the 'A' Not Followed by 'B' Pattern? | 2-10 |
| 2.8.24 | What is the Down Trend Pattern? | 2-11 |
| 2.8.25 | What is the Union Pattern? | 2-11 |
| 2.8.26 | What is the Fluctuation Pattern? | 2-11 |
| 2.8.27 | What is the Inverse W Pattern? | 2-11 |
| 2.8.28 | What is the Eliminate Duplicates Pattern? | 2-11 |
| 2.8.29 | What is the Detect Missing Event Pattern? | 2-11 |
| 2.8.30 | What is the Left Outer Join Pattern? | 2-12 |
| 2.9 | Understanding Shapes | 2-12 |
| 2.10 | Understanding Target | 2-12 |
| 2.11 | Understanding the Predictive Model | 2-13 |
| 2.12 | Understanding Custom JARs | 2-13 |
| 2.13 | Understanding Export and Import | 2-13 |

3 Overview of the Components of a Pipeline

| | | |
|-------|------------------------------|-----|
| 3.1 | Understanding Query Stage | 3-1 |
| 3.1.1 | What is Filter | 3-2 |
| 3.1.2 | What is Correlation | 3-2 |
| 3.1.3 | What is Summary | 3-2 |
| 3.1.4 | What is Group By | 3-2 |
| 3.1.5 | What is Retain All Columns | 3-3 |
| 3.1.6 | What is Range | 3-3 |
| 3.1.7 | What is Evaluation Frequency | 3-4 |
| 3.1.8 | What is Visualization | 3-4 |
| 3.2 | Understanding Rules | 3-5 |
| 3.3 | Understanding Rule Stage | 3-5 |
| 3.4 | Understanding Pattern Stage | 3-5 |

| | | |
|---------|--|------|
| 3.5 | Understanding Custom Stage | 3-5 |
| 3.6 | Understanding Scoring Stage | 3-5 |
| 3.7 | Understanding Query Group | 3-6 |
| 3.7.1 | What is Query Group Stage: Stream? | 3-6 |
| 3.7.2 | What is Query Group Stage: Table? | 3-6 |
| 3.8 | Understanding the Live Output Table | 3-6 |
| 3.9 | Understanding Visualizations | 3-7 |
| 3.9.1 | What is a Bar Type of Visualization? | 3-8 |
| 3.9.2 | What is a Line Type of Visualization? | 3-8 |
| 3.9.3 | What is An Area Type of Visualization? | 3-9 |
| 3.9.4 | What is a Stacked Bar Type of Visualization? | 3-9 |
| 3.9.5 | What is a Spatial Type of Visualization? | 3-9 |
| 3.9.6 | What is a Pie Chart? | 3-9 |
| 3.9.7 | What is a Bubble Chart? | 3-9 |
| 3.9.8 | What is a Thematic Map? | 3-9 |
| 3.9.9 | What is a Scatter Chart? | 3-9 |
| 3.10 | Understanding the Topology Viewer | 3-10 |
| 3.10.1 | What is Immediate Family? | 3-10 |
| 3.10.2 | What is Extended Family? | 3-10 |
| 3.11 | Understanding Expression Builder Functions | 3-11 |
| 3.11.1 | What are Bessel Functions? | 3-11 |
| 3.11.2 | What are Conversion Functions? | 3-12 |
| 3.11.3 | What are Date Functions? | 3-12 |
| 3.11.4 | What are Geometry Functions? | 3-13 |
| 3.11.5 | What are Interval Functions? | 3-13 |
| 3.11.6 | The YM Interval Functions | 3-14 |
| 3.11.7 | What are Math Functions? | 3-15 |
| 3.11.8 | What are Null-related Functions? | 3-17 |
| 3.11.9 | What are Statistical Functions? | 3-17 |
| 3.11.10 | What are String Functions? | 3-19 |

Preface

Understanding Oracle Stream Analytics describes what is Oracle Stream Analytics and how it works. It explains the artifacts and components of a Stream Analytics pipeline.

Topics:

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

Audience

This document is intended for users who are looking to build pipelines in Oracle Stream Analytics.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Accessible Access to Oracle Support

Oracle customers who have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Related Documents

Documentation for Oracle Stream Analytics is available on [Oracle Help Center](#).

Also see the following documents for reference:

- *Understanding Oracle Stream Analytics*
- *Quick Installer for Oracle Stream Analytics*
- *Known Issues in Oracle Stream Analytics*
- *Spark Extensibility for CQL in Oracle Stream Analytics*
- *Using Oracle Stream Analytics*

Conventions

The following text conventions are used in this document:

| Convention | Meaning |
|-----------------|--|
| boldface | Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary. |
| <i>italic</i> | Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values. |
| monospace | Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter. |

1

Overview of Oracle Stream Analytics

Oracle Stream Analytics allows for the creation of custom operational dashboards that provide real-time monitoring and analyses of event streams in an Apache Spark based system. Oracle Stream Analytics enables customers to identify events of interest in their Apache Spark based system, execute queries against those event streams in real time and drive operational dashboards or raise alerts based on that analysis. Oracle Stream Analytics runs as a set of native Spark pipelines.

Topics:

- [About Oracle Stream Analytics](#)
- [Why Oracle Stream Analytics?](#)
- [How Does Oracle Stream Analytics Work?](#)

1.1 About Oracle Stream Analytics

Stream Analytics is an in-memory technology for real-time analytic computations on streaming data. The streaming data can originate from IoT sensors, web pipelines, log files, point-of-sale devices, ATM machines, social media, or from any other data source.

Oracle Stream Analytics is used to identify business threats and opportunities by filtering, aggregating, correlating, and analyzing high volume of data in real time.

More precisely, Oracle Stream Analytics can be used in the following scenarios:

- Build complex event processing pipelines by blending and transforming data from disparate transactional and non-transactional sources.
- Perform temporal analytics based on time and event windows.
- Perform location-based analytics using built-in spatial patterns.
- Detect patterns in time-series data and execute real-time actions.
- Build operational dashboards by visualizing processed data streams.
- Use Machine Learning to score current event and predict next event.
- Run ad-hoc queries on results of processed data streams.

Some industry specific examples include:

- Detecting real-time fraud based on incoming transaction data.
- Tracking transaction losses and margins in real-time to renegotiate with vendors and suppliers.
- Improving asset maintenance by tracking healthy operating parameters and proactively scheduling maintenance.
- Improving margins by continuously tracking demand and optimizing markdowns instead of randomly lowering prices.

- Readjusting prices by continuously tracking demand, inventory levels, and product sentiment on social media etc.
- Marketing and making real-time offers based on customer location and loyalty.
- Instantly identifying shopping cart defections and improving conversion rates.
- Upselling products and services by instantly identifying customer's presence on company website.
- Improving asset utilization by tracking average time it takes to load and unload merchandise.
- Improving turnaround time by preparing dock and staff based on estimated arrival time of fleet.
- Revising schedule estimates based on actual time to enter and exit loading zones, and so on.

1.2 Why Oracle Stream Analytics?

Various reasons and advantages encourage you to use Oracle Stream Analytics instead of similar products available in the industry.

Simplicity

Author powerful data processing pipelines using self-service web-based tool in Oracle Stream Analytics. The tool automatically generates a Spark pipeline along with instant visual validation of pipeline logic.

Built on Apache Spark

Oracle Stream Analytics can attach to any version-compliant Yarn cluster running Spark and is first in the industry to bring event-by-event processing to Spark Streaming.

Enterprise Grade

Oracle Stream Analytics is built on Apache Spark to provide full horizontal scale out and 24x7 availability of mission-critical workloads. Automated check-pointing ensures exact-once processing and zero data loss. Built-in governance provides full accountability of who did what and when to the system. As part of management and monitoring, Oracle Stream Analytics provides a visual representation of pipeline topology/relationships along with dataflow metrics to indicate number of events ingested, events dropped, and throughput of each pipeline.

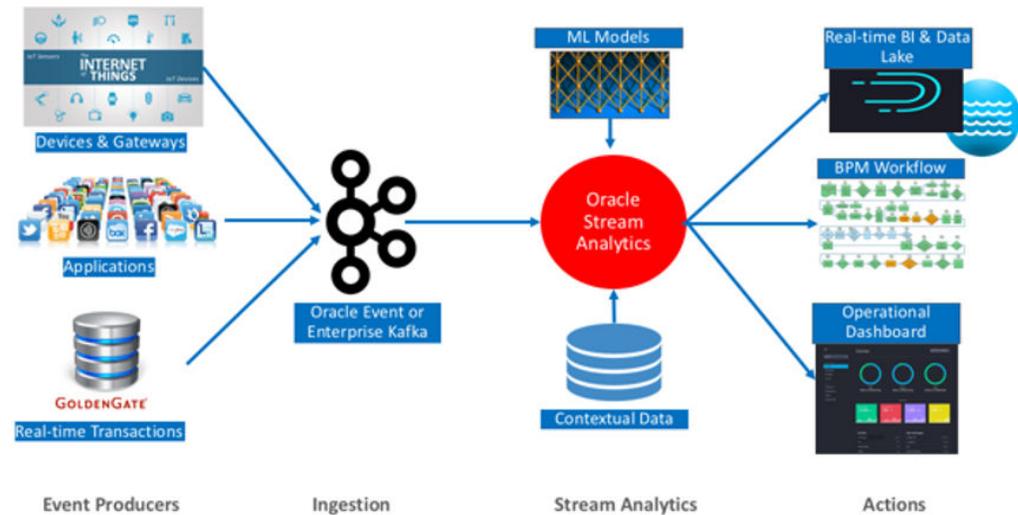
1.3 How Does Oracle Stream Analytics Work?

Stream Analytics starts with ingesting data from Kafka with first-class support for Oracle GoldenGate change data capture. Examining and analyzing the stream is performed by creating data pipelines.

A data pipeline can query data using time windows, look for patterns, and apply conditional logic while the data is still in motion. The query language used in Stream Analytics is called Continuous Query Language (CQL) and is similar to SQL, but CQL includes additional constructs for pattern matching and recognition. Though CQL is declarative, there is no need to write any code in Stream Analytics. The web-based tool automatically generates queries and the Spark Streaming pipeline. Once data is

analyzed and situation is detected, the pipeline can terminate to trigger BPM workflows in Oracle Integration or save results into a Data Lake for deeper insights and intelligence using Oracle Analytics Cloud.

The following diagram illustrates the architecture of Stream Analytics:



The analyzed data is used to build operational dashboards, trigger workflows, and it is saved to Data Lakes for business intelligence and ad-hoc queries.

2

Getting to Know Artifacts in Oracle Stream Analytics

Stream Analytics has various artifacts like connections, references, streams, targets, and many more. You create pipelines using these artifacts.

Topics

- [Understanding Different Types of Connections](#)
- [Understanding Stream](#)
- [Understanding Reference](#)
- [Understanding Geo Fence](#)
- [Understanding Pipeline](#)
- [Understanding Dashboard](#)
- [Understanding Cube](#)
- [About Stream Analytics Patterns](#)
- [Understanding Shape](#)
- [Understanding Target](#)
- [Understanding PMML](#)
- [Understanding Custom Jar](#)
- [Understanding Export and Import](#)

2.1 Understanding Different Types of Connections

A *connection* is a very basic artifact and the first entity that you need to create in the Catalog. It is a collection of metadata (such as URLs, credential and the like) required to connect to an external system. A connection is the basis for creation of sources (Streams, References or Geo Fences) and Targets.

It is possible to reuse the same connection to create multiple sources and/or targets. In other words, it can be reused to access different resources in the same system: for example different Kafka topics in the same Kafka cluster, or different database tables in the same Oracle database.

Kafka Connection

A Kafka connection has just a single parameter, the Zookeeper server URL above all the standard properties (name, description, tags) of catalog objects.

The Zookeeper URL is of the *format host:port*. If the port is not provided by the user, the system will assume the default Zookeeper port, i.e. 2181. Authentication to Kafka is not supported in this release.

Oracle Database Connection

To connect to an Oracle database, you must provide the following parameters:

- Connect using
- Service name/SID
- hostname
- port
- username
- password

Oracle Coherence Connection

Oracle Stream Analytics can use Oracle Coherence cache as a reference to look up data to enrich a stream that is being processed.

To connect to an Oracle Coherence connection, you must provide the following parameters:

- Server Url(s)

JNDI Connection

Oracle Stream Analytics can use JNDI as a source of streaming data.

To connect to an JNDI connection, you must provide the following parameters:

- JNDI Provider
- Server Url(s)
- Username
- Password
- Jndi Other Properties

Druid Connection

Oracle Stream Analytics can use Druid as a source of streaming data.

A Druid connection has just a single parameter, the Zookeeper server URL.

2.2 Understanding Streams

A *stream* is a source of dynamic data. The data is flowing, it is not static or frozen. For example, stock prices of a particular company can be considered as a stream as the data arrives in every second or even more frequently. Another example of streaming data is the position (geographical location) of vehicles (e.g. trucks) which again can change continuously as each vehicle is moving. Each vehicle reports its own position to a central system periodically, e.g. every second, and the central system receives the position messages as a stream.

Streams can be transmitted using different network protocols, messaging systems as well as using many different message formats.

For more information on the different types of streams that you can create, refer to Creating a Stream in the *Using Oracle Stream Analytics* guide.

To create a Kafka stream, you must create a Kafka connection first, and then select that connection in the stream creation wizard. In addition to the connection, the user needs to specify the Kafka topic that represents the stream of data.

2.3 Understanding References

A *reference* is a source of static data that provides contextual information about the event data. There are several different types of references, such as a reference to a database table, or to coherence cache.

References are used to enrich data that arrives from a Stream. For example, an Order stream contains order events and each event contains a product Id and a customer Id. Assume that there are two database tables, each containing information about the products and the customers, respectively. After creating two references, one for the products table and one for the customer table, Oracle Stream Analytics can use these references to enrich the incoming streams with information from these tables, such as product name, manufacturer, customer name, address, etc.

If references take their data from a database table, a caching mechanism can be applied. (You can use the Coherence cache directly.) By turning on caching (a configuration option of the reference), it is possible to add a caching layer in between the pipeline and the database table. This improves the performance of accessing static data, at the price of higher memory consumption by the pipeline. Once the data is loaded into cache, the reference fetches data from the cache only. Any update on the reference table does not take effect if expiration policy is set to "Never".

Note:

To create a Database reference, you should first create a Database connection. Similarly, you have to create a Coherence connection to be able to create a Coherence reference.

2.4 Understanding Geo Fences

A *geo fence* is a virtual boundary in a real world geographical area. This virtual boundary can be used to find object position with respect to the geo fence.

For example, the object position can be:

- Near to geo fence
- Exit geo fence
- Based on Stay Duration in geo fence
- Enters geo fence
- Present inside geo fence

2.4.1 What is a Manual Geo Fence?

User-created geo fences are called as manual geo fences. You can create, edit, and update manual geofence using the built-in map editor. Only polygon geo fences are allowed.

2.4.2 What is a Database-based Geo Fence?

Geo fences for which you import geometry from database are known as database-based geo fences. Geo fence geometry can be seen in Geo Fence Editor. The standard create, delete and update operations using geo fence editor are not allowed in database-based geo fence. Polygon and circular geo fences are supported.

2.5 Understanding Pipelines

A *pipeline* defines the data processing logic and is a sequence of pipeline stages. A stage can be one of the following types – Query, Pattern, Rule, Query Group, Custom, Scoring, Target.

A pipeline starts with a stream stage, which is the only default stage. You cannot remove the default stage. A stage can have one or more children of any type such as Query, Pattern, Rule, and so on. That is, the pipeline does not have to be linear. You can have multiple branches in accordance with your use case logic. Optionally, a branch can end with one or more targets. You cannot add other stages to a target.

You can edit any stage in your draft pipeline at any time. The changes affecting downstream stages are propagated automatically.

Draft Pipelines

Pipelines in the draft state possess the following characteristics:

- Are visible only to the owner
- Can be edited
- Works only when the pipeline editor is open. When you exit the Pipeline editor or close your browser, the draft pipeline is removed from the Spark cluster.
- Do not send events to a downstream target even if a target is configured

A newly created pipeline is in draft state. This is where you can explore your streams and implement the business logic. You do not have to do the implementation all at once; the pipeline will not run between your editing sessions.

Published pipelines

Pipelines in the published state possess the following characteristics:

- Are visible to any user
- Cannot be edited
- Will continue to run in the Spark cluster even after you exit the pipeline
- Send events to a downstream target

After you are done with the implementation and satisfied, you can add a target and publish your pipeline. The published pipeline runs continually on the Spark Cluster.

If you want to edit a published pipeline, you must unpublish it first.

2.6 Understanding Dashboards

Dashboards are a collection of inter-related visualizations based on a common underlying theme. For example, a Sales dashboard shows observations related to various sales-related activities such as quarterly sales, prospective customers, and so on. Dashboards enables the users to have a single page view of all the important and correlated analysis that provides meaningful insights and assists in the decision making process.

A dashboard is first class citizen of the catalog. Any user with appropriate privileges can build the dashboards just by assembling the outcomes of the different pipeline stages without writing a single line of code. Dashboards display the live data.

Oracle Stream Analytics consists of more than one data pipeline stages where the outcome of each stage serves as input to the next stage. At the end of every stage, supporting inline visualizations enable you to visualize the result of the active stage. In Oracle Stream Analytics, you can visualize the outcome of the various application stages at a single place.

Combining dashboards with cube, users can create visualizations based on the data exploration activities performed in the analytics section of Oracle Stream Analytics. You can include these visualizations in the dashboards. With the dashboard feature, user can create mashup like showcases where both operational and analytics visualizations are intermixed to present the complete picture of the underlying business operation.

2.7 Understanding Cubes

A *cube* is a data structure that helps you quickly analyze data related to business problems on multiple dimensions. Oracle Stream Analytics cubes are powered by Druid, which is a distributed, in-memory OLAP data store.

Oracle Stream Analytics pipelines enable users to perform realtime data processing on the streaming data. Whereas, cube is the mechanism by which users can perform interactive analysis on the historical data. For this purpose, the pipeline outputs the processed data into the Kafka streams which in turn feeds the cube. Using cubes, users can carry out univariate, bivariate, and multivariate data analysis. Cube enables the users to carry out data exploration on the historical data with a rich set of 30 visualizations. These visualizations ranges from simple table, line, bar to the advanced visualizations such as sankey, boxplot, maps, and so on. Users can save the result of these cube explorations and make them available on dashboards that are embedded with Oracle Stream Analytics visualizations. This collaboration of dashboards and cubes serves both operational and strategic analytics needs of the business users. The visualizations available in the cubes also have a rich set of look and feel related properties for enhancing the value of the results of exploratory data analysis.

2.8 Understanding Stream Analytics Patterns

The visual representation of the event stream varies from one pattern type to another based on the key fields you choose.

A *pattern* provides you with the results displayed in a live output stream based on common business scenarios.

The following table lists the categories of patterns:

| Category | Pattern |
|---------------|---|
| Enrichment | Reverse Geo Code: Near By Left Outer Join |
| Outlier | Fluctuation |
| Inclusion | Union Left Outer Join |
| Missing Event | 'A' Not Followed by 'B' Detect Missing Event |
| Spatial | Proximity: Stream with Geo Fence Geo Fence Spatial: Speed Interaction: Single Stream Reverse Geo Code: Near By Geo Code Spatial: Point to Polygon Interaction: Two Stream Proximity: Two Stream Direction Reverse Geo Code: Near By Place Proximity: Single Stream Geo Filter |
| Filter | Eliminate Duplicates Fluctuation |
| State | 'A' Not Followed by 'B' Inverse W Detect Missing Event W 'A' Followed by 'B' |
| Finance | Inverse W W |

| Category | Pattern |
|----------------|---|
| Trend | 'A' Not Followed by 'B' Top N Change Detector Up Trend Detect Missing Event Down Trend 'A' Followed by 'B' Detect Duplicates Bottom N |
| Shape Detector | Inverse W W |
| Statistical | Correlation Quantile |

2.8.1 What is the Spatial: Speed Pattern?

This pattern lets you get the output average speed over the selected window range of a moving object.

2.8.2 What is the Geo Code Pattern?

When analyzing data, you may encounter situations where you need to obtain the latitude and longitude of a moving object based on street address, zip code, address, etc.

You can use this pattern to get geographic coordinates (like latitude and longitude) for an address.

2.8.3 What is the Interaction: Single Stream Pattern?

Two shapes are said to interact with each other if any part of the shape overlaps.

The Single Stream pattern lets you get interaction of an object with every other object in the same stream.

2.8.4 What is the Interaction: Two Stream Pattern?

If two shapes interact, the distance between them is *zero*.

You can use the Two Stream pattern to get interaction of an object in one stream with objects in another stream.

2.8.5 What is the Spatial: Point to Polygon Pattern?

Using this pattern you can get an object shape based on geographical coordinates, fixed length and breadth of an object.

For example, if you know the length and breadth of a ship, you can get the shape or geofence of the ship using its position coordinates, where the coordinates keep changing as the ship moves.

2.8.6 What is the Proximity: Single Stream Pattern?

You can use this pattern to get proximity of each object with every other object in a stream.

For example, if there is a single stream of flying airplanes and the distance buffer is specified as 1000 meters, the output in the table shows planes that are less than 1000 meters apart.

2.8.7 What is the Proximity: Two Stream Pattern?

You can use this pattern to get the proximity between objects of two streams.

The distance buffer acts as a filter in this pattern stage. For example, if there is a driver and passenger stream, you can get the proximity of each passenger with every other driver using a filter criteria of 'within a distance of 1 km'.

2.8.8 What is the Proximity: Stream with Geo Fence Pattern?

You can use this pattern to get proximity of an object with a virtual boundary or geo fence.

For example, if you have certain stores in the city of California, you can send promotional messages as soon as the customer comes into a proximity of 1000 meters from any of the stores.

2.8.9 What is the Direction Pattern?

You can use this pattern to get the direction of a moving object.

For example, you can evaluate the direction of a moving truck.

2.8.10 What is the Geo Fence Pattern?

You can use this pattern when you want to track the relation of an object with a virtual boundary called geo fence.

Relations can be *Enter*, *Exit*, *Stay*, or *Near* with respect to a geo fence. For example, you can trigger an alert when an object enters the geo fence. You can also analyze a stream containing geo-location data. It helps in determining how events are related to a polygon in a geo fence.

The geo-location can be:

- Near to Geo Fence
- Exiting Geo Fence
- Staying within Geo Fence for a specified duration
- Entering Geo Fence

2.8.11 What is the Geo Fence Filter: Inside Pattern?

You can use this pattern to track objects inside one or more geo fences.

For example, if users move from one geographical location to another, you can send promotional messages to the users when they are inside a specified geo fence.

2.8.12 What is the Reverse Geo Code: Near By Pattern?

You can use this to obtain nearest place for the specified latitude/longitude or geographical coordinates.

2.8.13 What is the Reverse Geo Code: Near By Place Pattern?

This pattern lets you obtain the near by location with granular information like city, country, street etc. for the specified latitude and longitude.

2.8.14 What is the Correlation Pattern?

You can use this pattern if you need to identify correlation between two numeric parameters. An output of 0 implies no correlation, +1 is positive correlation, and -1 implies negative correlation.

2.8.15 What is the Quantile Pattern?

You should use this pattern if you need to calculate the value of quantile function. For example, when asked for the 3rd quantile of student scores, it could return a value of 80 to imply 75% of students scored less than 80.

2.8.16 What is the Detect Duplicates Pattern?

The Detect Duplicates pattern detects duplicate events in your stream according to the criteria you specify and within a specified time window. Events may be partially or fully equivalent to be considered duplicates.

You can use this pattern to understand how many duplicate events your stream has. For example, when you suspect that your aggregates are offset, you may want to check your stream for duplicate events.

2.8.17 What is the Change Detector Pattern?

The Change Detector pattern looks for changes in the values of your event fields and reports the changes once they occur within a specified range window. For example, an event arrives with value `value1` for field `field1`. If any of the following incoming events within a specified range window contains a value different from `value1`, an alert is triggered. You can designate more than one field to look for changes.

You can use it when you need to be aware of changes in a normally stable value. For example, a sensor reading that is supposed to be the same for certain periods of time and changes in readings may indicate issues.

The default configuration of this pattern stage is to alert on change of any selected fields.

2.8.18 What is the W Pattern?

The W pattern, also known as a double bottom chart pattern, is used in the technical analysis of financial trading markets.

You can use this pattern to detect when an event data field value rises and falls in “W” fashion over a specified time window. For example, use this pattern when monitoring a market data feed stock price movement to determine a buy/sell/hold evaluation.

2.8.19 What is the ‘A’ Followed by ‘B’ Pattern?

The ‘A’ Followed by ‘B’ pattern looks for particular events following one another and will output an event when the specified sequence of events occurs.

You can use it when you need to be aware of a certain succession of events happening in your flow. For example, if an order status `BOOKED` is followed by an order status `SHIPPED` (skipping status `PAID`), you need to raise an alert.

2.8.20 What is the Top N Pattern?

The Top N pattern will output N events with highest values from a collection of events arriving within a specified time window sorted not in the default order of arrival but the way you specify.

You can use it to get the highest values of fields in your stream within a specified time window. For example, use it to get N highest values of pressure sensor readings.

2.8.21 What is the Bottom N Pattern?

The Bottom N pattern will output N events with lowest values from a collection of events arriving within a specified time window sorted not in the default order of arrival but the way you specify.

You can use it to get the lowest values of fields in your stream within a specified time window. For example, use it to get N lowest values of pressure sensor readings.

2.8.22 What is the Up Trend Pattern?

The Up Trend pattern detects a situation when a numeric value goes invariably up over a period of time.

You can use the pattern if you need to detect situations of a constant increase in one of your numeric values. For example, detect a constant increase in pressure from one of your sensors.

2.8.23 What is the ‘A’ Not Followed by ‘B’ Pattern?

The ‘A’ Not Followed by ‘B’ pattern will look for a missing second event in a particular combination of events and will output the first event when the expected second event does not arrive within the specified time period.

You can use it when you need to be aware of a specific event not following its predecessor in your flow. For example, if an order status `BOOKED` is not followed by an order status `PAID` within a certain time period, you may need to raise an alert.

2.8.24 What is the Down Trend Pattern?

The Down Trend pattern detects a situation when a numeric value goes invariably down over a period of time.

You can use this pattern if you need to detect situations of a constant reduction in one of your numeric values. For example, detect a constant drop in pressure from one of your sensors.

2.8.25 What is the Union Pattern?

The Union pattern merges two streams with identical shapes into one.

You can use this pattern if you have two streams with identical shapes that you want to merge into one, for example when you have two similar sensors sending data into two different streams, and you want to process the streams simultaneously, in one pipeline.

2.8.26 What is the Fluctuation Pattern?

You can use this pattern to detect when an event data field value changes in a specific upward or downward fashion within a specific time window. For example, use this pattern to identify the variable changes in an Oil Pressure value are maintained within acceptable ranges.

2.8.27 What is the Inverse W Pattern?

The Inverse W pattern, also known as a double top chart pattern, is used in the technical analysis of financial trading markets.

You can use this pattern when you want to see the financial data in a graphical form.

2.8.28 What is the Eliminate Duplicates Pattern?

The Eliminate Duplicates pattern looks for duplicate events in your stream within a specified time window and removes all but the first occurrence. A duplicate event is an event that has one or more field values identical to values of the same field(s) in another event. It is up to you to specify what fields are analyzed for duplicate values. You can configure the pattern to compare just one field or the whole event.

You can use it to get rid of noise in your stream. If you know that your stream contains duplicates that might offset your aggregates, such as counts, use the Eliminate Duplicates pattern to cleanse your data.

2.8.29 What is the Detect Missing Event Pattern?

The Detect Missing Event pattern discovers simple situations when an expected event is missing.

You can use this pattern if you need to detect missing events in your feed. For example, you have a feed when multiple sensors send their readings every 5 seconds. Use this pattern to detect sensors that have stopped sending their readings, which may indicate that the sensor is broken or there is no connection to the sensor.

2.8.30 What is the Left Outer Join Pattern?

The Left Outer join pattern joins your flow with another stream or a reference using the left outer join semantics.

You can use this pattern to join a stream or a reference using the left outer join semantics. The result of this pattern always contains the data of the left table even if the join-condition does not find any matching data in the right table.

2.9 Understanding Shapes

A *shape* is the format of the data. In Oracle Stream Analytics, each message (or event, in stream processing terminology) in a Stream or Target must have the same format and this format must be specified when creating the Stream or Target. You can think of the shape as the streaming analogy of the database table structure for static data. Each shape consists of a number of fields and each field has a name and a data type. In the Stream creation wizard, it is possible to assign an alias to a field, so that the field can later be referenced by this user-given alias.

Assume that the stream contains data about orders. In this case, the shape may contain the following fields: an order id of type string, a customer id of type integer, product id of type integer, a quantity of type integer and a unit price of type Number.

2.10 Understanding Target

A *target* represents an external system where the results of the stream processing is being directed to. Just like streams, targets are the links to the outside world. Streams are the input to a pipeline, whereas targets are the output. While a pipeline can consume and process multiple streams.

It can have no target, but that configuration does not really make sense, as the purpose of creating a pipeline is to process streaming data and direct the output to an external system, i.e a target.

JMS target type

To create JMS type target, you must provide the following parameters:

- Connection
- Jndi name
- Data Format

Kafka target type

To create Kafka type target, you must provide the following parameters:

- Connection
- Topic name

- Data Format

REST target type

To create REST type target, you must provide the following parameters:

- URL
- Custom HTTP headers
- Batch processing
- Data Format

2.11 Understanding the Predictive Model

The *predictive model* is an algorithm that you apply to streaming data to predict outcomes. In Oracle Stream Analytics, a predictive model is a PMML file, that you upload and store in the system.

Oracle Stream Analytics supports PMML versions 3.0, 3.1, 3.2, 4.0, and 4.1. In a pipeline, you use a predictive model in a scoring stage to do probability scoring.

2.12 Understanding Custom JARs

A *custom JAR* is a Oracle Stream Analytics catalog artifact. It's a JAR file containing a custom event-processing algorithm implemented in Java, or a POJO file in Java..

You need to create your own custom event processing logic if you cannot build this logic using the Oracle Stream Analytics web tooling. This may be required in complex use cases in particular industries. Generally, you can address complex use cases with patterns. A custom JAR contains one or more custom stage types and/or one or more custom functions. You can use a custom stage type in the Custom Stage, and you can use custom functions in calculated field expressions. For details on how to build custom JARs, see [Oracle Stream Analytics Developer's Guide](#).

2.13 Understanding Export and Import

The *export and import* feature lets you migrate your pipeline and its contents between Oracle Stream Analytics systems (such as development and production) in a matter of few clicks. You also have the option to migrate only select artifacts.

To export and import a pipeline, you need admin privileges. The export will result in a ZIP file containing the metadata for the catalog object being exported along with all its dependencies. You can import only ZIP files that you previously exported using this export / import functionality. Any other type of files won't be imported successfully.

 **Note:**

- You cannot import a pipeline developed with earlier versions of Oracle Stream Analytics.
- On reimport, the existing metadata is overwritten with the newly imported metadata.
- Artifacts of a published pipeline can't be overwritten. You must first unpublish the pipeline and then retry.

3

Overview of the Components of a Pipeline

A Oracle Stream Analytics pipeline is comprised of many components that define the pipeline.

These components are described in the following topics:

- [Understanding Query Stage](#)
- [Understanding Rule](#)
- [Understanding Rule Stage](#)
- [Understanding Pattern Stage](#)
- [Understanding Custom Stage](#)
- [Understanding Scoring Stage](#)
- [Understanding Query Group](#)
- [Understanding Live Output Table](#)
- [Understanding Visualizations](#)
- [Understanding Topology Viewer](#)
- [Understanding Expression Builder Functions](#)

3.1 Understanding Query Stage

A *query stage* is used to configure a SQL-like query on the data stream and comprises additional sources for joins, filters, summaries, group by, time windows, and so on.

For example, the query below calculates hourly total sales where transaction amount is greater than a dollar and outputs the result every 1 second.

```
Select sum (TransactionAmount) As HourlySales  
From SalesStream [Range 1 Hour Slide 1 Second]  
Where TransactionAmount > 1
```

Queries like above or more complex queries can all be configured in the query stage with zero coding and with no intimate knowledge of Continuous Query Language or CQL. The CQL language is similar to SQL but with additional constructs for temporal analytics and pattern matching.

A query stage has the following sub sections:

- Filters
- Correlation
- Summary/Group By
- Window Type - Range and Evaluation Frequency
- Visualizations

3.1.1 What is Filter

The *filter* section in a query stage or query group stage allows events in the data stream to be filtered out.

Only events which satisfy the filter condition are passed to the downstream stage. For example, in a data stream containing `SensorId` and `Temperature`, you can filter events where `Temperature` is lower than or equal to 70 degrees by setting the filter condition to `Temperature > 70`.

3.1.2 What is Correlation

A *correlation* is used to enrich the incoming event in the data stream with static data in a database table or with data from other streams.

For example, if the event in the data stream only includes `SensorId` and `SensorTemperature`, the event could be enriched with data from a table to obtain `SensorMake`, `SensorLocation`, `SensorThreshold`, and many more.

Correlating an event with other sources requires the join condition to be based on a common key. In the above example, the `SensorId` from the stream can be used to correlate with `SensorKey` in the database table. The following query illustrates the above data enrichment scenario producing sensor details for all sensors whose temperature exceeds their pre-defined threshold.

```
Select T.SensorId, T.Temperature, D.SensorName, D.SensorLocation
From TemperatureStream[Now] T, SensorDetailsTable D
Where T.SensorId = D.SensorKey And T.Temperature > D.SensorThreshold
```

Queries like above and more complex queries can be automatically generated by configuring sources and filter sections of the query stage.

3.1.3 What is Summary

A data stream is a continuous sequence of events but we can summarize the data over any time range including an unbounded range.

For example, you can continuously compute the maximum temperature for each sensor from the beginning of time by configuring a query like the one below in a Query stage.

```
Select SesnsorId, max(Temperature)
From TemperatureStream
Group By SensorId
```

3.1.4 What is Group By

A *group by* collects the data of all the rows with an identical column value. Group by is used in conjunction with Summaries (aggregate functions) to provide information about each group.

Here is an example configuration that generates a query for computing the average temperature of each sensor at the end of the hour and using readings from last one hour.

```
Select SesnsorId, avg(Temperature)
From TemperatureStream [Range 1 Hour Slide 1 Hour]
Group By SensorId
```

Example

If you add multiple group bys, the data is grouped on multiple columns. For example, you have a stream that gives you sales numbers for geographical locations. You have the following columns BEFORE group by:

| COUNTRY | CITY | REVENUE |
|---------|--------|---------|
| US | SF | 500 |
| US | NY | 1000 |
| INDIA | BOMBAY | 800 |
| INDIA | BOMBAY | 1500 |
| INDIA | BOMBAY | 700 |
| | | |

Calculate sum of revenue (summary) by country (groupby) to get:

| COUNTRY | SUM_OF_REVENUE |
|---------|----------------|
| US | 1500 |
| INDIA | 3000 |

Add CITY as another group by, to get your aggregations grouped by city in addition to country:

| COUNTRY | CITY | SUM_OF_REVENUE |
|---------|-----------|----------------|
| US | NY | 1000 |
| US | SF | 500 |
| INDIA | BOMBAY | 1500 |
| INDIA | BANGALORE | 1500 |

3.1.5 What is Retain All Columns

The **Retain All Columns** is feature to allow the fields that are not in summary and group by be present in the output. The values of the retained columns(or fields) are the values of the last event in the stream.

3.1.6 What is Range

A *range* is a window applied on the data stream. Since data stream is an unbounded sequence of events it is often necessary to apply a window when computing aggregates.

Examples of ranges include – Last 1 Hour of events, Last 5 Minutes of events, Last 10 Events, and many more. Applying a range retains data in memory so be cautious with use of window ranges. For example, if data is arriving at the rate of 2000 events

per second and if each event is 1KB then we have 2MB of data per second. Applying a 1-hour window on this data stream consumes 2MB times 3600 or 7.2GB of memory.

The supported time units in a range are:

- now
- nanoseconds
- microseconds
- milliseconds
- seconds
- minutes
- hours
- events

3.1.7 What is Evaluation Frequency

Evaluation Frequency or a Window Slide (commonly referred to) determines the frequency at which results are desired.

For example, the configured query below outputs total sales every 1 second but using transactions from last 1 hour.

```
Select sum (TransactionAmount) As HourlySales  
From SalesStream [Range 1 Hour Slide 1 Second]
```

In other words, Evaluation Frequency determines how often you want to see the results. In the above query, if result is only desired at the end of the hour then we set the Evaluation Frequency to 1 hour.

3.1.8 What is Visualization

Visualizations are graphical representation of the streaming data in a pipeline. You can add visualizations on all stages in the pipeline except a target stage.

Select an appropriate visualization from the following available options, that suits your requirement.

- Bar
- Line
- Geo Spatial
- Area
- Pie
- Scatter
- Bubble
- Thematic Map
- Stacked Bar
- Table

3.2 Understanding Rules

A *rule* is a set of conditions applied to the incoming stream and a set of actions performed on the stream when conditions are true. Each event is analyzed independently of other events.

For example, assume that your stream is a stream from pressure sensors and has the following fields:

- `sensor_id`
- `pressure`
- `status`

If you want to assign a status value based on the pressure, you can define the following rules:

- if the pressure is less than or equal to 50, the status must be set to GREEN
- if the pressure is between 50 and 100, the status must be set to YELLOW
- if the pressure is greater than 100, the status must be set to RED.

3.3 Understanding Rule Stage

A *rule stage* is a stage in the pipeline where you apply conditional (`IF - THEN`) logic to the events in the stream. You can check for specific conditions and assign values to fields based on the results of your checks.

You can add multiple rules to the stage and they will get applied to pipeline in the sequence they are added. A *rule* is a set of conditions applied to the incoming stream and a set of actions performed on the stream when conditions are true. Each event is analyzed independently of other events.

3.4 Understanding Pattern Stage

Patterns are a stage within a pipeline. When working from a pattern, you need to specify a few key fields to discover an interesting result. You can create pattern stages within the pipeline. Patterns are not stand-alone artifacts. They need to be embedded within a pipeline.

3.5 Understanding Custom Stage

Custom Stage is a type of stage where you can apply custom Java/Scala code to streaming data in the pipeline. It behaves like any other type of stage with data flowing into and out of it. It is close to a pattern stage in the way that you are asked to configure a few parameters before its logic applies to the stream.

3.6 Understanding Scoring Stage

Scoring Stage is a type of stage where you apply a machine learning model to your streaming data to do predictive analysis. Scoring Stage is the infrastructure that Oracle

Stream Analytics provides to data scientists for machine learning model deployment against streaming data.

3.7 Understanding Query Group

A query group stage lets you do aggregations on multiple group bys and multiple windows. It is a collection of groups, where each of the group has its own window, filters that affect the data processing only within that group.

A query group has two types of stages:

- Stream
- Table

3.7.1 What is Query Group Stage: Stream?

A query group stage of the type stream is where you can apply aggregate functions with different group-bys and window ranges to your streaming data. You can have multiple query groups in one stage.

3.7.2 What is Query Group Stage: Table?

A query group stage of the type table is where you can apply aggregate functions with different group bys and window ranges to a database table data recreated in memory. Use this stage on a change data capture stream, such as GoldenGate. You can have multiple query groups in one stage.

3.8 Understanding the Live Output Table

The Live Output table is the main feedback mechanism from the pipelines that you build. The Live Output table will display events that go out of your pipeline, after your processing logic has been applied on the incoming stream or streams.

The Live Output table will be displayed for each stage of your pipeline and will include output of that particular stage. On the source stage the Live Output table will display events as they arrive in the stream. On the target stage, the Live Output stage will display events as they will flow to the target.

The Live Output table is also a powerful tool for event shape manipulation. With the Live Output table you can:

- Add new fields to the event using an extensive library of functions in the expression builder, and remove new fields
- Change the order of the event fields
- Rename event fields
- Remove existing fields from the output of the stage
- Add a timestamp field to each event
- Hide fields from view (but retain them in the output)
- Pause and restart event display in the browser (not affecting downstream stages or targets)

The interaction with the table should be intuitively clear to anyone who has worked with popular spreadsheet pipelines.

Expression Builder

The expression builder provides functionality to add new fields to your output based on existing fields. You can use a rich library of functions to manipulate your event data. A simple example is string concatenation; you can construct a full name from first and last names:



Note:

The event shape manipulation functionality is available on the table in the query stage.

The expression builder has syntax highlighting and code completion. You can also see the function signature, input parameters and the return value in the Expression Editor.

```
=concat()  
CONCAT  
concat(value1, ...)  
Concat(value1,...) - Concatenation of values converted to strings  
For example  
concat(value1, ...)  
value1  
A part of string to concatenate with others. Value can be one of the following types:  
string, bigdecimal, byte, double, float, bigint, int, timestamp. Required.  
vararg1  
A part of string to concatenate with others. Value can be one of the following types:  
string, bigdecimal, byte, double, float, bigint, int, timestamp. Optional.  
Return type  
string
```

3.9 Understanding Visualizations

Visualization is mapping of the data (information) to a graphical or tabular format which can be used to answer a specific analytical question.

It translates data, its properties and relationships into an easy to interpretable visual object consisting of points, lines, shapes and colors. It effectively represents the

results of the meaningful multi-dimensional questions. It also enables to discover the influential patterns out of the represented data (information) using the visual analysis.

Visualizations

Visualizations are divided into two categories:

- Axis based

Axis based visualizations display series and groups of data. Series and groups are analogous to the rows and columns of a grid of data. Typically, the rows in the grid appear as a series in visualization, and the columns in the grid appear as groups.

Axis based visualizations enables users to visualize the data along two graph axis x and y like sum of sales over regions or sum of sales over time period. X axis values can be categorical in nature like regions or can be based on time series values whereas Y axis represents the measured value like sum(sales). These charts are useful for visualizing trends in a set of values over time and comparing these values across series.

- Spatial

Spatial visualizations are used when geography is especially important in analyzing an event. It represents business data superimposed on a single geo fence.

3.9.1 What is a Bar Type of Visualization?

Bar visualization is one of the widely used visualization types which represents data as a series of vertical bars. It is best suited for comparison of the values represented along y axis where different categories are spread across x axis. In a Bar visualization vertical columns represent metrics (measured values). The horizontal axis displays multiple or non-consecutive categories.

In Horizontal Bar, the axis positions are switched. The vertical axis displays multiple or non-consecutive categories. The horizontal columns represents metrics (measured values). It is preferable when the category names are long text values and requires more space in order to be displayed.

3.9.2 What is a Line Type of Visualization?

Line visualization represents data as a line, as a series of data points, or as data points that are connected by a line. Line visualization require data for at least two points for each member in a group. The X-axis is a single consecutive dimension, such as a date-time field, and the data lines are likely to cross. X axis can also have non date-time categories. Y axis represents the metrics (measured value). It is preferred to use line visualization when data set is continuous in nature. It is best suited for trend-based plotting of data over a period of time. In Line visualization, emphasis is on the continuation or the flow of the values (a trend) but individual value comparison can also be carried out. Multiple series can also be compared with the line visualizations.

It can have a horizontal orientation where axis are switched i.e. y axis holds categories whereas x axis shows metrics.

3.9.3 What is An Area Type of Visualization?

Area visualization represents data as a filled-in area. Area visualization requires at least two groups of data along an axis. The X-axis is a single consecutive dimension, such as a date-time field, and the data lines are unlikely to cross. Y axis represents the metrics (measured value). X axis can also have non date-time categories. This visualization is mainly suitable for presenting accumulative value changes over time.

It can have a horizontal orientation where axis are switched i.e. y axis holds categories whereas x axis shows metrics.

3.9.4 What is a Stacked Bar Type of Visualization?

A Stacked visualization displays sets of values stacked in a single segmented column instead of side-by-side in separate columns. It is used to show a composition. Bars for each set of data are appended to previous sets of data. The size of the stack represents a cumulative data total.

3.9.5 What is a Spatial Type of Visualization?

Geo Spatial visualization allows displaying location of an object on a geo fence and takes user to the area where events are occurring. User can configure visualization to specify latitude, longitude, identifier etc. Customization of visualization by specifying different pins like arrows with different colors based on certain condition is also allowed.

3.9.6 What is a Pie Chart?

A pie chart is a circular graphic divided into slices that indicate numerical proportions. The arc length of each slice is proportionate to the quantity it represents.

You can use Pie charts to compare parts of a whole.

3.9.7 What is a Bubble Chart?

A bubble chart displays three dimensions of data. Each entity with its several versions (mostly three) of associated data is plotted as a disk. This disk shows two of the values through the disk's xy location and the third through its size.

3.9.8 What is a Thematic Map?

A thematic map focuses on a specific theme or subject area. It includes some locational or reference information and emphasizes spatial variation of one or a small number of geographic distributions. These distributions may be physical phenomena such as climate, population density, traffic congestion, and so on.

3.9.9 What is a Scatter Chart?

A scatter chart is a mathematical diagram that uses Cartesian coordinates to display values for multiple variables for a set of data. You can color code the points to display data for an additional variable. This chart shows how much one variable is affected by another. The relationship between two variables is called their correlation.

3.10 Understanding the Topology Viewer

Topology is a graphical representation and illustration of the connected entities and the dependencies between the artifacts.

3.10.1 What is Immediate Family?

Immediate Family context displays the dependencies between the selected entity and its child or parent.

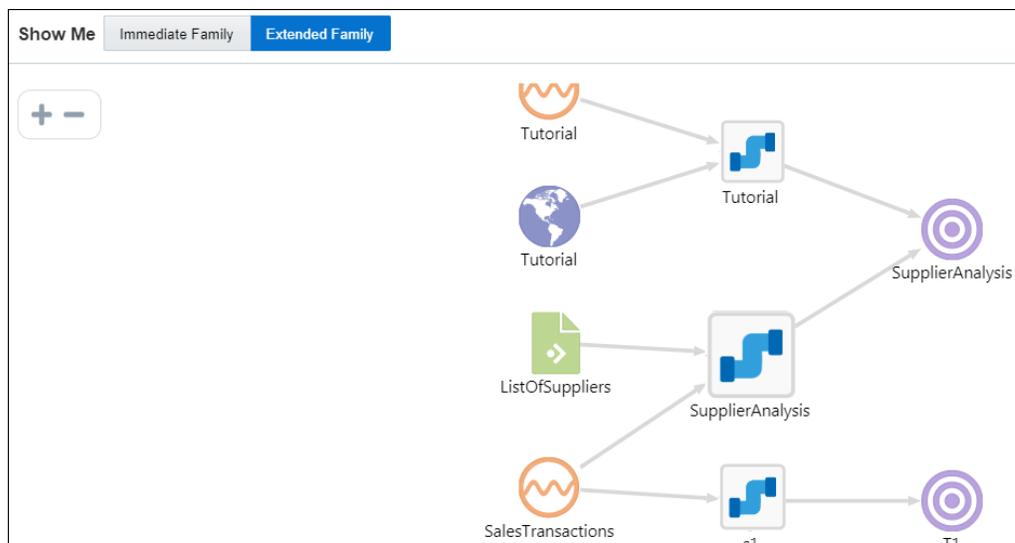
The following figure illustrates how a topology looks in the **Immediate Family**.



3.10.2 What is Extended Family?

Extended Family context displays the dependencies between the entities in a full context, that is if an entity has a child entity and a parent entity, and the parent entity has other dependencies, all the dependencies are shown in the Full context.

The following figure illustrates how a topology looks in the **Extended Family**.



3.11 Understanding Expression Builder Functions



This topic applies only to Oracle user-managed services.

Expression Builder is an editor that allows you to build expressions using various existing functions. The expressions help you in achieving the required results for your pipelines.

Topics:

- [What are Bessel Functions?](#)
- [What are Conversion Functions?](#)
- [What are Date Functions?](#)
- [What are Geometry Functions?](#)
- [What are Interval Functions?](#)
- [What are Math Functions?](#)
- [What are Null-related Functions?](#)
- [What are Statistical Functions?](#)
- [What are String Functions?](#)

3.11.1 What are Bessel Functions?

The mathematical cylinder functions for integers are known as Bessel functions.

The following Bessel functions are supported in this release:

| Function Name | Description |
|--------------------------|--|
| <code>BesselI0(x)</code> | Returns the modified Bessel function of order 0 of the double argument as a double |

| Function Name | Description |
|------------------------------|---|
| <code>BesselI0_exp(x)</code> | Returns the exponentially scaled modified Bessel function of order 0 of the double argument as a double |
| <code>BesselI1(x)</code> | Returns the modified Bessel function of order 1 of the double argument as a double |
| <code>BesselI1_exp(x)</code> | Returns the exponentially scaled modified Bessel function of order 1 of the double argument as a double |
| <code>BesselJ(x,x)</code> | Returns the Bessel function of the first kind of order n of the argument as a double |
| <code>BesselK(x,x)</code> | Returns the modified Bessel function of the third kind of order n of the argument as a double |
| <code>BesselK0_exp(x)</code> | Returns the exponentially scaled modified Bessel function of the third kind of order 0 of the double argument as a double |
| <code>BesselK1_exp(x)</code> | Returns the exponentially scaled modified Bessel function of the third kind of order 1 of the double argument as a double |
| <code>BesselY(x)</code> | Returns the Bessel function of the second kind of order n of the double argument as a double |

3.11.2 What are Conversion Functions?

The conversion functions help in converting values from one data type to other.

The following conversion functions are supported in this release:

| Function Name | Description |
|------------------------------------|--|
| <code>bigdecimal(value1)</code> | Converts the given value to bigdecimal |
| <code>boolean(value1)</code> | Converts the given value to logical |
| <code>date(value1,value2)</code> | Converts the given value to datetime |
| <code>double(value1)</code> | Converts the given value to double |
| <code>float(value1)</code> | Converts the given value to float |
| <code>int(value1)</code> | Converts the given value to integer |
| <code>long(value1)</code> | Converts the given value to long |
| <code>string(value1,value2)</code> | Converts the given value to string |

3.11.3 What are Date Functions?

The following date functions are supported in this release:

| Function Name | Description |
|-------------------------------|-------------------------------------|
| <code>day(date)</code> | Returns day of the date |
| <code>eventtimestamp()</code> | Returns event timestamp from stream |
| <code>hour(date)</code> | Returns hour of the date |
| <code>minute(date)</code> | Returns minute of the date |

| Function Name | Description |
|--|--|
| <code>month(date)</code> | Returns month of the date |
| <code>nanosecond(date)</code> | Returns nanosecond of the date |
| <code>second(date)</code> | Returns second of the date |
| <code>systimestamp()</code> | Returns the system's timestamp on which the application is running |
| <code>timeformat(value1,value2)</code> | Returns the provided timestamp in required time format |
| <code>timestamp()</code> | Returns the current output time |
| <code>year(date)</code> | Returns year of the date |

3.11.4 What are Geometry Functions?

The Geometry functions allow you to convert the given values into a geometrical shape.

The following interval functions are supported in this release:

| Function Name | Description |
|---|---|
| <code>CreatePoint(lat, long, SRID)</code> | Returns a 2-dimensional point type geometry from the given latitude and longitude. The default SRID is 8307. The return value is of the datatype <code>sdo geometry</code> . |
| <code>distance(lat1, long1, lat2, long2, SRID)</code> | Returns distance between the first set of latitude, longitude and the second set of latitude, longitude values. The default SRID is 8307. |



Note:

Only SRID 8307 is supported in the current release.

The return value is of the datatype `double`.

3.11.5 What are Interval Functions?

The Interval functions help you in calculating time interval from given values.

The following interval functions are supported in this release:

| Function Name | Description |
|---|---|
| <code>dsintervaltonum (c1 INTERVAL DAY TO SECOND, c2 char)</code> | Converts the given value to a numeric value. User must provide unit for the output numeric value as second argument to this function. Allowed values for the unit is: DAY,HOUR,MINUTE,SECOND |
| <code>numtodsinterval(n, interval_unit)</code> | Converts the given value to an INTERVAL DAY TO SECOND literal. The value of the interval_unit specifies the unit of n and must resolve to one of the string values: DAY, HOUR, MINUTE, or SECOND. The return value is of the datatype interval. |
| <code>numtoyminterval(n, interval_unit)</code> | Converts the given value to an INTERVAL YEAR TO MONTH literal. The value of the interval_unit specifies the unit of n and must resolve to one of the following string values: YEAR, MONTH. |
| <code>to_dsinterval(string)</code> | Converts a string in format DD HH:MM:SS into a INTERVAL DAY TO SECOND data type. The DD indicates the number of days between 0 to 99. The HH:MM:SS indicates the number of hours, minutes and seconds in the interval from 0:0:0 to 23:59:59.999999. The seconds part can accept upto six decimal places. The return value is of the datatype interval. |
| <code>to_yminterval(string)</code> | Converts a string in format YY-MM into a INTERVAL YEAR TO MONTH data type. The YY part indicates the number of years between 0 to 99. The MM part indicates the number of months between 0-11. The return value is of the datatype interval. |
| <code>ymintervaltonum(c1 INTERVAL YEAR TO MONTH, c2 char)</code> | Converts the given value to a numeric value. User must provide unit for the output numeric value as second argument to this function. Allowed values for the unit is: YEAR,MONTH |

3.11.6 The YM Interval Functions

The YM Interval functions help you in calculating time interval from year to month.

The following are the YM interval functions:

| Function Name | Description |
|--|---|
| <code>to_ymininterval (string)</code> | <p>TO_YMINTERVAL converts a character string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 datatype to an INTERVAL YEAR TO MONTH type. TO_YMINTERVAL accepts argument in one of the two formats:</p> <ul style="list-style-type: none"> • SQL interval format compatible with the SQL standard (ISO/IEC 9075:2003) • ISO duration format compatible with the ISO 8601:2004 standard <p>In the SQL format, years is an integer between 0 and 999999999, and months is an integer between 0 and 11. Additional blanks are allowed between format elements.</p> <p>In the ISO format, years and months are integers between 0 and 999999999. Days, hours, minutes, seconds, and frac_secs are non-negative integers, and are ignored, if specified. No blanks are allowed in the value.</p> |
| <code>numtoyminterval(n, interval_unit)</code> | <p>NUMTOYMINTERVAL converts number <i>n</i> to an INTERVAL YEAR TO MONTH literal. The argument <i>n</i> can be any NUMBER value or an expression that can be implicitly converted to a NUMBER value. The argument <i>interval_unit</i> can be of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 datatype. The value for <i>interval_unit</i> specifies the unit of <i>n</i> and must resolve to one of the following string values: YEAR and MONTH</p> <p>The return value is of the datatype interval.</p> |

3.11.7 What are Math Functions?

The math functions allow you to perform various mathematical operations and calculations ranging from simple to complex.

The following math functions are supported in this release:

| Function Name | Description |
|--|--|
| <code>IEEEremainder(value1, value2)</code> | Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard |
| <code>abs(value1)</code> | Returns the absolute value of a number |
| <code>acos(value1)</code> | Returns arc cosine of a value |
| <code>asin(value1)</code> | Returns arc sine of a value |
| <code>atan(value1)</code> | Returns arc tangent of a value |
| <code>atan2(arg1, arg2)</code> | Returns polar angle of a point (arg2, arg1) |

| Function Name | Description |
|---|--|
| <code>binomial(base, power)</code> | Returns binomial coefficient of the base raised to the specified power |
| <code>bitMaskWithBitsSetFromTo(x)</code> | BitMask with BitsSet (From, To) |
| <code>cbrt(value1)</code> | Returns cubic root of the specified value |
| <code>ceil(value1)</code> | Rounds to ceiling |
| <code>copySign(value1, value2)</code> | Returns the first floating-point argument with the sign of the second floating-point argument |
| <code>cos(value1)</code> | Returns cosine of a value |
| <code>cosh(value1)</code> | Returns cosine hyperbolic of a value |
| <code>exp(x)</code> | Returns exponent of a value |
| <code>expm1(x)</code> | More precise equivalent of <code>exp(x)</code> ; Returns 1 when x is around zero |
| <code>factorial(value1)</code> | Returns factorial of a natural number |
| <code>floor(value1)</code> | Rounds to floor |
| <code>getExponent(value1)</code> | Returns the unbiased exponent used in the representation of a double |
| <code>getSeedAtRowColumn(value1, value2)</code> | Returns a deterministic seed as an integer from a (seemingly gigantic) matrix of predefined seeds |
| <code>hash(value1)</code> | Returns an integer hashcode for the specified double value |
| <code>hypot(value1, value2)</code> | Returns square root of sum of squares of the two arguments |
| <code>leastSignificantBit(value 1)</code> | Returns the least significant 64 bits of this UUID's 128 bit value |
| <code>log(value1, value2)</code> | Calculates the log value of the given argument to the given base, where <code>value 1</code> is the value and <code>value 2</code> is the base |
| <code>log1(value1)</code> | Returns the natural logarithm of a number |
| <code>log10(value1)</code> | Calculates the log value of the given argument to base 10 |
| <code>log2(value1)</code> | Calculates the log value of the given argument to base 2 |
| <code>logFactorial(value1)</code> | Returns the natural logarithm (base e) of the factorial of its integer argument as a double |
| <code>longFactorial(value1)</code> | Returns the factorial of its integer argument (in the range <code>k >= 0 && k < 21</code>) as a long |
| <code>maximum(value1, value2)</code> | Returns the maximum of 2 arguments |
| <code>minimum(value1, value2)</code> | Returns the minimum of 2 arguments |
| <code>mod(value1, value2)</code> | Returns modulo of a number |
| <code>mostttSignificantBit(value 1)</code> | Returns the most significant 64 bits of this UUID's 128 bit value |
| <code>nextAfter(value1, value2)</code> | Returns the floating-point number adjacent to the first argument in the direction of the second argument |
| <code>nextDown(value1)</code> | Returns the floating-point value adjacent to the input argument in the direction of negative infinity |

| Function Name | Description |
|---|---|
| <code>nextUp(value1)</code> | Returns the floating-point value adjacent to the input argument in the direction of positive infinity |
| <code>Pow(m,n)</code> | Returns m raised to the nth power |
| <code>rint(value1)</code> | Returns the double value that is closest in value to the argument and is equal to a mathematical integer |
| <code>round(value1)</code> | Rounds to the nearest integral value |
| <code>Scalb(d,scaleFactor)</code> | Returns $d \times 2^{\text{scaleFactor}}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the double value set |
| <code>signum(value1)</code> | Returns signum of an argument as a double value |
| <code>sin(value1)</code> | Returns sine of a value |
| <code>sinh(value1)</code> | Returns sine hyperbolic of a value |
| <code>sqrt(value1)</code> | Returns square root of a value |
| <code>stirlingCorrection(value1)</code> | Returns the correction term of the Stirling approximation of the natural logarithm (base e) of the factorial of the integer argument as a double |
| <code>tan(value1)</code> | Returns tangent of a value |
| <code>tanh(value1)</code> | Returns tangent hyperbolic of a value |
| <code>toDegrees(value1)</code> | Converts the argument value to degrees |
| <code>toRadians(value1)</code> | Returns the measurement of the angle in radians |
| <code>ulp(value1)</code> | Returns the size of an ulp of the argument |

3.11.8 What are Null-related Functions?

The following null-related functions are supported in this release:

| Function Name | Description |
|---------------------------------|---|
| <code>nvl(value1,value2)</code> | Replaces null with a value of the same type |

3.11.9 What are Statistical Functions?

Statistical functions help you in calculating the statistics of different values.

The following statistical functions are supported in this release:

| Function Name | Description |
|---|---|
| <code>beta1(value1,value2,value3)</code> | Returns the area from zero to value3 under the beta density function |
| <code>betaComplemented(value1,value2,value3)</code> | Returns the area under the right hand tail (from value3 to infinity) of the beta density function |

| Function Name | Description |
|---|--|
| <code>binomial2(value1,value2,value3)</code> | Returns the sum of the terms 0 through <code>value1</code> of the Binomial probability density. All arguments must be positive. |
| <code>binomialComplemented(value1,value2,value3)</code> | Returns the sum of the terms <code>value1+1</code> through <code>value2</code> of the binomial probability density. All arguments must be positive. |
| <code>chiSquare(value1,value2)</code> | Returns the area under the left hand tail (from 0 to <code>value2</code>) of the chi square probability density function with <code>value1</code> degrees of freedom. The arguments must both be positive. |
| <code>chiSquareComplemented(value1,value2)</code> | Returns the area under the right hand tail (from <code>value2</code> to infinity) of the chi square probability density function with <code>value1</code> degrees of freedom. The arguments must both be positive. |
| <code>errorFunction(value1)</code> | Returns the error function of the normal distribution |
| <code>errorFunctionComplemented(value1)</code> | Returns the complementary error function of the normal distribution |
| <code>gamma(value1,value2,value3)</code> | Returns the gamma function of the arguments |
| <code>gammaComplemented(value1,value2,value3)</code> | Returns the integral from <code>value3</code> to infinity of the gamma probability density function |
| <code>incompleteBeta(value1,value2,value3)</code> | Returns the incomplete beta function evaluated from zero to <code>value3</code> |
| <code>incompleteGamma(value1,value2)</code> | Returns the incomplete gamma function |
| <code>incompleteGammaComplemented(value1,value2)</code> | Returns the complemented incomplete gamma function |
| <code>logGamma(value1)</code> | Returns the natural logarithm of the gamma function |
| <code>negativeBinomial(value1,value2,value3)</code> | Returns the sum of the terms 0 through <code>value1</code> of the negative binomial distribution. All arguments must be positive. |
| <code>negativeBinomialComplemented(value1,value2,value3)</code> | Returns the sum of the terms <code>value1+1</code> to infinity of the negative binomial distribution. All arguments must be positive. |
| <code>normal(value1,value2,value3)</code> | Returns the area under the normal (Gaussian) probability density function, integrated from minus infinity to <code>value1</code> (assumes mean is zero, variance is one) |
| <code>normalInverse(value1)</code> | Returns the value for which the area under the normal (Gaussian) probability density function is equal to the argument <code>value1</code> (assumes mean is zero, variance is one) |
| <code>poisson(value1,value2)</code> | Returns the sum of the first <code>value1</code> terms of the Poisson distribution. The arguments must both be positive. |
| <code>poissonComplemented(value1,value2)</code> | Returns the sum of the terms <code>value1+1</code> to infinity of the poisson distribution |
| <code>studentT(value1,value2)</code> | Returns the integral from minus infinity to <code>value2</code> of the Student-t distribution with <code>value1 > 0</code> degrees of freedom |

| Function Name | Description |
|--|---|
| <code>studentTInverse(value1, value2)</code> | Returns the value, for which the area under the Student-t probability density function is equal to $1 - \text{value1}/2$. The function uses the studentT function to determine the return value iteratively. |

3.11.10 What are String Functions?

The following String functions are supported in this release:

| Function Name | Description |
|--|---|
| <code>coalesce(value1, ...)</code> | Returns the first non-null expression in the list. If all expressions evaluate to null, then the COALESCE function will return null |
| <code>concat(value1, ...)</code> | Returns concatenation of values converted to strings |
| <code>indexOf(string, match)</code> | Returns first index of <code>'match'</code> in <code>'string'</code> or 1 if not found |
| <code>initcap(value1)</code> | Returns a specified text expression, with the first letter of each word in uppercase and all other letters in lowercase |
| <code>length(value1)</code> | Returns the length of the specified string |
| <code>like(value1, value2)</code> | Returns a matching pattern |
| <code>lower(value1)</code> | Converts the given string to lower case |
| <code>lpad(value1, value2, value3)</code> | Pads the left side of a string with a specific set of characters (when <code>string1</code> is not null) |
| <code>ltrim(value1, value2)</code> | Removes all specified characters from the left hand side of a string |
| <code>replace(string, match, replacement)</code> | Replaces all <code>'match'</code> with <code>'replacement'</code> in <code>'string'</code> |
| <code>rpadd(value1, value2, value3)</code> | Pads the right side of a string with a specific set of characters (when <code>string1</code> is not null) |
| <code>rtrim(value1, value2)</code> | Removes all specified characters from the right hand side of a string |
| <code>substr(string, from)</code> | Returns substring of a <code>'string'</code> when indices are between <code>'from'</code> (inclusive) and up to the end of the string |
| <code>substring(string, from, to)</code> | Returns substring of a <code>'string'</code> when indices are between <code>'from'</code> (inclusive) and <code>'to'</code> (exclusive) |
| <code>translate(value1, value2, value3)</code> | Replaces a sequence of characters in a string with another set of characters. However, it replaces a single character at a time. |
| <code>upper(value1)</code> | Converts given string to uppercase |