115
DBMS_KAFKA

The `DBMS_KAFKA` package provides a PL/SQL interface for enabling Oracle SQL access to topics in Kafka clusters.

Users granted `READ` access to an Oracle SQL access to Kafka (OSAK) cluster can use the `DBMS_KAFKA` package to create applications that query Kafka data from Oracle Database views and tables.

DBMS_KAFKA Overview

The `DBMS_KAFKA` packages enable you to access and process Kafka data with Oracle SQL access to Kafka (OSAK).

To enable applications to consume Kafka data, you use the `DBMS_KAFKA` packages. Before you can use Kafka resources, an administrator for Kafka must have registered and enabled access to a Kafka cluster.

To access the Kafka topics, you create an Oracle SQL Access to Kafka (OSAK) application. You first have to decide what mode to use.

- Loading mode: Use to load data from a Kafka topic into an Oracle Database table.
- Streaming mode: Use to read sequentially through a Kafka topic.
- Seekable mode: Use to randomly access a Kafka topic between starting and ending timestamps.

You then create the application using the appropriate OSAK package:

- `CREATE_LOAD_APP`: Creates an application that can be used in Loading mode.
- `CREATE_STREAMING_APP`: Creates an application that can be used in Streaming mode.
- `CREATE_SEEKABLE_APP`: Creates an application that can be used in Seekable mode.

Other DBMS_KAFKA packages enable you to manage the Kafka data.

The following is an overview of the procedure for each package

Loading Data Into a Load Application

- Use `DBMS_KAFKA.CREATE_LOAD_APP` to create an Oracle SQL Access to Kafka Load application
- Optionally, use `DBMS_KAFKA_INIT_OFFSET_TS` or `DBMS_KAFKA_INIT_OFFSET` to set the first Kafka record that you want to be read.
- Run `LOOP` until done.
  - Use `DBMS_KAFKA.EXECUTE_LOAD_APP` to load Kafka data starting from where a previous read left off to the current high water mark.
- Use `DBMS_KAFKA.DROP_LOAD_APP` to drop the load application after you are finished with the Kafka data.

Loading Data Into a Streaming Mode Application

To query Kafka data in Streaming mode to read sequentially through a Kafka topic, the procedure is as follows:

1. Use `DBMS_KAFKA.CREATE_STREAMING_APP` to create the Oracle SQL Access to Kafka streaming application.
2. Optionally, use `DBMS_KAFFA_INIT_OFFSET_TS` or `DBMS_KAFKA_INIT_OFFSET` to set the first Kafka record that you want to be read.
3. Run `LOOP` on the data in SQL:
   a. Call `DBMS_KAFKA.LOAD_TEMP_TABLE` to load the global temporary table with the next set of rows from Kafka
   b. Use `SELECT` from the OSAK global temporary table.
   c. Process the data retrieved
   d. If the processing was successful, then use `DBMS_KAFKA.UPDATE_OFFSET` to advance to the next set of Kafka records.
   e. Use `COMMIT` to commit the offset tracking information.
4. When finished with the application, use `DBMS_KAFKA.DROP_STREAMING_APP` to drop the application.

Loading Data into a Seekable Mode Application

To query Kafka data in Seekable mode, so that you can access Kafka records between two timestamps, an overview of the procedure is as follows:

1. `DBMS_KAFKA.CREATE_SEEKABLE_APP` Use to create the Oracle SQL Access to Kafka seekable application
2. Run `LOOP` on the Kafka data in SQL:
   a. Use `DBMS_KAFKA.SEEK_OFFSET_TS` to seek to a defined window of time in a Kafka topic.
   b. Call `DBMS_KAFKA.LOAD_TEMP_TABLE` to load a global temporary table with the set of rows from Kafka that you want to analyze.
   c. Use `SELECT` from the OSAK global temporary table.
   d. Process the data.
3. When done with the application, use `DBMS_KAFKA.DROP_SEEKABLE_APP` to drop the application.

DBMS_KAFKA LOADING Mode

Use the `DBMS_KAFKA LOADING` mode packages to load Kafka data incrementally into Oracle Database.

Loading procedures enable you to load available Kafka records into Oracle Database, which can then serve as a data warehouse for that data. You can then combine that data with Oracle Database tables for analytics.

An application declares that it is a loading application by calling the PL/SQL procedure `DBMS_KAFKA.CREATE_LOAD_APP` to initialize state for subsequent calls to `DBMS_KAFKA.EXECUTE_LOAD_APP`. `DBMS_KAFKA.CREATE_LOAD_APP` creates a single view over all partitions of the topic. An application can optionally call the `DBMS_KAFKA.INIT_OFFSET[_TS]` procedure to set the starting point in Kafka topic partitions.

The `DBMS_KAFKA.EXECUTE_LOAD_APP` procedure is called in an application loop to load data from where the previous call left off to the current high water mark of the Kafka topic. This procedure runs in an autonomous transaction.

When you are finished working with the Kafka data, you can remove the application by using `DBMS_KAFKA.DROP_LOAD_APP`.

CREATE_LOAD_APP

This procedure creates an Oracle SQL Access to Kafka Load application that retrieves data from all partitions in a Kafka topic to load that Kafka data into an Oracle Database table. It also creates, if not already present, a metadata view that is used to inspect the Kafka cluster for live topic and partition information regarding the Kafka topic. This view is created once, and serves all applications that are sharing the same cluster. This model is restrictive in that only one application instance is allowed to call `DBMS_KAFKA.EXECUTE_LOAD_APP` for the created `LOAD` application.

Parameters

Parameter	Description
`cluster_name`	The name of a registered Oracle SQL access to Kafka cluster that has the topic that you want to associate with this application. Case-insensitive. The registered cluster names can be obtained from the OSAK Administrator, by using the following statement: `SELECT cluster_name FROM sys.user_kafka_clusters;`
`application_name`	The application name. This parameter is also used as the Kafka group that can read the topic. Case-insensitive.
`topic_name`	The topic name in the Kafka cluster whose contents you want to retrieve. Case-sensitive.

Parameter	Description
options	Includes a list of properties formatted as a JSON document. Options are described in more detail in the topic "DBMS_KAFKA OPTIONS Passed to CREATE_XXX_APP".

Examples

Example 115-1 CREATE_LOAD_APP Procedure for Oracle SQL Access to Kafka

In the following example, a load application called ExampleApp is created for data from the Kafka cluster ExampleCluster, using the Kafka topic my-company-app-event1 and the option ProducerRecord.

```
PROCEDURE CREATE_LOAD_APP (
                           ExampleCluster         IN  VARCHAR2,
                           ExampleApp             IN  VARCHAR2,
                           my-company-app-event1  IN  VARCHAR2,
                           ProducerRecord         IN  CLOB
```

Related Topics

- DBMS_KAFKA OPTIONS Passed to CREATE_xxx_APP

DROP_LOAD_APP

The DBMS_KAFKA_ADM procedure DROP_LOAD_APP drops the Oracle SQL for Kafka (OSAK) LOAD application, and removes related metadata.

Syntax

```
EXEC DROP_LOAD_APP (
                          cluster_name     IN VARCHAR2,
                          application_name IN VARCHAR2
  );
```

Parameters

Table 115-1 DROP_LOAD_APP Procedure Parameters for DBMS_KAFKA_ADM.

Parameter	Description
cluster_name	Name of an existing Kafka cluster Case-insensitive.
application_name	Name of an existing application associated with the Kafka cluster. Case-insensitive.

Usage Notes

Use this procedure to drop an Oracle SQL access to Kafka (OSAK) application when you no longer want to load data from a Kafka topic into an Oracle Database table.

Examples

Suppose you have completed your work with the application called `ExampleApp` with data from the Kafka cluster `ExampleCluster`. Use this procedure to drop the application:

```
EXEC DROP_LOAD_APP (
                          ExampleCluster   IN VARCHAR2,
                          ExampleApp       IN VARCHAR2
  );
```

EXECUTE_LOAD_APP

The `DBMS_KAFKA` procedure `EXECUTE_LOAD_APP` loads a user table from a dedicated Oracle SQL access to Kafka (OSAK) view. To use this procedure, you must previously have created a load application with `CREATE_LOAD_APP`.

Syntax

```
PROCEDURE EXECUTE_LOAD_APP (
                             cluster_name        IN  VARCHAR2,
                             application_name    IN  VARCHAR2,
                             target_table        IN  VARCHAR2,
                             records_loaded      OUT INTEGER,
                             parallel_hint       IN INTEGER DEFAULT 0
);
```

Parameters

Table 115-2 EXECUTE_LOAD_APPS Procedure Parameters for DBMS_KAFKA

Parameter	Description
cluster_name	Name of a registered Oracle SQL access to Kafka cluster that has the topic that you want associated with this application. Case-insensitive. The registered cluster names can be obtained from the OSAK Administrator, or by using or by using the following SQL statement: `SELECT cluster_name from SYS.USER_KAFKA_CLUSTERS;`
application_name	The name of an existing application associated with the Kafka cluster Case-insensitive.
target_table	A target table in the Oracle Database that will be loaded with data from a Kafka topic. It must be consistent with some or all of the columns retrieved from the Kafka cluster topic by the OSAK view created by the `LOAD` operation.
records_loaded	(OUT) The number of Kafka records loaded

Table 115-2 (Cont.) EXECUTE_LOAD_APPS Procedure Parameters for DBMS_KAFKA

Parameter	Description
parallel_hint	(IN) (Optional) The degree of parallelism to use when loading the application that maps exclusively to a particular OSAK view. If a parallel hint is not specified, or it is less than or equal to 1, then parallelism is not used to load the table. **Note:** Only use a parallel hint when `PARALLEL_DEGREE_POLICY` for the user session or the system is set to `MANUAL`. For all other policies (for example, `AUTO`), no parallel hint should be passed. If parallel hint exceeds the granule count of an OSAK view, then an exception will be raised.

Usage Notes

Oracle SQL access to Kafka (OSAK) views are used transparently by PL/SQL calls to load records from a topic in a Kafka cluster previously created by the `CREATE_LOAD_APP` procedure. Applications call `EXECUTE_LOAD_APP` to load an Oracle Database user table from a dedicated OSAK view.

> **Note:**
>
> Only one user can query an OSAK temporary table using a single database application instance serving the OSAK application. However by using `EXECUTE_LOAD_APP`, you load Kafka data into a permanent database table that is accessible by multiple applications.

Each call to the `EXECUTE_LOAD_APP` procedure reads new records from the Kafka topic, and inserts these records into the Oracle Database table. `EXECUTE_LOAD_APP` also advances offsets of all Kafka partitions, so that the next time `EXECUTE_LOAD_APP` is run, it will insert new rows. Using EXECUTE_LOAD_APP enables you to perform incremental loads, so that you can update the Oracle Database table with updates to the Kafka topic. Because that Kafka data is moved into standard Oracle Database tables, they become available for processing and analysis by multiple applications.

Examples

Suppose you have completed your initial cluster definition for the Kafka cluster `ExampleCluster`, and registered the cluster. Next, you use this procedure to load data from the Kafka cluster `ExampleCluster` into the application `ExampleLoadApp`:

```
DECLARE
    v_records_inserted INTEGER;
BEGIN
    SYS.DBMS_KAFKA.EXECUTE_LOAD_APP (

'ExampleCluster',

'ExampleLoadApp',
```

```
'ExampleLoadTable',

v_records_inserted);
END;
```

DBMS_KAFKA Global Temporary Tables

Use the `DBMS_KAFKA LOAD_TEMP_TABLE` mode packages to load Kafka data into a temporary table, from which the Kafka data view in Oracle Database is created.

The `DBMS_KAFKA.LOAD_TEMP_TABLE` procedure is called in an application loop to load data into an Oracle SQL access to Kafka (OSAK) application. For both `STREAMING` and `SEEKABLE` applications, you use `CREATE_APP_xxx` (where xxx is either `STREAMING` or `SEEKABLE`) to create the application. Next, you use an application loop while calling `LOAD_TEMP_TABLE` for the application, and process the data loaded into the temporary tables.

LOAD_TEMP_TABLE

The `DBMS_KAFKA` procedure `LOAD_TEMP_TABLE` selects all data from an Oracle SQL access to Kafka view into a temporary table.Use this procedure to create an Oracle SQL access to Kafka (OSAK) dedicated temporary table that an application can use with SQL queries to analyze the data, or to or join with Oracle Database tables.

Syntax

```
FUNCTION LOAD_TEMP_TABLE(
                   temporary-table-name   IN VARCHAR2
  ) RETURN INTEGER;
```

Parameters

Table 115-3 LOAD_TEMP_TABLE Procedure Parameters for DBMS_KAFKA

Parameter	Description
temporary-table-name	Name of the temporary table that you want to create Type: VARCHAR Case-insensitive.
parallel-hint	(Optional) the degree of parallelism when loading the global temporary table that maps exclusively to a particular OSAK view. If a parallel hint is not specified, or it is less than or equal to 1, then parallelism is not used to load the table. **Note:** Only use a parallel hint when `PARALLEL_DEGREE_POLICY` for the user session or the system is set to `MANUAL`. For all other policies (for example, `AUTO`), no parallel hint should be passed. If parallel hint exceeds the granule count of an OSAK view, then an exception will be raised.

Usage Notes

Oracle SQL access to Kafka (OSAK) views are used transparently by PL/SQL calls to load records from Kafka topics into either a dedicated OSAK global temporary table or into a user-defined table. `STREAMING` and `SEEKABLE` applications call `LOAD_TEMP_TABLE` which loads an OSAK global temporary table from a dedicated OSAK view, while `LOAD` applications call `EXECUTE_LOAD_APP` to load a user table from a dedicated OSAK view.

> **Caution:**
>
> OSAK views are dedicated views, which can serve only one application instance. They are not shared by other instances of the application, and they should not be queried by a generalized tool (for example, SQL*Plus or SQL Developer). Concurrent access of an OSAK view can cause a race condition where a tool could inadvertently read more rows than the dedicated application, and inadvertently advance Kafka offsets beyond what the dedicated application has read. As a result, the application can then fail to obtain from the Kafka topic records that should have been processed.

Examples

Suppose you want to load and process records associated with a streaming application called `ExampleApp` from Kafka cluster `ExampleCluster`. By default, the OSAK views are configured to read from the earliest record present to last record currently published when they are initially created. After you create the streaming application with `DBMS_KAFKA.CREATE_STREAMING_APP`, the following is an example Kafka data processing loop for that streaming application:

```
BEGIN
   LOOP
       SYS.DBMS_KAFKA.LOAD_TEMP_TABLE

(ORA$DKVGTT_EXAMPLECLUSTER_EXAMPLEAPP_0);
       FOR kafka_record IN (
             SELECT kafka_offset offset
                    FROM ORA$DKVGTT_EXAMPLECLUSTER_EXAMPLEAPP_0)
       LOOP
              SYS.DBMS_OUTPUT.PUT_LINE ('Processing record: ' ||
kafka_record.offset);
              --application logic to process the Kafka records
       END LOOP;
       IF (application logic was successful) THEN
            --Update internal metadata to confirm Kafka records were
successfully processed
            SYS.DBMS_KAFKA.UPDATE_OFFSET

('ORA$DKV_EXAMPLECLUSTER_EXAMPLEAPP_0');
            COMMIT;
       ELSE
            --add your application logic to correct for any failures
       END IF;
   END LOOP;
END;
```

DBMS_KAFKA Streaming Mode

Use the `DBMS_KAFKA STREAMING` mode packages to stream data from a Kafka topic into an Oracle SQL access to Kafka (OSAK) global temporary table from a dedicated OSAK view. To use this package, you must first use `LOAD_TEMP_TABLE` to create the dedicated OSAK view. Use `STREAMING` mode for applications that require access to Kafka topics in a sequential manner from the beginning, or from a specific starting point in a Kafka topic. This mode allows a SQL query using an OSAK temporary table to access Kafka records sequentially in an application processing loop. An application declares that it is a streaming application by calling the PL/SQL procedure `DBMS_KAFKA.CREATE_STREAMING_APP` to initialize state for subsequent queries of OSAK views.

CREATE_STREAMING_APP

The `DBMS_KAFKA` procedure `CREATE_STREAMING_APP` creates an Oracle SQL access to Kafka (OSAK) streaming application. The application includes a set of dedicated OSAK global temporary tables and OSAK views that are used for retrieving new and unread records from partitions in a Kafka topic. It also creates, if not already present, a metadata view that is used to inspect the Kafka cluster for live topic and partition information regarding the Kafka topic. This view is created once, and serves all applications that are sharing the same cluster.

Syntax

```
PROCEDURE CREATE_STREAMING_APP (
                                  cluster_name          IN  VARCHAR2,
                                  application_name      IN  VARCHAR2,
                                  topic_name            IN  VARCHAR2,
                                  options               IN  CLOB,
                                  view_count            IN  INTEGER DEFAULT 1
  );
```

Parameters

Table 115-4 CREATE_STREAMING_APP Procedure Parameters for DBMS_KAFKA

Parameter	Description
cluster_name	Name of a registered Oracle SQL access to Kafka cluster that has the topic that you want associated with this application. Case-insensitive. The registered cluster names can be obtained from the OSAK Administrator, or by using or by using the following SQL statement: `SELECT cluster_name from SYS.USER_KAFKA_CLUSTERS;`
application_name	The name of the application. Also used as the Kafka group that will read the topic. Case-insensitive.
topic_name	The topic name in the Kafka cluster whose contents will be retrieved.

Table 115-4 (Cont.) CREATE_STREAMING_APP Procedure Parameters for DBMS_KAFKA

Parameter	Description
options	Includes a list of properties formatted as a JSON document. Options are described in more detail in the topic "DBMS_KAFKA OPTIONS Passed to CREATE_XXX_APP".
view_count	(OPTIONAL) Identifies the number of Oracle SQL access to Kafka (OSAK) view pairs to create. Valid values are 1 to *N*, where *N* is the number of Kafka partitions in a topic, or 0, which defaults to *N*. (Default is 1).

Usage Notes

Each OSAK view is exclusively used by one instance of an Oracle SQL access to Kafka application. Each application instance call populates the view with Kafka rows. The application then can then run one or more SQL queries against the content in the OSAK view A STREAMING application is different from a LOAD or SEEKING application in that it can choose how many OSAK views need to be created. The number of OSAK views must be between 1 and *N* where *N* is the number of partitions in the Kafka topic.

As with other types of OSAK applications, each application instance exclusively queries one unique OSAK temporary table. Each OSAK view includes the cluster name, the application name, and an application instance identifier (ID). Creating multiple application instances enables applications to scale out and divide the workload of analyzing Kafka data across application instances running concurrently on one or more threads, processes, or systems.

The number of Kafka partitions bound to a specific OSAK view and its associated OSAK global temporary table will vary depending upon how many views are created and how many partitions exist. If *N* OSAK view/temporary table pairs are created, then the application user must have been allocated at least *N* sessions per user, so that *N* application instances can run concurrently.

Examples

Suppose you want to create a set of four views for a streaming application called `ExampleApp`, streamed from a Kafka topic called `ExampleTopic` in the Kafka cluster `ExampleCluster` that has four partitions, where each view is associated with one partition. You can enter the following statement:

```
DECLARE
   v_options  VARCHAR2;
BEGIN
  v_options := '{"fmt" : "DSV", "reftable" : "user_reftable_name"}';
  SYS.DBMS_KAFKA.CREATE_STREAMING_APP (
                                     'ExampleCluster',
                                     'ExampleApp',
                                     'ExampleTopic',
                                      v_options,
                                      4);
END;
/
```

Alternatively, to create one view for an application that is associated with all four partitions of the topic, you enter the following statement:

```
DECLARE
v_options VARCHAR2;
BEGIN
              v_options := '{"fmt" : "DSV", "reftable" :
"user_reftable_name"}';
 SYS.DBMS_KAFKA.CREATE_STREAMING_APP (

'ExampleCluster',

'ExampleApp',

'ExampleTopic',
                                                        v_options,
                                                        1);

END;
/
```

Related Topics

- DBMS_KAFKA OPTIONS Passed to CREATE_xxx_APP

DROP_STREAMING_APP

The `DBMS_KAFKA` procedure `DROP_STREAMING_APP` drops the streaming application.This function removes the Oracle SQL access to Kafka (OSAK) view, and drops all associated database objects.

Syntax

```
PROCEDURE DROP_STREAMING_APP (
                                cluster_name     IN VARCHAR2,
                                application_name IN VARCHAR2
  );
```

Parameters

Table 115-5 DROP_STREAMING_APP Procedure Parameters for DBMS_KAFKA

Parameter	Description
cluster_name	Name of an existing Kafka cluster Case-insensitive.
application_name	The name of an existing application using the Kafka cluster. Case-insensitive.

Usage Notes

When you are done with an application, you use this function to drop an OSAK view associated with an application.

Examples

Suppose you have completed your work with the Kafka cluster `ExampleCluster`, which was being used by Streaming application `ExampleApp`. You can then use this procedure to drop the application::

```
EXEC SYS.DBMS_KAFKA.DROP_STREAMING_APP (
                                        'ExampleCluster', 'ExampleApp');
```

INIT_OFFSET

The `DBMS_KAFKA_ADM` procedure `INIT_OFFSET` enables you to select a particular offset as the starting point for reading Kafka data. Use this option when you want to select a particular starting point in the data to load, instead of loading data from the first record available.

Syntax

```
PROCEDURE INIT_OFFSET (
         view_name IN VARCHAR2,
         record_count IN INTEGER,
         water_mark IN VARCHAR2 ;
```

Parameters

Table 115-6 INIT_OFFSET Procedure Parameters for DBMS_KAFKA

Parameter	Description
view_name	Name of an existing OSAK view (VARCHAR2)
record_count	record number (INTEGER)
water_mark	Watermark (VARCHAR) The high or low watermark that indicates the desired relative positioning. Values are restricted to `WATER_MARK_HIGH` (`'WMH'`) or `WATER_MARK_LOW` (`'WM'`) constants. Default: `WATER_MARK_HIGH`

Usage Notes

The `INIT_OFSET` procedure enables `STREAMING` or `LOAD` applications to start reading current records after the application has been created without being forced to first read a backlog of old records that are no longer interesting.

Use this function to specify a starting point based on the difference (delta) number of records from either the high or low water mark of every partition where you want your application to read data, instead of starting the read point from the beginning of the data records in the Kafka topic.

Examples

Suppose that you want your application to restart the processing with the last 100 records available and continue on from that point. You can achieve this result by running the following

procedure before the application is restarted, or as part of the application logic before data retrieval loop:

```
SYS.DBMS_KAFKA.INIT_OFFSET (

'ORA$DKV_EXAMPLECLUSTER_EXAMPLEAPP_0',
                                        100,
SYS.DBMS_KAFKA.WATER_MARK_HIGH);
```

INIT_OFFSET_TS (Milliseconds Since Epoch)

The `DBMS_KAFKA` procedure `INIT_OFFSET_TS` using milliseconds since epoch specifies a starting offset using a timestamp.

Using milliseconds since epoch initializes the starting offset related to a timestamp for each Kafka partition belonging to the OSAK view. INIT_OFFSET_TS would typically be called at the outset of a new application instance dedicated to processing the view or recovering after an application instance shutdown or failure.

.

Syntax

```
PROCEDURE INIT_OFFSET_TS (
                            view_name           IN VARCHAR2,
                            start_timestamp_ms  IN INTEGER);
```

Parameters

Table 115-7 INIT_OFFSET_TS (Millseconds Since Epoch) Procedure Parameters for DBMS_KAFKA

Parameter	Description
view_name	Name of an existing Kafka cluster (`VARCHAR`) Case-insensitive.
start_timestamp	The timestamp of the offset from which you want to start your application (INTEGER). The first record returned will have a timestamp equal to the timestamp provided, or the nearest timestamp greater than the timestamp provided.

Usage Notes

`INIT_OFFSET_TS` using milliseconds since a Kafka epoch initializes the starting offset related to a timestamp for each Kafka partition belonging to the OSAK view. A typical use case is to call `INIT_OFFSET_TS` with a specified epoch starting point at the outset of a new application instance dedicated to processing the view, or to recovering after an application instance shutdown or failure.

This procedure serves to position the processing of Kafka topic records to a point that is relatively current, potentially skipping unprocessed older records in the Kafka partitions.

> **Note:**
>
> Be aware that the time between initializing the offset and the first fetch can be delayed. During this gap in time, it is possible that the record for the chosen offset can be deleted due to either the record exceeding the Kafka retention time, or to the record being explicitly removed.

Examples

Suppose that you want to calculate and return the number of milliseconds since the epoch time from the input `TIMESTAMP`. The timestamp is considered to be in the session's timezone unless the timezone is provided. You provide the parameter `datetime` (an integer) Timestamp to convert to milliseconds since epoch time. The parameter timezone (integer) is optional, providing the timezone of the timestamp. If you do not specify a timezone, then the timezone defaults to the session's timezone:

```
SYS.DBMS_KAFKA.SEEK_OFFSET_TS ('ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0',
                               1593829800,
                               1593833400);

SYS.DBMS_KAFKA.LOAD_TEMP_TABLE ('ORA$DKVGTT_EXAMPLECLUSTER_SEEKABLEAPP_0');
SELECT <columns> FROM ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0;
```

INIT_OFFSET_TS (Timestamp with Separate Timezone Parameter)

The `DBMS_KAFKA` procedure `INIT_OFFSET_TS` using a TIMESTAMP with a string parameter specifying the TIMESTAMP TIME ZONE. specifies a starting position to return dataUsing a timestamp specified with a timestamp with time zone positions the processing of Kafka data to a timezone related to a timestamp for each Kafka partition belonging to the Oracle SQL access to Kafka (OSAK) view. `INIT_OFFSET_TS` would typically be called at the outset of a new application instance dedicated to processing the view or recovering after an application instance shutdown or failure.

Syntax

```
PROCEDURE INIT_OFFSET_TS (
                            view_name           IN VARCHAR2,
                            start_timestamp     IN TIMESTAMP,
                            timezone            IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 115-8 INIT_OFFSET_TS Timestamp with Time Zone Procedure Parameters for DBMS_KAFKA

Parameter	Description
view_name	Name of an existing Kafka cluster (`VARCHAR`) Case-insensitive.

Table 115-8 (Cont.) INIT_OFFSET_TS Timestamp with Time Zone Procedure Parameters for DBMS_KAFKA

Parameter	Description
start_timestamp	The timestamp of the offset from which you want to start your application (INTEGER). The first record returned will have a timestamp equal to the timestamp provided, or the nearest timestamp greater than the timestamp provided.
timezone	The timezone of the timestamp (INTEGER). If no value is provided, then the default is to use the session timezone.

Usage Notes

`INIT_OFFSET_TS` using a timestamp with a separate time zone parameter initializes the starting offset related to a timestamp and a time zone for each Kafka partition belonging to the OSAK view. A typical use case is to call `INIT_OFFSET_TS` with a timestamp with a separate time zone parameter at the outset of a new application instance dedicated to processing the view, or to recovering after an application instance shutdown or failure.

This procedure serves to position the processing of Kafka topic records to a point that is relatively current, potentially skipping unprocessed older records in the Kafka partitions.

> **Note:**
>
> Be aware that the time between initializing the offset and the first fetch can be delayed. During this gap in time, it is possible that the record for the chosen offset can be deleted due to either the record exceeding the Kafka retention time, or to the record being explicitly removed.

Examples

Suppose that you want to select a later time to process data from a Kafka topic than at the first record available. For example, if a data center was down for maintenance for a weekend, and you only want to process new Kafka data generated after the data center was reopened at 6 P.M. (18:00:00), then you want to start the data record timestamped after 6 PM. To achieve this with an OSAK view, run the following procedure before the application is restarted, or as part of the application logic before the data retrieval loop:

```
SYS.DBMS_KAFKA.INIT_OFFSET_TS (

'ORA$DKV_EXAMPLECLUSTER_EXAMPLEAPP_0',
                                        TO_DATE ('2023/07/05 18:00:00',

'YYYY/MM/DD HH:MI:SS'))
                                        timestamp=1603507387101;
```

INIT_OFFSET_TS (Timestamp with Time Zone)

The `DBMS_KAFKA` procedure `INIT_OFFSET_TS` using timestamp with timezone specifies a starting offset using a timestamp.Using a timestamp specified with a time zone initializes the starting

offset related to a timestamp for each Kafka partition belonging to the Oracle SQL access to Kafka (OSAK) view. `INIT_OFFSET_TS` would typically be called at the outset of a new application instance dedicated to processing the view or recovering after an application instance shutdown or failure.

Syntax

```
PROCEDURE INIT_OFFSET_TS (
                            view_name           IN VARCHAR2,
                            start_timestamp     IN TIMESTAMP WITH TIME ZONE);
```

Parameters

Table 115-9 INIT_OFFSET_TS Procedure Parameters for DBMS_KAFKA

Parameter	Description
view_name	Name of an existing Kafka cluster (`VARCHAR`) Case-insensitive.
start_timestamp	The timestamp of the offset from which you want to start your application (INTEGER). The first record returned will have a timestamp equal to the timestamp provided, or the nearest timestamp greater than the timestamp provided.
timezone	(Optional). The timezone of the timestamp (INTEGER). If not provided, then it defaults to the session timezone

Usage Notes

`INIT_OFFSET_TS` initializes the starting offset related to a timestamp for each Kafka partition belonging to the OSAK view. A typical use case is to call INIT_OFFSET_TS at the outset of a new application instance dedicated to processing the view, or to recovering after an application instance shutdown or failure.

This procedure serves to position the processing of Kafka topic records to a point that is relatively current, potentially skipping unprocessed older records in the Kafka partitions.

> **Note:**
>
> Be aware that the time between initializing the offset and the first fetch can be delayed. During this gap in time, it is possible that the record for the chosen offset can be deleted due to either the record exceeding the Kafka retention time, or to the record being explicitly removed.

Examples

Timestamp with specified time zone

Suppose that you want to select a later time to process data from a Kafka topic than at the first record available. For example, if a data center was down for maintenance for a weekend, and you only want to process new Kafka data generated after the data center was reopened at 6 P.M. (18:00:00), then you want to start the data record timestamped after 6pm. To achieve this

with an OSAK view, run the following procedure before the application is restarted, or as part of the application logic before the data retrieval loop:

```
SYS.DBMS_KAFKA.SEEK_OFFSET_TS ('ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0',
                               TO_TIMESTAMP ('2020/07/04 02:30:00',
'YYYY/MM/DD HH:MI:SS',
                               TO_TIMESTAMP ('2020/07/04 03:30:00',
'YYYY/MM/DD HH:MI:SS'),
                               'UTC');

SYS.DBMS_KAFKA.LOAD_TEMP_TABLE ('ORA$DKVGTT_EXAMPLECLUSTER_SEEKABLEAPP_0');
SELECT <columns> FROM ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0;
```

Timestamp without a specified time zone

```
SYS.DBMS_KAFKA.SEEK_OFFSET_TS ('ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0',
                               TO_TIMESTAMP_TZ ('2020/07/04 02:30:00 -8:00',
'YYYY/MM/DD HH:MI:SS TZH:TZM',
                               TO_TIMESTAMP_TZ ('2020/07/04 03:30:00 -8:00',
'YYYY/MM/DD HH:MI:SS TZH:TZM'));

SYS.DBMS_KAFKA.LOAD_TEMP_TABLE ('ORA$DKVGTT_EXAMPLECLUSTER_SEEKABLEAPP_0');
SELECT <columns> FROM ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0;
```

UPDATE_OFFSET

The `DBMS_KAFKA` procedure `UPDATE_OFFSET` update the last Kafka offsets read so the next pass in the loop will retrieve and process new unread Kafka records. `UPDATE_OFFSET` transparently advances Kafka partition offsets of the Kafka group ID for all of the partitions belonging to the Oracle SQL access to Kafka (OSAK) view, so that for every call to `DBMS_KAFKA.LOAD_TEMP_TABLE`, a new set of unread Kafka records is retrieved and processed.

Syntax

```
PROCEDURE UPDATE_OFFSET (view_name IN VARCHAR2);
```

Parameters

Table 115-10 UPDATE_OFFSET Procedure Parameters for DBMS_KAFKA

Parameter	Description
view_name	The name of an OSAK view Case-insensitive.

Usage Notes

`UPDATE_OFFSET` advances the Kafka offset read point to the next set of Kafka records if the processing of the earlier records was successful. `UPDATE_OFFSET` also initiates an Oracle Database transaction, if a transaction is not already started, and records the last offsets in metadata tables. For this reason, applications should commit a transaction after each call to `UPDATE_OFFSET`. Because OSAK manages offsets within an Oracle Database transaction, in which ACID (Atomicity, Consistency, Isolation, Durability) is preserved in the transaction, no

records are lost or reread. If the transaction fails to complete successfully, then offsets are not advanced, and the application will pick up where it previously off off when it resumes.

Examples

A typical use case is to advance an offset read to the next set of Kafka records after the previous set has been processed successfully, such as in a Streaming mode application. For example:

```
BEGIN
   LOOP
       SYS.DBMS_KAFKA.LOAD_TEMP_TABLE

(ORA$DKVGTT_EXAMPLECLUSTER_EXAMPLEAPP_0);
       FOR kafka_record IN (
             SELECT kafka_offset offset
                    FROM ORA$DKVGTT_EXAMPLECLUSTER_EXAMPLEAPP_0)
       LOOP
              SYS.DBMS_OUTPUT.PUT_LINE ('Processing record: ' ||
kafka_record.offset);
              --application logic to process the Kafka records
       END LOOP;
       IF (application logic was successful) THEN
            --Update internal metadata to confirm Kafka records were
successfully processed
            SYS.DBMS_KAFKA.UPDATE_OFFSET

('ORA$DKV_EXAMPLECLUSTER_EXAMPLEAPP_0');
            COMMIT;
       ELSE
            --application logic to correct what failed
       END IF;
   END LOOP;
END;
```

You can also use it as part of the process of access an OSAK view directly to retrieve updates. For example, suppose you have a Streaming mode application that is monitoring average temperatures over time in a laboratory called `Lab1`. You can use `UPDATE_OFFSET` to retrieve the average temperature since it was last checked:

```
EXEC DBMS_KAFKA.ENABLE_VIEW_QUERY(

'ORA$DKV_EXAMPLECLUSTER_STREAMINGAPP_0');
SELECT AVG(temperature) FROM ORA$DKV_EXAMPLECLUSTER_STREAMINGAPP_0
                                                    WHERE sensor_name =
'LAB1_SENSOR';
EXEC DBMS_KAFKA.UPDATE_OFFSET(

'ORA$DKV_EXAMPLECLUSTER_STREAMINGAPP_0');
COMMIT;
```

DBMS_KAFKA Seekable Mode

Use the DBMS_KAFKA SEEKABLE mode packages to read Kafka records that exist between two points in time.
An application declares that it is a loading application by calling the PL/SQL procedure DBMS_KAFKA.CREATE_SEEKABLE_APP to set up and create a seekable Oracle SQL access to Kafka (OSAK) view and temporary table.

Use the Seekable mode view with SEEK_OFFSET_TS to search through the Kafka data to find a set of data whose timestamps exist between two points in time. You can then use LOAD_TEMP_TABLE to load that data range into an OSAK temporary table, and run one or more application queries against the OSAK global temporary table containing the Kafka records in that timestamp range. When you have completed your queries, you can then use DROP_SEEKABLE_APP to drop the application.

.

CREATE_SEEKABLE_APP

The DBMS_KAFKA procedure CREATE_SEEKABLE_APP Creates one Oracle SQL access to Kafka (OSAK) view and an associated global temporary table You can use CREATE_SEEKABLE_APP to seek and load Kafka records between a particular timeframe. Use of this procedure is restricted to one application instance to perform seek operations.

Syntax

```
PROCEDURE CREATE_SEEKABLE_APP (
                                cluster_name          IN  VARCHAR2,
                                application_name      IN  VARCHAR2,
                                topic_name            IN  VARCHAR2,
                                options               IN  CLOB
```

Parameters

Parameter	Description
cluster_name	The name of a registered Oracle SQL access to Kafka cluster that has the topic that you want to associate with this application. Case-insensitive. The registered cluster names can be obtained from the OSAK Administrator, by using the following statement: `SELECT cluster_name FROM sys.user_kafka_clusters;`
application_name	The application name. This parameter is also used as the Kafka group that can read the topic. Case-insensitive.
topic_name	The topic name in the Kafka cluster whose contents you want to retrieve. Case-sensitive.

Parameter	Description
options	Includes a list of properties formatted as a JSON document. Options are described in more detail in the topic "DBMS_KAFKA OPTIONS Passed to CREATE_XXX_APP".

Examples

Example 115-2 CREATE_SEEKABLE_APP Procedure for Oracle SQL Access to Kafka

```
DECLARE
   v_options  VARCHAR2;
BEGIN
  v_options := '{"fmt" : "DSV", "reftable" : "user_reftable_name"}';
  SYS.DBMS_KAFKA.CREATE_SEEKABLE_APP (
                                     'ExampleCluster',
                                     'ExampleApp',
                                     'ExampleTopic',
END;
/
```

Related Topics

- DBMS_KAFKA OPTIONS Passed to CREATE_xxx_APP

DROP_SEEKABLE_APP

The DBMS_KAFKA procedure DROP_SEEKABLE_APP drops the seeking application. This function removes the Oracle SQL access to Kafka (OSAK) view, and its related metadata.

Syntax

```
PROCEDURE DROP_SEEKABLE_APP (
                                cluster_name     IN VARCHAR2,
                                application_name IN VARCHAR2
  );
```

Parameters

Table 115-11 DROP_SEEKABLE_APP Procedure Parameters for DBMS_KAFKA

Parameter	Description
cluster_name	Name of an existing Kafka cluster Case-insensitive.
application_name	The name of an existing application using the Kafka cluster. Case-insensitive.

Usage Notes

When you are done with an application, you use this function to drop an OSAK view associated with an application.

Examples

Suppose you have completed your work with the Kafka cluster `ExampleCluster`, which was being used by the Seekable application `ExampleApp`. You can then use this procedure to drop the application:

```
EXEC SYS.DBMS_KAFKA.DROP_SEEKABLE_APP (
                                        'ExampleCluster', 'ExampleApp');
```

SEEK_OFFSET_TS (Timestamp with Separate Timezone Parameter)

The `DBMS_KAFKA` procedure `SEEK_OFFSET_TS` using a timestamp with a timezone parameter positions an Oracle SQL access to Kafka (OSAK) view to start reading Kafka records between two Kafka epoch timestamps that you specify.Use this procedure to specify a timeframe for records within a Kafka topic that you want to seek, as specified by epoch time in milliseconds.

Syntax

```
PROCEDURE SEEK_OFFSET_TS(
                view_name           IN VARCHAR2,
                start_timestamp_ms  IN INTEGER,
                end_timestamp_ms    IN INTEGER,
                timezone  IN INTEGER);
```

Parameters

Table 115-12 SEEK_OFFSET_TS (Timestamp with Separate Timezone Parameter)Procedure Parameters for DBMS_KAFKA

Parameter	Description
view_name	Name of an existing Kafka cluster (`VARCHAR`) Case-insensitive.
start_timestamp	The timestamp of the offset from which you want to start your application (INTEGER). The last record returned will have a timestamp equal to the timestamp provided, or the nearest timestamp greater than the timestamp provided.

Table 115-12 (Cont.) SEEK_OFFSET_TS (Timestamp with Separate Timezone Parameter)Procedure Parameters for DBMS_KAFKA

Parameter	Description
end_timestamp	The timestamp of the offset from which you want to end your application (INTEGER). The last record returned will have a timestamp equal to the timestamp provided, or the nearest timestamp less than the timestamp provided. The range of records returned is inclusive of the start timestamp and exclusive on the end timestamp. For example, a range of 2:00PM to 3:00PM returns records with timestamps in the range of 2:00:00.000000 PM to 2:59:59.999999PM This range enables the application to search from 2:00 to 3:00, and then 3:00 to 4:00, without having overlapping records between the sets.
timezone	The timezone of both of the timestamp arguments (INTEGER). If no value is provided, then the default is to use the session timezone.

Usage Notes

`SEEK_OFFSET_TS` seeks Kafka topic records that exist within a specific timeframe that you select, specified by Kafka epoch time in milliseconds. The last record returned is the last record published.

The purpose of `SEEK_OFFSET_TS` is to position an OSAK view to start reading Kafka records within a given timeframe, as defined by the epoch timestamps that you specify. If the window of records exceeds the range of actual records in a Kafka topic, this procedure will return whatever records do exist.

For example:

- If the timestamps are either both below the low water mark, or both above the high water mark, then no records will be returned.
- If the start timestamp is below the low water mark, then the first record returned will be the low water mark.
- If the end timestamp is above the high water mark, then the last record returned will be the difference of the high water mark (HWM), minus 1 (HWM - 1). For example, in a new topic has 100 records, the offset range is from 0 to 99, and the HWM is set at 100.

> **Note:**
>
> The data retrieved by a SEEKABLE application using Kafka epoch timeframes as the specification for retrieval cannot include any outlier records that are delivered after the load of the OSAK global temporary table, even if they have a timestamp that is within the timestamp window provided.

Examples

Suppose that you want to investigate issues that have occurred in the past. If the data is still present in the Kafka steam, then you can create a Seekable application by calling

DBMS_KAFKA.CREATE_SEEKABLE_APP. You can then call the SEEK_OFFSET_TS procedure to request the OSAK view to retrieve a range of data records. For example, if an IT consultant was informed that a production issue occurred around 3:00 in the morning, then the consultant can use the following procedure to load the temporary table, and then select to retrieve an hour's worth of data around that time:

```
SYS.DBMS_KAFKA.SEEK_OFFSET_TS (

'ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0',
                                          TO_DATE ('2023/04/02 02:30:00',
'YYYY/MM/DD HH:MI:SS',
                                          TO_DATE ('2023/04/02 03:30:00',
'YYYY/MM/DD HH:MI:SS'));
SYS.DBMS_KAFKA.LOAD_TEMP_TABLE

(ORA$DKVGTT_EXAMPLECLUSTER_SEEKABLEAPP_0);
SELECT <columns> FROM ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0;
```

Another use case might be if an application with sequential access to a Kafka stream detected a potential anomaly, and was configured to insert a row into an anomaly table. The anomaly table would include the Kafka timestamp, as well as any other data it was configured to record. In that case, another application can then use this information to retrieve records around the suspected record to see if there were any other issues. To achieve this goal, run the following procedure, load the temporary table, and then select and apply application logic to the results:

```
SYS.DBMS_KAFKA.SEEK_OFFSET_TS (

'ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0',
                                          TO_DATE ('2020/07/04 02:30:00',
'YYYY/MM/DD HH:MI:SS',
                                          TO_DATE ('2020/07/04 03:30:00',
'YYYY/MM/DD HH:MI:SS'));

SYS.DBMS_KAFKA.LOAD_TEMP_TABLE

(ORA$DKVGTT_EXAMPLECLUSTER_SEEKABLEAPP_0);
SELECT <columns> FROM ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0;
--application logic
```

SEEK_OFFSET_TS (Timestamp in Milliseconds)

The DBMS_KAFKA procedure SEEK_OFFSET_TS (timestamp in milliseconds) seeks Kafka records in a topic that exist between a window of time specified by epoch time in milliseconds.

Syntax

```
PROCEDURE SEEK_OFFSET_TS(
                view_name           IN VARCHAR2,
                start_timestamp_ms  IN INTEGER,
                end_timestamp_ms    IN INTEGER);
```

Parameters

Table 115-13 SEEK_OFFSET_TS (Timestamp in Milliseconds) Procedure Parameters for DBMS_KAFKA

Parameter	Description
view_name	Name of an existing Oracle SQL access to Kafka (OSAK) view. Case-insensitive.
start_timestamp_ms	The timestamp in milliseconds where you want to seek to the first record
end_timestamp_ms	The timestamp in milliseconds where you want to seek to the last record

Usage Notes

`SEEK_OFFSET_TS` seeking a timestamp range in milliseconds)seeks Kafka topic records that exist within a starting timestamp and ending timestamp window that you select, specified by Kafka epoch time in milliseconds. The last record returned is the last record published.

The purpose of `SEEK_OFFSET_TS` using milliseconds to specify the window is to position an OSAK view to start reading Kafka records within a given timeframe, as defined by the epoch timestamps that you specify. If the window of records exceeds the range of actual records in a Kafka topic, this procedure will return whatever records do exist.

For example:

- If the timestamps are either both below the low water mark, or both above the high water mark, then no records will be returned.
- If the start timestamp is below the low water mark, then the first record returned will be the low water mark.
- If the end timestamp is above the high water mark, then the last record returned will be the difference of the high water mark (HWM), minus 1 (HWM - 1). For example, in a new topic has 100 records, the offset range is from 0 to 99, and the HWM is set at 100.

> **Note:**
>
> Kafka record timestamps are either assigned by Kafka (that is, by transaction time), or are assigned by the application (that is, valid or decision time). The data retrieved by a SEEKABLE application will not include outlier records that were delivered after the load of the OSAK global temporary table that have a timestamp that is within the timestamp window provided.

Examples

Suppose that you want to investigate issues that have occurred in the past. If the data is still present in the Kafka steam, then you can create a Seekable application by calling `DBMS_KAFKA.CREATE_SEEKABLE_APP`. You can then call the `SEEK_OFFSET_TS` procedure to request the OSAK view to retrieve a range of data records. For example, if an IT consultant was informed that a production issue occurred around 3:00 in the morning, then the consultant

can use the following procedure to load the temporary table, and then select to retrieve an hour's worth of data around that time:

```
SYS.DBMS_KAFKA.SEEK_OFFSET_TS (

'ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0',
                                          TO_DATE ('2023/04/02 02:30:00',
'YYYY/MM/DD HH:MI:SS',
                                          TO_DATE ('2023/04/02 03:30:00',
'YYYY/MM/DD HH:MI:SS'));
SYS.DBMS_KAFKA.LOAD_TEMP_TABLE

(ORA$DKVGTT_EXAMPLECLUSTER_SEEKABLEAPP_0);
SELECT <columns> FROM ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0;
```

Another use case might be if an application with sequential access to a Kafka stream detected a potential anomaly, and was configured to insert a row into an anomaly table. The anomaly table would include the Kafka timestamp, as well as any other data it was configured to record. In that case, another application can then use this information to retrieve records around the suspected record to see if there were any other issues. To achieve this goal, run the following procedure, load the temporary table, and then select and apply application logic to the results:

```
SYS.DBMS_KAFKA.SEEK_OFFSET_TS (

'ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0',
                                          TO_DATE ('2020/07/04 02:30:00',
'YYYY/MM/DD HH:MI:SS',
                                          TO_DATE ('2020/07/04 03:30:00',
'YYYY/MM/DD HH:MI:SS'));

SYS.DBMS_KAFKA.LOAD_TEMP_TABLE

(ORA$DKVGTT_EXAMPLECLUSTER_SEEKABLEAPP_0);
SELECT <columns> FROM ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0;
--application logic
```

SEEK_OFFSET_TS (Timestamp with Timezone)

The `DBMS_KAFKA` procedure `SEEK_OFFSET_TS` seeks Kafka records in a topic that exist between a window of time specified by `TIMESTAMP WITH TIME ZONE`.Use this procedure to position an Oracle SQL access to Kafka (OSAK) view to start reading Kafka records between two specified `TIMESTAMP WITH TIME ZONE` timestamps that you specify.

Syntax

```
PROCEDURE SEEK_OFFSET_TS(
                view_name           IN VARCHAR2,
                start_timestamp_ms  IN INTEGER,
                end_timestamp_ms    IN INTEGER);
```

Parameters

Table 115-14 SEEK_OFFSET_TS Procedure Parameters for DBMS_KAFKA

Parameter	Description
view_name	Name of an existing Kafka view name (VARCHAR) Case-insensitive.
start_timestamp	The timestamp where you want to seek the first record (INTEGER)
end_timestamp	The timestamp where you want to seek to the last record (INTEGER).

Usage Notes

The purpose of SEEK_OFFSET_TS defined by start and end timestamps is to position an OSAK view to start reading Kafka records within a given timeframe, as defined by the epoch timestamps that you specify. If the window of records exceeds the range of actual records in a Kafka topic, this procedure will return whatever records do exist.

For example:

- If the timestamps are either both below the low water mark, or both above the high water mark, then no records will be returned.
- If the start timestamp is below the low water mark, then the first record returned will be the low water mark.
- If the end timestamp is above the high water mark, then the last record returned will be the difference of the high water mark (HWM), minus 1 (HWM - 1). For example, in a new topic has 100 records, the offset range is from 0 to 99, and the HWM is set at 100.

> **Note:**
>
> Kafka record timestamps are either assigned by Kafka (that is, transaction time), or are assigned by applications (that is, valid or decision time).The data retrieved by a SEEKABLE application using Kafka epoch timeframes as the specification for retrieval cannot include any outlier records that are delivered after the load of the OSAK global temporary table, even if they have a timestamp that is within the timestamp window provided.

Examples

Suppose that you want to investigate issues that have occurred in the past. If the data is still present in the Kafka steam, then you can create a Seekable application by calling DBMS_KAFKA.CREATE_SEEKABLE_APP. You can then call the SEEK_OFFSET_TS procedure to request the OSAK view to retrieve a range of data records. For example, if an IT consultant was informed that a production issue occurred around 3:00 in the morning, then the consultant can use the following procedure to load the temporary table, and then select to retrieve an hour's worth of data around that time:

```
SYS.DBMS_KAFKA.SEEK_OFFSET_TS (

'ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0',
```

```
                                          TO_DATE ('2023/04/02 02:30:00',
'YYYY/MM/DD HH:MI:SS',
                                          TO_DATE ('2023/04/02 03:30:00',
'YYYY/MM/DD HH:MI:SS'));
SYS.DBMS_KAFKA.LOAD_TEMP_TABLE

(ORA$DKVGTT_EXAMPLECLUSTER_SEEKABLEAPP_0);
SELECT <columns> FROM ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0;
```

Another use case might be if an application with sequential access to a Kafka stream detected a potential anomaly, and was configured to insert a row into an anomaly table. The anomaly table would include the Kafka timestamp, as well as any other data it was configured to record. In that case, another application can then use this information to retrieve records around the suspected record to see if there were any other issues. To achieve this goal, run the following procedure, load the temporary table, and then select and apply application logic to the results:

```
SYS.DBMS_KAFKA.SEEK_OFFSET_TS (

'ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0',
                                          TO_DATE ('2020/07/04 02:30:00',
'YYYY/MM/DD HH:MI:SS',
                                          TO_DATE ('2020/07/04 03:30:00',
'YYYY/MM/DD HH:MI:SS'));

SYS.DBMS_KAFKA.LOAD_TEMP_TABLE

(ORA$DKVGTT_EXAMPLECLUSTER_SEEKABLEAPP_0);
SELECT <columns> FROM ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0;
--application logic
```

ADD_PARTITIONS

The `DBMS_KAFKA_ADM` procedure `ADD_PARTITIONS` adds additional Kafka partitions to an existing set of Oracle SQL access to Kafka (OSAK) views.

Syntax

```
PROCEDURE ADD_PARTITIONS (
                      cluster_name         IN  VARCHAR2,
                      application_name     IN  VARCHAR2
```

Parameters

Table 115-15 ADD_PARTITIONS procedure parameters for DBMS_KAFKA

Parameter	Description
cluster_name	Name of an existing Kafka cluster Case-insensitive.

Table 115-15 (Cont.) ADD_PARTITIONS procedure parameters for DBMS_KAFKA

Parameter	Description
application_name	Name of an existing application associated with the Kafka cluster. Case-insensitive.

Usage Notes

The semantics for `ADD_PARTITIONS` are similar to the `CREATE_xxx_APP` calls, except it preserves state information about existing Kafka topic partitions (for example, `committed_offset`), and binds new partitions either to either existing views, or to new OSAK views.

> **Caution:**
>
> Before calling `ADD_PARTITIONS`, all application instances for this application must be shut down. After `ADD_PARTITIONS` is called successfully, the application instances can be restarted. If the application is a streaming application configured to create one view instance per partition, then new views will be created. Using these views will require starting additional application instances, each dedicated to processing exclusively one of the new views created.

If no partitions have been added to the topic, then the procedure will terminate successfully without altering the existing views.

When an OSAK view is created, and the partitions are assigned, the view manages a list of partitions. For example, suppose an OSAK view was previously assigned to read partitions 0,1,2,3 in a Kafka topic. If `ADD_PARTITIONS` later adds a new partition to this view (partition 16, for example), then the OSAK view is now configured to fetch Kafka records from partitions 0,1,2,3, and 16.

This procedure runs both DDL and DML, which are transactional. This call should only be run outside of any existing transactional context.

Examples

```
BEGIN
  SYS.DBMS_KAFKA.ADD_PARTITIONS ('ExampleCluster',
                                                    'ExampleApp');
END;
/
```

DROP_ALL_APPS

The DBMS_KAFKA procedure DROP_ALL_APPS drops all applications for the Kafka cluster.Use this procedure to confirm that a connection can be established with the configured security information. The function returns the state of the cluster.

Syntax

```
PROCEDURE DROP_ALL_APPS (
                            \cluster_name    IN VARCHAR2
);
```

Parameters

Table 115-16 DROP_ALL_APPS Procedure Parameters for DBMS_KAFKA

Parameter	Description
cluster_name	Name of an existing Kafka cluster Case-insensitive.

Usage Notes

The DROP_ALL_APPS procedure runs both Data Definition Language (DDL) and Data Manipulation Language (DML) changes, which are transactional. Only run this call outside of any existing transactional context.

> **Note:**
>
> This is an autonomous transaction.

Examples

Suppose that one or more of the OSAK applications no longer exist in the Kafka cluster ExampleCluster. You can then drop all of the applications in that cluster by using the following statement:

```
EXEC SYS.DBMS_KAFKA.DROP_ALL_APPS ('ExampleCluster');
```

ENABLE_VIEW_QUERY

The DBMS_KAFKA procedure ENABLE_VIEW_QUERY sets a context within the current Oracle Database session, enabling an application to query a view. Use this procedure to enable an application to query an Oracle SQL access to Kafka (OSAK) view directly.

Syntax

```
PROCEDURE ENABLE_VIEW_QUERY(view_name IN VARCHAR2);
```

Parameters

Table 115-17 ENABLE_VIEW_QUERY Procedure Parameters for DBMS_KAFKA

Parameter	Description
view_name	Name of an existing Kafka view Case-insensitive.

Usage Notes

Only one query against the OSAK view should be done within a transaction. Multiple queries of an OSAK view cannot guarantee repeatable read behavior. Because Kafka is a streaming service, it is possible that new records can be published to a Kafka topic in between the `SELECT` queries. If new records are published, then this can result in additional rows being seen by the application.

> **Caution:**
>
> This is an advanced procedure. Oracle strongly recommends that this call is only used by developers who have a deep understanding of the Oracle SQL access to Kafka (OSAK) processing model and are knowledgeable about debugging Oracle access plans. This includes understanding how Kafka offsets are advanced by OSAK after an OSAK view is queried.

In general, Oracle recommends that you use `LOAD_TEMP_TABLE` or `EXECUTE_LOAD_TABLE`. These procedures query the Kafka views transparently, and load them a single time into either a global temporary table, or into an Oracle Database table for further analysis using SQL.

Examples

Enabling a user to query an OSAK view directly should only be used in two cases:

1. When a query is only referencing the OSAK view in a `FROM` clause to scan a Kafka stream. For example, suppose you want to access an OSAK view directly and retrieve the average temperature of a sensor in laboratory 1 since it was last checked. You can use the following query:

```
EXEC DBMS_KAFKA.ENABLE_VIEW_QUERY(

'ORA$DKV_EXAMPLECLUSTER_STREAMINGAPP_0');
SELECT AVG(temperature)
FROM ORA$DKV_EXAMPLECLUSTER_STREAMINGAPP_0WHERE sensor_name =
'LAB1_SENSOR';
EXEC DBMS_KAFKA.UPDATE_OFFSET(
                                'ORA$DKV_EXAMPLECLUSTER_STREAMINGAPP_0');
COMMIT;
```

2. When a query creates a simple table join between the OSAK view and an Oracle Database table, where the OSAK view is forced to be the outer table of a join by using the `ORDERED` hint in a query. This case ensures that the Kafka data is retrieved only once. For example:

```
SELECT /*+ ORDERED *\/ COUNT(*)
FROM thermostat_spec s, ora$dkv_thermostat_0 t
WHERE s.device_type = t.device_type
AND  t.current_setting >
 t.temperature_setting +
s.device_max_variation;
```

SET_TRACING

The `DBMS_KAFKA` procedure `SET_TRACING` enables or disables debug level tracing for the external table driver code associated with an Oracle SQL access to Kafka (OSAK) application.This function generates logging output to the trace file for the session.

Syntax

```
PROCEDURE SET_TRACING(
                        cluster_name        IN VARCHAR2,
                        application_name IN VARCHAR2,
                        enable                   IN BOOLEAN);
```

Parameters

Table 115-18 DROP_SEEKABLE_APP Procedure Parameters for DBMS_KAFKA

Parameter	Description
cluster_name	Name of an existing Kafka cluster Case-insensitive.
application_name	The name of an existing application using the Kafka cluster. Case-insensitive.
enable	`[true |false]` Default: `false`. Set `true` to enable the debug output, or `false` to disable the debug output

Usage Notes

Use to generate debug logging output to the trace file for the session. Set to `true` to enable the debug output, or `false` to disable the debug output.

> **Note:**
>
> The following event must already be enabled for the database:
>
> ```
> event="39431 trace name context forever, level 1" # Enable external
> table debug tracing
> ```

Examples

Suppose you want to start tracing on the Kafka cluster `ExampleCluster`, which was being used by the application `ExampleApp`. You can then use this procedure:

```
EXEC SYS.DBMS_KAFKA.SET_TRACING('KAFKACLUS1', 'KAFKACLUS1', True);
```

DBMS_KAFKA OPTIONS Passed to CREATE_xxx_APP

The options parameters for the `DBMS_KAFKA.CREATE_LOAD_APP`, `CREATE_STREAMING_APP`, and `CREATE_SEEKABLE_APP` packages are provided to the packages in a JSON document
The `DBMS_KAFKA` options can be used with each of the `DBMS_KAFKA` packages to create and define Oracle SQL access to Kafka (OSAK) applications.

> **Note:**
>
> 54 bytes are reserved for the maximum Kafka metadata chunk size in the buffer. For example, a bufsize of 1M can hold a record of 1M – 54 bytes. Bufsize defaults to 1000 KB with a max of 10,000 KB, so the largest Kafka record size supported is 10000 kb – 54.

Table 115-19 Options for DBMS_KAFKA

Option Name	Description
`avrodecimaltype`	Specifies the representation of a decimal stored in the byte array. Valid values: `int`, `integer`, `str`, `string` Default: If this parameter is not used, then an Avro decimal column is read assuming byte arrays store the numerical representation of the values (that is, default to `int`), which is as the Avro specification defines. Only allowed if the `fmt` option is specified as `AVRO`. Related access parameter: `com.oracle.bigdata.avro.decimaltpe`
`avroschema`	JSON document representing the schema of the Kafka value payload. The schema must define an Avro record type. There is no default value. A value must be specified if the `fmt` option is specified as `AVRO`

Table 115-19 (Cont.) Options for DBMS_KAFKA

Option Name	Description
`blankasnull`	When set to `true`, loads fields consisting of spaces as null. Values: `[true | false]` Default: `false` Only allowed if the `fmt` option is specified as `DSV` Related access parameter: `com.oracle.bigdata.blankasnull`
`bufsize`	Sets the buffer size in kilobytes for large record reads. Set this value if you need to read records that are greater than the default buffer size. Default (in kilobytes): 1000 Related access parameter: `com.oracle.bigdata.buffersize`
`conversionerrs`	If a row has data type conversion errors, then the related columns are stored as `null`, or if the row is rejected. Values: `[reject_record | store_null]` Default: `store_null` Only allowed if the `fmt` option is specified as `DSV` Related access parameter: `com.oracle.bigdata.conversionerrors`

`datefmt`	Specifies the date format in the source file. The value `auto` checks for the following formats.
	`J, MM-DD-YYYYBC, MM-DD-YYYY, YYYYMMDD HHMISS, YYMMDD HHMISS, YYYY.DDD, YYYY-MM-DD`
	Default: `yyyy-mm-dd hh24:mi:ss`
	Only allowed if the
	`fmt` option is specified as `DSV`
	Related access parameter: `com.oracle.bigdata.dateformat`

`escapedby`	Specifies the character used to escape any embedded field terminators or line terminators in the value for fields. The character value must be wrapped in single-quotes. For example: `'\'` If the parameter is not specified, then there is no value. Only allowed if `fmt` option is specified as `DSV` Related access parameter: `com.oracle.bigdata.csv.rowformat.fields.escapedby`
`fmt`	Format of the value payload of a Kafka record. Values: `[DSV | JSON | AVRO]` There is no default value. The format must be specified in the options passed. Related access parameter: `com.oracle.bigdata.kafka.format`
`jsondt`	Store JSON in `varchar2` (max size) or `clob` to support large JSON records. Values: `[varchar2 | clob]` Default: `varchar2` Only allowed if the `fmt` option is specified as JSON.

Table 115-19 (Cont.) Options for DBMS_KAFKA

Option Name	Description
nulldefinedas	Specifies the character used to indicate the value of a field is NULL. If the parameter is not specified, then there is no value. Only allowed if the fmt option is specified as DSV Related access parameter: com.oracle.bigdata.csv.rowformat.nulldefinedas
quote	Specifies the quote character for the fields. When specified, the characters defined as the quote characters are removed during loading. Valid values: character Default: Null, meaning no quote character Only allowed if the fmt option is specified as DSV Related access parameter: com.oracle.bigdata.quote
rejectlmt	The operation errors out after a specified number of rows (*number*, an integer) are rejected. This operation only applies when rejecting records due to conversion errors. To allow all records to be processed even if conversion errors exist, pass the value 'unlimited'. Valid values: [*number* | 'unlimited'] Default: 0, which means that no conversion errors are allowed.
removequotes	Removes any quotes that are around any field in the source file. Values: [true | false] Default: false Only allowed if the fmt option is specified as DSV Related access parameter: com.oracle.bigdata.removequotes
separator	Specifies the character used to separate the field values. The character value must be wrapped in single quotes. For example: '|' Default: ',' Related access parameter: com.oracle.bigdata.csv.rowformat.fields.terminator
terminator	Specifies the character used to separate the record values. The character value must be wrapped in single quotes. For example: '|' Default: '\n' Related access parameter: com.oracle.bigdata.csv.rowformat.lines.terminator
trimspaces	Specifies how the leading and trailing spaces of the fields are trimmed. Valid values: rtrim, ltrim, notrim, ltrim, ldrtrim Default: notrim Only allowed if the fmt option is specified as DSV Related access parameter: com.oracle.bigdata.trimspaces
truncatecol	If the data in the file is too long for a field, then this option truncates the value of the field rather than rejecting the row or setting the field to NULL. Values: [true | false] Default: false Only allowed if the fmt option is specified as DSV Related access parameter: com.oracle.bigdata.truncatecol

Table 115-19 (Cont.) Options for DBMS_KAFKA

Option Name	Description
tsfmt	Specifies the timestamp format in the source file. The value auto checks for the following formats: YYYY-MM-DD HH:MI:SS.FF, YYYY-MM-DD HH:MI:SS.FF3, MM/DD/YYYY HH:MI:SS.FF3 Default: yyyy-mm-dd hh24:mi:ss.ff Only allowed if the fmt option is specified as DSV Related access parameter: com.oracle.bigdata.timestampformat
tslzfmt	Specifies the timestamp with local timezone format in the source file. The value auto checks for the following formats: DD Mon YYYY HH:MI:SS.FF TZR, MM/DD/YYYY HH:MI:SS.FF TZR, YYYY-MM-DD HH:MI:SS+/-TZR, YYYY-MM-DD HH:MI:SS.FF3, DD.MM.YYYY HH:MI:SS TZR Default: yyyy-mm-dd hh24:mi:ss.ff Only allowed if the fmt option is specified as DSV Related access parameter: com.oracle.bigdata.timestampltzformat
tstzfmt	Specifies the timestamp with timezone format in the source file. The value auto checks for the following formats: DD Mon YYYY HH:MI:SS.FF TZR, MM/DD/YYYY HH:MI:SS.FF TZR, YYYY-MM-DD HH:MI:SS+/-TZR, YYYY-MM-DD HH:MI:SS.FF3, DD.MM.YYYY HH:MI:SS TZR Default: yyyy-mm-dd hh24:mi:ss.ff Only allowed if the fmt option is specified as DSV Related access parameter: com.oracle.bigdata.timestamptzformat

Example 115-3 AVRO Schema Record Type

The following is an example of an AVRO data schema. In this case, the type of AVRO data defined is a record, and the data is a record of sensor values for a temperature monitor as monitored over a set period of time.

```
{
  "type" : "record",
  "name" : "sensor_value",
  "namespace" : "example.sensor",
  "fields" : [ {
    "name" : "EventTime",
    "type" : "long",
    "logicalType" : "timestamp-millis"
  }, {
    "name" : "IotDeviceType",
    "type" : "int"
  }, {
    "name" : "IotDeviceUnitId",
    "type" : "int"
  }, {
    "name" : "TempSetting",
    "type" : "double"
```

```
  }, {
    "name" : "TempReading",
    "type" : "double"
  } ]
}
```