22

Oracle SQL Access to Kafka

Starting with Oracle Database 23ai, you can use Oracle SQL APIs to query Kafka topics dynamically using Oracle SQL.

Oracle SQL Access to Kafka integrates Kafka and OCI Streaming Service streams with Oracle Database 23ai in several important ways. First, it enables you to connect Oracle Database to one or more Kafka topics. After the database is connected, you can then query that topic dynamically using Oracle SQL, without persisting the Kafka data in Oracle Database. This feature enables you to analyze real time data in combination with data captured in your Oracle Database. In addition, Oracle SQL Access to Kafka enables fast, scalable and lossless loading of Kafka topics into Oracle Database. The `DBMS_KAFKA` APIs simplify the management of this entire process.

- About Oracle SQL Access to Kafka Version 2
  Oracle SQL access to Kafka (`OSaK`) provides a native feature of Oracle Database that enables Oracle SQL to query Kafka topics.
- Global Tables and Views for Oracle SQL Access to Kafka
  Learn about how Oracle SQL Access to Kafka accesses Kafka STREAMING, SEEKING, and LOAD applications, and the unique `ORA$` prefixes used with global temporary tables.
- Understanding how Oracle SQL Access to Kafka Queries are Performed
  Oracle SQL Access to Kafka accesses Kafka streaming data, but queries are performed on Oracle Database global temporary tables, which provides several advantages.
- Streaming Kafka Data Into Oracle Database
  Oracle SQL Access to Kafka enables Kafka streaming data to be processed with Oracle Database tables using standard SQL semantics.
- Querying Kafka Data Records by Timestamp
  Oracle SQL Access to Kafka in Seekable mode assists you to query older data stored in Kafka, based on timestamps associated with the Kafka data.
- About the Kafka Database Administrator Role
  To administer Oracle SQL access to Kafka, grant the Oracle Database role `OSAK_ADMIN_ROLE` and grant required administration privileges to the administrator role and the Kafka administration API package.
- Enable Kafka Database Access to Users
  The application user accounts are granted the `DBMS_KAFKA` database privileges required to access OSAK.
- Data Formats Supported with Oracle SQL Access to Kafka
  Oracle SQL access to Kafka supports Kafka records represented in three formats: delimited text data (for example, csv), JSON, and Avro
- Configuring Access to a Kafka Cluster
  You can configure access to secured Kafka clusters, or non-secured Kafka clusters
- Creating Oracle SQL Access to Kafka Applications
  To create an application to access Apache Cluster data, create the type of application that you require.

- Security for Kafka Cluster Connections
  Oracle SQL Access to Kafka supports access to Kafka and Oracle Streaming Service (OSS), using various security mechanisms, such as SSL, SASL, and Kerberos.
- Configuring Access to Unsecured Kafka Clusters
  To configure access to non-secure Kafka clusters, the OSAK administrator (Oracle Database user with osak_admin_role) must complete this procedure.
- Configuring Access to Secure Kafka Clusters
  To configure access to secure Kafka clusters use this procedure.
- Administering Oracle SQL Access to Kafka Clusters
  See how to update, temporarily disable, and delete Kafka cluster definitions with Oracle SQL access to Kafka
- Guidelines for Using Kafka Data with Oracle SQL Access to Kafka
  Review guidelines, restrictions, and recommendations as part of your application development plan.
- Choosing a Kafka Cluster Access Mode for Applications
  To use Oracle SQL access to Kafka, decide what mode of data access you require for your applications.
- Creating Oracle SQL Access to Kafka Applications
  To query Kafka data in a LOAD application, load Kafka data into an Oracle Database table using these procedures.
- Using Kafka Cluster Access for Applications
  Learn how to use Kafka cluster data access with your applications.

22.1 About Oracle SQL Access to Kafka Version 2

Oracle SQL access to Kafka (`OSaK`) provides a native feature of Oracle Database that enables Oracle SQL to query Kafka topics.

Starting with Oracle Database 23ai, version 2 of Oracle SQL access to Kafka is installed with Oracle Database. It provides a native Oracle Database connector service to Kafka clusters. It consists of a set of features accessed through the `DBMS_KAFKA` and `DBMS_KAFKA_ADM` packages.

What it does

Oracle SQL Access to Kafka Version 2 enables Kafka streaming data to be processed with Oracle Database tables using standard Oracle Database SQL semantics, and enables data to be processed by standard Oracle application logic (for example, Oracle JDBC applications). Oracle SQL access to Kafka is integrated in Oracle Database. This integration of Kafka access in Oracle Database enables you to relate tables in Oracle Database using data streams produced by Kafka or an OCI Streaming Service, without requiring an external client connector application. Oracle SQL access to Kafka can scale up data streams for Oracle Database in the same fashion as Kafka applications.

Oracle SQL access to Kafka enables you to do the following:

- Create and use a streaming application to process unread Kafka records one time, where these records do not need to be retained after they are processed.
- Create and use a loading application to capture unread Kafka records permanently in an Oracle Database table, for access by various Oracle applications. In this case, Kafka records are captured and persisted in user tables in Oracle Database. This use case is helpful for data warehouses.

- Create and use a seeking application to reread records that are in a Kafka topic, based on a user-supplied timestamp interval.
- Create and use two or more streaming applications. These applications can be used to stream data from two or more Kafka topics, where you can then join them using SQL in Oracle Database.

How It Works

Oracle SQL access to Kafka version 2 provides access to Kafka data using Oracle Database system-generated views, and external tables. These views and external tables use the DBMS_KAFKA package to define a named Oracle SQL access to Kafka application. In general, these views and external tables are transparent for streaming, loading, and seeking applications.

Your application can perform and control operations as an Oracle Database transaction, complying with the ACID (Atomicity, Consistency, Isolation, Durability) requirements for the database, ensuring that either all parts of the transaction are committed, or all rolled back, with a unique identifier (a **transaction ID**) for each transaction. This transaction ID includes timestamps that you can use to identify and roll back errors. The ACID feature of Oracle Database transactions provides support for data recovery in case of a failure, without losing or repeating records.

The Oracle transaction performed with Oracle SQL access to Kafka includes managing the Kafka partition offsets, and committing them to database metadata tables in Oracle Database.

Without Oracle SQL Access to Kafka, the Kafka partition offsets need to be managed either by the application, or by Kafka, neither of which support transaction semantics. This means that after a system failure, Kafka records can be lost or reprocessed by an application. Managing offsets in an Oracle Database transaction avoids these problems, and enhances the isolation and durability of the Kafka data.

Because Oracle SQL Access to Kafka is available with Oracle Database, and is used with PL/SQL and SQL queries, no external client application is required to provide a connector to Oracle Database.

The `ORA_KAFKA` PL/SQL package has functions and procedures to register a Kafka cluster in a database schema, query Kafka topics, query data from specified offsets or specified timestamps, and more. You can choose either to use global temporary tables without storing the data, or store the data into user tables in the target Oracle Database.

How you can use it

You can use Oracle SQL access to Kafka application to access global temporary tables or user tables created in Oracle Database, so that your application can obtain data. That data can be streams of data, or snapshots of the data from other databases, which can be accessed directly, or loaded into Oracle Database tables and be used within your application.

Kafka global temporary tables have the following characteristics:

- The global temporary table is loaded once at the outset of an application instance, and used as a snapshot of Kafka records for the duration of the application instance. The application can use standard Oracle SQL with the global temporary table.
- Each query from a global temporary table results in a trip to the Kafka cluster, re-retrieving the same rows, and perhaps additional rows.

The corresponding global temporary table receives a snapshot from an Oracle SQL access to Kafka view. Applications use this temporary table for one or more queries within a transaction: a global temp table is loaded once, and used. The Kafka offsets are advanced, and then the

app commits, indicating that it is finishee\d with the Kafka records loaded in the global temporary table.

Reading from the temporary table is beneficial for the following reasons:

- Repeatable reads are supported, either explicitly from multiple queries or implicitly within a join
- Reliable statistics are gathered for the query optimizer
- Only one trip is made to Kafka when loading the temporary table. Subsequent queries do not result in a trip to the Kafka cluster.
- Global temporary tables can be joined with standard Oracle tables. Joining Oracle SQL access to Kafka temporary tables with Oracle Database tables increases your ability to use Oracle Database capabilities with Kafka data.
- You can leverage the mature optimization and processing strategies in Oracle Database to minimize code paths needed to join tables efficiently.

22.2 Global Tables and Views for Oracle SQL Access to Kafka

Learn about how Oracle SQL Access to Kafka accesses Kafka STREAMING, SEEKING, and LOAD applications, and the unique `ORA$` prefixes used with global temporary tables.

Applications using Oracle SQL Access to Kafka (OSAK) for `STREAMING` and `SEEKING` of Kafka topics use PL/SQL to call an OSAK procedure to load global temporary tables with the results of a query from the corresponding Oracle SQL Access to Kafka view. LOAD applications do not require global temporary tables, because the `LOAD` application performs incremental loads into an existing Oracle Database table using the `EXECUTE_LOAD_APP` procedure. For STREAMING, SEEKING and LOAD applications, OSAK creates the views and external tables in all three cases.

Both Oracle SQL Access to Kafka views and temporary tables have unique `ORA$` prefixes that identify them as objects created by Oracle SQL Access to Kafka.

`ORA$DKV` (for views) and `ORA$DKX` (for tables) are prefixes for Oracle SQL access to Kafka generated views and external tables that serve calls to `DBMS_KAFKA` to load data from Kafka into a user-owned table or into a global temporary table. Typically, these views and external tables are treated as internal objects, which are not directly manipulated by an Oracle application.

`ORA$DKVGTT` is a prefix that designates that it is a global temporary table that is loaded from a streaming or seeking app. This global temporary table is loaded transparently when calling `DBMS_KAFKA.LOAD_TEMP_TABLE`.

22.3 Understanding how Oracle SQL Access to Kafka Queries are Performed

Oracle SQL Access to Kafka accesses Kafka streaming data, but queries are performed on Oracle Database global temporary tables, which provides several advantages.

A typical application does not query Oracle SQL Access to Kafka views directly. Instead:

- Each query from an Oracle SQL Access to Kafka view fetches data directly from Kafka from the current offset to the current high water mark. Because rows are continually being added, each query from a view will likely retrieve more rows. Therefore,Oracle SQL Access to Kafka views do not support repeatable reads, either explicitly from multiple queries or implicitly within a join.

- There are no reliable statistics gathered from Oracle SQL Access to Kafka views for the query optimizer
- Each query from an Oracle SQL Access to Kafka view results in a trip to the Kafka cluster, re-retrieving the same rows and perhaps additional rows. These query retrievals can affect performance.

The corresponding temporary table receives a snapshot from an Oracle SQL Access to Kafka view. Applications use this temporary table for one or more queries within a transaction. Reading from the temporary table is beneficial for the following reasons:

- Repeatable reads are supported, either explicitly from multiple queries or implicitly within a join
- Reliable statistics are gathered for the query optimizer
- Only one read is made to Kafka when loading the temporary table. Subsequent queries do not require returning to the Kafka cluster to access the data.

The global temporary tables can be joined with standard Oracle tables (for example, Oracle customer relationship management (CRM) tables.

By joining Oracle SQL access to Kafka temporary tables with Oracle Database tables, you obtain the following advantages:

- Leveraging the mature optimization and execution strategies in Oracle Database to minimize code path required to join tables efficiently
- Obtaining Oracle Database transaction semantics, with the security of Oracle Database ACID transaction processing (atomicity, consistency, isolation, and durability), ensuring that all changes to data are performed as if they are a single operation, controlled by the application
- Managing the Kafka partition offsets and committing them to database metadata tables in Oracle Database, so that after a system failure, these Oracle Database transactions with Kafka records are not subject to being lost or reprocessed by an application.

22.4 Streaming Kafka Data Into Oracle Database

Oracle SQL Access to Kafka enables Kafka streaming data to be processed with Oracle Database tables using standard SQL semantics.

Apache Kafka is commonly used to capture and consolidate data from many streaming sources, so that analytics can be performed on this data. Typically, this requires loading of all the Kafka records into the database, and then combining the data with database tables for analytics, either for short-term study or for longer analysis.

With Oracle SQL access to Kafka, you can use standard SQL, PL/SQL and other database development tools to accomplish the load from Kafka to an Oracle Database, and process that data using standard Oracle application logic, such as JDBC applications. Oracle SQL access to Kafka can create a view that maps to all partitions of the Kafka topic that you want to load. Each Oracle SQL access to Kafka call to load more data queries this view, which in turn queries all partitions of the Kafka topic from the previous point last read to the current data **high watermark offset** (the offset of the last message that was fully inserted to all Kafka partitions). Data retrieved from the Kafka partitions is loaded into a temporary Oracle Database table.

These Oracle SQL Access to Kafka views behave much like a Kafka application instance. They read records from Kafka starting at a given offset until it reaches the high watermark offset

When Oracle SQL Access to Kafka creates a view, it also creates a corresponding global temporary table. The application calls an Oracle SQL Access to Kafka PL/SQL procedure to load this global temporary table with the results of a query from the corresponding Oracle SQL Access to Kafka view.

The global temporary tables can be joined with standard Oracle tables (for example, Oracle customer relationship management (CRM) tables.

By joining Oracle SQL access to Kafka temporary tables with Oracle Database tables, you obtain the following advantages:

- Leveraging the mature optimization and execution strategies in Oracle Database to minimize code path required to join tables efficiently
- Obtaining Oracle Database transaction semantics, with the security of Oracle Database ACID transaction processing (atomicity, consistency, isolation, and durability), ensuring that all changes to data are performed as if they are a single operation, controlled by the application
- Managing the Kafka partition offsets and committing them to database metadata tables in Oracle Database, so that after a system failure, these Oracle Database transactions with Kafka records are not subject to being lost or reprocessed by an application.

22.5 Querying Kafka Data Records by Timestamp

Oracle SQL Access to Kafka in Seekable mode assists you to query older data stored in Kafka, based on timestamps associated with the Kafka data.

In the event of anomalies, you can use Oracle SQL access to Kafka to assist with identifying Kafka data associated with the anomaly in a specified window of time.

For example, suppose a computer company has multiple sites. Each site has labs, and all access to the building and labs are protected by key card access. The company has a vast array of employees, some who just need office space, some who maintain the machines in the labs, and some who monitor the building for issues such as ventilation issues, unpermitted access, and general usages of the sites. In this scenario, Kafka topics can consist of the following:

- Key card usage (`KCdata`)
- Facility monitoring (`Fdata`)
- System monitoring, such as uptime, access, intrusion detection (`Sdata`)

If an usual event is detected while reading through Kafka data and combining it with Oracle data, the application can log the anomaly along with the timestamp of the record containing the unusual event. A second application can then read through these errors and process them. For each unusual event, the application might seek to a window of timestamps 10 seconds before and after the event. This is similar to analyzing exceptions in log files. It is common to look at log entries before and after the event to see if the exception was caused by an earlier issue, or if the exception led to downstream problems.

To evaluate a site issue, you can load the key card readers data (`KCdata`) to a permanent table. For example, if multiple applications use this data, then it would make sense to load that date into an Oracle Database table that can be used by multiple applications, to assist the real estate team to track building and office usage. The IT department uses the data to determine who is on site to handle issues.

Using a Streaming query, you can scan the facility data (`Fdata`) to determine if there are any atyplical or unusual events in the data. This could be a spike in lab temperature, a door that did

not close and is raising an alarm, the fire detection system sounding an alarm, or other data points associated with the timeframe, suhc as a door that was left ajar.

The security team is given an alert of a door that did not close. They use the streaming data to determine the door was left ajar at 3:17 AM. They can then use a Seeking query to seek multiple other data points (KCdata, Fdata, Sdata) in a 30 minute window (3:02 to 3:32) to determine who accessed the building, what doors or labs were accessed, what machines went offline or were directly accessed, and other data records, so that they can take the proper response to the developing situation.

In this scenario, you can use Oracle SQL Access to Kafka to create a single view that maps to all partitions of the Kafka topic. When Oracle SQL access to Kafka creates a view, it also creates a corresponding global temporary table. The application first specifies a starting and ending timestamp and then calls Oracle SQL access to Kafka to load the global temporary table with the rows in the specified window of time. You can leverage standard Oracle Database SQL transaction processing to parse large volumes of data to identify relevant device data within the anomalous event.

22.6 About the Kafka Database Administrator Role

To administer Oracle SQL access to Kafka, grant the Oracle Database role `OSAK_ADMIN_ROLE` and grant required administration privileges to the administrator role and the Kafka administration API package.

To provide role-based authentication to grant the Oracle SQL access for Kafka administration privileges to an administrative user, Oracle provides the `OSAK_ADMIN_ROLE` starting with Oracle Database 23ai. You can grant this role to an administrator user for Oracle SQL Access to Kafka. This role grants the system privileges required for users that you designate as Oracle SQL access for Kafka administrators to configure, register, and manage Kafka clusters. The system privileges granted by this role are as follows:

- CREATE CREDENTIAL, to create a Kafka SASL-SSL (Simple Authentication and Security Layer) password or OSS (Oracle Streaming Service) authToken
- CREATE ANY DIRECTORY, to create cluster access and cluster configuration directory
- DROP ANY DIRECTORY, to drop cluster access and cluster configuration directory
- READ privileges to `sys.dbms_kafka_clusters`
- READ privileges to `sys.dbms_kafka_applications`
- READ privileges to `sys.dbms_kafka_messages`

22.7 Enable Kafka Database Access to Users

The application user accounts are granted the `DBMS_KAFKA` database privileges required to access OSAK.

As a DBA, you create and grant users privileges to administer and use Oracle SQL access to Kafka. There are two categories of users:

- Oracle SQL Access to Kafka administrators are privileged users. To simplify management of Oracle SQL access to Kafka, Oracle recommends that the Oracle DBA grant the `OSAK_ADMIN_ROLE` to designated Kafka administrators. This role is precreated in the database starting with Oracle Database 23ai.

  Administrators run the `DBMS_KAFKA_ADM` package methods to configure and manage the Kafka cluster information. Either users granted `OSAK_ADMIN_ROLE` or the Oracle DBA can

create the operating system level cluster configuration directory, and populate that directory with configuration files. Oracle SQL Access to Kafka administrators create the Oracle directory object for the Kafka cluster configuration and access directories.

- Application users of Kafka topic data are granted the `READ` privileges required to access to the `DBMS_KAFKA` packages, so that they can access and use data accessed from Kafka cluster topics.

Example 22-1 Grant OSAK_ADMIN_ROLE to Kafka Administrator Users

In this example, the `OSAK_ADMIN_ROLE` is granted to user `kafka-admin`:

```
GRANT OSAK_ADMIN_ROLE
   TO kafka-admin;
```

Example 22-2 Grant User Access to Kafka Users

To enable applications to use Oracle SQL access to Kafka, you grant `DBMS_KAFKA` access. These application users must already have the following privileges on the source Kafka cluster and target Oracle Database:

- `CREATE SESSION`
- `CREATE TABLE`
- `CREATE VIEW`
- Available quota on the tablespace where they access Kafka data
- Read access on the cluster access directory of a registered Kafka cluster

22.8 Data Formats Supported with Oracle SQL Access to Kafka

Oracle SQL access to Kafka supports Kafka records represented in three formats: delimited text data (for example, csv), JSON, and Avro

Kafka is without schemas, and format-neutral. Application data is stored as opaque byte arrays in the key and value fields of a Kafka record. Because the Kafka key is used mainly for hashing data into Kafka partitions, only the value field of a Kafka record is retrieved and rendered as Oracle rows. The application is responsible for serialization and deserialization of the data and for supplying a schema that defines the structure of the data format. In Oracle SQL Access for Kafka, the data format and schema are specified in the options argument to the `DBMS_KAFKA.CREATE_[LOAD|STREAMING|SEEKABLE]_APP()` procedures.

> **Note:**
>
> Regardless of the format type, the tables and views created contain three additional columns: `KAFKA_PARTITION`, `KAFKA_OFFSET`, and `KAFKA_EPOCH_TIMESTAMP`.

- JSON Format and Oracle SQL Access to Kafka
  For JSON, Oracle SQL access to Kafka determines the columns for the table or view.
- Delimited Text Format and Oracle SQL Access to Kafka
  For delimited text formats, Oracle SQL access to Kafka creates views and temporary tables in the user schema with Kafka data.

- Avro Formats and Oracle SQL Access to Kafka
  For Avro formats, Oracle SQL access to Kafka uses the Avro schema to determine the data columns and the three metadata columns.

22.8.1 JSON Format and Oracle SQL Access to Kafka

For JSON, Oracle SQL access to Kafka determines the columns for the table or view.

The following is an example of using options to display data for a JSON streaming application:

```
DECLARE
 v_options VARCHAR2;
BEGIN
 v_options := '{"fmt" : "JSON"}';
 SYS.DBMS_KAFKA.CREATE_STREAMING_APP (
                'ALPHA1',
                'MYAPP',
                'ExampleTopic',
                v_options);
END;
/
```

With Javascript Object Notation (JSON) data, Oracle SQL Access to Kafka creates views and global temporary tables in the user schema over Kafka data. These views are prefixed by `ORA$DKV_` The temporary tables are prefixed by `ORA$DKVGTT_`. The package `DBMS_KAFKA.CREATE_xxx_APP` uses a fixed schema to return JSON data from a Kafka record.

For example:

```
SQL> describe ORA$DKVGTT_ALPHA1_MYAPP_0;
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 KAFKA_PARTITION                                    NUMBER(38)
 KAFKA_OFFSET                                       NUMBER(38)
 KAFKA_EPOCH_TIMESTAMP                              NUMBER(38)
 VALUE                                              VARCHAR2(4000)
```

With the `VARCHAR2` type, the length of the `VALUE` column is restricted by the maximum `varchar2` length of your database. Note that the `VALUE` column has the option to be of type `CLOB`.

The `KAFKA_` columns identify the partition id, the offset, and the timestamp of the Kafka record. (The underlying timestamp representation is an integer representing the number of milliseconds since Unix epoch.)

The data in the value portion of the Kafka record is returned as text to the *VALUE* column. The character encoding of the external text is fixed as `AL32UTF8`. Oracle SQL access to Kafka logic does not check for valid JSON syntax in the `VALUE` columns. However, faulty JSON is discovered when JSON operators in a SQL query attempt to parse the `VALUE` data.

22.8.2 Delimited Text Format and Oracle SQL Access to Kafka

For delimited text formats, Oracle SQL access to Kafka creates views and temporary tables in the user schema with Kafka data.

With delimited data, such as CSV or comma-delimited data, Oracle SQL Access to Kafka creates views and global temporary tables in the user schema over Kafka data. These views are prefixed by `ORA$DKV_`. The temporary tables are prefixed by `ORA$DKVGTT_`. With DSV format, the data columns are based on the reference table passed in the options plus the three metadata columns

The temporary tables and views created with Oracle SQL access to Kafka delimited text format data have columns that reflect the shape of the delimited text data in the value field of a Kafka record. Oracle SQL access to Kafka converts text data into the native Oracle datatypes expressed in the table and view definition. The character encoding of the external text is fixed as `AL32UTF8`.

When a Kafka record is retrieved, a canonical layout is created, starting with the Kafka partition identifier (`INTEGER`), Kafka record offset (`INTEGER`), and Kafka record timestamp (`INTEGER`), followed by delimited text data in the Kafka value. In other words, the Kafka data is flattened out and streamed as rows of pure delimited text fields, using the order of the view schema definition.

The following Oracle data types are supported:

- `INTEGER`, `INT`, `NUMBER`
- `CHAR`, `VARCHAR2`
- `NCHAR`, `NVARCHAR2`
- `CLOB`, `NCLOB`, `BLOB`
- `FLOAT`, `BINARY_FLOAT`, `BINARY_DOUBLE`
- `TIMESTAMP`, `DATE`
- `TIMESTAMP WITH TIME ZONE`, `TIMESTAMP WITH LOCAL TIME ZONE`
- `INTERVAL`
- `RAW`
- `BOOLEAN`

To simplify the specification of delimited text at application creation time, you provide the name of a table that describes the columns of the user data in the order that they are physically ordered in the Kafka record value field. Oracle SQL Access to Kafka uses that name in views and temporary tables.

The following example shows the shape of the delimited text data table (a reference table, or **reftable**) provided when you create an Oracle SQL Access to Kafka application. Again, the Kafka value field reflects the identical physical order and the desired data type conversion from the delimited text.

You should preserve reftables after they are used for a `CREATE_xxx_APP` call to create Oracle SQL Access to Kafka views and temporary tables reflecting the shape. You will require the reftable to recreate views.

```
SQL> describe FIVDTI_SHAPE;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 F1                                                 NUMBER
 I2                                                 NUMBER
 V3                                                 VARCHAR2(50)
 D4                                                 DATE
```

```
 T5                                                TIMESTAMP(6)
 V6                                                VARCHAR2(200)
 I7                                                NUMBER
```

The reference table describes the fields in the Kafka record value only. For example, the reftable `FIVDTI_SHAPE` could support Kafka records where F1, I2, V3, D4, T5, V6, I7 are fields in the Kafka record value. The fields in the Kafka record value must be separated by delimiters (for example, comma delimiters).

> **Note:**
>
> The reference table cannot include invisible (hidden) columns. The ordering of the columns must match the order of the data values from the Kafka record. An invisible column has a `COLUMN_ID` of `NULL`, so its position in the column list cannot be determined.

Oracle SQL Access to Kafka temporary tables created for data described by the `FIVDTI_SHAPE` table will have the following schema:

```
SQL> describe ORA$DKVGTT_ALPHA1_MYAPP__0;
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 KAFKA_PARTITION                                    NUMBER(38)
 KAFKA_OFFSET                                       NUMBER(38)
 KAFKA_EPOCH_TIMESTAMP                              NUMBER(38)
 F1                                                 NUMBER
 I2                                                 NUMBER
 V3                                                 VARCHAR2(50)
 D4                                                 DATE
 T5                                                 TIMESTAMP(6)
 V6                                                 VARCHAR2(200)
 I7                                                 NUMBER
```

22.8.3 Avro Formats and Oracle SQL Access to Kafka

For Avro formats, Oracle SQL access to Kafka uses the Avro schema to determine the data columns and the three metadata columns.

- About Using Avro Format with Oracle SQL Access to Kafka
  Learn how Oracle SQL access to Kafka makes Kafka data in the Avro format available for use in Oracle Database tables and views.
- Primitive Avro Types Supported with Oracle SQL Access to Kafka
  To use Apache Avro Schema primitive type names in the database, Oracle converts these types to SQL data types.
- Complex Avro Types Supported with Oracle SQL Access to Kafka
  To use Apache Avro Schema complex type names in the database, Oracle converts these types to supported SQL data types.
- Avro Logical Types Supported with Oracle SQL Access to Kafka
  To use Apache Avro Schema logical type names in the database, Oracle converts these types to supported SQL data types.

22.8.3.1 About Using Avro Format with Oracle SQL Access to Kafka

Learn how Oracle SQL access to Kafka makes Kafka data in the Avro format available for use in Oracle Database tables and views.

To enable the use of the Apache Avro formatted data by applications in Oracle Database table and views, Oracle SQL Access for Kafka converts the data format based on the schema specified in the options argument to the `DBMS_KAFKA.CREATE_[LOAD|STREAMING|SEEKABLE]_APP()` procedures.

An Apache Avro record is an ordered list of named fields and types. The schema for a record defines the structure of the data and how it can be read. The Avro schema must be passed when the Oracle SQL access to Kafka application is created. This means that an Oracle SQL access to Kafka application can only support a single Avro schema for a Kafka topic. It is not supported to use more than one schema type in the topic stream. If the schema evolves, then you must create a new Oracle SQL access to Kafka application. Oracle SQL access to Kafka does not support the Confluent Schema Registry. If Kafka records in Avro format include a Confluent header, then that header is stripped off and ignored by Oracle SQL access to Kafka.

Kafka is without schemas, and format-neutral. Application data is stored as opaque byte arrays in the key and value fields of an Apache Avro record. Because the Kafka key is used mainly for hashing data into Kafka partitions, only the value field of an Apache Avro record is retrieved and rendered in Oracle Database tables, as Oracle rows. The application is responsible for serialization and deserialization of the data, and for supplying a schema that defines the structure of the data format.

You can use both primitive and complex Avro types with Oracle SQL access to Kafka, but you can use only one type for each application.

22.8.3.2 Primitive Avro Types Supported with Oracle SQL Access to Kafka

To use Apache Avro Schema primitive type names in the database, Oracle converts these types to SQL data types.

Table 22-1 Avro Primitive types and Oracle Type Conversions for Oracle SQL Access to Kafka

Type Description	Avro Primitive Type	Oracle Type
null/no value	`null`	`VARCHAR2(1)`
(not applicable)	`boolean`	`NUMBER(1)`
32-bit signed integer	`int`	`INTEGER`
64-bit signed integer	`long`	`INTEGER`
IEEE 32-bit floating point	`float`	`BINARY_FLOAT`
IEEE 64-bit floating point	`double`	`BINARY_DOUBLE`
byte array/binary	`bytes`	`BLOB`
UTF-8 encoded character string	`string`	`VARCHAR2`

The following example Avro schema defines a record that uses all Avro primitive types:

```
{
 "type" : "record",
```

```
 "name" : "primitives",
 "fields" : [
 { "name" : "f_null", "type" : "null" },
 { "name" : "f_boolean", "type" : "boolean" },
 { "name" : "f_int", "type" : "int"}
 { "name" : "f_long", "type" : "long" },
 { "name" : "f_float", "type" : "float" },
 { "name" : "f_double", "type" : "double" },
 { "name" : "f_bytes", "type" : "bytes" },
 { "name" : "f_string", "type" : "string"}
 ]
}
```

If you created Oracle SQL access to Kafka temporary tables for Avro data by using this example Avro schema, then the temporary tables have the following schema:

```
describe ORA$DKVGTT_ALPHA1_MYAPP__0;
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 KAFKA_PARTITION                                    NUMBER(38)
 KAFKA_OFFSET                                       NUMBER(38)
 KAFKA_EPOCH_TIMESTAMP                              NUMBER(38)
 F_NULL                                             CHAR(1)
 F_BOOLEAN                                          NUMBER(1)
 F_INT                                              NUMBER(38)
 F_LONG                                             NUMBER(38)
 F_FLOAT                                            BINARY_FLOAT
 F_DOUBLE                                           BINARY_DOUBLE
 F_BYTES                                            BLOB
 F_STRING                                           VARCHAR2(4000)
```

The `VARCHAR2` type length (in this example, for the `F_STRING` column) is determined by the maximum `varchar2` length of your database.

22.8.3.3 Complex Avro Types Supported with Oracle SQL Access to Kafka

To use Apache Avro Schema complex type names in the database, Oracle converts these types to supported SQL data types.

Description

The Apache Avro complex data types take specified attributes. To use the Avro complex types, Oracle SQL access to Kafka convertes them to Oracle types, as specified in the following table.

Table 22-2 Avro Complex types and Oracle Type Conversions for Oracle SQL Access to Kafka

Avro Complex Type	Oracle Type	Type Description
`fixed`	`BLOB`	A fixed type is used to declare a fixed-length field that can be used for storing binary data. It has two required attributes: the field's name, and the size in 1-byte quantities.
`enum`	`VARCHAR2`	An Avro enum field. Avro enums are enumerated types. They consist of are JSON strings with the type name `enum`, taking the name of the enum, and can take additional optional attributes.
`record`	`VARCHAR2`	Struct field. The struct field corresponds to a field in the input Avro records. A record represents an encapsulation of attributes that, all combined, describe a single thing.
`map`	`VARCHAR2`	A map is an associative array, or dictionary, that organizes data as key-value pairs. The key for an Avro map must be a string. Avro maps supports only one attribute: values. This attribute is required and it defines the type for the value portion of the map. Values can be of any type.
`array`	`VARCHAR2`	An array of any type The array type defines an array field. It only supports the items attribute, which is required. The items attribute identifies the type of the items in the array.

Note:

The Avro complex types `record`, `map`, and `array` are converted to a JSON format string before conversion to a `VARCHAR2` type.

The following example Avro schema defines a record that uses all Avro complex types:

```
{
 "type" : "record",
 "name" : "complex",
 "fields" : [
  { "name" : "f_fixed",
```

```
    "type" : { "type" : "fixed", "name" : "ten", "size" : 10}
  },
  { "name" : "f_enum",
    "type" : { "type" : "enum", "name" : "colors",
               "symbols" : [ "red", "green", "blue" ]   }
  },
  { "name" : "f_record",
    "type" : {"type" : "record", "name" : "person",
              "fields" : [ { "name" : "first_name", "type" : "string" },
                           { "name" : "last_name",  "type" : "string"} ] }
  },
  { "name" : "f_map",
    "type" : { "type" : "map", "values" : "int" }
  },
  { "name" : "f_array",
    "type" : {"type" : "array", "items" : "string"  }
  }]
}
```

If you created Oracle SQL access to Kafka temporary tables for Avro data by using this example Avro schema, then the temporary tables have the following schema:

```
describe ORA$DKVGTT_ALPHA1_MYAPP__0;
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 KAFKA_PARTITION                                    NUMBER(38)
 KAFKA_OFFSET                                       NUMBER(38)
 KAFKA_EPOCH_TIMESTAMP                              NUMBER(38)
 F_FIXED                                            BLOB
 F_ENUM                                             VARCHAR2(4000)
 F_RECORD                                           VARCHAR2(4000)
 F_MAP                                              VARCHAR2(4000)
 F_ARRAY                                            VARCHAR2(4000)
```

The `VARCHAR2` type length (in this example, for the `F_ENUM`, `F_RECORD`, `F_MAP` and `F_ARRAY` columns) is determined by the maximum `varchar2` length of your database.

22.8.3.4 Avro Logical Types Supported with Oracle SQL Access to Kafka

To use Apache Avro Schema logical type names in the database, Oracle converts these types to supported SQL data types.

Description

An Avro logical type is an Avro primitive or complex type with extra attributes to represent a derived type. Logical types are converted to Oracle types as specified in the following table.

Table 22-3 Avro Complex types and Oracle Type Conversions for Oracle SQL Access to Kafka

Type Description	Avro Logical Type	Oracle Type
decimal: arbitrary-precision signed decimal number of the form unscaled × 10^{-scale}	decimal (bytes, fixed)	`NUMBER`
UUIDs (Universally Unique Identifiers), also known as GUIDS (Globally Unique Identifiers): These IDs are randomly generated, in conformity with RFC-4122.	UUID (string)	Not supported.
date A date within the calendar, with no reference to a particular time zone or time of day Number of days from the Unix epoch, 1 January 1970	date (int)	`DATE`
time (millis): A time of day, with no reference to a particular calendar, time zone or date, represented as number of milliseconds after midnight: 00:00:00.000	time-millis (int)	`TIMESTAMP`
time (micros): A time of day, with no reference to a particular calendar, time zone or date number of microseconds after midnight: 00:00:00.000000	time-micros (long)	`TIMESTAMP`
timestamp (millis) UTC: An instant on the global timeline, independent of a particular time zone or calendar number of milliseconds from the Unix epoch, 1 January 1970: 00:00:00.000 UTC	timestampmillis (long)	`TIMESTAMP`
timestamp (micros) UTC: An instant on the global timeline, independent of a particular time zone or calendar number of microseconds from the Unix epoch, 1 January 1970: 00:00:00.000000 UTC	timestampmicros (long)	`TIMESTAMP`
duration An amount of time defined by a number of months, days and milliseconds.	fixed (size:12)	Not supported.

> **Note:**
>
> Decimal types, which are used with the logical types time-millis, time-macros, timestampmillis and timestampmicros, are internally stored as byte arrays (fixed or not). Depending on the Avro writer, some of these arrays store the string representation of the decimal, while others store the unscaled value. To avoid presenting ambiguous data, Oracle recommends that you use the option `avrodecimaltype` to declare explicitly which representation is used. If this option is not explicitly specified, then the default option for Oracle SQL access to Kafka is that the unscaled representation of the data is stored in the decimal columns of the file.

The following example Avro schema defines a record that uses all Avro logical types:

```
{
  "type" : "record",
  "name" : "logical",
  "fields" : [ {
  "name" : "f_decimal",
  "type" : {
      "type" : "bytes",
      "logicalType" : "decimal",
      "precision" : 4,
      "scale" : 2
    }
  }, {
  "name" : "f_date",
  "type" : {
      "type" : "int",
      "logicalType" : "date"
    }
  }, {
  "name" : "f_time_millis",
  "type" : {
      "type" : "int",
      "logicalType" : "time-millis"
    }
  }, {
  "name" : "f_time_micros",
  "type" : {
      "type" : "long",
      "logicalType" : "time-micros"
    }
  }, {
  "name" : "f_timestamp_millis",
  "type" : {
      "type" : "long",
      "logicalType" : "timestamp-millis"
    }
  }, {
  "name" : "f_timestamp_micros",
  "type" : {
      "type" : "long",
      "logicalType" : "timestamp-micros"
    }
```

```
  } ]
}
```

If you created Oracle SQL access to Kafka temporary tables for Avro data by using this example Avro schema, then the temporary tables have the following schema:

```
describe ORA$DKVGTT_ALPHA1_MYAPP__0;
 Name                                      Null?    Type
 ----------------------------------------- --------
----------------------------
 KAFKA_PARTITION                                    NUMBER(38)
 KAFKA_OFFSET                                       NUMBER(38)
 KAFKA_EPOCH_TIMESTAMP                              NUMBER(38)
 F_DECIMAL                                          NUMBER
 F_DATE                                             DATE
 F_TIME_MILLIS                                      TIMESTAMP(3)
 F_TIME_MICROS                                      TIMESTAMP(6)
 F_TIMESTAMP_MILLIS                                 TIMESTAMP(3)
 F_TIMESTAMP_MICROS                                 TIMESTAMP(6)
```

22.9 Configuring Access to a Kafka Cluster

You can configure access to secured Kafka clusters, or non-secured Kafka clusters

- Create a Cluster Access Directory
  The Oracle SQL access to Kafka administrator must create a cluster access directory object for each Kafka cluster to control database user access to the cluster.
- The Kafka Configuration File (osakafka.properties)
  To access Kafka clusters, you must create and a configuration file that contains the information required to access the Kafka cluster.
- Kafka Configuration File Properties
  The properties described here are used in the Kafka Configuration File `osakafka.properties`.
- Security Configuration Files Required for the Cluster Access Directory
  Identify the configuration files you require, based on your security protocol.

22.9.1 Create a Cluster Access Directory

The Oracle SQL access to Kafka administrator must create a cluster access directory object for each Kafka cluster to control database user access to the cluster.

The **Cluster Access Directory** is the Oracle directory object that contains the Kafka cluster configuration files. This directory is required for all clusters. For access to Kafka clusters, each Kafka cluster requires its own Cluster Access Directory. As the Oracle SQL access to Kafka administrator, you administer access to the Kafka cluster through creating the Cluster Access Directory object, and then granting `READ` access to this directory to the database users who need to access the Kafka cluster. You must create the Cluster Access Directory before you call the `DBMS_KAFKA_ADM.REGISTER_CLUSTER()` procedure.

Example 22-3 Creating a Cluster Access Directory Object and Granting READ Access

First create a cluster access directory object. In this example, the object is `osak_kafkaclus1_access`:

```
CREATE DIRECTORY osak_kafkaclus1_access AS '';;
```

After the Kafka Cluster is successfully registered, the Oracle SQL access to Kafka administrator grants READ access on this directory to users.

In this example, the user `example_user` is granted access to `osak_kafkaclus1_access`:

```
GRANT READ ON DIRECTORY osak_kafkaclus1_access TO example_user;
```

22.9.2 The Kafka Configuration File (osakafka.properties)

To access Kafka clusters, you must create and a configuration file that contains the information required to access the Kafka cluster.

- About the Kafka Configuration File
  The `osakafka.properties` file contains configuration information required to access secured Kafka Clusters, as well as additional information about Oracle SQL access to Kafka.
- Oracle SQL Access for Kafka Configuration File Properties
  To create an `osakafka.properties` file, review and specify the properties as described here.
- Creating the Kafka Access Directory
  To access secure Kafka clusters, you must create a Kafka Access Directory for each Kafka cluster.

22.9.2.1 About the Kafka Configuration File

The `osakafka.properties` file contains configuration information required to access secured Kafka Clusters, as well as additional information about Oracle SQL access to Kafka.

The **Kafka Configuration File**, `osakafka.properties`, is created in the Cluster Access Directory. The Oracle SQL access to Kafka administrator (granted `OSAK_ADMIN_ROLE`) creates the `osakafka.properties` file. This file is used by the `DBMS_KAFKA_ADM` package to make connections to an Apache Kafka cluster.

The Oracle SQL access to Kafka administrator creates a Cluster Access Directory directory in which to store the configuration files for each Kafka Cluster. Each Cluster Access Directory has its own Kafka Configuration File. To manage access to Apache Kafka clusters, only an Oracle SQL access to Kafka administrator has read and write access to Cluster Access Directories for Kafka clusters. No other users are granted any privileges on Cluster Access Directories, or Kafka Configuration Files.

Functions of the Kafka Configuration File

The `osakafka.properties` file is similar to the consumer properties file used by a Kafka Consumer using `librdkafka`. Secure Apache Kafka clusters require credential files, such as certificate authority, and client private key and client public certificate (PEM). These additional files again are like the ones required by a Kafka Consumer using `librdkafka`. The `osakafka.properties` file has the following properties:

- It is created and managed by the Oracle SQL access to Kafka administrator as part of the setup and configuration needed to access a Kafka cluster.
- It consists of a text file of key-value pairs. Each line has the format *key=value* describing the *key* and the *value*, and is terminated with a new line. The new line character cannot be part of the key or value.
- It contains Oracle SQL access for Kafka parameters, which are identified with the `osak` prefix.
- It contains debugging properties for Oracle SQL access to Kafka.
- It is used by the `DBMS_KAFKA_ADM` package to make connections to a Kafka cluster using `librdkafka` APIs.
- It is required for secure Kafka clusters, to store security configuration properties required to connect to Kafka clusters using `librdkaka` interfaces, Oracle SQL access to Kafka tuning properties, which are identified with the `osak` prefix, and debugging properties. For secure cluster access, the key-value pairs contain include cluster configuration files such as SSL/TLS certificates and client public and private keys.
- It is optional for non-secure Kafka clusters, to contain the tuning and debugging properties for cluster connections

The `osakafka.properties` file is stored in the Oracle SQL access for Kafka Cluster Access directory, in the path *ORACLE_base*`/osak/clusters/`*cluster-name*`/config`, where *Oracle_base* is the Oracle base directory of the target Oracle Database, and *cluster-name* is the name of the Kafka Cluster whose access information is stored in the configuration file.

Guidelines for Creating Kafka Configuration Files

As part of the setup and configuration required to access an Apache Kafka cluster, an Oracle SQL access for Kafka administrator The information in this file is used to set session context in C interfaces, which make connections to a Kafka cluster using `librdkafka` APIs.

The `SYS.DBMS_KAFKA_SEC_ALLOWED_PROPERTIES` system table contains a pre-populated list of supported consumer configuration properties, including security properties. For extensibility, `SYS` can add more properties to this table with certain restrictions

The `DBMS_KAFKA_ADM.REGISTER_CLUSTER()` procedure reads only those properties from the `osakafka.properties` file that are also listed in the `SYS.DBMS_KAFKA_SEC_ALLOWED_PROPERTIES` system table. Any extra properties are ignored.

22.9.2.2 Oracle SQL Access for Kafka Configuration File Properties

To create an `osakafka.properties` file, review and specify the properties as described here.

osakafka.properties File Processing

The properties specified in the `osakafka.properties` must be those listed in the table that follows. If you provide any other key-value pairs, then these values are ignored.

Note the following:

- Property names with the `osak` prefix are internal tuning properties or debugging properties.
- Property names without the `osak` prefix are Kafka consumer properties, which are used by `librdkafka`. For a complete list of properties, refer to the documentation for the Apache Kafka C/C++ client library (`librdkafka`) documentation.

Property	Allowed Values	Description
`security.protocol`	PLAINTEXT SSL SASL_PLAIN_TEXT SASL_SSL	Security Protocol used to communicate with Kafka brokers
`sasl.mechanisms`	GSSAPI PLAIN SCRAM-SHA-256 SCRAM-SHA-512	SASL mechanism to use for authentication **NOTE**: Despite the plural name, only one mechanism must be configured. This property is allowed to provide backward compatibility for older Kafka clusters. Where possible, Oracle recommends that you use the property `sasl.mechanism`instead.
`sasl.mechanism`	GSSAPI PLAIN SCRAM-SHA-256 SCRAM-SHA-512	Simple Authentication and Security Layer (SASL) mechanism to use for authentication
`ssl.ca.location`	File in the cluster configuration directory	File name of Certification Authority (CA) certificate for verifying the broker key. If an absolute path is specified, then the last token of path is taken as the file name.
`ssl.key.location`	File in the cluster configuration directory	File name of client private key If an absolute path is specified, then the last token of path is taken as the file name. The corresponding password value must be stored as a database credential using the `DBMS_CREDENTIALCREATE_CREDENTIAL()` procedure
`ssl.certificate.location`	File in the cluster configuration directory	File name of client public (PEM) key If an absolute path is specified, then the last token of path is taken as the file name.
`ssl.endpoint.identification.algorithm`	Valid Values: https none	Endpoint identification algorithm to validate the Kafka broker hostname, using a Kafka broker certificate. Values are as follows: `https`: Server (Kafka broker) hostname verification, as specified in RFC2818. `none`: No endpoint verification. Default Value: `none`

Property	Allowed Values	Description
sasl.username	Username	The username required for authenticating to the Kafka cluster. The corresponding password value for this username must be stored as a database credential, using the DBMS_CREDENTIALCREATE_CREDENTIAL() procedure
sasl.kerberos.principal	Client Kafka Kerberos principal name	The Client Kerberos principal name
sasl.kerberos.ccname	Kerberos ticket cache file name	The Kerberos ticket cache file Example: krb5ccname_osak This file must exist in the cluster configuration directory.
sasl.kerberos.config	Kerberos Configuration file name	The Kerberos configuration of the Kafka Cluster. Example krb5.conf This file must exist in the cluster configuration directory
sasl.kerberos.service.name	Kerberos principal name (Kafka primary name)	The primary name of the Kerberos principal, which is the name that appears before the slash (/). For example, kafka is the primary name of the Kerberos principal kafka/broker1.example.com@EXAMPLE.
max.partition.fetch.bytes	1024 * 1024	For librdkafkaSDK clients, OSS recommends that you allocate 1MB for each partition.
debug	all	Used to debug connectivity issues.

Example

The following is an example osakafka.properties file that specifies security protocol SSL,and provides authentication by using a Certification Authority (CA) certificate on the client:

```
security.protocol=ssl
ssl.ca.location=ca-cert
ssl.certificate.location=client_myhostname_client.pem
ssl.key.location=client_myhostname_client.key
```

Related Topics

- librdkafka The Apache Kafka C/C++ client library

22.9.2.3 Creating the Kafka Access Directory

To access secure Kafka clusters, you must create a Kafka Access Directory for each Kafka cluster.

The Oracle SQL access to Kafka administrator creates the operating system directory *Oracle-base*/osak/cluster_name/config, where *Oracle-base* is the Oracle base directory, and cluster_name is the value of the cluster name parameter passed to the SYS.DBMS_KAFKA_ADM.REGISTER_CLUSTER call. Each Kafka cluster requires its own dedicated Kafka Cluster Directory.

This directory must contain all the configuration files needed to access the Kafka Cluster:

- osakafka.properties file.
- Security files listed in the osakafka.properties file

In the following example, the Oracle base directory is /u01/app/oracle, and the cluster name is kafkaclus1:

```
mkdir u01/app/oracle/osak/kafkaclus1/config;
CREATE DIRECTORY osak_kafkaclus1_config AS
'u01/app/oracle/osak/kafkaclus1/config' ;
```

22.9.3 Kafka Configuration File Properties

The properties described here are used in the Kafka Configuration File osakafka.properties.

Description

The properties in the Kafka Configuration File contain configuration information for the Apache Kafka cluster. There are two categories of property names in the Kafka Configuration File:

- **consumer configuration property parameters** are properties used by the Apache Kafka broker. These files
- **Oracle properties** are the property names with the osak prefix. These properties are used for internal tuning or debugging.

The properties listed in the Kafka Configuration File are cross-checked against the system table SYS.DBMS_KAFKA_SEC_ALLOWED_PROPERTIES which contains all the supported properties. Any properties specified in the osakafka.properties file but not listed in the SYS.DBMS_KAFKA_SEC_ALLOWED_PROPERTIES table will be ignored by OSAK. The properties and values allowed in the osakafka.properties file are listed below:

Table 22-4 Property Names and Descriptions for Kafka Configuration Files

Property Name	Allowed Values	Description
security.protocol	PLAINTEXT, SSL, SASL_PLAIN_TEXT, SASL_SSL	Security Protocol used to communicate with Kafka brokers

Table 22-4 (Cont.) Property Names and Descriptions for Kafka Configuration Files

Property Name	Allowed Values	Description
`sasl.mechanisms`	`GSSAPI`, `PLAIN`, `SCRAM-SHA-256`, `SCRAM-SHA-512`	The SASL mechanism to use for authentication **NOTE**: Despite the plural name, only one mechanism must be configured. This property is allowed to provide backward compatibility for older Kafka clusters. Where possible, Oracle recommends that you use use the property `sasl.mechanism` instead.
`sasl.mechanism`	`GSSAPI`, `PLAIN`, `SCRAM-SHA-256`, `SCRAM-SHA-512`	The SASL mechanism to use for authentication
`ssl.ca.location`	File in cluster config directory	File name of Certification Authority (CA) certificate for verifying the broker key. If the absolute path is specified, then the last token of path is taken as the file name
`ssl.key.location`	File in cluster config directory	File name of client private key If an absolute path is specified, then the last token of path is taken as the file name The corresponding password value must be stored as a database credential using the `DBMS_CREDENTIALCREATE_CREDENTIAL()` procedure
`ssl.certificate.location`	File in cluster config directory	File name of client public (PEM) key If an absolute path is specified, then the last token of path is taken as the file name.
`ssl.endpoint.identification.algorithm`	Valid Values: `https`, `none` Default Value: `none`	The endpoint identification algorithm to validate the Kafka broker hostname, using the Kafka broker certificate. `https`: Server (Kafka broker) hostname verification as specified in RFC2818. `none`: No endpoint verification..
`sasl.username`	Username required for authenticating with Kafka cluster	Username required for authenticating with Kafka cluster. The corresponding password value must be stored as a database credential using the `DBMS_CREDENTIALCREATE_CREDENTIAL()` procedure
`sasl.kerberos.principal`	Client Kafka Kerberos principal name	Client Kerberos principal name

Table 22-4 (Cont.) Property Names and Descriptions for Kafka Configuration Files

Property Name	Allowed Values	Description
`sasl.kerberos.ccname`	Kerberos ticket cache file name	Kerberos ticket cache file Example: `krb5ccname_osak` This file must exist in the cluster configuration directory.
`sasl.kerberos.config`	Kerberos Configuration file name	Kerberos configuration of the Kafka Cluster. Example `krb5.conf` This file must exist in the cluster configuration directory
`sasl.kerberos.service.name`	The Kerberos principal name with which Kafka runs.	The Kerberos principal name with which Kafka runs.
`max.partition.fetch.bytes`	1024 * 1024	OSS recommends that you allocate 1MB for each partition for `librdkafkaSDK` clients.
`debug`	All	Used to debug connectivity issues

Example 22-4 Configuration File with Properties

```
osakafka.properties file for security protocol: SSL with client authentication
security.protocol=ssl
ssl.ca.location=ca-cert
ssl.certificate.location=client_myhostname_client.pem
ssl.key.location=client_myhostname_client.key
```

22.9.4 Security Configuration Files Required for the Cluster Access Directory

Identify the configuration files you require, based on your security protocol.

To configure access to a secure Kafka Cluster, the Oracle SQL access to Kafka administrator must add several configuration files from the Kafka Cluster Access Directory. The list of required files depends on which security protocol is used to configure security on the Kafka cluster. The file list can include files such as the certificate authority file, the SSL client public certificate file (PEM format), and the SSL client private key file.

> **Note:**
>
> The Kerberos `keytab` file is not required, because Kerberos ticket management is handled outside of Oracle SQL access to Kafka.

- SASL_SSL/GSSAPI
  Apache clusters with the `SASL_SSL` using `GSSAPI` authentication protocol required files for the Cluster Access Directory

- SASL_PLAINTEXT/GSSAPI
  Apache clusters with the `SASL_PLAINTEXT` using `GSSAPI` authentication protocol required files for the Cluster Access Directory
- SASL_PLAINTEXT/SCRAM-SHA-256
  Apache clusters with the `SASL_PLAINTEXT` using `SCRAM-SHA-256` authentication protocol required files for the Cluster Access Directory
- SASL_SSL/PLAIN
  Apache clusters with the `SASL_SSL` using `PLAIN` authentication protocol required files for the Cluster Access Directory
- SSL with Client Authentication
  Apache clusters with the `SSL` authentication protocol required files for the Cluster Access Directory
- SSL without Client Authentication
  Apache clusters with the `SSL` authentication protocol and without cliet authentication that are required files for the Cluster Access Directory

22.9.4.1 SASL_SSL/GSSAPI

Apache clusters with the `SASL_SSL` using `GSSAPI` authentication protocol required files for the Cluster Access Directory

Description

The SASL_SSL/GSSAPI protocol specifies Kerberos authentication with encryption. The Kerberos tickets must be managed externally (outside Oracle SQL access To Kafka).

DBMS_CREDENTIAL

Not required, because Kerberos tickets are managed externally.

Required Files in the Cluster Access Directory

1. The certificate authority (CA) file
2. The `osakafka.properties` file, with `ssl.ca.location` specifying the CA file is the SSL certificate authority .

In the following example, the property `security.protocol` specifies `SASL_SSL`. The property `sasl.mechanism` specifies `GSSAPI`. The CA file is `ca-cert.pem`, and it is specified by the property `ssl.ca.location`.

```
security.protocol=SASL_SSL
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
sasl.kerberos.config=krb5.conf
sasl.kerberos.ccname=krb5ccname_osak
sasl.kerberos.principal=kafkaclient/<FQDN-hostname>@<Realm>
ssl.ca.location=ca-cert.pem
ssl.endpoint.identification.algorithm=https
```

22.9.4.2 SASL_PLAINTEXT/GSSAPI

Apache clusters with the `SASL_PLAINTEXT` using `GSSAPI` authentication protocol required files for the Cluster Access Directory

Description

The `SASL_PLAINTEXT/GSSAPI` protocol specifies Kerberos authentication with no encryption. The Kerberos tickets must be managed externally (outside Oracle SQL access to Kafka).

DBMS_CREDENTIAL

Not required, because Kerberos tickets are managed externally.

Required Files in the Cluster Access Directory

1. The `osakafka.properties` file, with `ssl.ca.location` specifying the CA file is the SSL certificate authority .

In the following example, the property `security.protocol` specifies `SASL_PLAINTEXT`, and the property `sasl.mechanism` specifies `GSSAPI`.

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
sasl.kerberos.principal=kafkaclient/FQDN-hostname@Realm
sasl.kerberos.config=krb5.conf
sasl.kerberos.ccname=krb5ccname_osak
```

22.9.4.3 SASL_PLAINTEXT/SCRAM-SHA-256

Apache clusters with the `SASL_PLAINTEXT` using `SCRAM-SHA-256` authentication protocol required files for the Cluster Access Directory

Description

The `SASL_PLAINTEXT/SCRAM-SHA-256` protocol specifies SASL SCRAM authentication with no encryption.

DBMS_CREDENTIAL

Required, to store the password for the SASL user name.

Required Files in the Cluster Access Directory

1. The `osakafka.properties` file.

In the following example, the property `security.protocol` specifies `SASL_PLAINTEXT`, and the property `sasl.mechanism` specifies `SCRAM-SHA-256`.

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=SCRAM-SHA-256
sasl.username=testuser
```

22.9.4.4 SASL_SSL/PLAIN

Apache clusters with the `SASL_SSL` using `PLAIN` authentication protocol required files for the Cluster Access Directory

Description

The `SASL_SSL/PLAIN` protocol specifies settings for used OSS Kafka clusters

DBMS_CREDENTIAL

Required to store the r `sasl.password`.

Required Files in the Cluster Access Directory

1. The `osakafka.properties` file.

Example 22-5 OSS Cluster `osakafka.properties` File

In the following example, the property `security.protocol` specifies `SASL_SSL`, and the property `sasl.mechanism` specifies `PLAIN`.

```
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.username=<tenancyName>/<username>/<streamPoolID>
 #-- limit request size to 1 MB per partition
max.partition.fetch.bytes=1048576
```

Example 22-6 Non-OSS Cluster `osakafka.properties` File

In the following example, the property `security.protocol` specifies `SASL_SSL`, and the property `sasl.mechanism` specifies `PLAIN`.The ssl.ca.location property specifies a certificate authority (CA) file. The CA file is `ca-cert.pem`.

```
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.username=kafkauser
ssl.ca.location=ca-cert.pem
ssl.endpoint.identification.algorithm=https
```

22.9.4.5 SSL with Client Authentication

Apache clusters with the `SSL` authentication protocol required files for the Cluster Access Directory

Description

The `SSL` protocol specifies SSL with client authorization.

DBMS_CREDENTIAL

Required, to store the password for the SSL key.

Required Files in the Cluster Access Directory

1. The `osakafka.properties` file.

2. The configuration authority (CA) file
3. The rdkafka client PEM file (*rdkafka.client*.pem)
4. The rdkafka client key (*rdkafka.client*.key)

Example 22-7 SSL `osakafka.properties` File

In the following example, the property `security.protocol` specifies `SSL`, the property `ssl.key.location` specifies the rdkafka client key, and the sa.ca.location property specifies the certificate authority file.

```
security.protocol=SSL
ssl.certificate.location=rdkafka.client.pem
ssl.key.location=rdkafka.client.key
ssl.ca.location=ca-cert.pem
ssl.endpoint.identification.algorithm=https
```

22.9.4.6 SSL without Client Authentication

Apache clusters with the `SSL` authentication protocol and without cliet authentication that are required files for the Cluster Access Directory

Description

The `SSL` protocol specifies SSL without client authorization.

DBMS_CREDENTIAL

Not required.

Required Files in the Cluster Access Directory

1. The `osakafka.properties` file.
2. The configuration authority (CA) file

Example 22-8 SSL `osakafka.properties` File

In the following example, the property `security.protocol` specifies `SSL`, and the `sa.ca.location` property specifies the certificate authority file.

```
security.protocol=SSL
ssl.ca.location=ca-cert.pem
ssl.endpoint.identification.algorithm=https
```

22.10 Creating Oracle SQL Access to Kafka Applications

To create an application to access Apache Cluster data, create the type of application that you require.

Oracle SQL access to Kafka provides the following application modes that you can use to attach to the Apache Kafka cluster:

- **Loading**: Use to load data from a Kafka Topic into an Oracle Database table.
- **Streaming**: Use to read sequentially through a Kafka topic.

- **Seekable**: Use to access a Kafka topic randomly between starting and ending timestamps that you designate.

Choose the type of application that you want to create, depending on the kind of access to Kafka topics that you require:

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

Example 22-9 Creating a Streaming Application with Four Views for a Kafka Topic

In the following example, a streaming application is created to use a set of four views with temporary tables for a Kafka topic that has four (4) partitions. Each view creates a temporary table. Each view (and temporary table) is associated with one partition of the Kafka topic:

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

Example 22-10 Creating a Streaming Application with One View for a Kafka Topic

In the following example, a streaming application is created to use one view (1) with a temporary table where the temporary tables for a Kafka topic has four partitions. The view (a temporary table) is associated with the entire Kafka topic:

```
DECLARE
 v_options VARCHAR2;
BEGIN
 v_options := '{"fmt" : "DSV", "reftable" : "user_shape_table_name"}';
 SYS.DBMS_KAFKA.CREATE_STREAMING_APP (
                'ExampleCluster',
                'ExampleApp',
                'ExampleTopic',
                v_options,
                1);
END;
/
```

22.11 Security for Kafka Cluster Connections

Oracle SQL Access to Kafka supports access to Kafka and Oracle Streaming Service (OSS), using various security mechanisms, such as SSL, SASL, and Kerberos.

> **Note:**
>
> The credentials used to access the Kafka cluster must have access to both the Kafka broker metadata, as well as any topics that will be part of any Oracle SQL access to Kafka application. If there are access control lists (ACLs) enabled for the credentials, then ensure that access is granted to both the brokers and to the Kafka topics. In a shared Oracle Real Application Clusters (Oracle RAC) environment, securiy credentials should be in a shared location, not local to a cluster member node.

Secure Kafka Clusters

To maintain securely encrypted data transmission between Oracle Database and clusters, Oracle SQL access to Kafka employs several security protocols. For access to secure Kafka clusters and Oracle Streaming Services (OSS) clusters, security configuration files are used. These operating system files must exist in the cluster configuration directory. The cluster configuration Oracle directory object is created to access the cluster configuration files. Only the `osak_admin_role` is granted `READ` access to this directory. The cluster configuration files are readable only by the `osak_admin_role`. The cluster configuration files include the `osakafka.properties` file, and additional security files, such as SSL/TLS/PEM files and certificates. Keys and Certificates for SSL are stored in the Oracle keystore.

The cluster access Oracle directory object is used to control access to the Kafka cluster. This directory object does not contain any configuration files. Kafka sessions are exclusive to individual PDBs in the multitenant environment. Each PDB where you want to create an application to connect to a Kafka broker must create its own application.

No passwords must be embedded in files. Any embedded password properties in the `osakafka.properties` file will be ignored. All passwords must be stored as database credentials using the `DBMS_CREDENTIAL` package.

Kafka Clusters Using Kerberos Authentication

For Kafka clusters using Kerberos Authentication, the Kerberos ticket for the Kafka principal specified in the `osakafka.properties` file must be acquired on the database system, and renewed periodically outside of Oracle SQL access to Kafka.

The cluster configuration directory object and the cluster access directory object and database credential name must be supplied as input parameters to the `DBMS_KAFKA_ADM.REGISTER_CLUSTER()` call.

The Oracle SQL Access to Kafka administrator (a user with the `osak_admin_role`, the OSAK_ADMIN) performs the cluster registration and administration tasks.

22.12 Configuring Access to Unsecured Kafka Clusters

To configure access to non-secure Kafka clusters, the OSAK administrator (Oracle Database user with osak_admin_role) must complete this procedure.

Access to non-secure Kafka clusters requires that you create a cluster access database directory object to control access to the Kafka cluster. The grants on this database directory are used to control which Oracle Database users can access the Kafka cluster. This database directory has an empty path: it does not a need a corresponding operating system directory, and it also does not contain any files. Oracle recommends that the Oracle Directory Object Name for a cluster access database directory object takes the form `OSAK_CLUSTER_NAME_ACCESS`, where `CLUSTER_NAME` is the name of the Kafka cluster.

Procedure:

1. Create the cluster access database directory with an empty path. This directory is used to control which Oracle users can access the Kafka cluster.

   For example, create a cluster access database directory object called `oaskaccess_kafkaclust1` with an empty path. This directory is used to control which Oracle users can access the Kafka cluster.

   ```
   SQL> CREATE DIRECTORY OSAK_KAFKACLUS2_ACCESS AS '';
   ```

2. On the target Oracle Database server, create the cluster configuration operating system directory in the Oracle base path directory, using the path *`Oracle_base`*`/osak/`*`cluster_name`*`/config` where *`Oracle_base`* is the Oracle base directory, and *`cluster_name`* is the Kafka cluster name. For example:

   ```
   mkdir /u01/app/oracle/osak/kafkaclus2/config
   ```

   Log in to the database as SYSDBA, start SQL, and create the corresponding Oracle directory object. In this example, the Kafka cluster name is `KAFKACLUS2`:

   ```
   SQL> CREATE DIRECTORY OSAK_KAFKACLUS2_CONFIG AS  'u01/app/oracle/osak/
   kafkaclus2/config';
   ```

3. Create an empty `osakafka.properties` file, or an osakafka.properties file with OSAK tuning or debugging properties.
4. In SQL, register the Kafka cluster using `DBMS_KAKFA_ADM.REGISTER_CLUSTER()`. For example, using the server hostname `mykafkabootstrap-host`, port 9092, for Kafka cluster `KAFKACLUS2`:

   ```
   SQL> select DBMS_KAFKA_ADM.REGISTER_CLUSTER (
             cluster_name  => 'KAFKACLUS2',
             bootstrap_servers =>'Kafka-example-host:9092',
             kafka_provider =>
   DBMS_KAFKA_ADM.KAFKA_PROVIDER_APACHE,
             cluster_access_dir => 'OSAK_KAFKACLUS2_ACCESS',
             credential_name => NULL,
             cluster_config_dir => 'OSAK_KAFKACLUS2_CONFIG',
              cluster_description =>  'My test cluster kafkaclus2',
   ```

```
            options  => NULL)
from dual;
```

If configuration is successful, then the registration return is 0 (zero):

```
SQL> DBMS_KAFKA_ADM_RE…..
              0
```

5. Grant read access to a Kafka user. In the following example, user `app2-usr` is granted access to the Kafka cluster named `KAFKACLUS2`:

```
SQL> grant read on directory osak_kafkaclus2_access to app2-usr;
```

22.13 Configuring Access to Secure Kafka Clusters

To configure access to secure Kafka clusters use this procedure.

Access to secure Kafka clusters requires configuration files, such as `osakafka.properties`, and additional security files such as SSL/TLS PEM files and certificates. These files are stored in a cluster configuration database directory object. The configuration files and directory are protected by the operating system directory and file access privileges,

The cluster configuration operating system directory is configured in the Oracle base directory, and is owned by the Oracle Installatoin owner, or Oracle user (oracle), and the Oracle Inventory Group (oinstall). The Oracle user and Oracle Inventory group must have directory privileges set to 750 (`rwxr-x---`) on and the `osakafka.properties` file in the directory must have privileges set to 540 (`rw-r-----`). All other files in the cluster configuration directory must have privileges set to and 440 (`r--r-----`).

- The Oracle SQL Access for Kafka configuration file (`osakafka.properties`) is created and stored in a cluster configuration database directory object.
- Security files for your chosen security method, such as Kerberos, SSL, TLS/SSL with PEM files, and the certificates created for them, are stored in a cluster configuration database directory object.

Procedure:

1. Create a cluster access database directory object to control access to the Kafka cluster. The grants on this database directory object are used to control which Oracle Database users can access the Kafka cluster. This database directory has an empty path. That is, it does not need a corresponding operating system directory, and does not contain any files.

   For example, create a cluster access database directory object called `oaskaccess_kafkaclust1` with an empty path. This directory is used to control which Oracle users can access the Kafka cluster. :

   ```
   SQL> CREATE DIRECTORY osakaccess_kafkaclus1 AS '';
   ```

2. On the target Oracle Database server, create the cluster configuration operating system directory in the Oracle base path directory, using the path *`Oracle_base`*`/osak/`*`cluster_name`*`/config` where *`Oracle_base`* is the Oracle base directory, and *`cluster_name`* is the Kafka cluster name. For example:

   ```
   mkdir /u01/app/oracle/osak/kafkaclus1/config
   ```

3. Log in to the database as SYSDBA, start SQL, and create the corresponding Oracle directory object in the target Oracle Database. Oracle recommends that you use `OSAK_clustername_access` for the database object name, where clusternamne is the name of the Kafka cluster. For example:

```
CREATE DIRECTORY OSAK_KAFKACLUS1_CONFIG
   AS '/u01/app/oracle/osak/kafkaclus1/config';
```

4. Create the `osakafka.properties` file in the cluster configuration directory, based on the security protocol you use. This file is similar to `librdkafka` client properties file.

   In the following example, the `osakafka.properties` file is configured to use Secure Socket Layer (SSL) for the security protocol, with client authentication:

```
security.protocol=ssl
ssl.ca.location=ca-cert
ssl.certificate.location=client_myhostname_client.pem
ssl.key.location=client_myhostname_client.key
ssl.key.password=password-that-is-ignored
```

5. Copy the security files referred to by `osakafka.properties` into the cluster configuration directory. For example, where the ca-cert path is `/etc/ssl/certs/`:

```
$cp /etc/ssl/certs/ca-cert /u01/app/oracle/osak/kafkaclus1/config;
$cp /etc/ssl/certs/client-myhostname-client.pem /u01/app/oracle/osak/
kafkaclus1/config;
$cp /etc/ssl/certs/client-myhostname-client.key /u01/app/oracle/osak/
kafkaclus1/config;
```

6. Set up credentials:

   - If you are using either `SSL.key.location` or `sasl.username` properties in the `osakafka.properties` file:

     Create a database credential to store the password required for authentication with the Kafka cluster using SSL SASI authentication. The corresponding password properties `ssl.key.password` or `sasl.password` are added automatically by `DBMS_KAFKA` during the cluster registration process. For example:

```
begin
    dbms_credential.create_credential(
        credential_name => 'KAFKACLUS1CRED1',
        username => 'KAFKACLUS1',
        password => 'enter-ssl-key-password-or-sasl-password);
end;
/
```

   - If your Kafka cluster uses GSSAPI/Kerberos as its authentication mechanism:

     Acquire the Kerberos ticket on the databases system for the Kafka principal listed in the `osakafka.properties` file

7. Log in as `SYSDBA`, start SQL, and register the Kafka cluster using the `SYS.DBMS_KAFKA_ADM.REGISTER_CLUSTER()` procedure. In the following example, the Kafka cluster `KAFKACLUS1` is registered:

```
select DBMS_KAFKA_ADM.REGISTER_CLUSTER ('KAFKACLUS1',
             'mykafkabootstrap-host:9092',
             DBMS_KAFKA_ADM.KAFKA_PROVIDER_APACHE,
             'OSAK_KAFKACLUS1_ACCESS'
             'KAFKACLUS1CRED1',
             'OSAK_KAFKACLUS1_CONFIG',
             'My test cluster kafkaclus1') from dual;
```

If successful, then the output should return 0 (zero). For example:

```
SQL> DBMS_KAFKA_ADM_RE…..
             0
```

8. Grant read access to a Kafka user. In the following example, user `app1-usr` is granted access to the Kafka cluster named `KAFKACLUS1`:

```
SQL> grant read on directory OSAK_KAFKACLUS1_ACCESS to app1-usr;
```

22.14 Administering Oracle SQL Access to Kafka Clusters

See how to update, temporarily disable, and delete Kafka cluster definitions with Oracle SQL access to Kafka

- Updating Access to Kafka Clusters
  If the Kafka cluster environment changes, you can update the cluster definition and configuration for those changes.
- Disabling or Deleting Access to Kafka Clusters
  You can temporarily disable an Oracle SQL access to a Kafka cluster, or delete the connection if it is no longer required.

22.14.1 Updating Access to Kafka Clusters

If the Kafka cluster environment changes, you can update the cluster definition and configuration for those changes.

During the lifetime of the Kafka cluster definition, if you need to update the cluster definition, then you can use DBMS_KAFKA_ADM.UPDATE_CLUSTER_INFO and DBMS_KAFKA_ADM.CHECK_CLUSTER.

22.14.2 Disabling or Deleting Access to Kafka Clusters

You can temporarily disable an Oracle SQL access to a Kafka cluster, or delete the connection if it is no longer required.

Example 22-11 Disabling a Kafka Cluster

During temporary outages of the Kafka environment, you can temporarily disable access to the Kafka cluster

- DBMS_KAFKA_ADM.DISABLE_CLUSTER followed by
- DBMS_KAFKA_ADM.ENABLE_CLUSTER when the Kafka environment is back up

Example 22-12 Deleting a Kafka Cluster

When a cluster definition is no longer needed, the OSAK Administrator can remove the cluster definition

- DBMS_KAFKA_ADM.DEREGISTER_CLUSTER

22.15 Guidelines for Using Kafka Data with Oracle SQL Access to Kafka

Review guidelines, restrictions, and recommendations as part of your application development plan.

- Kafka Temporary Tables and Applications
  Oracle SQL access to Kafka views and their corresponding temporary tables are bound to a unique Kafka application (a group ID), and must exclusively access one or more partitions in a topic on behalf of that application.
- Sharing Kafka Data with Multiple Applications Using Streaming
  To enable multiple applications to use Kafka data, use Oracle SQL access to Kafka to stream Kafka tables to a user table.
- Dropping and Recreating Kafka Tables
  Because the Kafka offsets are managed by the `DBMS_KAFKA` metadata tables, changes to a Kafka topic configuration can require manual updates to Oracle SQL access to Kafka applications.

22.15.1 Kafka Temporary Tables and Applications

Oracle SQL access to Kafka views and their corresponding temporary tables are bound to a unique Kafka application (a group ID), and must exclusively access one or more partitions in a topic on behalf of that application.

Use these guidelines to assist you with constricting your applications.

Kafka Group IDs and Oracle SQL Access to Kafka Temporary Tables

Unlike standard Oracle tables and views, in accordance with the rules for consuming Apache Kafka data, Kafka temporary tables cannot be shared across multiple applications. With Kafka data, each temporary table is a snapshot of data fetched directly from Kafka at a particular point of time, and has a canonical name format that identifies the Kafka cluster, the application name, and a **view ID**, an integer identifying a particular view accessing one or more partitions in the cluster or topic on behalf of an application associated with a consumer group ID (`groupID`) in Kafka. The temporary views and tables created in Oracle Database are bound to a unique Kafka application (identified by `groupID`), and must exclusively access one or more partitions in a topic on behalf of that application. It cannot share access to these partitions simultaneously with other applications. This restriction extends to an Oracle application instance. An Oracle SQL Access to Kafka view and its associated temporary table must be exclusive to that application. If you want to configure multiple applications to query the same Kafka topic or partition data, then these applications must identify themselves as a different application (that is, with different, unique Kafka group IDs), and create their own Oracle SQL access to Kafka applications, reflecting their own group ID and applcation identity, and their own set of offsets to track.

Guidelines for Using Views and Tables with Oracle SQL Access to Kafka

Create views and tables for your applications in accordance with the kinds of analytics you want to perform with that data.

If you want your application to use Oracle SQL for analytics, then Oracle recommends that you create an Oracle SQL access to Kafka view for that application that captures all partitions of the data that you want to query. Each visit by a single application instance captures all new Kafka data in a topic, and generates aggregate information that the application can then store or display.

If you do not want to perform analytics using Oracle SQL, but instead use complex logic in the application itself, then Oracle recommends that you scale out the application instances, and have each Oracle SQL access to Kafka view access a single partition on behalf of a single application instance. For this case, typically the Kafka data is joined with standard Oracle tables to enrich the data returned to the application.

In cases where some SQL analytics and joins are performed before more analysis is done by the application, views mapping to some subset of the partitions in a topic can be a good option to choose.

22.15.2 Sharing Kafka Data with Multiple Applications Using Streaming

To enable multiple applications to use Kafka data, use Oracle SQL access to Kafka to stream Kafka tables to a user table.

To share Kafka data with multiple Oracle users, so that table is not tied to a specific Group ID, Oracle recommends that you have an application user run the Oracle SQL access to Kafka in Loading mode, with the PL/SQL procedure `DBMS_KAFKA.EXECUTE_LOAD_APP`, to create a table owned by that user. With this option, a single application instance runs the Loading PL/SQL procedure on a regular basis to load all new data incrementally from a Kafka topic into an Oracle Database table. After the data is loaded into the table, it can then be made accessible to standard Oracle Database applications granted access to that table, without the restrictions that apply to temporary tables.

22.15.3 Dropping and Recreating Kafka Tables

Because the Kafka offsets are managed by the `DBMS_KAFKA` metadata tables, changes to a Kafka topic configuration can require manual updates to Oracle SQL access to Kafka applications.

To ensure that Oracle application instances can identify what Kafka table content has been read, and where it has been read, partition offsets of a Kafka topic must tracked on *a per application instance* basis.

Kafka supports three models for committing offsets:

- Auto-commit, where Kafka commits the last offset fetched on a short time schedule
- Manual commit, where applications send a request for Kafka to commit an offset
- Application-managed commits, where Kafka commits are entirely managed by the applications.

Oracle uses application-managed commits. In these commits, Kafka sees this as an application declaring manual commits without ever explicitly committing to Kafka. Offsets are recorded and maintained exclusively in `DBMS_KAFKA` metadata tables. These tables are protected by the ACID transaction properties of Oracle Database. To insure the integrity of

transactions, Oracle does not support Kafka auto-commit or Kafka manual commit in Oracle SQL Access to Kafka.

If a Kafka topic is dropped and recreated, then you must update that table manually, depending on the scenario:

Example 22-13 Dropping and Resetting a View with the Same Partitions

If the number of partitions remains the same as the original Kafka topic configuration, then you must reset the view reset the Oracle SQL access to Kafka view to begin processing from the beginning of the Kafka partition within the recreated topic. To reset the view, call the procedure `DBMS_KAFKA.INIT_OFFSET(view_name, 0, 'WML')`, where *view_name* is the name of the view.

Example 22-14 Dropping and Resetting a View with Fewer Partitions

This option is not available. If the number of partitions is less than the original Kafka topic configuration, then the Oracle SQL access to Kafka applications associated with this topic must be dropped and recreated.

Example 22-15 Dropping and Resetting a View with More Partitions

If the number of partitions is greater than the original Kafka topic configuration, then you must reset the Oracle SQL Access to Kafka view by calling the procedure `DBMS_KAFKA.INIT_OFFSET(view_name, 0, 'WML')`, where *view_name* is the name of the view, and then call the procedure `DBMS_KAFKA.ADD_PARTITIONS` for each Oracle SQL Access to Kafka application using this topic.

22.16 Choosing a Kafka Cluster Access Mode for Applications

To use Oracle SQL access to Kafka, decide what mode of data access you require for your applications.

- Configuring Incremental Loads of Kafka Records Into an Oracle Database Table
  To enable applications to load data incrementally from a Kafka topic into an Oracle Database table, you use Oracle SQL Access to Kafka in Loading mode.
- Streaming Access to Kafka Records in Oracle SQL Queries
  To access Kafka topics in a sequential manner from the beginning of the topic, or from a specific starting point in a Kafka topic, you can use Oracle SQL Access to Kafka in Streaming mode.
- Seekable access to Kafka Records in Oracle SQL queries
  To access Kafka records randomly between two timestamps, you use Oracle SQL Access to Kafka in Seekable mode

22.16.1 Configuring Incremental Loads of Kafka Records Into an Oracle Database Table

To enable applications to load data incrementally from a Kafka topic into an Oracle Database table, you use Oracle SQL Access to Kafka in Loading mode.

Configuring Oracle SQL Access to Kafka to perform incremental loads using the `EXECUTE_LOAD_APP` procedure enables you to move Kafka data into standard Oracle tables, which are accessible by multiple applications without the one reader constraint imposed when using Oracle SQL access to Kafka temporary tables.

To load Kafka data incrementally into an Oracle Database table, an application declares that it is a loading application by calling the PL/SQL procedure `DBMS_KAFKA.CREATE_LOAD_APP` to

initialize a state for subsequent calls to`DBMS_KAFKA.EXECUTE_LOAD_APP`. The `DBMS_KAFKA.CREATE_LOAD_APP` procedure creates a single view over all partitions of the topic.

If you do not require data from the entire topic, then you also have the option to configure the application to call the `DBMS_KAFKA.INIT_OFFSET[_TS]` procedure to set the starting point in Kafka topic partitions for loading the Kafka data.

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.

To load data into an Oracle Database table from a Kafka topic:

- DBMS_KAFKA.CREATE_LOAD_APP to create an Oracle SQL Access to Kafka Load application
- Optionally, DBMS_KAFFA_INIT_OFFSET_TS or DBMS_KAFKA_INIT_OFFSET to set the first Kafka record to be read
- LOOP until done
  - DBMS_KAFKA.EXECUTE_LOAD_APP to load Kafka data starting from where we left off to the current high water mark
- DBMS_KAFKA.DROP_LOAD_APP to drop the load application

22.16.2 Streaming Access to Kafka Records in Oracle SQL Queries

To access Kafka topics in a sequential manner from the beginning of the topic, or from a specific starting point in a Kafka topic, you can use Oracle SQL Access to Kafka in Streaming mode.

If your application requires access to Kafka topics in a sequential manner, you can configure Oracle SQL Access to Kafka in Streaming mode. This mode enables a SQL query using an Oracle SQL access to Kafka temporary table to access Kafka records sequentially in an application processing loop. With this use case, the application declares that it is a streaming application by calling the PL/SQL procedure `DBMS_KAFKA.CREATE_STREAMING_APP` to initialize the state for subsequent queries of Oracle SQL access to Kafka views. In addition to creating views, this procedure also creates a global temporary table for each view. You also have the option to use the INIT_OFFSET[_TS] procedure to set the starting point in Kafka topic partitions for your application. When you set as starting point, the initial query reads the Kafka data from the starting point. The application then can perform the following steps, in a processing loop:

1. Call `DBMS_KAFKA.CREATE_STREAMING_APP` to create the Oracle SQL access to Kafka streaming application.
2. (Optional) call `DBMS_KAFFA_INIT_OFFSET_TS` or `DBMS_KAFKA_INIT_OFFSET` to set the first Kafka record that you want to be read.
3. LOOP until done:
   a. Call DBMS_KAFKA.LOAD_TEMP_TABLE to load the global temporary table with the next set of rows from Kafka
   b. SELECT from the OSAK global temporary table Process data retrieved
   c. If the processing was successful, call `DBMS_KAFKA.UPDATE_OFFSET` to update the last Kafka offsets read
   d. Commit the offset tracking information using `COMMIT`.
4. When finished, call DBMS_KAFKA.DROP_STREAMING_APP to drop the application.

The PL/SQL procedure `DBMS_KAFKA.UPDATE_OFFSET` transparently advances Kafka partition offsets of the Kafka group ID for all of the partitions that are identified with the Oracle SQL access to Kafka view, so that for every call to `DBMS_KAFKA.LOAD_TEMP_TABLE`, a new set of unread Kafka records is retrieved and processed

Note that `UPDATE_OFFSET` initiates an Oracle transaction if a transaction is not already started, and records the last offsets in metadata tables. Because of this, to ensure that the transaction does not lose its session information you should configure your application to commit the transaction after every call to `UPDATE_OFFSET`. After you commit the transaction, because Oracle SQL access to Kafka manages offsets within an Oracle transaction, no records are lost or reread. If the transaction fails to complete, then offsets are not advanced. When the application resumes data reads, it can then restart the data reads of the Kafka data from where it stopped its previous reads.

22.16.3 Seekable access to Kafka Records in Oracle SQL queries

To access Kafka records randomly between two timestamps, you use Oracle SQL Access to Kafka in Seekable mode

The Seekable mode of Oracle SQL access to Kafka enables an application to read Kafka records between timestamps of interest, typically identified by a peer application doing streaming access. In this mode, you specify the start and end timestamps that define a window of time from which the `DBMS_KAFKA.LOAD_TEMP_TABLE` procedure will populate the temporary table. An application declares that it is a Seekable application by calling the PL/SQL procedure `DBMS_KAFKA.CREATE_SEEKABLE_APP` to initialize the state for accessing Kafka in Seekable mode. This procedure creates a view and a corresponding global temporary table over all partitions of the topic. The `DBMS_KAFKA.SEEK_OFFSET_TS` procedure is called to specify the time window from which to query. The application calls `SEEK_OFFSET_TS` before calling the `DBMS_KAFKA.LOAD_TEMP_TABLE` procedure to load the temporary table with the next set of rows.

To query Kafka data in ”Seekable” mode in order to access Kafka records between two timestamps

- DBMS_KAFKA.CREATE_SEEKABLE_APP to create the Oracle SQL Access to Kafka seekable application
- LOOP until done
  - DBMS_KAFKA.SEEK_OFFSET_TS to seek to a user defined window of time in a Kafka topic
  - Call DBMS_KAFKA.LOAD_TEMP_TABLE to load the global temporary table with the set of rows from Kafka
  - SELECT from the OSAK global temporary table
  - Process the data
- DBMS_KAFKA.DROP_SEEKABLE_APP when done with the application

22.17 Creating Oracle SQL Access to Kafka Applications

To query Kafka data in a LOAD application, load Kafka data into an Oracle Database table using these procedures.

Typical uses of load procedures include:

`DBMS_KAFKA.CREATE_LOAD_APP`: This procedure is used to set up loading into an Oracle table

`DBMS_KAFKA.INIT_OFFSET[_TS] (OPTIONAL)`: This procedure is used to set offsets in all topic partitions to control the starting point of a sequence of load operations. You repeat this procedure until you no longer want to load new rows from the Kafka topic on which you run the procedure.

`DBMS_KAFKA.EXECUTE_LOAD_APP`: This procedure is used to load new unread records from a Kafka topic to high water mark of all topic partitions

`DBMS_KAFKA. DROP_LOAD_APP`: This procedure is used when loading is complete from the Kafka topic on which you are running procedures.

- Creating Load Applications with Oracle SQL Access to Kafka
  If you want to load data into an Oracle Database table, then use the Loading mode of `DBMS_KAFKA`.
- Creating Streaming Applications with Oracle SQL Access to Kafka
  If you want to load data into an Oracle Database table, then use the Loading mode of `DBMS_KAFKA`.
- Creating Seekable Applications with Oracle SQL Access to Kafka
  If you want to investigate issues that occurred in the past, and randomly access a Kafka topic between starting and ending timestamps, then use the Seekable mode of `DBMS_KAFKA`.

22.17.1 Creating Load Applications with Oracle SQL Access to Kafka

If you want to load data into an Oracle Database table, then use the Loading mode of `DBMS_KAFKA`.

An Oracle SQL access to Kafka load application retrieves data from all partitions of a Kafka topic, and places that data into an Oracle Database table for processing. 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. Only one application instance is allowed to call `DBMS_KAFKA.EXECUTE_LOAD_APP` for the created `LOAD` application.

Example 22-16 Loading Data Into a Table with DBMS_KAFKA.EXECUTE_LOAD_APP

In this example, you create create one view and associated temporary table for a loading application. The Kafka cluster name is ExampleCluster, the application name is ExampleApp. The Kafka Topic is ExampleTopic, which is a topic that has four partitions:

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

Example 22-17 Loading Data Periodically Into a Table with DBMS_KAFKA.EXECUTE_LOAD_APP

As an alternative to processing Kafka data from a set of application views, you can choose simply to load the data from Kafka into an Oracle Database table, periodically fetching the latest data into the table. The `DBMS_KAFKA.EXECUTE_LOAD_APP` procedure in this example obtains the latest data from the Kafka cluster, and inserts the data into the table, `ExampleLoadTable`. An application that uses the data in this table has the option to calll `DBMS_KAFKA.INIT_OFFSET[_TS]` to set the starting point for the load.

```
DECLARE
    v_records_inserted INTEGER;
BEGIN
    SYS.DBMS_KAFKA.EXECUTE_LOAD_APP (
                            'ExampleCluster',
                            'ExampleLoadApp',
                            'ExampleLoadTable',
                           v_records_inserted);
END;
```

Example 22-18 Dropping the Kafka View and Metadata with DBMS_KAFKA.DROP_LOAD_APP or DBMS_KAFKA.DROP_ALL_APPS

If the Oracle SQL access to Kafka Load application is no longer needed, then you can drop the views and metadata by calling `DBMS_KAFKA.DROP_LOAD_APP`. In the following example, the Kafka cluster is `ExampleCluster`, and the application is `ExampleApp`.

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

If the Kafka cluster for one or more Oracle SQL access to Kafka applications no longer exists, then you can drop all of the applications for a given cluster by calling `DBMS_KAFKA.DROP_ALL_APPS`

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

22.17.2 Creating Streaming Applications with Oracle SQL Access to Kafka

If you want to load data into an Oracle Database table, then use the Loading mode of `DBMS_KAFKA`.

Streaming enables the ability to process data at scale. You can use Oracle SQL access to Kafka in streaming mode to create multiple application instances. Multiple instances enables applications to scale out and divide the workload of analyzing Kafka data across the application instances running concurrently on one or more threads, processes, or systems.

An Oracle SQL access to Kafka streaming application includes a set of dedicated Oracle SQL access to Kafka global temporary tables and Oracle SQL access to Kafka views. These temporary tables and views can be used for retrieving new, 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 active topic and partition information regarding the Kafka topic. This view is created once, and serves all applications that are sharing the same cluster.

Each Oracle SQL access to Kafka global temporary table and its related view is exclusively used by one instance of an Oracle SQL access to Kafka application.

Each application instance calls `LOAD_TEMP_TABLE`, which populates the dedicated Oracle SQL access to Kafka global temporary table with Kafka rows retrieved from the associated view. The application then can run one or more SQL queries against the content in the Oracle SQL access to Kafka global temporary table. When the application is done with the current set of Kafka rows, it calls `UPDATE_OFFSET` and `COMMIT`.

A `STREAMING` mode application is different from a `LOAD` or `SEEKING` application in that you can configure the application to select how many Oracle SQL access to Kafka views and temporary tables are required for your application purpose. As with other types of Oracle SQL access to Kafka applications, each application instance exclusively queries one unique Oracle SQL access to Kafka temporary table. Each Oracle SQL access to Kafka view and global temporary table name includes the cluster name, the application name, and an application instance identifier (ID).

In creating your application, be aware that the number Oracle SQL access to Kafka views and temporary table pairs you create must be between 1 and *N* where *N* is the number of partitions in the Kafka topic.

During runtime, each application instance runs in its own user session, and processes one Oracle SQL access to Kafka global temporary table and its associated view. Accordingly, to run application instances concurrently, you must allocate at least as many sessions to the user as there are partitions in the Kafka topic (that is, the value of `N`). If the `view_count` exceeds the maximum sessions per user, then this call fails with an error indicating that there are insufficient sessions allocated to the user. The number of Kafka partitions bound to a specific Oracle SQL access to Kafka view and its associated global temporary table varies, depending on how many views are created, and on how many partitions exist. Oracle SQL access to Kafka balances the number of partitions assigned to each view.

Example 22-19 Streaming Data Into a Table with DBMS_KAFKA.CREATE_STREAMING_APP

In this example, you create a set of four views and associated temporary tables for a Streaming mode application using data from a topic called `ExampleTopic`. The topic has four partitions, and each view and temporary table is associated with one partition:

```
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;
/
```

Example 22-20 Loading Data Into a Single Table with DBMS_KAFKA.CREATE_STREAMING_APP

In this example, Streaming mode is used to create one view and associated temporary table for an application that is associated with all four partition of the topic:

```
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;
/
```

Example 22-21 Dropping the Kafka View and Metadata with DBMS_KAFKA.DROP_STREAMING_APP or DBMS_KAFKA.DROP_ALL_APPS

If the Oracle SQL access to Kafka Load application is no longer needed, then you can drop the views and metadata by calling `DBMS_KAFKA.DROP_STREAMING_APP`. In the following example, the Kafka cluster is `ExampleCluster`, and the application is `ExampleApp`.

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

If the Kafka cluster for one or more Oracle SQL access to Kafka applications no longer exists, then you can drop all of the applications for a given cluster by calling `DBMS_KAFKA.DROP_ALL_APPS`

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

22.17.3 Creating Seekable Applications with Oracle SQL Access to Kafka

If you want to investigate issues that occurred in the past, and randomly access a Kafka topic between starting and ending timestamps, then use the Seekable mode of `DBMS_KAFKA`.

Before accessing Kafka topics in Seekable mode, you must create an Oracle SQL Access to Kafka application with `DBMS_KAFKA.CREATE_SEEKABLE_APP` This package creates an application that you can use in Seekable mode.

Using Oracle SQL access to Kafka in Seekable mode enables you to use Kafka data to investigate issues that have occurred in the past. Provided that the data is still present in the Kafka steam, you can create a Seekable application by calling `DBMS_KAFKA.CREATE_SEEKABLE_APP`. When you have created a Seekable mode application, you can then call the procedure `DBMS_KAFKA.SEEK_OFFSET_TS` to request the Oracle SQL access to Kafka view to retrieve a range of data records. For example, suppose that an IT consultant was informed that a production issue occurred around 03:00 in the morning, and needed to investigate the cause. The consultant could use the following procedure, load the temporary table, and then select to retrieve an hour's worth of data around that time:

In creating your application, be aware that the number Oracle SQL access to Kafka views and temporary table pairs you create must be between 1 and *N* where *N* is the number of partitions in the Kafka topic.

During runtime, each application instance runs in its own user session, and processes one Oracle SQL access to Kafka global temporary table and its associated view. Accordingly, to run application instances concurrently, you must allocate at least as many sessions to the user as there are partitions in the Kafka topic (that is, the value of `N`). If the `view_count` exceeds the maximum sessions per user, then this call fails with an error indicating that there are insufficient sessions allocated to the user. The number of Kafka partitions bound to a specific Oracle SQL access to Kafka view and its associated global temporary table varies, depending on how many views are created, and on how many partitions exist. Oracle SQL access to Kafka balances the number of partitions assigned to each view.

Example 22-22 Searching a Date Range in Kafka Data Using DBMS_KAFKA.CREATE_SEEKABLE_APP

In this example, suppose that an IT consultant was informed that a production issue occurred around 03:00 in the morning, and needed to investigate the cause. The consultant could use the following procedure, load the temporary table, and then select to retrieve an hour's worth of data around that time, where the Kafka cluster is `EXAMPLECLUSTER`, and the columns are `EventCol` and `ExceptionCol`:

```
SYS.DBMS_KAFKA.SEEK_OFFSET_TS (
                         'ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0',
                         TO_DATE ('2022/07/04 02:30:00', 'YYYY/MM/DD
HH:MI:SS',
                         TO_DATE ('2022/07/04 03:30:00', 'YYYY/MM/DD
HH:MI:SS'));
SYS.DBMS_KAFKA.LOAD_TEMP_TABLE
                   (ORA$DKVGTT_EXAMPLECLUSTER_SEEKABLEAPP_0);
SELECT EventCol, ExceptionCol FROM ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0;
```

Example 22-23 Locating Records Associated with Anomalies Using DBMS_KAFKA.CREATE_SEEKABLE_APP

Suppose that when an application using sequential access to a Kafka stream detected a potential anomaly, the application inserts a row into an anomaly table. The anomaly table includes the Kafka timestamp, as well as any other data specified as important to trace. Another application could use this information to retrieve records around the suspected record to see if there were any other issues associated with the anomaly. In this example, the columns associated with an anomaly that an IT consultant wants to examine are `UserCol` and `ReqeustCol`. To achieve this, 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$DKVGTT_EXAMPLECLUSTER_SEEKABLEAPP_0);
SELECT UserCol, RequestCol FROM ORA$DKV_EXAMPLECLUSTER_SEEKABLEAPP_0;
--application logic
```

Example 22-24 Dropping the Kafka View and Metadata with DBMS_KAFKA.DROP_SEEKABLE_APP or DBMS_KAFKA.DROP_ALL_APPS

If the Oracle SQL access to Kafka Load application is no longer needed, then you can drop the views and metadata by calling `DBMS_KAFKA.DROP_SEEKABLE_APP`. In the following example, the Kafka cluster is `ExampleCluster`, and the application is `ExampleApp`.

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

If the Kafka cluster for one or more Oracle SQL access to Kafka applications no longer exists, then you can drop all of the applications for a given cluster by calling `DBMS_KAFKA.DROP_ALL_APPS`

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

22.18 Using Kafka Cluster Access for Applications

Learn how to use Kafka cluster data access with your applications.

- How to Diagnose Oracle SQL Access to Kafka Issues
  If you encounter issues with Oracle SQL access to Kafka, then use these guidelines to determine the cause, and resolve the issue.
- Identifying and Resolving Oracle SQL Access to Kafka Issues
  To assist with identifying and resolving issues, Oracle SQL access to Kafka provides trace files, message tables, operation results tables, and a state column in the cluster table.

22.18.1 How to Diagnose Oracle SQL Access to Kafka Issues

If you encounter issues with Oracle SQL access to Kafka, then use these guidelines to determine the cause, and resolve the issue.

The following are the main diagnostic issues for Oracle SQL access to Kafka:

Failures to establish an initial connection

Errors of this type are as follows:

- Incorrect startup server list
- Incorrect credential information
- Networking configuration issues

Failures on first access

Failures on first access when calling `DBMS_KAFKA CREATE_LOAD_APP`, `CREATE_STREAMING_APP`, or `CREATE_SEEKABLE_APP` typically have the following causes:

- Missing or incorrect topic
- Connection issues

Failures during record selection

Failures of this type typically have the following causes:

- Connection issues
- Internal metadata or logic issues
- Missing records
- Parsing errors where the Oracle SQL access to Kafka view shape does not match the input.

Failure for an Oracle application and Oracle SQL access to Kafka views to keep up with Kafka data input.

Failures of this type require resource tuning. They occur when the ingestion rate of rows into a topic in a Kafka cluster comes close to or exceeds the Oracle Database ability to consume Kafka records, such that after a period of time, unread records in Kafka become aged out by Kafka before they are consumed by Oracle Database.

Avoid or correct this kind of error by determining the workload. For example, check the frequency of querying, the typical number of records processed per query per Oracle SQL access to Kafka view, the degree of parallelism being used, and the time spent by an application performing analysis. When you have determined the workload, then ensure that the application stack can meet it. Size your resources so that the application and Oracle Database can process peak Kafka records without stressing either the application or Oracle Database resources.

If you find that throughput rates start increasing, then several things can help. For example: increase the degree of parallelism for the application user, start more application instances, or add partitions to the Kafka cluster.

Example 22-25 Resolving an Oracle SQL Access to Kafka (OSAK) Application Error

Suppose your OSAK application `EXAMPLEAPP` is loading data from the Kafka cluster `EXAMPLECLUSTER`, and you receive an error such as the following:

```
ORA-62721: The specified parallel hint [%0!s] exceeds the granule count {%1!
s}.
```

The cause of this error is that the specified value was greater than the maximum possible parallelism, which is determined by the granule count. How do you resolve such an error?

The `parallel_hint` parameter on `LOAD_TEMP_TABLE` and `EXECUTE_LOAD_APP` is related to the degree or parallelism (or DOP), which determines how many parallel process can be run for a given select statement to fetch the data. To leverage parallel queries to their potential, the `parallel_hint` parameter must be set between 2 and the maximum allowed DOP. The maximum DOP is either the maximum allowed for the user making the call, or the number of partitions associated with the OSAK view, whichever is smaller. The cause is that either the database or the user account running the application has exceeded the maximum allowed DOP.

To resolve this issue, specify a value less than or equal to the granule count. The granule count can be determined by calling the `DBMS_KAFKA.GET_GRANULE_COUNT` function:

```
DECLARE
  v_dop INTEGER;
BEGIN
   LOOP
       v_dop :=
SYS.DBMS_KAFKA.GET_GRANULE_COUNT('ORA$DKVGTT_EXAMPLECLUSTER_EXAMPLEAPP_0');
```

```
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;
```

22.18.2 Identifying and Resolving Oracle SQL Access to Kafka Issues

To assist with identifying and resolving issues, Oracle SQL access to Kafka provides trace files, message tables, operation results tables, and a state column in the cluster table.

Determine the nature of the issue you see, and then use the utility available to you to identify and address the issue:

- **Connection issue, logic issue, or Kafka access layer (Oracle executables called by a Kafka data select)** Check the trace file. Also, you can check the state column in the `sys.user_kafka_clusters` table.
- **Exceptions from DBMS_KAFKA and DBMS_KAFKA_ADM APIs**: Review error messages in the `sys.user_kafka_messages` table.
- **Operations runtime issue**: Review messages in the `sys.user_kafka_ops_results` table when the performance of Oracle SQL access to Kafka data retrieval is not as expected.

Example 22-26 Connection issue, Logic Issue or Kafka access layer issue

Use the trace file to identify the issue.

- For connection related issues, the details are available from the view object tracing. To enable, either add the event to the `init.ora` file or use the alter system command to update the system during runtime:

  Add the following entry to the initialization file (`init.ora`):

  ```
  event='trace[KGRK] disk highest'
  ```

  Alter the system:

  ```
  alter system set events 'trace[KGRK] disk highest';
  ```

> **Note:**
>
> Updates to the `init.ora` file require a restart of the database to take effect.

- For logic-related errors, all error paths contain tracing. All messages are prefaced with by the string `kubsCRK`. These logic errors will also result in SQL exceptions being raised.
- The tracing output for the Kafka access layer of an Oracle SQL access to Kafka application is enabled by calling `DBMS_KAFKA.SET_TRACING` with the enable argument passed as `TRUE`. The tracing output is disabled by calling the same function with the enable argument passed as `FALSE`.

  For example:

  To enable tracing for a cluster named `ExampleCluster`, with the application is `ExampleApp`, enter the following:

  ```
  DBMS_KAFKA.SET_TRACING('ExampleCluster', 'ExampleApp', true)
  ```

  To disable tracing for that cluster, enter the following:

  ```
  DBMS_KAFKA.SET_TRACING('ExampleCluster', 'ExampleApp', false)
  ```

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

If you determine that the issue is a connection issue, then check the **State** column in the `sys.user_kafka_clusters` table. The connection levels are designated by numeric values:

- **CONNECTED (0)**: This state indicates that the connection to the Kafka cluster has been established. Errors that occur while the connection is established indicate an issue with requesting the Kafka data. To identify the issue, enable tracing by using the `DBMS_KAFKA.SET_TRACING` API, reproduce the problem, and then check the associated trace file for the session for messages containing '`kubsCRK`". Also check for messages in the `user_kafka_messages` table.
- **MAINTENANCE (1)**: This state indicates that the connection to the Kafka cluster has been established, but errors that occur while the connection is established indicate an issue requesting the Kafka data. To resolve this issue, enable tracing using the `DBMS_KAFKA.SET_TRACING API`, reproduce the problem, and then check the associated trace file for the session for messages containing `kubsCRK`. Also check for messages in the `user_kafka_messages` table.
- **BROKEN (2)**: This state indicates that a connection cannot be reestablished to the Kafka cluster. Look for errors in the trace file for the facility KUBD, and in the message table.
- **DEREGISTERED (3)**: This state indicates that the OSAK administrator has forced the cluster to be deregistered, and the associated Oracle SQL access to Kafka views should no longer be accessed. This is expected behavior, and not an error.

Example 22-27 PL/SQL Package issues

Check the `Sys.user_kafka_messages table`. This table contains any messages logged within the last three days. The data is automatically purged of older data once a day. The messages are also removed if the OSAK views associated with the data are dropped.

Example 22-28 Operations Runtime Issue

If the number of rows retrieved using a `SELECT` statement appears to be less than expected, then use the data in the `sys.user_kafka_ops_results` table to review the number of records read from Kafka for the last selection.

The `SELECT` only contains rows that parsed correctly, so the difference between the rows retrieved and Kafka records read indicates that not all data in the Kafka topic is in the format specified during the `DBMS_KAFKA CREATE_LOAD_APP`, `CREATE_STREAMING_APP`, or `CREATE_SEEKABLE_APP` call.

If the Kafka topic data is not in the specified format, then the answers are as follows:

1. Fix the producers publishing to the Kafka cluster.
2. Drop and recreate the application so that it provides the proper format (reference table for DSV, Avro schema for AVRO).
3. For JSON data, before you drop and recreate the application, check to see if the data exceeds the maximum column length in the `VARCHAR2 VALUE` column. If the data is larger than the maximum, then you can drop and recreate the application, but this time add the option `"jsond" : "clob"` to the options parameter. This option enables OSAK to create the column as a character large object (CLOB) column, instead of the default maximum sized `VARCHAR2`.