17
ORACLE_BIGDATA Access Driver

With the `ORACLE_BIGDATA` access driver, you can access data stored externally (in object stores or file systems) as if that data was stored in tables in an Oracle Database.

- Accessing External Data Using the ORACLE_BIGDATA Driver
  You can use the `ORACLE_BIGDATA` driver to access data located in external file systems and object stores.
- Enabling and Configuring your Database for Object Storage Access
  Accessing data residing in object stores or other external places with HTTPS requires the installation and configuration of `DBMS_CLOUD`.
- Setting Up Credentials and Location Parameters for Object Stores
  You create credential objects and then specify the object store URI.
- ORACLE_BIGDATA Accessing Files
  To use `ORACLE_BIGDATA`, you provide information in an access parameter to indicate how to access and parse the data.
- ORACLE_BIGDATA Accessing Apache Iceberg
  See how to use the Apache Iceberg open table format with the Oracle Big Data Access Driver
- ORACLE_BIGDATA Accessing Delta Sharing
  See how to use the Linux Foundation Projects Delta Sharing data sharing protocol with the Oracle Big Data Access Driver
- ORACLE_BIGDATA Accessing JSON Documents File Type
  See how to use a native JSON reader format (`jsondoc`) for documents stored in object storage or local directories.

17.1 Accessing External Data Using the ORACLE_BIGDATA Driver

You can use the `ORACLE_BIGDATA` driver to access data located in external file systems and object stores.

If you are accessing files residing in the file system, then you must have access to the directory object where the files reside. More details can be found in "Location of Data Files and Output Files"

If you are using Object Storage, then there are three steps required to access data in an object store:

1. **Enable your database, users, and roles to safely access data in Object stores**

   .

2. **Create a credential object (not required for public buckets).**

   A **credential object** stores object store credentials in an encrypted format. The identity specified by the credential must have access to the underlying data in the object store.

3. **Create an external table or query using an inline external table**. The access driver type must be `ORACLE_BIGDATA`. The `CREATE TABLE` statement must reference the credential object, which provides authentication to access the object store. The table that you create also requires a `LOCATION` clause, which provides the URI to the files within the object store.

   For public buckets, the `CREDENTIAL` is not required.

17.2 Enabling and Configuring your Database for Object Storage Access

Accessing data residing in object stores or other external places with HTTPS requires the installation and configuration of `DBMS_CLOUD`.

For details of how to install and configure `DBMS_CLOUD` and how to enable safe access for individual users and roles, see:

Related Topics

- Using the DBMS_CLOUD Family of Packages

17.3 Setting Up Credentials and Location Parameters for Object Stores

You create credential objects and then specify the object store URI.

- How to Create a Credential for Object Stores
  To create your credential object, use the `DBMS_CLOUD.CREATE_CREDENTIAL` procedure.
- How to Define the Location Clause for Object Storage
  Use these examples to see how you can specify the object store URI, depending on its source.

17.3.1 How to Create a Credential for Object Stores

To create your credential object, use the `DBMS_CLOUD.CREATE_CREDENTIAL` procedure.

The credential object contains the username and password information needed to access the object store. Depending on your use case, you can use either an authorizatoin (auth) token, or use Oracle Cloud Infrastructure (OCI) native credentials. If you work with OCI Object Storage, then Oracle recommends that you use the OCI native method.

> **Note:**
>
> You must have the `DBMS_CLOUD` package installed.

- Creating the Credential Object with DBMS_CREDENTIAL.CREATE_CREDENTIAL
  The `DBMS_CLOUD.CREATE_CREDENTIAL` procedure enables you to authenticate access to an external object store.

17.3.1.1 Creating the Credential Object with DBMS_CREDENTIAL.CREATE_CREDENTIAL

The `DBMS_CLOUD.CREATE_CREDENTIAL` procedure enables you to authenticate access to an external object store.

- Auth Token-Based Credentials
  When you are working with Cloud services that require username and an auth token for access, use this method, replacing the values with the values required for your service.
- Native Oracle Cloud Infrastructure (OCI) Credentials
  When you are working with OCI Object Storage, use this method.

17.3.1.1.1 Auth Token-Based Credentials

When you are working with Cloud services that require username and an auth token for access, use this method, replacing the values with the values required for your service.

Example 17-1 Auth Token-Based Credentials

```
BEGIN
DBMS_CLOUD.CREATE_CREDENTIAL(
    credential_name => 'AUTH_TOKEN_CRED',
    username        => 'username@example.com',
    password        => 'auth_token');
END;
```

Related Topics

- CREATE_CREDENTIAL Procedure

17.3.1.1.2 Native Oracle Cloud Infrastructure (OCI) Credentials

When you are working with OCI Object Storage, use this method.

Example 17-2 Native Oracle Cloud Infrastructure (OCI) Credentials (Preferred for OCI Object Storage)

Using OCI credentials enables you to provide tenancy and user details in a secure way.

In the following example, *`OCI_CRED`* is the Oracle Cloud Infrastructure user name, *`ocid1.user.oc1..aaaaa...`* is the Oracle Cloud Identifier (OCID), *`ocid1.tenancy.oc1..aabbb...`* is the Oracle Cloud tenancy identifier, *`MIIEogIBAAKCAQEAtUnx...JEBg=`* is the SSH private key, and *`f2:db:f9:18:a4:aa:...`* is the public key fingerprint:

```
BEGIN
DBMS_CLOUD.CREATE_CREDENTIAL (
       credential_name => 'OCI_CRED',
       user_ocid       => 'ocid1.user.oc1..aaaaa...',
       tenancy_ocid    => 'ocid1.tenancy.oc1..aabbb...',
       private_key     => 'MIIEogIBAAKCAQEAtUnx...JEBg=',
       fingerprint     => 'f2:db:f9:18:a4:aa:...');
END;
```

Related Topics

- CREATE_CREDENTIAL Procedure

17.3.2 How to Define the Location Clause for Object Storage

Use these examples to see how you can specify the object store URI, depending on its source.

`LOCATION` is a URI pointing to data in the object store. Currently supported object stores are Oracle Object Store, Amazon S3 and Azure Blob Storage. To see a full list, refer to "CREATE_CREDENTIAL Procedure" in *Oracle Database PL/SQL Packages and Types Reference*:
DBMS_CLOUD CREATE_CREDENTIAL Procedure

In the examples, the following variables are used:

- *`region`* – tenancy region
- *`container`* – name of a container resource
- *`namespace`* – namespace in a region
- *`bucket`* – a logical container for storing objects that has a globally unique identifier
- *`objectname`* – a unique identifier for an object in a bucket
- *`storage_account`* – the name of the Azure Storage account used to access the Azure Blob Storage.

Example 17-3 Native Oracle Cloud Infrastructure Object Storage

```
location ('https://objectstorage.region.oraclecloud.com/n/namespace/b/
bucket/o/objectname')
```

Example 17-4 Oracle Cloud Infrastructure Object Storage

```
location ('https://swiftobjectstorage.region.oraclecloud.com/v1/namespace/
bucket/objectname'
```

Example 17-5 Amazon Web Service AWS S3 Storage Format

```
location ('https://s3.region.amazonaws.com/bucket/objectname')
```

Example 17-6 Microsoft Azure Blob Storage Format

```
location ('https://storage_account.blob.core.windows.net/container/
objectname')
```

Related Topics

- Oracle Cloud Infrastructure Overview of Object Storage

17.4 ORACLE_BIGDATA Accessing Files

To use `ORACLE_BIGDATA`, you provide information in an access parameter to indicate how to access and parse the data.

To access the external object store, you define the file format type in the access parameter `com.oracle.bigdata.fileformat`, using one of the following values: `csv`, `textfile`, `avro`, `parquet`, `jsondoc`, or `orc`:

```
com.oracle.bigdata.fileformat=[csv|textfile|avro|parquet|orc|jsondoc]
```

You can also use `ORACLE_BIGDATA` to access local files for testing or simple querying. In this case, the `LOCATION` field value is the same as what you would use for `ORACLE_LOADER`. You can use an Oracle directory object followed by the name of the file in the `LOCATION` field. For local files, a credential object is not required. However, you must have privileges over on the directory object in order to access the file. For a list of all files, see:

ORACLE_BIGDATA Access Parameters

- Syntax Rules for Specifying Properties
  The properties are set using keyword-value pairs in the SQL `CREATE TABLE ACCESS PARAMETERS` clause and in the configuration files.
- ORACLE_BIGDATA Common Access Parameters
  There is a set of access parameters that are common to all file formats. Some parameters are unique to specific file formats.
- ORACLE_BIGDATA Specific Access Parameters
  Avro, Parquet, Textfile and CSV all have specific access parameters.

17.4.1 Syntax Rules for Specifying Properties

The properties are set using keyword-value pairs in the SQL `CREATE TABLE ACCESS PARAMETERS` clause and in the configuration files.

The syntax must obey these rules:

- The format of each keyword-value pair is a **keyword**, which can be a colon or equal sign, and a **value**. The following are valid keyword-value pairs:

  ```
  keyword=value
  keyword:value
  ```

  The value is everything from the first non-whitespace character after the separator to the end of the line. Whitespace between the separator and the value is ignored. Trailing whitespace for the value is retained.
- A property definition can span multiple lines. When this happens, precede the line terminators with a backslash (escape character), except on the last line. For example:

  ```
  Keyword1=  Value part 1 \
             Value part 2 \
             Value part 3
  ```

- Special characters can be embedded in a property name or property value by preceding the character with a backslash (escape character). The following table describes the special characters:

Table 17-1 Special Characters in Properties

Escape Sequence	Character
\b	Backspace (\u0008)
\t	Horizontal tab (\u0009)
\n	Line feed (\u000a)
\f	Form feed (\u000c)
\r	Carriage return (\u000d)
"	Double quote (\u0022)
'	Single quote (\u0027)
\	Backslash (\u005c) When multiple backslashes are at the end of the line, the parser continues the value to the next line only for an odd number of backslashes.
\u*xxxx*	Unicode code point.

> **Note:**
>
> When multiple backslashes are at the end of a line, the parser continues the value to the next line only for an odd number of backslashes.

17.4.2 ORACLE_BIGDATA Common Access Parameters

There is a set of access parameters that are common to all file formats. Some parameters are unique to specific file formats.

Common Access Parameters

The following table lists parameters that are common to all file formats accessed through ORACLE_BIGDATA. The first column identifies each access parameter common to all data file types. The second column describes each parameter.

> **Note:**
>
> Parameters that are specific to a particular file format cannot be used for other file formats

Table 17-2 Common Access Parameters

Common Access Parameter	Description
com.oracle.bigdata.fileformat	Specifies the format of the file. The value of this parameter identifies the reader that processes the file. Each reader can support additional access parameters that may or may not be supported by other readers. **Valid values:** parquet, orc, textfile, avro, csv, jsondoc • parquet - file uses Parquet data file format • orc - file uses ORC columnar storage file format • textfile - file uses text file format • avro - file uses Avro file format • csv - file uses CSV text file format • jsondoc - reads a JSON file. The JSON values are mapped to a single JSON column that may be queried using SQL/JSON. **Default:** parquet
com.oracle.bigdata.log.opt	Specifies whether log messages should be written to a log file. When none is specified, then no log file is created. If the value is normal, then log file is created when the file reader decides to write a message. It is up to the file reader to decide what is written to the log file. **Valid values:** normal, none **Default:** none.
com.oracle.bigdata.log.qc	Specifies the name of the log file created by the parallel query coordinator. This parameter is used only when com.oracle.bigdata.log.opt is set to normal. The valid values are the same as specified for com.oracle.bigdata.log.qc .
com.oracle.bigdata.log.exec	Specifies the name of the log file created during query execution. This value is used (and is required) only when com.oracle.bigdata.log.exec is set to normal. The valid values are the same as specified for in ORACLE_HIVE and ORACLE_HDFS. **Valid values:** normal, none **Default:** none.
com.oracle.bigdata.credential.name	Specifies the credential object to use when accessing data files in an object store. This access parameter is required for object store access. It is not needed for access to files through a directory object or for data stored in public buckets. The name specified for the credential must be the name of a credential object in the same schema as the owner of the table. Granting a user SELECT or READ access to this table means that credential will be used to access the table. Use DBMS_CREDENTIAL.CREATE_CREDENTIAL in the DBMS_CREDENTIAL PL/SQL package to create the credential object. For example: exec dbms_credential.create_credential(credential_name => '*MY_CRED*',username =>'*username*', password => '*password*'); In the CREATE TABLE statement, set the value of the credential parameter to the name of the credential object. For example: com.oracle.bigdata.credential.name=MY_CRED

Table 17-2 (Cont.) Common Access Parameters

Common Access Parameter	Description
`com.oracle.bigdata.credential.schema`	Specifies the schema in which the credential object for accessing **Object Stores** is created. This parameter is used in the `ACCESS PARAMETERS` clause when creating external tables with the `ORACLE_BIGDATA` access driver.

17.4.3 ORACLE_BIGDATA Specific Access Parameters

Avro, Parquet, Textfile and CSV all have specific access parameters.

- Avro-Specific Access Parameters
  In addition to common access parameters, there are parameters that apply only to the Avro file format.
- Examples of Creating External Tables with Avro Files
  The following examples demonstrate how to query and create external tables using Avro files stored in Oracle Cloud Object Storage.
- Parquet-Specific Access Parameters
  Some access parameters are only valid for the Parquet file format.
- Examples of Creating External Tables with Avro Files
  The following examples demonstrate how to query and create external tables using Avro files stored in Oracle Cloud Object Storage.
- Textfile and CSV-Specific Access Parameters
  The text file and comma-separated value (`csv`) file formats are similar to the hive text file format.
- Examples of Creating External Tables
  The following examples demonstrate different methods for creating tables and querying external CSV data stored in Oracle Cloud Object Storage.

17.4.3.1 Avro-Specific Access Parameters

In addition to common access parameters, there are parameters that apply only to the Avro file format.

The first column in this table identifies the access parameters specific to the Avro file format and the second column describes the parameter. There is only one Avro-specific parameter at this time.

> **Note:**
>
> Parameters that are specific to a particular file format cannot be used for other file formats.

Table 17-3 Avro-Specific Access Parameters

Avro-Specific Parameter	Description
`com.oracle.bigdata.avro.decimaltpe`	Specifies the representation of a decimal stored in the byte array. **Valid values:** `int`, `integer`, `str`, `string` **Default:** If this parameter is not used, then an Avro decimal column is read assuming byte arrays store the numerical representation of the values (that is, default to `int`) as the Avro specification defines.

17.4.3.2 Examples of Creating External Tables with Avro Files

The following examples demonstrate how to query and create external tables using Avro files stored in Oracle Cloud Object Storage.

- Creating an External Table with Avro File
  This example creates a database object that references external Avro files.
- Creating an Avro File External Table Using DBMS_CLOUD
  This example uses the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure to define an external table with an Avro file format.

17.4.3.2.1 Creating an External Table with Avro File

This example creates a database object that references external Avro files.

Example 17-7 Creating an External Table

```
CREATE TABLE CUSTOMERS_AVRO_XT
(
    CUSTOMER_ID NUMBER,
    FIRST_NAME VARCHAR2(64),
    LAST_NAME VARCHAR2(64),
    EMAIL VARCHAR2(64),
    SIGNUP_DATE DATE
)
ORGANIZATION EXTERNAL
(
    TYPE ORACLE_BIGDATA
    ACCESS PARAMETERS
    (
        com.oracle.bigdata.fileformat=AVRO
        com.oracle.bigdata.credential.name="OCI_CRED"
    )
    LOCATION ('https://objectstorage.us-ashburn-1.oraclecloud.com/n/
my_namespace/b/sales_data/o/customers_avro/')
);
```

17.4.3.2.2 Creating an Avro File External Table Using DBMS_CLOUD

This example uses the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure to define an external table with an Avro file format.

Example 17-8 Creating an External Table Using DBMS_CLOUD

```
BEGIN
    DBMS_CLOUD.CREATE_EXTERNAL_TABLE(
        table_name => 'CUSTOMERS_AVRO_XT',
        credential_name => 'OCI_CRED',
        file_uri_list => 'https://objectstorage.us-
ashburn-1.oraclecloud.com/n/my_namespace/b/sales_data/o/customers_avro/
*.avro',
        format => '{"type": "avro"}'
    );
END;
```

> **Note:**
>
> You don't have to specify column list and types, The column list and types are automatically derrived from file itself.

17.4.3.3 Parquet-Specific Access Parameters

Some access parameters are only valid for the Parquet file format.

The first column in this table identifies the access parameters specific to the Parquet file format and the second column describes the parameter.

> **Note:**
>
> Parameters that are specific to a particular file format can not be used for other file formats.

Table 17-4 Parquet-Specific Access Parameters

Parquet-Specific Access Parameter	Description
`com.oracle.bigdata.prq.binary_as_string`	This is a Boolean property that specifies if the binary is stored as a string. **Valid values:** `true`, `t`, `yes`, `y`, `1`, `false`, `f`, `no`, `n`, `0` **Default:** `true`
`com.oracle.bigdata.prq.int96_as_timestamp`	This is a Boolean property that specifies if `int96` represents a timestamp. **Valid values:** `true`, `t`, `yes`, `y`, `1`, `false`, `f`, `no`, `n`, `0` **Default:** `true`

17.4.3.4 Examples of Creating External Tables with Avro Files

The following examples demonstrate how to query and create external tables using Avro files stored in Oracle Cloud Object Storage.

- Querying Parquet External Data with Inline External Table
  This example queries data directly from an external Parquet file without creating a database object.
- Creating an External Table Referencing Parquet Files
  This example creates a database object that references external Parquet files.
- Creating an External Table Using DBMS_CLOUD Referencing Parquet Files
  This example uses the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure to define an external table with a Parquet file format.

17.4.3.4.1 Querying Parquet External Data with Inline External Table

This example queries data directly from an external Parquet file without creating a database object.

Example 17-9 Querying External Data with Inline External Table

```
SELECT *
FROM EXTERNAL
(
    (
        CUSTOMER_ID NUMBER,
        FIRST_NAME VARCHAR2(64),
        LAST_NAME VARCHAR2(64),
        EMAIL VARCHAR2(64),
        SIGNUP_DATE VARCHAR2(64)
    )
    TYPE ORACLE_BIGDATA
    ACCESS PARAMETERS
    (
        com.oracle.bigdata.fileformat=parquet
        com.oracle.bigdata.credential.name=OCI_CRED
    )
    LOCATION ('https://objectstorage.us-ashburn-1.oraclecloud.com/n/
my_namespace/b/sales_data/o/customers_parquet/')
) t;
```

17.4.3.4.2 Creating an External Table Referencing Parquet Files

This example creates a database object that references external Parquet files.

Example 17-10 Creating an External Table

```
CREATE TABLE CUSTOMERS_PARQ_XT
(
    CUSTOMER_ID NUMBER,
    FIRST_NAME VARCHAR2(64),
    LAST_NAME VARCHAR2(64),
    EMAIL VARCHAR2(64),
    SIGNUP_DATE VARCHAR2(64)
)
ORGANIZATION EXTERNAL
(
    TYPE ORACLE_BIGDATA
    ACCESS PARAMETERS
    (
```

```
        com.oracle.bigdata.fileformat=parquet
        com.oracle.bigdata.credential.name=OCI_CRED
    )
    LOCATION ('https://objectstorage.us-ashburn-1.oraclecloud.com/n/
my_namespace/b/sales_data/o/customers_parquet/')
);
```

17.4.3.4.3 Creating an External Table Using DBMS_CLOUD Referencing Parquet Files

This example uses the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure to define an external table with a Parquet file format.

Example 17-11 Creating an External Table Using DBMS_CLOUD

```
BEGIN
    DBMS_CLOUD.CREATE_EXTERNAL_TABLE(
        table_name => 'CUSTOMERS_PARQ_XT',
        credential_name => 'OCI_CRED',
        file_uri_list => 'https://objectstorage.us-
ashburn-1.oraclecloud.com/n/my_namespace/b/sales_data/o/customers_parquet/
*.parquet',
        format => '{"type": "parquet"}'
    );
END;
```

> **Note:**
>
> You don't have to specify column list and types, The column list and types are automatically derrived from file itself.

17.4.3.5 Textfile and CSV-Specific Access Parameters

The text file and comma-separated value (`csv`) file formats are similar to the hive text file format.

Text file and CSV file formats read text and `csv` data from delimited files. `ORACLE_BIGDATA` automatically detects the line terminator (either `\n`, `\r`, or `\r\n`). By default, it assumes the fields in the file are separated by commas, and the order of the fields in the file match the order of the columns in the external table.

Note:

Parameters that are specific to a particular file format cannot be used for other file formats.

Table 17-5 Textfile and CSV-Specific Access Parameters

Textfile-Specific Access Parameter	Description
com.oracle.bigdata.buffersize	Specifies the size of the input/output (I/O) buffer used for reading the file. The value is the size of the buffer in kilobytes. Note that the buffer size is also the largest size that a record can be. If a format reader encounters a record larger than this value, then it will return an error. **Default:** 1024
com.oracle.bigdata.blankasnull	When set to true, loads fields consisting of spaces as null. **Valid values**: true, false **Default**: false **Example**: com.oracle.bigdata.blankasnull=true
com.oracle.bigdata.characterset	Specifies the character set of source files. **Valid values**: UTF-8 **Default**: UTF-8 **Example**: com.oracle.bigdata.characterset=UTF-8
com.oracle.bigdata.compressiontype	If this parameter is specified, then the code tries to decompress the data according to the compression scheme specified. **Valid values:** gzip, bzip2 , zlib, detect **Default:** no compression If detect is specified, then the format reader tries to determine which of the supported compression methods was used to compress the file.
com.oracle.bigdata.conversionerrors	If a row has data type conversion errors, then the related columns are stored as null, or the row is rejected. **Valid values**: reject_record, store_null **Default**: store_null **Example**: com.oracle.bigdata.conversionerrors=reject_record
com.oracle.bigdata.csv.rowformat.nulldefinedas	Specifies the character used to indicate the value of a field is NULL. If the parameter is not specified, then there is no value.

Table 17-5 (Cont.) Textfile and CSV-Specific Access Parameters

Textfile-Specific Access Parameter	Description
com.oracle.bigdata.csv.rowformat.fields.terminator	Specifies the character used to separate the field values. The character value must be wrapped in single-quotes. Example: '|'. **Default:** ','
com.oracle.bigdata.csv.rowformat.fields.escapedby	Specifies the character used to escape any embedded field terminators or line terminators in the value for fields. The character value must be wrapped in single quotes. Example: '\'.
com.oracle.bigdata.dateformat	Specifies the date format in the source file. The format option Auto checks for the following formats: J, MM-DD-YYYYBC, MM-DD-YYYY, YYYYMMDD HHMISS, YYMMDD HHMISS, YYYY.DDD, YYYY-MM-DD **Default**: yyyy-mm-dd hh24:mi:ss **Example**: com.oracle.bigdata.dateformat= "MON-RR-DDHH:MI:SS"
com.oracle.bigdata.fields	Specifies the order of fields in the data file. The values are the same as for com.oracle.bigdata.fields in ORACLE_HDFS, with one exception – in this case, the data type is optional. Because the data file is text, the text file reader ignores the data types for the fields, and assumes all fields are text. Because the data type is optional, this parameter can be a list of field names.
com.oracle.bigdata.ignoreblanklines	Blank lines are ignored when set to true. **Valid values**: true, false **Default**: false **Example**: com.oracle.bigdata.ignoreblanklines=true
com.oracle.bigdata.ignoremissingcolumns	Missing columns are stored as null. **Valid values**: true **Default**: true **Example**: com.oracle.bigdata.ignoremissingcolumns=true
com.oracle.bigdata.json.ejson	Specifies whether to enable extended JSON. **Valid values**: true, t, yes, y, 1, false, f, no, n, 0 **Default**: true

Table 17-5 (Cont.) Textfile and CSV-Specific Access Parameters

Textfile-Specific Access Parameter	Description
	Example: `com.oracle.bigdata.jason.ejson=yes`
`com.oracle.bigdata.json.path`	A JSON path expression that identifies a sequence of nested JSON values, which will be mapped to table rows. **Valid values**: String property **Default**: `null` **Example**: `'$.data[*]'`
`com.oracle.bigdata.json.unpackarrays`	Specifies whether to unbox the array found in JSON files. The file consists of an array of JSON objects. The entire file is a grammatically valid JSON doc. An example of such a file is `[{"a":1},{"a":2},{"a":3}]`. **Valid values**: `true`, `t`, `yes`, `y`, `1`, `false`, `f`, `no`, `n`, `0` **Default**: `false` **Example**: `com.oracle.bigdata.json.unpackarrays=true`
`com.oracle.bigdata.quote`	Specifies the quote character for the fields. The quote characters are removed during loading when specified. **Valid values**: character **Default**: Null, meaning no quote **Example:** `com.oracle.bigdata.csv.rowformat.quotecharacter='"'`
`com.oracle.bigdata.rejectlimit`	The operation errors out after specified number of rows are rejected. This only applies when rejecting records due to conversion errors. **Valid values**: number **Default**: `0` **Example:** `com.oracle.bigdata.rejectlimit=2`
`com.oracle.bigdata.removequotes`	Removes any quotes that are around any field in the source file. **Valid values**: `true`, `false` **Default**: `false` **Example:**`com.oracle.bigdata.removequotes=true`
`com.oracle.bigdata.csv.skip.header`	Specifies how many rows should be skipped from the start of the files. **Valid values**: number **Default**: `0`, if not specified

Table 17-5 (Cont.) Textfile and CSV-Specific Access Parameters

Textfile-Specific Access Parameter	Description
	Example: `com.oracle.bigdata.csv.skip.header=1`
`com.oracle.bigdata.timestampformat`	Specifies the timestamp format in the source file. The format option `AUTO` checks for the following formats: `YYYY-MM-DD HH:MI:SS.FF`, `YYYY-MM-DD HH:MI:SS.FF3`, `MM/DD/YYYY HH:MI:SS.FF3` **Valid values**: `auto` **Default**: `yyyy-mm-dd hh24:mi:ss.ff` **Example:** `com.oracle.bigdata.timestamptzformat="auto"`
`com.oracle.bigdata.timestampltzformat`	Specifies the timestamp with local timezone format in the source file. The format option `AUTO` checks for the following formats: `DD Mon YYYY HH:MI:SS.FF TZR`, `MM/DD/YYYY HH:MI:SS.FF TZR`, `YYYY-MM-DD HH:MI:SS+/-TZR`, `YYYY-MM-DD HH:MI:SS.FF3`, `DD.MM.YYYY HH:MI:SS TZR` **Valid values**: `auto` **Default**: `yyyy-mm-dd hh24:mi:ss.ff` **Example:** `com.oracle.bigdata.timestampltzformat="auto"`
`com.oracle.bigdata.timestamptzformat`	Specifies the timestamp with timezone format in the source file. The format option `AUTO` checks for the following formats: `DD Mon YYYY HH:MI:SS.FF TZR`, `MM/DD/YYYY HH:MI:SS.FF TZR`, `YYYY-MM-DD HH:MI:SS+/-TZR`, `YYYY-MM-DD HH:MI:SS.FF3`, `DD.MM.YYYY HH:MI:SS TZR` **Valid values**: `auto` **Default**: `yyy-mm-dd hh24:mi:ss.ff` **Example:** `com.oracle.bigdata.timestamptzformat="auto"`
`com.oracle.bigdata.trimspaces`	Specifies how the leading and trailing spaces of the fields are trimmed. **Valid values**: `rtrim`, `ltrim`, `notrim`, `ltrim`, `ldrtrim` **Default**: `notrim`

Table 17-5 (Cont.) Textfile and CSV-Specific Access Parameters

Textfile-Specific Access Parameter	Description
	Example: `com.oracle.bigdata.trimspaces=rtrim`
`com.oracle.bigdata.truncatecol`	If the data in the file is too long for a field, then this option truncates the value of the field rather than rejecting the row or setting the field to NULL. **Valid values**: `true`, `false` **Default**: `false` **Example**: `com.oracle.bigdata.truncatecol=true`

17.4.3.6 Examples of Creating External Tables

The following examples demonstrate different methods for creating tables and querying external CSV data stored in Oracle Cloud Object Storage.

- Creating an External Table Referencing CSV Files
  This example creates a database object that references the external CSV file.
- Creating an External Table with CSV Files Using DBMS_CLOUD
  This example uses the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure to define an external table with detailed specifications for columns and fields.

17.4.3.6.1 Creating an External Table Referencing CSV Files

This example creates a database object that references the external CSV file.

Example 17-12 Creating an External Table

```
CREATE TABLE CUSTOMERS_CSV_XT
(
    CUSTOMER_ID NUMBER,
    FIRST_NAME VARCHAR2(64),
    LAST_NAME VARCHAR2(64),
    EMAIL VARCHAR2(64),
    SIGNUP_DATE DATE
)
ORGANIZATION EXTERNAL
(
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY DATA_PUMP_DIR
    ACCESS PARAMETERS
    (
        RECORDS DELIMITED BY NEWLINE
        FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
        MISSING FIELD VALUES ARE NULL
        SKIP 1
```

```
        (
            CUSTOMER_ID CHAR(10),
            FIRST_NAME CHAR(64),
            LAST_NAME CHAR(64),
            EMAIL CHAR(64),
            SIGNUP_DATE DATE 'YYYY-MM-DD'
        )
    )
    LOCATION ('customers.csv')
)
REJECT LIMIT UNLIMITED;
```

17.4.3.6.2 Creating an External Table with CSV Files Using DBMS_CLOUD

This example uses the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure to define an external table with detailed specifications for columns and fields.

Example 17-13 Creating an External Table Using DBMS_CLOUD

```
BEGIN
    DBMS_CLOUD.CREATE_EXTERNAL_TABLE(
        table_name => 'CUSTOMERS_CSV_XT',
        credential_name => 'OCI_CRED',
        file_uri_list => 'https://objectstorage.us-
ashburn-1.oraclecloud.com/n/my_namespace/b/sales_data/o/customers_csv/*.csv',
        column_list => 'CUSTOMER_ID NUMBER,
                        FIRST_NAME VARCHAR2(256),
                        LAST_NAME VARCHAR2(256),
                        EMAIL VARCHAR2(256),
                        SIGNUP_DATE DATE',
        field_list => 'CUSTOMER_ID CHAR,
                        FIRST_NAME CHAR(256),
                        LAST_NAME CHAR(256),
                        EMAIL CHAR(256),
                        SIGNUP_DATE CHAR date_format DATE MASK "YYYY-MM-DD"',
        format => '{
                        "type": "csv",
                        "delimiter": ",",
                        "skipheaders": 1
                    }'
    );
END;
/
```

17.5 ORACLE_BIGDATA Accessing Apache Iceberg

See how to use the Apache Iceberg open table format with the Oracle Big Data Access Driver

- Apache Iceberg Tables Overview
  The `ORACLE_BIGDATA` Access Driver allows users to access data stored in object stores as if the data resided in Oracle Database tables.

- Supported Configurations for Apache Iceberg
  Oracle supports catalog-based and non-catalog based data, and supports Parquet data formats with the Apache Iceberg table format
- Iceberg-Specific Access Parameters
  Oracle supports `ORACLE_BIGDATA` access parameters that defines Apache Iceberg table access.
- Examples of Table Creation and Inline External Table SQL for Iceberg Tables
  The following examples demonstrate how to create tables and perform queries on Iceberg tables in Oracle Database, leveraging both manifest files and AWS Glue catalogs.

17.5.1 Apache Iceberg Tables Overview

The `ORACLE_BIGDATA` Access Driver allows users to access data stored in object stores as if the data resided in Oracle Database tables.

The `ORACLE_BIGDATA` functionality now extends to include support for Apache Iceberg, a widely adopted open table format that introduces features such as schema evolution, time-travel queries, and fast query planning. Iceberg integration enables efficient data management for external data sets in data lakes.

Key features of Apache Iceberg:

- **Updates/Deletes**: Serializable isolation of updates and deletes enhances consistency.
- **Time-Travel Queries**: You can query historical snapshots.
- **Schema Evolution**: You can manage changes in table schemas without data migration.
- **Partition Evolution**: You can perform logical partitioning without having to perform physical data movement.
- **Extensive Metadata**: With enhanced metadata, you can set up advanced optimizations, such as partition pruning and column statistics.

17.5.2 Supported Configurations for Apache Iceberg

Oracle supports catalog-based and non-catalog based data, and supports Parquet data formats with the Apache Iceberg table format

Catalog-Based

- **AWS Glue Catalog**: Metadata and data are stored in Amazon S3, managed by AWS Glue.

Non-Catalog (File-Based)

1. **Manifest File**: Directly specify the path to the metadata/manifest file for Iceberg tables.

File Formats

Oracle Database supports IParquet data format for Iceberg tables.

17.5.3 Iceberg-Specific Access Parameters

Oracle supports `ORACLE_BIGDATA` access parameters that defines Apache Iceberg table access.

Table 17-6 Iceberg-Specific Access Parameters

Parameter	Description	Mandatory
`com.oracle.bigdata.access_protocol`	Table protocol defnition. This must be set to `iceberg`.	Yes
`com.oracle.bigdata.access_protocol.config`	JSON configuration for catalog details (For example: AWS Glue or OCI Hadoop Catalog).	Optional
`com.oracle.bigdata.fileformat`	File format used in Iceberg tables (For example: `Parquet`	Yes

17.5.4 Examples of Table Creation and Inline External Table SQL for Iceberg Tables

The following examples demonstrate how to create tables and perform queries on Iceberg tables in Oracle Database, leveraging both manifest files and AWS Glue catalogs.

- Creating a Table Pointing to the Manifest File
  This example creates an external table that references a specific manifest file for Iceberg:
- Inline External Table Query (Manifest File Reference)
  In this example, no database object is created. Instead, the query directly references the manifest file.
- Creating a Table Using DBMS_CLOUD
  This example uses the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure to define an external table.
- Creating a Table Using AWS Glue as a Catalog
  This example uses the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure to define an external table.
- Inline External Table Query Using AWS Glue Catalog
  This query uses an inline external table referencing the Amazon Web Service data integration service AWS Glue as the catalog.
- Creating an External Table Using DBMS_CLOUD with AWS Glue Catalog
  This example demonstrates using the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure with the Amazon Web Service data integration service AWS Glue as the catalog.

17.5.4.1 Creating a Table Pointing to the Manifest File

This example creates an external table that references a specific manifest file for Iceberg:

Example 17-14 Creating a Table Pointing to the Manifest File

```
CREATE TABLE CUSTOMERS_ICEBERG
(
    CUSTOMER_ID NUMBER,
    FIRST_NAME VARCHAR2(64),
    LAST_NAME VARCHAR2(64),
    EMAIL VARCHAR2(64),
    SIGNUP_DATE DATE
)
```

```
ORGANIZATION EXTERNAL
(
    TYPE ORACLE_BIGDATA
    DEFAULT DIRECTORY DATA_PUMP_DIR
    ACCESS PARAMETERS
    (
        com.oracle.bigdata.fileformat=parquet
        com.oracle.bigdata.credential.name=AWS_S3_CREDENTIAL
        com.oracle.bigdata.access_protocol=iceberg
    )
    LOCATION ('iceberg:https://sales-data.s3.us-west-2.amazonaws.com/
customers/metadata/00001-27da..ef5.metadata.json')
)
PARALLEL;
```

17.5.4.2 Inline External Table Query (Manifest File Reference)

In this example, no database object is created. Instead, the query directly references the manifest file.

Example 17-15 Inline External Table Query (Manifest File Reference)

```
SELECT *
FROM EXTERNAL
(
    (
        CUSTOMER_ID NUMBER,
        FIRST_NAME VARCHAR2(64),
        LAST_NAME VARCHAR2(64),
        EMAIL VARCHAR2(64),
        SIGNUP_DATE DATE
    )
    TYPE ORACLE_BIGDATA
    ACCESS PARAMETERS
    (
        com.oracle.bigdata.fileformat=parquet
        com.oracle.bigdata.credential.name=AWS_S3_CREDENTIAL
        com.oracle.bigdata.access_protocol=iceberg
    )
    LOCATION ('iceberg:https://sales-data.s3.us-west-2.amazonaws.com/
customers/metadata/00001-27da..ef5.metadata.json')
) t;
```

17.5.4.3 Creating a Table Using DBMS_CLOUD

This example uses the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure to define an external table.

Example 17-16 Creating a Table Using DBMS_CLOUD

```
BEGIN
    DBMS_CLOUD.CREATE_EXTERNAL_TABLE(
        table_name => 'CUSTOMERS_ICEBERG',
        credential_name => 'AWS_S3_CREDENTIAL',
        file_uri_list => 'https://sales-data.s3.us-west-2.amazonaws.com/
```

```
customers/metadata/00001-27da..ef5.metadata.json',
        format =>'{"access_protocol":{"protocol_type":"iceberg"}}'
    );
END;
```

17.5.4.4 Creating a Table Using AWS Glue as a Catalog

This example uses the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure to define an external table.

Example 17-17 Creating a Table Using AWS Glue as a Catalog

This example defines a table that uses the Amazon Web Service data integration service AWS Glue as the Iceberg catalog:

```
CREATE TABLE CUSTOMERS_ICEBERG
(
    CUSTOMER_ID NUMBER,
    FIRST_NAME VARCHAR2(64),
    LAST_NAME VARCHAR2(64),
    EMAIL VARCHAR2(64),
    SIGNUP_DATE DATE
)
ORGANIZATION EXTERNAL
(
    TYPE ORACLE_BIGDATA
    ACCESS PARAMETERS
    (
        com.oracle.bigdata.fileformat=parquet
        com.oracle.bigdata.credential.name=AWS_S3_CREDENTIAL
        com.oracle.bigdata.access_protocol=iceberg
        com.oracle.bigdata.access_protocol.config='{"iceberg_catalog_type":
"aws_glue", "iceberg_glue_region": "us-west-2", "iceberg_table_path":
"sales_db.customers"}'
    )
    LOCATION ('iceberg:')
)
PARALLEL;
```

17.5.4.5 Inline External Table Query Using AWS Glue Catalog

This query uses an inline external table referencing the Amazon Web Service data integration service AWS Glue as the catalog.

Example 17-18 Inline External Table Query Using AWS Glue Catalog

```
SELECT *
FROM EXTERNAL
(
    (
        CUSTOMER_ID NUMBER,
        FIRST_NAME VARCHAR2(64),
        LAST_NAME VARCHAR2(64),
        EMAIL VARCHAR2(64),
        SIGNUP_DATE DATE
```

```
    )
    TYPE ORACLE_BIGDATA
    ACCESS PARAMETERS
    (
        com.oracle.bigdata.fileformat=parquet
        com.oracle.bigdata.credential.name=AWS_S3_CREDENTIAL
        com.oracle.bigdata.access_protocol=iceberg
        com.oracle.bigdata.access_protocol.config='{"iceberg_catalog_type":
"aws_glue", "iceberg_glue_region": "us-west-2", "iceberg_table_path":
"sales_db.customers"}'
    )
    LOCATION ('iceberg:')
) t;
```

17.5.4.6 Creating an External Table Using DBMS_CLOUD with AWS Glue Catalog

This example demonstrates using the `DBMS_CLOUD.CREATE_EXTERNAL_TABLE` procedure with the Amazon Web Service data integration service AWS Glue as the catalog.

Example 17-19 Creating an External Table Using DBMS_CLOUD with AWS Glue Catalog

```
BEGIN
    DBMS_CLOUD.CREATE_EXTERNAL_TABLE(
        table_name => 'CUSTOMERS_ICEBERG',
        credential_name => 'AWS_S3_CREDENTIAL',
        file_uri_list => '',
        format =>'{"access_protocol":
{"protocol_type":"iceberg","protocol_config":{"iceberg_catalog_type":
"aws_glue", "iceberg_glue_region": "us-west-2", "iceberg_table_path":
"sales_db.customers"}}}'
    );
END;
```

17.6 ORACLE_BIGDATA Accessing Delta Sharing

See how to use the Linux Foundation Projects Delta Sharing data sharing protocol with the Oracle Big Data Access Driver

- Delta Sharing Protocol Overview
  The `ORACLE_BIGDATA` Access Driver now supports the Delta Sharing open protocol.
- Supported Configurations for Delta Sharing Protocol
  Oracle supports catalog-based and non-catalog based data, and supports Parquet data formats with the Apache Iceberg table format
- Creating Credentials for Delta Sharing
  Use these examples to see how you can create Bearer Token credential objects or client ID/Secret credential objects to access Delta Sharing.
- Listing and Describing Delta Share Metadata
  See how to use the `LIST` file format to list and the `DESC` file format to describe delta shares, schemas, and tables.
- Delta Sharing Access Parameters
  Oracle supports `ORACLE_BIGDATA` access parameters that define Delta Sharing access.

- Examples of Creating External Tables for Delta Sharing
  The following examples demonstrate how to create tables using Bearer Token credentials and using client ID/Secret credentials.

17.6.1 Delta Sharing Protocol Overview

The `ORACLE_BIGDATA` Access Driver now supports the Delta Sharing open protocol.

The `ORACLE_BIGDATA` Access Driver supports the Linux Open Project Delta Sharing protocol, which is an open protocol based on Parquet that provides secure and scalable real-time sharing of large datasets.

17.6.2 Supported Configurations for Delta Sharing Protocol

Oracle supports catalog-based and non-catalog based data, and supports Parquet data formats with the Apache Iceberg table format

Access Protocol: `delta-sharing` This protocol is required for accessing Delta Share datasets.

Credential Types

- **Bearer Token**: This credential type is suitable for temporary access. It requires an explicit token refresh.
- **Client ID/Secret**: This credential type enables automatic token refresh. Oracle recommends that you use this type for long-term access to Oracle-managed Delta Share servers.

17.6.3 Creating Credentials for Delta Sharing

Use these examples to see how you can create Bearer Token credential objects or client ID/Secret credential objects to access Delta Sharing.

Example 17-20 Creating Bearer Token Credentials for Delta Sharing

```
BEGIN
    DBMS_CREDENTIAL.CREATE_CREDENTIAL (
        credential_name => 'DATABRICKS',
        username        => 'BEARER_TOKEN',
        password        => 'faaie590d541265bcab1f2de9813274bf233'
    );
END;
```

Example 17-21 Creating Client ID/Secret Credentials for Oracle-Managed Delta Sharing

```
BEGIN
    DBMS_CREDENTIAL.CREATE_CREDENTIAL (
        credential_name => 'DRIVER_CLIENT_ID',
        username        => '_GEVd3cSVYYJtZ...68Q0VINQ..',   -- client ID
        password        => 'IV3gncgr0p6Mk...1WwmQ2uUg..'   -- client secret
    );
END;
```

17.6.4 Listing and Describing Delta Share Metadata

See how to use the `LIST` file format to list and the `DESC` file format to describe delta shares, schemas, and tables.

Example 17-22 Listing Delta Share Content

You can list the content of delta shares, schemas, and tables by using the `LIST` file format. The location path determines the level of detail retrieved. For example:

```
SELECT DISTINCT url
FROM EXTERNAL (
     ( url VARCHAR2(200) )
     TYPE ORACLE_BIGDATA
     ACCESS PARAMETERS (
          com.oracle.bigdata.fileformat = list,
          com.oracle.bigdata.credential.name = DATABRICKS,
          com.oracle.bigdata.access_protocol = delta_sharing
     )
     LOCATION (
          'https://sharing.delta.io/delta-
sharing/#',                          -- Shares
          'https://sharing.delta.io/delta-sharing/
#delta_sharing',             -- Share schemas
          'https://sharing.delta.io/delta-sharing/
#delta_sharing.default'      -- Share tables
     )
     REJECT LIMIT UNLIMITED
);

URL
--------------------------------------------------------------------------------
--
https://sharing.delta.io/delta-sharing/#delta_sharing.default.COVID_19_NYT
https://sharing.delta.io/delta-sharing/#delta_sharing.default.boston-housing
https://sharing.delta.io/delta-sharing/#delta_sharing.default.flight-asa_2008
https://sharing.delta.io/delta-sharing/#delta_sharing.default.lending_club
https://sharing.delta.io/delta-sharing/#delta_sharing.default.nyctaxi_2019
https://sharing.delta.io/delta-sharing/
#delta_sharing.default.nyctaxi_2019_part
https://sharing.delta.io/delta-sharing/#delta_sharing.default.owid-covid-data
https://sharing.delta.io/delta-sharing/#delta_sharing.default
https://sharing.delta.io/delta-sharing/#delta_sharing
```

Example 17-23 Describing Delta Share Tables

To retrieve column definitions for a specific delta share table, use the `DESC` file format. For example:

```
SELECT *
FROM EXTERNAL (
    (
        "path"       VARCHAR2(4000 BYTE),
        "oratype"    VARCHAR2(40 BYTE),
        scale         NUMBER,
```

```
        precision     NUMBER,
        filetype      VARCHAR2(400),
        compression   VARCHAR2(400),
        "partoflist" NUMBER(1),
        "depth"      NUMBER(19)
    )
    TYPE ORACLE_BIGDATA
    ACCESS PARAMETERS (
        com.oracle.bigdata.credential.name = 'DATABRICKS',
        com.oracle.bigdata.fileformat = desc,
        com.oracle.bigdata.access_protocol = delta_sharing
    )
    LOCATION (
        'https://sharing.delta.io/delta-sharing/#DELTA_SHARING.DEFAULT.BOSTON-
HOUSING'
    )
    REJECT LIMIT UNLIMITED
)
ORDER BY "path";

path       oratype          SCALE    PRECISION    FILETYPE    COMPRESSION
partoflist    depth
ID         NUMBER(10)              0           10 Parquet
snappy                              1
age        BINARY_DOUBLE           0           15 Parquet
snappy                              1
black      BINARY_DOUBLE           0           15 Parquet
snappy                              1
chas       NUMBER(10)              0           10 Parquet
snappy                              1
crim       BINARY_DOUBLE           0           15 Parquet
snappy                              1
dis        BINARY_DOUBLE           0           15 Parquet
snappy                              1
indus      BINARY_DOUBLE           0           15 Parquet
snappy                              1
lstat      BINARY_DOUBLE           0           15 Parquet
snappy                              1
medv       BINARY_DOUBLE           0           15 Parquet
snappy                              1
nox        BINARY_DOUBLE           0           15 Parquet
snappy                              1
ptratio    BINARY_DOUBLE           0           15 Parquet
snappy                              1
rad        NUMBER(10)              0           10 Parquet
snappy                              1
rm         BINARY_DOUBLE           0           15 Parquet
snappy                              1
tax        NUMBER(10)              0           10 Parquet
snappy                              1
zn         BINARY_DOUBLE           0           15 Parquet
snappy                              1
```

17.6.5 Delta Sharing Access Parameters

Oracle supports `ORACLE_BIGDATA` access parameters that define Delta Sharing access.

Table 17-7 Iceberg-Specific Access Parameters

Parameter	Description	Mandatory
`com.oracle.bigdata.delta_sharing.token_endpoint`	Token endpoint as defined in the JSON profile file. Required for client ID/secret credentials. .	Optional
`com.oracle.bigdata.access_protocol`	**Value**: `delta-sharing` Protocol for Delta shares.	Yes
`com.oracle.bigdata.fileformat`	**Value**: `parquet` Used when accessing Delta Share tables. This parameter is optional, and the default for Delta Share access. **Value**: `list` Used to derive metadata of Delta shares, share schemas, and share tables **Value**: `desc` Used to obtain the column definitions of a Delta Share.	Optional

17.6.6 Examples of Creating External Tables for Delta Sharing

The following examples demonstrate how to create tables using Bearer Token credentials and using client ID/Secret credentials.

Example 17-24 Using Bearer Token Credential

In this example, we create an external table referencing a known Delta Share table:

```
CREATE TABLE BOSTONHOUSING (
    "ID"        NUMBER(10,0),
    "crim"      BINARY_DOUBLE,
    "zn"        BINARY_DOUBLE,
    "indus"     BINARY_DOUBLE,
    "chas"      NUMBER(10,0),
    "nox"       BINARY_DOUBLE,
    "rm"        BINARY_DOUBLE,
    "age"       BINARY_DOUBLE,
    "dis"       BINARY_DOUBLE,
    "rad"       NUMBER(10,0),
    "tax"       NUMBER(10,0),
    "ptratio"   BINARY_DOUBLE,
    "black"     BINARY_DOUBLE,
    "lstat"     BINARY_DOUBLE,
    "medv"      BINARY_DOUBLE
```

```
)
ORGANIZATION EXTERNAL (
    TYPE ORACLE_BIGDATA
    DEFAULT DIRECTORY "DATA_PUMP_DIR"
    ACCESS PARAMETERS (
        com.oracle.bigdata.credential.name = 'DATABRICKS',
        com.oracle.bigdata.fileformat = parquet,
        com.oracle.bigdata.access_protocol = delta_sharing
    )
    LOCATION (
        'https://sharing.delta.io/delta-sharing/#DELTA_SHARING.DEFAULT.BOSTON-
HOUSING'
    )
)
REJECT LIMIT UNLIMITED
PARALLEL;
Alternatively, user can create table with dbms_cloud package:

BEGIN
  DBMS_CLOUD.CREATE_EXTERNAL_TABLE
  ( TABLE_NAME        => 'BOSTONHOUSING'
   ,CREDENTIAL_NAME   => 'DATABRICKS'
   ,FILE_URI_LIST     => 'https://sharing.delta.io/delta-sharing/
#DELTA_SHARING.DEFAULT.BOSTON-HOUSING'
   ,COLUMN_LIST       => '"ID"       NUMBER(10)
    ,"crim"     BINARY_DOUBLE
    ,"zn"       BINARY_DOUBLE
    ,"indus"    BINARY_DOUBLE
    ,"chas"     NUMBER(10)
    ,"nox"      BINARY_DOUBLE
    ,"rm"       BINARY_DOUBLE
    ,"age"      BINARY_DOUBLE
    ,"dis"      BINARY_DOUBLE
    ,"rad"      NUMBER(10)
    ,"tax"      NUMBER(10)
    ,"ptratio"  BINARY_DOUBLE
    ,"black"    BINARY_DOUBLE
    ,"lstat"    BINARY_DOUBLE
    ,"medv"     BINARY_DOUBLE'
   ,FORMAT            => '{
       "type" : "parquet",
       "access_protocol" : "delta_sharing"
     }'
  );
END;
```

Example 17-25 Using Client ID/Secret Credentials

For Oracle-managed Delta Share tables, include the token endpoint for automatic token refresh. For example:

```
CREATE TABLE DRIVER_REFRESH (
    "DRIVER_ID" INTEGER,
    "NAME"      VARCHAR2(4000 BYTE),
    "POINTS"    INTEGER,
    "TEAM_ID"   INTEGER
```

```
)
ORGANIZATION EXTERNAL (
    TYPE ORACLE_BIGDATA
    DEFAULT DIRECTORY "DATA_PUMP_DIR"
    ACCESS PARAMETERS (
        com.oracle.bigdata.credential.name = 'DRIVER_CLIENT_ID',
        com.oracle.bigdata.delta_sharing.token_endpoint = 'https://
abcde...xyz.adb.ap-tokyo-1.oraclecloudapps.com/ords/jason/oauth/token',
        com.oracle.bigdata.fileformat = parquet,
        com.oracle.bigdata.access_protocol = delta_sharing
    )
    LOCATION (
        'https://abcde...xyz.adb.ap-tokyo-1.oraclecloudapps.com/ords/jason/
_delta_sharing#DELTA23AIPROD.JASON.DRIVER'
    )
)
REJECT LIMIT UNLIMITED
PARALLEL;
```

17.7 ORACLE_BIGDATA Accessing JSON Documents File Type

See how to use a native JSON reader format (`jsondoc`) for documents stored in object storage or local directories.

- Overview of JSON Document Support
  The `ORACLE_BIGDATA` Access Driver supports the `jsondoc` native JSON reader format.
- Access Parameters for JSON Document
  Oracle supports `ORACLE_BIGDATA` access parameters that define the jsondoc file format and are used in the `ACCESS PARAMETERS` clause of the `CREATE TABLE` statement:
- Examples of JSONDOC Usage
  The following examples demonstrate how to access JSON documents using ORACLE_BIGDATA with the `jsondoc` file type.

17.7.1 Overview of JSON Document Support

The `ORACLE_BIGDATA` Access Driver supports the `jsondoc` native JSON reader format.

The `ORACLE_BIGDATA` Access Driver support for `jsondoc` enables seamless interaction with JSON documents stored in object storage or local directories. The JSON reader is designed to parse and query JSON data in various structures, including the following:

- Line-delimited JSON documents
- JSON arrays (with optional path specifications for nested arrays)
- JSON documents with extended JSON (`EJSON`) annotations for specialized data types

This new capability provides the flexibility to handle complex JSON structures, and tp leverage Oracle Database's powerful JSON features for querying and analysis.

17.7.2 Access Parameters for JSON Document

Oracle supports `ORACLE_BIGDATA` access parameters that define the jsondoc file format and are used in the `ACCESS PARAMETERS` clause of the `CREATE TABLE` statement:

Table 17-8 JSON Document Access Parameters

Parameter	Description	Mandatory
`com.oracle.bigdata.fileformat`	Calls the new JSON Reader capabilities **Value**: `jsondoc`	Yes
`com.oracle.bigdata.json.ejson`	Specifies whether to enable extended JSON **Valid values**: `true`, `false` **Default**: `true`	Optional
`com.oracle.bigdata.json.path`	A valid JSON path expression that defines the location from which ORACLE_BIGDATA can load documents. **Default**: Read from the root of the document `$`.	Yes

17.7.3 Examples of JSONDOC Usage

The following examples demonstrate how to access JSON documents using ORACLE_BIGDATA with the `jsondoc` file type.

- Querying Line-Delimited JSON Documents
  The following is an example of a JSON file containing multiple line-delimited JSON documents, and the SQL statement using this file.
- Querying JSON Arrays
  The following is an example of a JSON file containing a single array of JSON objects, and the SQL statement using this file.
- Object wrapped JSON Arrays
  The following is an example of JSON documents wrapped in an outer JSON document.
- Extended JSON (EJSON) Support
  The SQL type JSON is capable of representing extended JSON types such as `TIMESTAMP`, `DOUBLE`, `FLOAT`, and `RAW`.
- Single-JSON Document with Multiline Files
  A single JSON document with multiline files can be mapped to a table, where each JSON file in the directory is mapped to a single row.

17.7.3.1 Querying Line-Delimited JSON Documents

The following is an example of a JSON file containing multiple line-delimited JSON documents, and the SQL statement using this file.

Example 17-26 Querying Line-Delimited JSON Documents

File: `fruit.json`

```
{"name": "apple", "count": 20} {"name": "orange", "count": 42} {"name":
"pear", "count": 10}
```

SQL Statement:

```
CREATE TABLE fruit (data JSON)

ORGANIZATION EXTERNAL

(TYPE ORACLE_BIGDATA ACCESS PARAMETERS

( com.oracle.bigdata.fileformat = jsondoc

   com.oracle.bigdata.credential.name = 'OCI_CRED' )

LOCATION ('https://<objectstorage-location>/fruit.json'));



SELECT f.data."name", f.data."count" FROM fruit f;

name            count
-----------------------------
"apple"         20
"orange"        42
"pear"          10
```

17.7.3.2 Querying JSON Arrays

The following is an example of a JSON file containing a single array of JSON objects, and the SQL statement using this file.

Example 17-27 Querying JSON Arrays

File: `fruit-array.json`

```
[
   {
      "name" : "apple",
      "count": 20
   },
   {
      "name" : "orange",
      "count": 42
   },
   {
      "name" : "pear",
      "count": 10
   }
]
```

SQL Statement:

```
CREATE TABLE fruit (data JSON) ORGANIZATION EXTERNAL (
  TYPE ORACLE_BIGDATA
  DEFAULT DIRECTORY default_dir
ACCESS PARAMETERS (
    com.oracle.bigdata.fileformat = jsondoc
```

```
    com.oracle.bigdata.json.path = $.[*]
    com.oracle.bigdata.credential.name = OCI_CRED
     )
  location ('https://<objectstorage-location>/fruit-array.json')
);

SQL> SELECT f.data."name", f.data."count"
     FROM fruit f;

name            count
-----------------------------
"apple"         20
"orange"        42
"pear"          10
```

17.7.3.3 Object wrapped JSON Arrays

The following is an example of JSON documents wrapped in an outer JSON document.

Example 17-28 Object wrapped JSON Arrays

To use this example, you provide a path (using `com.oracle.bigdata.json.path`) to the data that you want to load. The path must lead to an array. The rows are mapped as in the previous example.

File: `fruit-array.json`

```
{
  "last_updated": 1434054678,
  "ttl": 0,
  "version": "1.0",
  "fruit": [
    {"name" : "apple", "count": 20 },
    {"name" : "orange", "count": 42 },
    {"name" : "pear", "count": 10 }
  ]
}

CREATE TABLE fruit (data JSON) ORGANIZATION EXTERNAL (
  TYPE ORACLE_BIGDATA
  DEFAULT DIRECTORY default_dir
  ACCESS PARAMETERS
  (
    com.oracle.bigdata.fileformat = jsondoc
    com.oracle.bigdata.json.path = $.fruit[*]
    com.oracle.bigdata.credential.name = OCI_CRED
     )
  location ('https://objectstorage-location/fruit-wrapped.json')
);

SELECT f.data."name", f.data."count"
     FROM fruit f;

name            count
-----------------------------
"apple"        20
```

```
"orange"       42
"pear"         10
```

17.7.3.4 Extended JSON (EJSON) Support

The SQL type JSON is capable of representing extended JSON types such as `TIMESTAMP`, `DOUBLE`, `FLOAT`, and `RAW`.

Example 17-29 Extended JSON (EJSON) Support

. The JSON text can represent extended JSON types by using the extended JSON format. When set, these `ejson` annotations will be automatically converted to the corresponding types.

File: `fruit-extended.json`

```
{"name" : "apple", "count": 20, "modified":
{"$date":"2020-06-29T11:53:05.439Z"} }
{"name" : "orange", "count": 42 }
{"name" : "pear", "count": 10 }
```

SQL Statement:

```
CREATE TABLE fruit (data JSON) ORGANIZATION EXTERNAL (
  TYPE ORACLE_BIGDATA
  DEFAULT DIRECTORY default_dir
  ACCESS PARAMETERS
  (
    com.oracle.bigdata.fileformat = jsondoc
    com.oracle.bigdata.credential.name = oci_adwc4pm
  )
   location ('https://objectstorage-location/fruit-extended.json')
);

SELECT f.data."count", f.data."modified"
     FROM fruit f
     WHERE f.data."name" = "apple";

count       modified
-----------------------------
20          2020-06
```

17.7.3.5 Single-JSON Document with Multiline Files

A single JSON document with multiline files can be mapped to a table, where each JSON file in the directory is mapped to a single row.

Example 17-30 Single-JSON document, multiline files

A single JSON document with multiline files is a directory containing JSON files where each JSON file (document) in the directory is mapped to a single row in the table. In this case, the directory is `/data`, with the following files:

File: data/apple.json

```
{
   "name" : "apple",
   "count": 42
}
```

File: **data/orange.json**

```
{
   "name" : "orange",
   "count": 5
}
```

File: data/pear.json

```
{
   "name" : "pear",
   "count": 12
}
```

SQL Statement:

```
CREATE TABLE fruit (data JSON) ORGANIZATION EXTERNAL (
  TYPE ORACLE_BIGDATA
  DEFAULT DIRECTORY default_dir
  ACCESS PARAMETERS
  (com.oracle.bigdata.fileformat = jsondoc)
  location ('data/*.json')
);

SQL> SELECT f.data."name", f.data."count"
     FROM fruit f;

name            count
-----------------------------
"apple"         20
"orange"        42
"pear"          10
```