Oracle® Cloud
REST API Reference for Oracle Business Intelligence Cloud Service
E56163-03

November 2015
Preface ................................................................................................................................................................. v
   Audience ........................................................................................................................................................... v
   Related Resources ........................................................................................................................................ v
   Conventions .................................................................................................................................................. vi

1 Getting Started with the REST API
   URIs for Resources in Oracle BI Cloud Service .............................................................................................. 1-1
   Authentication and Authorization of Calls to Oracle BI Cloud Service ......................................................... 1-1
   Methods in the REST API ................................................................................................................................ 1-2
   Required Request Header .................................................................................................................................. 1-2
   Exceptions and Error Codes in the REST API .................................................................................................. 1-2

2 Basic Methods
   About ............................................................................................................................................................... 2-1
   Database TIMESTAMP ..................................................................................................................................... 2-2
   Service TIMESTAMP ...................................................................................................................................... 2-2
   Schema GET ................................................................................................................................................... 2-2
   Schema SIZE ................................................................................................................................................ 2-3

3 Table
   Table GET ..................................................................................................................................................... 3-1
   Table PUT .................................................................................................................................................... 3-4
   Table POST ................................................................................................................................................... 3-6
   Table DELETE ............................................................................................................................................... 3-8

4 Table Statistics
   Table Statistics GET ....................................................................................................................................... 4-1
   Table Statistics PUT ..................................................................................................................................... 4-2

5 Data
   Data PUT ....................................................................................................................................................... 5-1
   Data POST .................................................................................................................................................... 5-4
6 Index

Index GET .................................................................................................................................................. 6-1
Index PUT .................................................................................................................................................. 6-2
Index DELETE ........................................................................................................................................... 6-4
Oracle BI Cloud Service REST API (Representational State Transfer (REST) application programming interface (API)) allows you to programmatically create, manage, and load schemas, tables, and data into Oracle BI Cloud Service.

**Topics:**

- Audience
- Related Resources
- Conventions

**Audience**

REST API Reference for Oracle Business Intelligence Cloud Service is intended for developers who want to programmatically create, manage, and load schemas, tables, and data into Oracle BI Cloud Service.

**Note:** See Data Modeler REST APIs for Oracle BI Cloud Service if you want to programmatically interact with data models in Oracle BI Cloud Service. For example, you can force reports and model objects to display the most recent data by using the Data Modeler REST APIs to clear the cache.

**Related Resources**

For more information, see these Oracle resources:

- Oracle Public Cloud
  
  [http://cloud.oracle.com](http://cloud.oracle.com)
- Using Oracle Business Intelligence Cloud Service
- Preparing Data in Oracle Business Intelligence Cloud Service
- Data Modeler REST API for Oracle Business Intelligence Cloud Service
The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td>monospace</td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
Getting Started with the REST API

Oracle BI Cloud Service REST API allows you to programmatically create, manage, and load schemas, tables, and data into Oracle BI Cloud Service.

Topics

• URIs for Resources in Oracle BI Cloud Service
• Authentication and Authorization of API Calls to Oracle BI Cloud Service
• Methods in the REST API
• Required Request Header
• Exceptions and Error Codes in the REST API

URIs for Resources in Oracle BI Cloud Service
To perform operations on a resource in Oracle BI Cloud Service by using REST API calls, you must specify the fully qualified and unique URI of the resource. The fully qualified URI of a resource has this format:

https://<URL>/<resource_base_path>

• URL is the REST endpoint URL that you received when you subscribed to the service.

• resource_base_path is the base URI of the resource. For example, for tables, the base URI is dataload/v1/tables/TABLE_NAME.

For example, the following is the fully-qualified URI of a table:

https://<URL>/dataload/v1/tables/<TABLE_NAME>

Authentication and Authorization of Calls to Oracle BI Cloud Service
Methods of the Oracle BI Cloud Service REST API require basic authentication (username and password) and proper authorization by application role.

Authentication
The user name and password for a REST API request provide basic authentication.

Authorization
To make requests of the Oracle BI Cloud Service REST API, an authenticated user must have the BI Data Load Author application role.
Methods in the REST API

The Oracle BI Cloud Service REST API supports typical methods.

You can invoke REST API methods for objects in Oracle BI Cloud Service.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Retrieves information about the objects specified in the request URI.</td>
</tr>
<tr>
<td>PUT</td>
<td>Updates the attributes of an object by using the JSON-formatted data in the request body.</td>
</tr>
<tr>
<td>POST</td>
<td>Creates an object by using the JSON-formatted data in the request body.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Deletes the object specified in the request URI.</td>
</tr>
</tbody>
</table>

Required Request Header

Every request to the Oracle BI Cloud Service REST API requires an X-ID-TENANT-NAME header that is used for proper routing. The value for the header is the identity domain, which you find in the URL for the service, as in the following syntax:

<service>-<identity_domain>.<service_type>.<data_center>.oraclecloud.com

CURL is a command-line tool for getting and sending data using URL syntax. You can use cURL to invoke methods of the Oracle BI Cloud Service REST API, including the upload of CSV files.

See the following example of specifying the required request header with a cURL request for getting build information:

```
URL: curl -u <user_name>:<password> -X GET -H "X-ID-TENANT-NAME: <identity_domain>"
https://<URL>/dataload/v1/about
```

Exceptions and Error Codes in the REST API

In the case that a resource doesn’t respond to a method, the REST API returns errors to indicate the status and the nature of the error.

The REST API handles exceptions and returns errors as appropriate.

<table>
<thead>
<tr>
<th>HTTP Response Code</th>
<th>API Exception and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>401 (Unauthorized)</td>
<td>The URL is specified correctly and the user is authenticated, but the user isn’t assigned the proper application role (BI Data Load Author).</td>
</tr>
<tr>
<td>403 — Forbidden</td>
<td>The URL is specified correctly, but the user can’t be authenticated.</td>
</tr>
<tr>
<td>404 — Not Found</td>
<td>An incorrect URL is specified. The request doesn’t reach the REST API layer.</td>
</tr>
<tr>
<td>500 (Internal Server Error)</td>
<td>The request is properly routed to the REST API, but a failure occurs during execution.</td>
</tr>
</tbody>
</table>
You can use various basic methods with Oracle BI Cloud Service REST API.

**Topics:**
- About
- Database TIMESTAMP
- Service TIMESTAMP
- Schema GET
- Schema SIZE

**About**

The About method returns information about the build and usage of Oracle BI Cloud Service.

https://<URL>/dataload/v1/about

Various information is returned in the form of a map:

- **baseVersion:** The name of the version.
- **majorVersion:** A number that indicates the major version.
- **minorVersion:** A number that indicates the minor version.
- **build:** The build information
- **maxNumberOfParallelProcesses:** The maximum number of parallel invocations of the REST API that are allowed.
- **suggestedBatchSize:** The optimal number of rows allowed per each request to load data.
- **maximumBatchSize:** The maximum number of rows allowed per each request to load data.

---

**Note:** When uploading data with Oracle BI Cloud Service REST API, you use the methods that are described in Data. Refrain from sending batches in a size that is significantly larger than that specified in the suggestedBatchSize value. The time required to upload can significantly affect the failure points. Batches of smaller sizes are generally uploaded faster.
Example 2-1  Sample Information Returned

{"suggestedBatchSize":"3000","maxNumberOfParallelProcesses":"4","majorVersion":"1","baseVersion":"V1","minorVersion":"0","build":"11.1.1.8.0.20141217.0255","maxBatchSize":"1000000"}

Example 2-2  Sample cURL Command for About

The following is an example command for using cURL with the About method:


Database TIMESTAMP

The TIMESTAMP method returns the timestamp of the database server where customer data for Oracle BI Cloud Service is stored. The timestamp is returned as a number of milliseconds since January 1, 1970, 00:00:00 GMT.

https://<URL>/dataload/v1/schema/timestamp/

Example 2-3  Sample cURL Command for Timestamp

The following is an example command for using cURL with the TIMESTAMP method to obtain the timestamp of the database server:


Service TIMESTAMP

The TIMESTAMP method returns the timestamp of the web server where Oracle BI Cloud Service is running. The timestamp is returned as a number of milliseconds since January 1, 1970, 00:00:00 GMT.

https://<URL>/dataload/v1/timestamp/

Example 2-4  Sample cURL Command for Timestamp

The following is an example command for using cURL with the TIMESTAMP method to obtain the timestamp of the web server:


Schema GET

The GET method of the Schema resource obtains information about the schema for Oracle BI Cloud Service.

Request URI’s for Schema

Use URI’s to obtain various information about the schema for Oracle BI Cloud Service.

https://<URL>/dataload/v1/schema/ pings the database and returns a “Data Loader Service Alive” message if successful. If unsuccessful, then an error message is returned.

Example 2-5  Sample cURL Command for Get

The following is an example command for using cURL with the GET method to ping the database:
 Schema SIZE

The SIZE method of the Schema resource obtains information about the objects in the schema for Oracle BI Cloud Service.

Request URI’s for Schema

Use URI’s to obtain various information about the size of the objects in the schema for Oracle BI Cloud Service.

https://<URL>/dataload/v1/schema/size/ returns a JSON structure that contains information attributes that describe the objects in the schema.

Response Body for JSON Structure

If you use the URI for schema size, then the response for the objects in the schema contains a JSON structure that includes various attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableSizeInBytes</td>
<td>Specifies the total size in bytes of the table.</td>
</tr>
<tr>
<td>tableCount</td>
<td>Specifies the number of tables.</td>
</tr>
<tr>
<td>indexSizeInBytes</td>
<td>Specifies the total size in bytes of the index.</td>
</tr>
<tr>
<td>indexCount</td>
<td>Specifies the number of indexes.</td>
</tr>
</tbody>
</table>

**Example 2-6 Sample Information Returned**

{"tableSizeInBytes":38141952,"tableCount":10,"indexSizeInBytes":12648448,"indexCount":7"}

**Example 2-7 Sample cURL Commands for SIZE**

The following is an example command for using cURL with the SIZE method to describe the objects in the schema:

curl -u <user_name>:<password> -X GET -H "X-ID-TENANT-NAME: <identity_domain>" 
https://<URL>/dataload/v1/schema/size
A table stores data for Oracle BI Cloud Service. Use methods in the REST API to create, modify, and drop tables. A table name must not contain spaces.

https://<URL>/dataload/v1/tables/<TABLE_NAME>

**Topics:**
- Table GET
- Table PUT
- Table POST
- Table DELETE

**Table GET**

Use the GET method with the Table resource to obtain information about tables. You can list all tables in a schema, use a query filter to specify tables, or use a table name to obtain a table definition.

**Request Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;no table name&gt;</td>
<td>Optional</td>
<td>NA</td>
<td>Lists all tables in the schema.</td>
</tr>
<tr>
<td>&lt;Query_Filter&gt;</td>
<td>Optional</td>
<td>string</td>
<td>Use an asterisk (*) as a wildcard to return a list of all tables that match a pattern.</td>
</tr>
<tr>
<td>&lt;Table_Name&gt;</td>
<td>Optional</td>
<td>string</td>
<td>Specify the name of the table to return a table definition for the table, or an exception if no table with that name is found.</td>
</tr>
</tbody>
</table>

**Response Body**

In cases where the table exists, the GET method with a table name specified in the path returns a description of the table. The returned column attributes include:

- columnName
- dataType
- length
- precision
In cases where the index exists, the GET method returns the following index attributes:

- indexName
- tableName
- isUnique
- isBitMap

The index includes a list of columns with the following attributes:

- columnName
- postn — number
- sortCD — asc or desc

**Example 3-1   Sample Objects Returned**

See an example of objects that are returned.

```json
{
    "name":"SAMP_ACTIONS",
    "columns": [
        {
            "columnName":"MESSAGE",
            "dataType":"VARCHAR",
            "length":512,
            "precision":0,
            "nullable":true,
            "defaultValue":null,
            "sqlType":12
        },
        {
            "columnName":"ORDER_ID",
            "dataType":"DECIMAL",
            "length":32,
            "precision":10,
            "nullable":true,
            "defaultValue":null,
            "sqlType":3
        },
        {
            "columnName":"MESSAGE_TYPE",
            "dataType":"VARCHAR",
            "length":50,"precision":0,
            "nullable":true,
            "defaultValue":null,
            "sqlType":12
        },
        {
            "columnName":"ID",
            "dataType":"DECIMAL",
            "length":32,"precision":10,
            "nullable":true,
            "defaultValue":null,
            "sqlType":3
        }
    ]
}
```
### Example 3-2  Sample Request for All Tables

Get a list of all tables in the schema.

<URL>/dataload/v1/tables
Example 3-3  Sample Request for Tables of Certain Names

Using a query filter, get a list of all tables in the schema whose name starts with either MY or YOUR.

<URL>/dataload/v1/tables?name=MY*&name=YOUR*

Example 3-4  Sample cURL Commands for Get

The following is an example command for using cURL to obtain a list of tables:


The following is an example command for using cURL to obtain a filtered list of tables (for example, all tables whose names begin with SAMP*):


The following is an example command for using cURL to obtain a specific table definition:


Table PUT

Use the PUT method with the Table resource to create a table. Provide a JSON definition of the table with a list of columns, providing attributes for each.

Request Parameters

The PUT method takes a JSON definition of the column listing for a new table with the following column attributes:

- columnName
- dataType
- length
- precision
- nullable
- defaultValue

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;COLUMN_NAME&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies the name of the column in the JSON structure.</td>
</tr>
<tr>
<td>&lt;DATA_TYPE&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies the data type of the column such as VARCHAR, DATE/TIMESTAMP, and NUMBER.</td>
</tr>
<tr>
<td>&lt;Length of Column&gt;</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies the length of the column for VARCHAR and NUMBER data types.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Required?</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>-------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>&lt;PRECISION&gt;</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies the precision for NUMBER data types.</td>
</tr>
<tr>
<td>&lt;NULLABLE&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies whether the column is nullable.</td>
</tr>
<tr>
<td>&lt;DEFAULT VALUE&gt;</td>
<td>Optional</td>
<td>string</td>
<td>Specifies the default value of the column.</td>
</tr>
</tbody>
</table>

**Response Body**

In cases where a table with the specified name already exists, the “Table <TABLE-NAME> already exists” message is returned. When the table is created, the “Table <TABLE-NAME> created” message is returned.

**Sample Request**

Create an ORDERS table with ORDER_STATUS, ORDER_DAY_DT, REVENUE, and ORDER_NUMBER. The following example represents the JSON structure that is provided for the table definition.

```
[
  {
    "columnName":"ORDER_STATUS",
    "dataType":"VARCHAR",
    "length":30,
    "precision":0,
    "nullable":true,
    "defaultValue":null,
  },
  {
    "columnName":"ORDER_DAY_DT",
    "dataType":"TIMESTAMP",
    "length":0,
    "precision":0,
    "nullable":true,
    "defaultValue":null
  },
  {
    "columnName":"REVENUE",
    "dataType":"NUMBER",
    "length":38,
    "precision":7,
    "nullable":true,
    "defaultValue":null
  },
  {
    "columnName":"ORDER_NUMBER",
    "dataType":"DECIMAL",
    "length":38,
    "precision":0,
    "nullable":true,
    "defaultValue":null
  }
]
```
Example 3-5 Sample cURL Command for Put

The following is an example command for using cURL with the PUT method to create a table. In the example, `<table-file>` specifies the name of the file that contains the table definition and `<table-name>` specifies the table to load in the URI, such as SAMP_REVENUE.

```bash
```

Table POST

Use the POST method with the Table resource to modify a table to add columns or to modify their attributes. If the table does not exist, then a table is created. Provide a JSON definition of the table with a list of columns, providing attributes for each.

Request Parameters

The POST method takes a JSON definition of the column listing, with updates or additional columns for a new table with the following column attributes:

- columnName
- dataType
- length
- precision
- nullable
- defaultValue

Request Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;COLUMN_NAME&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies the name of the column.</td>
</tr>
<tr>
<td>&lt;DATA_TYPE&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies the data type of the column such as VARCHAR, DATE/TIMESTAMP, and NUMBER.</td>
</tr>
<tr>
<td>&lt;Length of Column&gt;</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies the length of the column for VARCHAR and NUMBER data types.</td>
</tr>
<tr>
<td>&lt;PRECISION&gt;</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies the precision for NUMBER data types.</td>
</tr>
<tr>
<td>&lt;NULLABLE&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies whether the column is nullable.</td>
</tr>
<tr>
<td>&lt;DEFAULT VALUE&gt;</td>
<td>Optional</td>
<td>string or integer</td>
<td>Specifies the default value of the column.</td>
</tr>
</tbody>
</table>
Response Body
In cases where a table with the same definition already exists, the “Table <TABLE_NAME> definition did not change. No alteration was done.” message is returned. When the table is successfully altered, the "Table <TABLE_NAME> altered" message is returned. When the table does not exist and is created, a “Table <TABLE_NAME> successfully created” message is returned. If the table can’t be altered, then a message with HTTP error code 500 is returned with text such as the following: “The table definition in the request has conflicts with the one in the schema.” The message provides an explanation of the conflict.

Exception
In some circumstances where changes are not permitted on tables with existing data, an exception is returned. Examples include:

- Data type changes
- Length or precision is updated to a length smaller than the existing data in the databases
- Change of a nullable column to not nullable

Example 3-6  Sample Request
Modify an ORDERS table with existing ORDER_STATUS, ORDER_DAY_DT, and REVENUE columns, adding a new ORDER_NUMBER column. The following example represents the JSON structure that is provided for the table definition to include the fourth column.

```
[ 
   { 
      "columnName": "ORDER_STATUS",
      "dataType": "VARCHAR",
      "length": 30,
      "precision": 0,
      "nullable": true,
      "defaultValue": null,
   },
   { 
      "columnName": "ORDER_DAY_DT",
      "dataType": "TIMESTAMP",
      "length": 0,
      "precision": 0,
      "nullable": true,
      "defaultValue": null
   },
   { 
      "columnName": "REVENUE",
      "dataType": "NUMBER",
      "length": 38,
      "precision": 7,
      "nullable": true,
      "defaultValue": null
   },
   { 
      "columnName": "ORDER_NUMBER",
      "dataType": "DECIMAL",
      "length": 38,
      "precision": 0,
   }
]
```
Example 3-7  Sample cURL Command for Post

The following is an example command for using cURL with the POST method to modify a table. In the example, <table-file> specifies the name of the file that contains the table definition and <table-name> specifies the table to modify in the URI, such as SAMP_REVENUE.


Table DELETE

Use the DELETE method with the Table resource to drop a table. This method doesn’t require a request body.

Request Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;TABLE_NAME&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies the name of the table.</td>
</tr>
</tbody>
</table>

Response Body

If the table exists, then a response of “Dropped the table <TABLE_NAME> successfully” is returned. If the table cannot be found, then a response of “Table <TABLE_NAME> does not exist” is returned.

Example 3-8  Sample cURL Command for Delete

The following is an example command for using cURL with the DELETE method to drop a table:

Table Statistics

Use methods in the REST API to collect statistics for tables and obtain information about them.

https://<URL>/dataload/v1/tables/<TABLE_NAME>/statistics

Topics:

• Table Statistics GET
• Table Statistics PUT

Table Statistics GET

Use the GET method with the Table Statistics resource to obtain information about table data, such as the number of rows and columns and when the table was last analyzed.

Request Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Table_Name&gt;</td>
<td>Optional</td>
<td>string</td>
<td>Specify the name of the table to return a table definition for the table, or an exception if no table with that name is found.</td>
</tr>
</tbody>
</table>

Response Body

In cases where the table exists, the GET method with a table name specified in the path returns information about table data. The returned column attributes include:

• numberOfRows
• numberOfColumns
• lastAnalyzedTimestamp

Response Example

See an example of the information that is returned.

{"numberOfRows":1999,"numberOfColumns":20,"lastAnalyzedTimestamp":"2015-01-09 00:27:03"}

Example 4-1 Sample cURL Command for Get

The following is an example command for using cURL with the GET method for the Table Statistics resource to obtain information about table data:
curl -u <user_name>:<password> -X GET -H "X-ID-TENANT-NAME: <identity_domain>"
https://<URL>/dataload/v1/tables/<table-name>/statistics

Table Statistics PUT

Use the PUT method with the Table Statistics resource to gather statistics for a table. Provide a JSON definition of the table with a list of columns, providing attributes for each.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Table_Name&gt;</td>
<td>Optional</td>
<td>string</td>
<td>Specify the name of the table to gather statistics for, or an exception if no table with that name is found.</td>
</tr>
</tbody>
</table>

If the table is analyzed successfully, then a message such as the following is displayed: "Table <TABLE_NAME> analyzed successfully". If there is an error analyzing the table or if the table doesn't exist, then an error is returned.

Example 4-2 Sample cURL Command for Put

The following is an example command for using cURL with the PUT method for the Table Statistics resource to gather statistics for a table:

curl -u <user_name>:<password> -X PUT -H "X-ID-TENANT-NAME: <identity_domain>"
https://<URL>/dataload/v1/tables/<table-name>/statistics
Use the Data resource to call methods of the REST API that insert, update, and delete data in a table.

\[URL\]/dataload/v1/tables/<\text{TABLE_NAME}>/data

**Topics**

- Data PUT
- Data POST
- Data DELETE

### Data PUT

Use the PUT method with the Data resource to insert data into a table that is specified in a URL.

Provide a JSON definition of the table with a list of columns, providing attributes for each. This method updates data in a table that is specified in the URL, using multi-part input (Content-Type: multipart/mixed). The first part is a JSON descriptor (Content-Type: application/json) of the data load. The second part is an input stream (Content-Type: application/octet-stream). Data in the stream can be text data read from comma-separated values (CSV) or in the form of Java objects, which support a wider range of data types. You can also specify either that only records that don’t exist in the table are updated or that all records are updated.

The PUT method supports content that is encoded with gzip. To use the PUT method, encode the data accordingly and add the following header to the request:

Content-Encoding: gzip

#### Request Parameters

Use the following parameters for text streaming; for example inserting data read from CSV values. The method takes a multi-part input (Content-Type: multipart/mixed) that begins with a JSON descriptor. The object includes a columnMaps section for one or more columns. Each column can include the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;\text{name}&gt;)&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies the name of a table column for insertion.</td>
</tr>
<tr>
<td>(&lt;\text{optionalJavaSqlTyp}&gt;)</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies the JDBC SQL type.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Required?</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>&lt;partOfUniqueKey&gt;</code></td>
<td>Optional</td>
<td>string</td>
<td>Specifies whether the column is part of a unique key.</td>
</tr>
<tr>
<td><code>&lt;position&gt;</code></td>
<td>Optional</td>
<td>integer</td>
<td>Specifies the position.</td>
</tr>
</tbody>
</table>

After the columnMaps section, you can include the following parameters that occur only once for the JSON descriptor.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;optionalMaximumErrors&gt;</code></td>
<td>Optional</td>
<td>integer</td>
<td>Specifies how many errors are allowed before process termination.</td>
</tr>
<tr>
<td><code>&lt;removeDuplicates&gt;</code></td>
<td>Optional</td>
<td>Boolean</td>
<td>If true, removes duplicates within the batch. If false, leaves the stream intact.</td>
</tr>
<tr>
<td><code>&lt;optionalWriteMode&gt;</code></td>
<td>Optional</td>
<td>string</td>
<td>Specifies how to write records. If present and set to &quot;Insert all&quot;, all records are inserted into the target table. If present and set to &quot;Insert new&quot;, only records not already present in the database are inserted into the target table. When the parameter is not present, the default mode of &quot;Insert all&quot; is assumed.</td>
</tr>
<tr>
<td><code>&lt;delimiter&gt;</code></td>
<td>Required</td>
<td>string</td>
<td>Specifies the delimiter that separates values in the data stream, if any. Applies only for CSV files and not for Java files. For example, for CSV files, you specify a comma as the delimiter.</td>
</tr>
<tr>
<td><code>&lt;timestampFormat&gt;</code></td>
<td>Required</td>
<td>integer</td>
<td>Specifies the format of timestamps in the data stream. Leave the parameter empty if no timestamps are available. Applies only for CSV files and not for Java files.</td>
</tr>
<tr>
<td><code>&lt;numberOfLinesToSkip&gt;</code></td>
<td>Optional</td>
<td>integer</td>
<td>Specifies how many lines in the stream of data to skip. Applies only for CSV files and not for Java files.</td>
</tr>
</tbody>
</table>

**Example 5-1  Sample Request**

Load a NAME column with data from a CSV file. The following example represents the JSON structure that defines the load.

```json
{
    "columnMaps": [
        {
        "column": {
            "name":"NAME",
            "optionalJavaSqlType":null,
            "partOfUniqueKey":true,
            "position":1,
        }
    ]
```
Request Parameters

Use the following parameters for Java object streaming updates. Java objects in the stream are specified by the presence of an optional parameter in the description JSON object, optionalStreamType, set to “Java”. If the parameter is not present or is set to “Text”, then CSV streaming is assumed. The method takes a multi-part input that begins with a JSON descriptor. The object includes a columnMaps section for one or more columns. Each column can include the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;name&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies the name of a table column for insertion.</td>
</tr>
<tr>
<td>&lt;optionalJavaSqlType&gt;</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies the JDBC SQL type.</td>
</tr>
<tr>
<td>&lt;partOfUniqueKey&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies whether the column is part of a unique key.</td>
</tr>
<tr>
<td>&lt;position&gt;</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies the position.</td>
</tr>
</tbody>
</table>

After the columnMaps section, you can include the following parameters that occur only once for the JSON descriptor.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;optionalMaximumErrors&gt;</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies how many errors are allowed before process termination.</td>
</tr>
<tr>
<td>&lt;removeDuplicates&gt;</td>
<td>Optional</td>
<td>Boolean</td>
<td>If true, removes duplicates within the batch. If false, leaves the stream intact.</td>
</tr>
<tr>
<td>&lt;optionalWriteMode&gt;</td>
<td>Optional</td>
<td>string</td>
<td>Specifies how to write records. If present and set to “Insert all”, all records are inserted into the target table. If present and set to “Insert new”, only records not already present in the database are inserted into the target table. Any other value is invalid and causes denial of the request. When the parameter is not present, the default mode of “Insert all” is assumed.</td>
</tr>
<tr>
<td>&lt;optionalStreamType&gt;</td>
<td>Optional</td>
<td>string</td>
<td>Specifies the type of stream, either CSV or Java object. If present and equal to “Text”, then CSV streaming is expected. If present and equal to “Java”, then Java object streaming is expected. Any other value is invalid and causes denial of the request. When the parameter is not present, the default mode of “Text” is assumed.</td>
</tr>
</tbody>
</table>
Example 5-2  Sample Request

Load a NAME column with data from a Java object. The following example represents the JSON structure that defines the load.

```
{
  "columnMaps":{
    "column":{
      "name":"NAME",
      "optionalJavaSqlType":null,
      "partOfUniqueKey":true,
      "position":1,
      "...
    },
    "optionalMaximumErrors":null,
    "removeDuplicates":true,
    "optionalStreamType":"Java"
  },
```

Example 5-3  Sample cURL Command for Put

The following is an example command for using cURL with the PUT method to insert data into a table. In the example, `<data-file>` specifies the name of the file that contains the data and `<table-name>` specifies the table to insert in the URI, such as SAMP_REVENUE.

```
```

Data POST

Use the POST method with the Data resource to update data in a table that is specified in a URL.

As with the PUT method, the POST method takes multi-part arguments. Data in the stream can be text or in the form of Java objects, which support a wider range of data types.

The POST method supports content that is encoded with gzip. To use the POST method, encode the data accordingly and add the following header to the request:

Content-Encoding: gzip

Request Parameters

Use the following parameters for text streaming updates; for example, updating with data read from CSV values. The method takes a multi-part input (Content-Type: multipart/mixed) that begins with a JSON descriptor. The object includes a columnMaps section for one or more columns. Each column can include the following parameters.
After the columnMaps section, you can include the following parameters that occur only once for the JSON descriptor.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;optionalMaximumErrors&gt;</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies how many errors are allowed before process termination.</td>
</tr>
<tr>
<td>&lt;removeDuplicates&gt;</td>
<td>Optional</td>
<td>Boolean</td>
<td>If true, removes duplicates within the batch. If false, leaves the stream intact.</td>
</tr>
<tr>
<td>&lt;optionalWriteMode&gt;</td>
<td>Optional</td>
<td>string</td>
<td>Specifies how to write records. If present and set to &quot;Upsert&quot;, then new records are inserted and existing records are updated. If present and equal to &quot;Update only&quot;, then only existing records are updated. Any other value is invalid and causes denial of the request. When the parameter is not present, the default mode of &quot;Upsert&quot; is assumed.</td>
</tr>
<tr>
<td>&lt;delimiter&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies the delimiter that separates values in the data stream, if any. Applies only for CSV files and not for Java files. For example, for CSV files, you specify a comma as the delimiter.</td>
</tr>
<tr>
<td>&lt;timestampFormat&gt;</td>
<td>Required</td>
<td>integer</td>
<td>Specifies the format of timestamps in the data stream. Leave the parameter empty if no timestamps are available. Applies only for CSV files and not for Java files.</td>
</tr>
<tr>
<td>&lt;numberOfLinesToSkip&gt;</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies how many lines in the stream of data to skip. Applies only for CSV files and not for Java files.</td>
</tr>
</tbody>
</table>

**Example 5-4 Sample Request**

Update a NAME column with data from a Java object. The following example represents the JSON structure that defines the update.

```json
{
  "columnMaps": {
    "column": {
      "name": "NAME",
      "optionalJavaSqlType": null,
      
```
"partOfUniqueKey":true,
"position":1,
...}
,"optionalMaximumErrors":null,
"removeDuplicates":true,
"optionalWriteMode":"Upsert",
"delimiter":",
"timestampFormat":"yyyy-MM-dd",
"numberOfLinesToSkip":0

Request Parameters

Use the following parameters for Java object streaming updates. Java objects in the stream are specified by the presence of an optional parameter in the JSON descriptor, optionalStreamType, set to "Java". If the parameter is not present or is set to "Text", then CSV streaming is assumed. The method takes a multi-part input (Content-Type: multipart/mixed) that begins with a JSON descriptor. The object includes a columnMaps section for one or more columns. Each column can include the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;name&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies the name of a table column for insertion.</td>
</tr>
<tr>
<td>&lt;optionalJavaSqlType&gt;</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies the JDBC SQL type.</td>
</tr>
<tr>
<td>&lt;partOfUniqueKey&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies whether the column is part of a unique key</td>
</tr>
<tr>
<td>&lt;position&gt;</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies the position.</td>
</tr>
</tbody>
</table>

After the columnMaps section, you can include the following parameters that occur only once for the JSON descriptor.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;optionalMaximumErrors&gt;</td>
<td>Optional</td>
<td>integer</td>
<td>Specifies how many errors are allowed before process termination.</td>
</tr>
<tr>
<td>&lt;removeDuplicates&gt;</td>
<td>Optional</td>
<td>Boolean</td>
<td>If true, removes duplicates within the batch. If false, leaves the stream intact.</td>
</tr>
<tr>
<td>&lt;optionalWriteMode&gt;</td>
<td>Optional</td>
<td>string</td>
<td>Specifies how to write records. If present and set to &quot;Upsert&quot;, then new records are inserted and existing records are updated. If present and set to &quot;Update only&quot;, then only existing records are updated. Any other value is invalid and causes denial of the request. When the parameter is not present, the default mode of &quot;Upsert&quot; is assumed.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Required?</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;optionalStreamType&gt;</td>
<td>Optional</td>
<td>string</td>
<td>Specifies the type of stream, either CSV or Java object. If present and equal to &quot;Text&quot;, then CSV streaming is expected. If present and equal to &quot;Java&quot;, then Java object streaming is expected. Any other value is invalid and causes denial of the request. When the parameter is not present, the default mode of &quot;Text&quot; is assumed.</td>
</tr>
</tbody>
</table>

**Example 5-5  Sample Request**

Update a NAME column with data from a Java object. The following example represents the JSON structure that defines the update.

```json
{
  "columnMaps": [
    {"column": {
      "name": "NAME",
      "optionalJavaSqlType": null,
      "partOfUniqueKey": true,
    },
    "position": 1,
    {...}
  ],
  "optionalMaximumErrors": null,
  "removeDuplicates": true,
  "optionalWriteMode": "Upsert",
  "optionalStreamType": "Java"
},
```

**Example 5-6  Sample cURL Command for Post**

The following is an example command for using cURL with the POST method to update data in a table. In the example, `<table-file>` specifies the name of the file that contains the table definition and `<table-name>` specifies the table to update in the URI, such as SAMP_REVENUE.

```
```

**Data DELETE**

Use the DELETE method with the Data resource to delete data from a table that is specified in a URL. If no arguments are specified, then the table is truncated. You can also specify a filter as JSON object.

**Request Parameters**

Use parameters to specify a JSON filter object to filter delete operations on a table.
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;column_name&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies the name of a table column for deletion.</td>
</tr>
<tr>
<td>&lt;operator&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies an operator. Operators include =, &lt;, &gt;, LIKE.</td>
</tr>
<tr>
<td>&lt;value&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies a filter value.</td>
</tr>
</tbody>
</table>

### Sample Request

Delete all records in a table where NAME = "Oregon". The following example represents the JSON structure that defines the filter.

```json
{
  "predicates": [
    {
      "columnName": "NAME",
      "operator": "=",
      "value": "Oregon"
    },
    {
      "columnName": "CONTACTS",
      "operator": ">",
      "value": "5"
    },
    ...
  ]
}
```

### Example 5-7  Sample cURL Command for Delete

The following is an example command for using cURL with the DELETE method to delete data from a table:

```
```
Indexes are treated as a sub-resource of tables. Use methods of the REST API to create, drop, and describe indexes.

<URL>/dataload/v1/tables/<TABLE_NAME>/indexes/<INDEX_NAME>

Topics:

• Index GET
• Index PUT
• Index DELETE

Index GET

Use the GET method with the Index resource to obtain information about index resources. You can list all tables in a schema, use a query filter to specify tables, or use a table name to obtain a table definition.

Get a description of a table index in the schema.

<URL>/dataload/v1/tables/<TABLE_NAME>/indexes/<INDEX_NAME>

Request Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Table_Name&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies the table for which the index is a sub-resource.</td>
</tr>
<tr>
<td>&lt;Index_Name&gt;</td>
<td>Required</td>
<td>string</td>
<td>Returns a Java object definition for the specified index, or an exception if no index with that name is found.</td>
</tr>
</tbody>
</table>

Response Body

In cases where the index exists, the GET method with a table name that is specified in the path name returns a description of the index. The attributes returned include:

• indexName
• tableName
• isUnique
• isBitmap

The JSON object also contains a list of the index columns with the following attributes:
Sample Request

Get an index named SAMP_REVENUE_I1 with a column named OFFICE_NAME on table SAMP_REVENUE. The following example represents the JSON structure provided for the index definition.

```json
[
    {
        "tableName": "SAMP_REVENUE",
        "indexName": "SAMP_REVENUE_I1",
        "columns": [
            {
                "columnName": "OFFICE_NAME",
                "postn": 1,
                "sortCD": "Asc"
            }
        ],
        "isUnique": "N",
        "isBitMap": "N"
    }
]
```

Example 6-1  Sample cURL Command for Get

The following is an example command for using cURL with the GET method to obtain information about index resources.

```
curl -u <user_name>:<password> -X GET -H "X-ID-TENANT-NAME: <identity_domain>"
https://<URL>/dataload/v1/tables/<table-name>/indexes/<index-name>
```

Index PUT

Use the PUT method with the Index resource to create an index. Provide a JSON definition of the index with a list of columns, providing attributes for each.

**Request Parameters**

The PUT method takes a JSON definition of the index with the following attributes:

- indexName
- tableName
- isUnique

The definition also contains a list of the index columns with the following attributes:

- columnName (String)
- postn (Number)
- sortCD (String – asc or desc)
Request JSON
The JSON definition is as follows:

```json
{
"columns": [
{
"columnName":"COLUMN 1",
"postn": "POSITION", (Specifies the position as an integer value)
"sortCD": "SORT ORDER" (Specifies the sort order for the column, either asc or desc)
},
"isUnique": "N",
"isBitMap": "N"
}
```

Response Body
When the index name does not exist, a response of “Index <INDEX_NAME> on table <TABLE_NAME> created successfully” is returned. If an index with the name already exists, then a response of “Index <INDEX_NAME> on table <TABLE_NAME> already exists” is returned.

Sample Request
Create an index named DEMO_REVENUE_I1 with two columns indexed on table DEMO_REVENUE, and ORDER_NUMBER. The following example represents the JSON structure provided for the index definition.

```json
{
"tableName": "SAMP_REVENUE",
"indexName": "SAMP_REVENUE_I1",
"columns": [
{
"columnName": "ORDER_NUMBER",
"postn": 1,
"sortCD": "Asc"
}
{
"columnName": "ORDER_DAY_DT",
"postn": 2,
"sortCD": "Asc"
}
],
"isUnique": "N",
"isBitMap": "N"
}
```

Example 6-2 Sample cURL Command for Put
The following is an example command for using cURL with the PUT method to create an index. In the example:

- `<index-file>` specifies the name of the file with the index definition.
- `<table-name>` specifies the name of the table in the URI on which to perform an index operation, such as SAMP_REVENUE.
Index DELETE

Use the DELETE method with the Index resource to delete an index. This method doesn’t require a request body.

Request Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;INDEX_NAME&gt;</td>
<td>Required</td>
<td>string</td>
<td>Specifies the name of the index.</td>
</tr>
</tbody>
</table>

Response Body

If the index exists, then a response of “Index <INDEX_NAME> dropped successfully” is returned. If the index cannot be found, then a response of “Index <INDEX_NAME> does not exist” is returned.

Example 6-3  Sample cURL Command for Delete

The following is an example command for using cURL with the DELETE method to delete an index: