Oracle® Cloud

Using Oracle Autonomous JSON Database

F32276-19
March 2025

Oracle Cloud Using Oracle Autonomous JSON Database,

F32276-19

Contents

5 Build an Application

6 Load JSON

7 Oracle Tools for Database Access

8 Oracle Extensions for IDEs

9 Code for High Performance

A Autonomous JSON Database for Experienced Oracle Database Users

B SODA Collection Metadata on Autonomous Database

Preface

This document describes how to work with Oracle Autonomous Databases to develop applications.

Audience

This document is intended for application developers whose applications store and retrieve data in Oracle Autonomous JSON Databases.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://support.oracle.com/portal/ or visit Oracle Accessibility Learning and Support if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documents

- *Getting Started with Oracle Cloud*
- *Oracle Cloud Infrastructure Object Storage*
- *Simple Oracle Document Access (SODA)*
- *Oracle as a Document Store*
- *Oracle Database JSON Developer's Guide*

Conventions

The following text conventions are used in this document.

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

1

Get Started Using Autonomous JSON Database

Oracle Autonomous JSON Database simplifies the task of developing applications that use JavaScript Object Notation (JSON) data. The following features, in particular, support the development of high-performing, high-security applications:

- **Automatic database administration.** Routine database administration tasks such as patching and taking backups are performed automatically, so you can concentrate on developing your application.
- **Automatic performance tuning.** You spend less time defining and tuning your database. See Autonomous Database – Oracle Database Features.
- **Preconfigured high performance.** When you connect to the database with an Oracle client using connection pools you take advantage of high performance features configured on the database side of your connection. See Code for High Performance.
- **Predefined, workload-specific database services.** Client applications can connect to the database using a connection service that best matches the type of database operations they need. (For most applications that use JSON documents you use the typical connection service for transaction processing, **tp**.) See Database Service Names for Autonomous Transaction Processing and Autonomous JSON Database in *Using Oracle Autonomous Database Serverless*.
- **Long-Term Backups.** You can create a long-term backup as a one-time backup or as scheduled long-term backup. You can create a new database by cloning from a long-term backup. You cannot use a long-term backup to recover or restore the same database where the long-term backup was created. See Long-Term Backups on Autonomous Database in *Using Oracle Autonomous Database Serverless*

Autonomous JSON Database is specialized for developing NoSQL-style applications that use JavaScript Object Notation (JSON) documents using Simple Oracle Document Access (SODA) APIs. You can also use Oracle Application Express (APEX) for low-code development of dashboards over your JSON data.

But just as for Autonomous Transaction Processing, JSON data stored in a JSON database is also fully accessible using Structured Query Language (SQL), for analytics and interfacing with relational tools. You can promote an Autonomous JSON Database anytime to an Autonomous Transaction Processing database, which lets you store more non-JSON data.

About Autonomous JSON Database

Oracle Autonomous JSON Database is Oracle Autonomous Transaction Processing, but designed for developing NoSQL-style applications that use JavaScript Object Notation (JSON) documents. You can promote an Autonomous JSON Database service to an Autonomous Transaction Processing service.

See About Autonomous Transaction Processing for a full description of the Autonomous Transaction Processing service. Autonomous JSON Database provides all of the same features, with this *important limitation*: you can store only up to 20 GB of data other than JSON document collections.[1] There is no storage limit for JSON collections.

Development of NoSQL-style, document-centric applications is particularly flexible because the applications use *schemaless* data. This lets you quickly react to changing application requirements. There's no need to normalize the data into relational tables, and no impediment to changing data structure or organization at any time, in any way. A JSON document has internal structure, but no relation is imposed on separate JSON documents.

With Oracle Autonomous JSON Database your JSON document-centric applications typically use Simple Oracle Document Access (SODA), which is a set of NoSQL-style APIs for various application-development languages and for the representational state transfer (REST) architectural style. You can use any SODA API to access any SODA collection.

SODA document collections are backed by ordinary database tables and views. To use other kinds of data, subject to the 20 GB limit, you typically need some knowledge of Structured Query Language (SQL) and how that data is stored in the database.

With Oracle Autonomous JSON Database, a SODA collection *can only contain JSON data*. For example, you cannot have a collection of image documents or a collection that contains both JSON documents and image documents. This is a limitation relative to Autonomous Transaction Processing, where you can define such heterogeneous collections.

No matter what kind of data your applications use, whether JSON or something else, you can take advantage of all Oracle Database features. This is true regardless of the kind of Oracle Autonomous Database you use.

JSON data is stored natively in the database. In a SODA collection on an Autonomous Database JSON data is stored in Oracle's native binary format, OSON.

Work with JSON Documents in Autonomous Database

Autonomous Database supports JavaScript Object Notation (JSON) data natively in the database. You can use NoSQL-style APIs to develop applications that use JSON document collections without needing to know Structured Query Language (SQL) or how the documents are stored in the database.

Oracle provides two sets of such APIs:

- Simple Oracle Document Access (SODA)
- Oracle Database API for MongoDB (also called the MongoDB API)

For example, this *SODA for Java* code opens a collection of cart documents, `carts`, then inserts and saves a new document:

```
OracleCollection coll = db.openCollection("carts");

// Insert and save a cart document.
OracleDocument doc = db.createDocumentFromString(
                       "{\"customerId\":123, \"items\":[...]}");
coll.save(doc);
```

And this code finds a document that has a field `customerId` with a value of `123`.

```
// Find and retrieve a document having customerID 123.
doc = coll.find().filter("{\"customerId\"}:123").getOne();
```

[1] You can subscribe to information event AJDNonJsonStorageExceeded, to be informed when the 20 GB limit is exceeded. See About Information Events on Autonomous Database in *Using Oracle Autonomous Database Serverless*.

Although SODA and the MongoDB API are your main ways of working with JSON documents when developing applications, the data in JSON collections, like other database data, can be accessed from outside an application, including using SQL and database clients such as Java Database Connectivity (JDBC), Oracle Call Interface, and Microsoft .NET Framework. For information about access using SQL see Oracle Tools for Database Access.

Oracle SQL and PL/SQL provide additional ways to use JSON data, beyond what is provided by SODA and the MongoDB API. All Oracle Autonomous Databases fully support the SQL/JSON standard, for example. See *Oracle Database JSON Developer's Guide* for complete information.

And because collections are backed by ordinary database tables and views, you can take advantage of all sorts of standard Oracle Database features, for use with the content of JSON documents.

With Autonomous JSON Database a collection can only contain JSON data. But you can combine (join) JSON data in collections with other data (JSON or non-JSON) that is not in a collection, in arbitrarily complex ways. Then, using features such as Oracle Machine Learning, you can analyze the data and create reports.

SODA and the MongoDB API give you fast, flexible, scalable application development without losing the ability to leverage SQL for analytics, machine learning, and reporting. There are no restrictions on the types of SQL queries that you can express over JSON data.

As a simple example of using SQL with a collection, here is a query that gets the `customerId` values of all documents in collection `carts`. (Database column `json_document` of table `carts` underlies collection `carts`.)

```
SELECT c.json_document.customerId FROM carts c;
```

And assuming fields `unitPrice` and `quantity`, this next query applies SQL aggregate function `sum` to the result of applying multiplication operator `*` to those field values for each document. That is, `sum` *aggregates* the products of unit price and quantity across all documents of the collection. (See https://github.com/oracle-quickstart/oci-cloudnative/blob/master/src/carts/sql/examples.sql for more such examples.)

```
SELECT sum(c.json_document.unitPrice.number()
           *
           c.json_document.quantity.number())
  FROM carts c;
```

In Autonomous Database, JSON data can be stored in Oracle's native binary format, OSON. OSON format is *always* used for JSON data in a collection. For other JSON data, which you store directly in a relational column of type `BLOB`, Oracle recommends that you specify OSON format for that column using a check constraint of `IS JSON FORMAT OSON` with `CREATE TABLE`. For example:

```
CREATE TABLE my_table (id NUMBER, json_doc BLOB
  CHECK (json_doc IS JSON FORMAT OSON))
```

If your database is release 19 or earlier and you use SQL/JSON function `json_query` to retrieve JSON data stored in OSON format, then by default (no `RETURNING` clause) native binary JSON values are automatically serialized to *textual* format (`VARCHAR2(4000)`).

But if you retrieve an entire JSON *document* then no such automatic serialization takes place. If you want the document in textual format then use SQL/JSON function `json_serialize` to serialize it. Here's an example:

```
SELECT json_serialize(c.json_document) FROM carts c;
```

SODA drivers are available for several languages and frameworks: Java, Node.js, Python, C (using Oracle Call Interface), PL/SQL, and REST. *SODA for REST* maps SODA operations to Uniform Resource Locator (URL) patterns, so it can be used with most programming languages.

> **Note:**
>
> If you use SODA to access collections in Oracle Database **19c**, Oracle recommends that you use the instant client for Oracle Database 21c or later, in order to smooth migration to the use of `JSON` data type when your database is upgraded to release 21 or greater.

To get started with SODA or the MongoDB API, see the following:

- Oracle video Demonstration: Using Autonomous Transaction Processing (ATP) Service as a JSON Document Store, which covers the examples shown here, and more, using an Always Free Autonomous Database
- Overview of SODA in *Oracle Database Introduction to Simple Oracle Document Access (SODA)*
- Overview of SODA Filter Specifications (QBEs) in *Oracle Database Introduction to Simple Oracle Document Access (SODA)*
- Overview of Oracle Database API for MongoDB in *Oracle Database API for MongoDB*

For complete information, see the following:

- Simple Oracle Document Access (SODA)
- Oracle Database API for MongoDB

Typical Workflow for Developing Applications with Autonomous JSON Database

Task	Description	More Information
Create and log in to your cloud account	Provide your information and sign up for an Oracle Cloud Service.	Request and Manage Free Oracle Cloud Promotions
Create a JSON database	Create the database that your application will use.	Create an Autonomous JSON Database
Begin building an application	Start building an application that takes advantage of the high-performance features of a JSON database.	Build an Application

Upgrade Autonomous JSON Database to Autonomous Transaction Processing

You can promote an Autonomous JSON Database to an Autonomous Transaction Processing database at any time.

An Autonomous JSON Database is the same as an Autonomous Transaction Processing database, except that an Autonomous JSON Database is limited in these respects:

- You can store only up to 20 GB of data other than JSON document collections.[2]

  (All Autonomous Databases, including , limit the storage of JSON data to 128 TB.)

- Collections cannot be **heterogeneous**. That is, they can only contain JSON documents. For example, you cannot have a collection of image documents or a collection that contains both JSON documents and image documents.

These limitations are appropriate if your use is primarily development of applications that use JSON documents. If you have a greater need to use data other than JSON data then follow these steps to promote your JSON database to an Autonomous Transaction Processing database:

1. Open the Oracle Cloud Infrastructure Console by clicking the ☰ next to Oracle Cloud.
2. From the Oracle Cloud Infrastructure left navigation menu click **Oracle Database**, and then click Autonomous JSON Database.
3. Choose your JSON Database from those listed in the compartment, by clicking its name in column **Display name**.
4. Do one of the following:
   - From the **More actions** drop-down list, select **Change workload type**.
   - In tab Autonomous Database information, under heading **General information**, item **Workload type**, click **Edit**.
5. Click **Convert** to confirm that you want to convert this JSON database to Autonomous Transaction Processing.
6. If you were using the refreshable clone feature with your Autonomous JSON Database then re-create the clone after promotion to Autonomous Transaction Processing. See Using Refreshable Clones with Autonomous Database.

See Autonomous Database Billing Summary for more information.

[2] You can subscribe to information event AJDNonJsonStorageExceeded, to be informed when the 20 GB limit is exceeded. See About Information Events on Autonomous Database.

2

Create an Autonomous JSON Database

If you have an Oracle Cloud user account then you can use one of the following interfaces to create an Oracle Autonomous JSON Database.

- **The Oracle Cloud console.** See Provision an Autonomous JSON Database Instance.
- **The Oracle Cloud Infrastructure REST API.** For information, see the CreateAutonomousDatabase endpoint in *Oracle Cloud Infrastructure API Documentation*. The body of the REST request must contain a single instance of CreateAutonomousDatabaseBase, with `dbWorkload` set to `AJD`.
- **The Oracle Cloud Infrastructure CLI.** For information, see the oci db autonomous-database create command in *OCI CLI Command Reference* . Set parameter `--db-workload [text]` to `AJD`. That is, use command `oci db autonomous-database create --db-workload AJD`.
- **An Oracle Cloud Infrastructure SDK or Plug-in.** For information, see Software Development Kits and DevOps Tools and Plug-ins in *Oracle Cloud Infrastructure API Documentation*.

You can promote an existing Always Free Autonomous JSON Database to a paid Autonomous JSON Database. For more information see Always Free Autonomous Database in *Using Oracle Autonomous Database Serverless*.

Provision an Autonomous JSON Database Instance

Follow these steps to provision a new Autonomous JSON Database instance using the Oracle Cloud Infrastructure Console.

Perform the following prerequisite steps as necessary:

- Open the Oracle Cloud Infrastructure Console by clicking the ☰ next to Oracle Cloud.
- From the Oracle Cloud Infrastructure left navigation menu click **Oracle Database**, and then click Autonomous JSON Database.
- Choose your region. See Switching Regions for information on switching regions and working in multiple regions.
- Choose your **Compartment**. See Compartments for information on using and managing compartments.

On the Autonomous Databases page, perform the following steps:

1. Click **Create Autonomous Database**.
2. Provide basic information for the Autonomous Database.
   - **Compartment**. See Compartments for information on using and managing compartments.
   - **Display name** Specify a user-friendly description or other information that helps you easily identify the resource. The display name does not have to be unique.

The default display name is a generated 16-character string that matches the default database name.

- **Database name** Specify the database name; it must consist of letters and numbers only. The maximum length is 30 characters. The same database name cannot be used for multiple Autonomous Databases in the same tenancy in the same region.

  The default database name is a generated 16-character string that matches the default display name.

3. Choose **JSON** from the workload-type choices:
   - **Data Warehouse**
   - **Transaction Processing**
   - **JSON**
   - **APEX**
4. Configure the database (ECPU compute model)
   - **Always Free**: Select to show Always Free options.

     You can only create a free instance in the tenancy's Home region.
   - **Developer**: Select to show Autonomous Database for Developers options.

     An Autonomous Database for Developers instance is a fixed shape database that is suited for development and testing use cases.

     Selecting this option sets the **Compute** resources to a fixed value of 4 ECPUs and the **Storage** to 20 GB.
   - **Choose database version**: Select the database version. The available database versions are Oracle Database 23ai and Oracle Database 19c.

     > **Note:**
     >
     > – In regions where Oracle Database 23ai is not available, Oracle Database 19c is the only choice.
     >
     > – Autonomous Database with Oracle Database 23ai in the Paid tier is available in all commercial public cloud regions *except* the following region: Singapore West: Singapore (XSP)
     >
     > – Always Free Autonomous Database with Oracle Database 23ai is available in all commercial public cloud regions *except* the following regions: Colombia Central: Bogota (BOG), Saudi Arabia Central (RUH), Singapore West: Singapore (XSP), Spain Central: Madrid (MAD)
   - **ECPU count**: Specify the number of CPUs for your database. The minimum value is 2.
   - **Compute auto scaling**: By default compute auto scaling is enabled to allow the system to automatically use up to three times more CPU and IO resources to meet workload demand. If you do not want to use compute auto scaling then deselect this option.

     See Use Auto Scaling for more information.
   - **Storage**: Specify your storage in Gigabytes (GB) or Terabytes (TB). Select **GB** or **TB** for the **Storage unit size**.

- By default, the IO capacity of your database depends on the number of ECPUs you provision. When you provision 384 TB of storage, your database is entitled to the full IO capacity of the Exadata infrastructure, independent of the number of ECPUs you provision.

  Autonomous Database uses Exadata Smart Flash Cache to automatically cache frequently accessed data, delivering the high I/O rates and fast response times of flash. The amount of flash cache for your database depends on the amount of storage you provision, or the amount of allocated storage if you enable storage auto scaling.

  If you want to provision more than 384 TB of storage, file a Service Request at Oracle Cloud Support.

- **Storage auto scaling**: By default storage auto scaling is disabled. Select if you want to enable storage auto scaling to allow the system to automatically expand to use up to three times more storage.

  With storage auto scaling disabled, the guaranteed minimum flash cache size is 10% of your database's provisioned storage size.

  With storage auto scaling enabled, the guaranteed minimum flash cache size is 10% of your database's provisioned base storage size or its allocated storage size, whichever is higher.

  See Use Auto Scaling for more information.

- **Show advanced options**: Click to show additional options

  – **Enable elastic pool:**

    See Create or Join a Resource Pool While Provisioning or Cloning an Instance for more information.

    An Autonomous JSON Database can join an elastic pool, but it cannot be a pool leader.

  – **Compute model**: Shows the selected compute model.

    Click **Change compute model** to change the compute model. After you select a different compute model, click **Save**.

    * **ECPU**
      Use the ECPU compute model for Autonomous Database. ECPUs are based on the number of cores elastically allocated from a pool of compute and storage servers.

    * **OCPU**
      Use the legacy OCPU compute model if your tenancy is using the OCPU model and you want to continue using OCPUs. The OCPU compute model is based on physical core of a processor with hyper threading enabled.

      > **Note:**
      >
      > OCPU is a legacy billing metric and has been retired for Autonomous Data Warehouse (**Data Warehouse** workload type) and Autonomous Transaction Processing (**Transaction Processing** workload type). Oracle recommends using ECPUs for all new and existing Autonomous Database deployments. See Oracle Support Document 2998742.1 for more information.

    See Compute Models in Autonomous Database for more information.

- **Bring your own license**: If you want to Bring Your Own License to the database, click **Enable** to show the **Update license and Oracle Database Edition** page.

  See Choose Bring Your Own License Option While Provisioning or Cloning for more information.

5. Backup retention

   Automatic backup retention period in days You have the option to select the automatic backup retention period, in a range from 1 to 60 days. You can restore and recover your database to any point-in-time in this retention period.

   Select **Immutable backup retention** to lock the backup retention period.

   After setting the immutable backup retention option, you cannot disable this option or change the retention period. To disable immutable backup retention or to change the backup retention period, file a Service Request at Oracle Cloud Support.

   See About Backup and Recovery on Autonomous Database for more information.

6. Create administrator credentials. Set the password for the Autonomous JSON Database Admin user.

   - **Username** This is a read only field.
   - **Password** Set the password for the Autonomous JSON Database Admin user.
   - **Confirm password** Enter the same password again to confirm your new password.

   The password must meet the strong password complexity criteria based on Oracle Cloud security standards. For more information on the password complexity rules, see About User Passwords on Autonomous Database.

7. Choose network access

   > **Note:**
   >
   > After you provision your Autonomous Database you can change the network access option you select for the instance.

   - **Secure access from everywhere**

     By default, secure connections are allowed from everywhere.

   - **Secure access from allowed IPs and VCNs only**

     This option restricts connections to the database according to the access control lists (ACLs) you specify. To add multiple ACLs for the Autonomous Database, click **Add access control rule**.

     See Configure Access Control Lists When You Provision or Clone an Instance for more information.

   - **Private endpoint access only**

     This option assigns a private endpoint, private IP, and hostname to your database. Specifying this option allows traffic only from the VCN you specify; access to the database from all public IPs or VCNs is blocked. This allows you to define security rules, ingress/egress, at the Network Security Group (NSG) level and to control traffic to your Autonomous Database.

     See Configure Private Endpoints When You Provision or Clone an Instance for more information.

8. (Optional) Provide contacts for operational notifications and announcements

   Click **Add contact** and in the **Contact email** field, enter a valid email address. To enter multiple **Contact email** addresses, repeat the process to add up to 10 customer contact emails.

   See View and Manage Customer Contacts for Operational Issues and Announcements for more information.

9. (Optional) Click Show advanced options to select advanced options.

   - **Encryption key**
     Encrypt using an Oracle-managed key: By default Autonomous Database uses Oracle-managed encryption keys. Using Oracle-managed keys, Autonomous Database creates and manages the encryption keys that protect your data and Oracle handles rotation of the TDE master key.

     Encrypt using a customer-managed key in this tenancy: If you select this option, a master encryption key from a Oracle Cloud Infrastructure Vault in the same tenancy is used to generate the TDE master key on Autonomous Database.

     Encrypt using a customer-managed key located in a remote tenancy: If you select this option, a master encryption key in the Oracle Cloud Infrastructure Vault located in a remote tenancy is used to generate the TDE master key on Autonomous Database.

     See Use Customer-Managed Encryption Keys on Autonomous Database for more information.

   - **Maintenance**
     Patch level By default the patch level is **Regular**. Select **Early** to configure the instance with the early patch level.

     See Set the Patch Level for more information.

   - **Tools**
     If you want to view or customize the tools configuration, select the tools tab.

     See Configure Autonomous Database Built-in Tools when you Provision or Clone an Instance for more information.

   - **Security attributes**
     Add a security attribute to control access for your resources using Zero Trust Packet Routing (ZPR) policies. To enter security attributes during provisioning you must already have set up security attributes with Zero Trust Packet Routing. You also can add security attributes after provisioning.

     > **Note:**
     >
     > You can apply Oracle Zero Trust Packet Routing (ZPR) policies to a private endpoint.

     Specify a **Namespace**, **Key**, and **Value** security attribute.

     Click **Add security attribute** to add additional security attributes.

     See Overview of Zero Trust Packet Routing for more information.

   - **Tags**
     If you want to use Tags, enter the **Tag key** and **Tag value**. Tagging is a metadata system that allows you to organize and track resources within your tenancy. Tags are composed of keys and values which can be attached to resources.

See Tagging Overview for more information.

10. Optionally, you can save the resource configuration as a stack by clicking **Save as Stack**. You can then use the stack to create the resource through the Resource Manager service.

    Enter the following details on the **Save as Stack** dialog, and click **Save**.

    - **Name:** Optionally, enter a name for the stack.
    - **Description:** Optionally, enter a description for this stack.
    - **Save in compartment:** Select a compartment where this Stack will reside.
    - **Tag namespace**, **Tag key**, and **Tag value**: Optionally, apply tags to the stack.

    For requirements and recommendations for Terraform configurations used with Resource Manager, see Terraform Configurations for Resource Manager. To provision the resources defined in your stack, apply the configuration.

11. Click **Create Autonomous Database**.

On the Oracle Cloud Infrastructure console the Lifecycle state shows **Provisioning** until the new database is available.

3

Use Oracle Database Actions with JSON Collections

Oracle Database Actions is a browser-based interface for Oracle SQL Developer. With it, you can run SODA commands or SQL code to create, query, index, and update collections of JSON documents, and to perform other tasks on your database.

Topics

About Database Actions (SQL Developer Web)

Database Actions provides a web-based interface with development, data tools, administration, monitoring, and download features for Autonomous Database.

The following table lists the main features of Database Actions. Database Actions on your Autonomous Database instance may include additional cards if you download applications from Oracle Cloud Marketplace.

Feature Area	Database Actions Cards
Development	SQL, Data Modeler, REST, JSON, Charts, Scheduling, Oracle Machine Learning, Graph Studio, and Oracle APEX
Data Studio	Data Load, Catalog, Data Insights, Data Transforms, and Data Analysis
Administration	Database Users, APEX Workspaces, Data Pump, Download Client Credentials, and Set Resource Management Rules
Monitoring	Performance Hub and Database Dashboard
Downloads	Download Oracle Instant Client and Download SODA Drivers
Related Services	Restful Data Services (ORDS) and SODA and Access Oracle Machine Learning Restful Services

See About Database Actions in *Using Oracle Database Actions* for more information.

Use Oracle Database Actions with SODA

You can use Oracle Database Actions to work with SODA collections directly, using its built-in SODA commands. Database Actions provides a browser-based development environment and a data modeler interface for Autonomous JSON Database.

See Connect with Built-In Oracle Database Actions for information about connecting to Database Actions.

A common set of SODA command-line commands are provided by Database Actions and Oracle SQL Developer Command Line, SQLcl. SQLcl is a Java-based command-line interface

for Oracle Database. You can use it to execute SQL and PL/SQL statements interactively or in batch. It provides inline editing, statement completion, command recall, and it supports existing SQL*Plus scripts.

See SODA Commands in *Using Oracle Database Actions* for complete information about the SODA command-line commands.

See Oracle video Demonstration: Using Autonomous Transaction Processing (ATP) Service as a JSON Document Store, for a demonstration of the examples shown here

You enter commands in the Worksheet area of Database Actions, click the green right-arrow, and see results and other information in the tabs below the worksheet. The simple examples here create a collection, insert a JSON document into the collection, and query the collection.

- Create a collection named `emp`.

  ```
  soda create emp
  ```

  The creation is echoed in tab Script Output, below the worksheet.

- List the existing SODA collections, using command `soda list`.

  ```
  soda list
  ```

  The list, shown in Script Output, includes collection `emp`.

- Insert five JSON documents into the collection, one by one.

```
soda insert emp {"name" : "Blake", "job" : "Intern", "salary" : 30000}
soda insert emp {"name" : "Smith", "job" : "Programmer", "salary" : 80000}
soda insert emp {"name" : "Miller", "job" : "Programmer", "salary" : 90000}
soda insert emp {"name" : "Clark", "job" : "Manager", "salary" : 100000}
soda insert emp {"name" : "King", "job" : "President", "salary" : 200000,
                 "email" : "king@example.com"}
```

  Each insertion is echoed in Script Output.

- Get (retrieve) documents, filtering the collection with a SODA query-by-example (QBE) pattern that matches `"Miller"` as the name. (Switch `-f` means list the documents that match the QBE.)

  ```
  soda get emp -f {"name":"Miller"}
  ```

  Script Output shows the one matching document that's selected (returned), along with the key for that document, which is a universally unique identifier (UUID) that identifies it.

- Get the documents that have a salary field whose value is at least 50,000. The QBE pattern uses SODA greater-than-or-equal operator, `$ge`, comparing target field `salary`, with the value 100,000.

  ```
  soda get emp -f {"salary" : {"$ge" : 100000}}
  ```

  Two documents are returned in this case, for employees Clark, and King, each of whose salary is at least 100,000.

Use Oracle Database Actions with SQL over SODA Collections

You can use Oracle Database Actions with SQL to work with SODA collections. In this case, you act directly on the backing-store tables or views that underlie SODA collections.

The examples here use the employees SODA collection, `emp`, created in topic Use Oracle Database Actions with SODA. (That topic creates the collection using Database Actions SODA commands, but the collection could be created, and it can be modified, using any supported SODA language or framework — Java, Node.js, Python, C, PL/SQL, or REST.)

Collection `emp` has these five employee documents:

```
{"name" : "Blake", "job" : "Intern", "salary" : 30000}
{"name" : "Smith", "job" : "Programmer", "salary" : 80000}
{"name" : "Miller", "job" : "Programmer", "salary" : 90000}
{"name" : "Clark", "job" : "Manager", "salary" : 100000}
{"name" : "King", "job" : "President", "salary" : 200000,
 "email" : "king@example.com"}
```

In Database Actions, you can see the complete backing-store database table that underlies this SODA collection in the Navigator tab to the left of the worksheet. In this case, expand `EMP` there to show the columns of that table.

- `ID` — Document key column.
- `CREATED_ON` — Creation timestamp column.
- `LAST_MODIFIED` — Last-modified timestamp column.
- `JSON_DOCUMENT` — JSON content column (in this case, employee data).

You can use Structured Query Language (SQL) directly on this underlying data.

You enter SQL statements in the Worksheet area of Database Actions, click the green right-arrow, and see results and other information in the tabs below the worksheet. The simple examples here select documents, project JSON fields from them, and perform aggregate operations on selected fields.

- Select each of the documents in the collection.

  ```
  SELECT json_serialize(json_document) FROM emp;
  ```

  The documents are listed in tab Script Output, below the worksheet.

  Because this query retrieves an entire JSON document you need to convert Oracle's native binary JSON format, OSON, to textual format using standard SQL/JSON function `json_serialize`. (When you use SQL to retrieve JSON objects or arrays from within JSON documents you need not use `json_serialize`; that data is automatically serialized to textual format.)

- Query the collection, projecting out the value of each of the fields from each document, as a SQL value.

  ```
  SELECT e.json_document.name,
         e.json_document.job,
         e.json_document.salary,
  ```

```
      e.json_document.email
  FROM emp e;
```

The projected field values are listed in Script Output in tabular form. The values for each document form one row of the table.

In the query, we give table `EMP` the alias `e`, and we use a simple dot notation *<table>.<JSON column>.<field>* to target each field.

The simple dot notation is handy for drilling down into JSON data. Just be aware of two particularities with respect to most SQL syntax: (1) A table alias is *required* when you use dot notation. (2) Although SQL is case-insensitive in general, with the dot notation *<field>* corresponds to *JSON* data, so it is interpreted case-sensitively (JSON, like JavaScript, is case-sensitive).

The value for each field except `salary` is a SQL string (`VARCHAR2` data type). The value for field `salary` is a SQL number (`NUMBER` data type). The value for field `email` for employee `King` is the `VARCHAR2` value `king@example.com`. The value for field `email` for the other employees is shown as `(null)`, meaning that the field is absent.

- Query the collection, projecting field `job` joining it with the result of an aggregate operation that counts employees that have each job (as a group) across the collection.

```
SELECT e.json_document.job, count(*) FROM emp e
  GROUP BY e.json_document.job;
```

SQL queries over SODA collections can perform arbitrarily complex joins and aggregate operations.

4

Develop RESTful Services

You can develop and deploy RESTful Services with native Oracle REST Data Services (ORDS) support on Autonomous Databases. Simple Oracle Document Access (SODA) for REST lets you use a JSON database as a simple JSON document store.

Topics

About Oracle REST Data Services in Autonomous Database

Oracle REST Data Services (ORDS) makes it easy to develop REST interfaces for relational data in a JSON database.

ORDS is a mid-tier Java application that maps HTTP(S) verbs, such as GET, POST, PUT, DELETE, and so on, to database transactions, and returns any results as JSON data.

> **Note:**
>
> The Oracle REST Data Services (ORDS) application in Autonomous JSON Database is preconfigured and fully managed. ORDS connects to the database using the `low` predefined database service with a fixed maximum number of connections (the number of connections for ORDS does not change based on the number of ECPUs (OCPUs if your database uses OCPUs). It is not possible to change the default ORDS configuration.
> See About Customer Managed Oracle REST Data Services on Autonomous Database for information on using an additional alternative ORDS deployment that enables flexible configuration options.

See Oracle REST Data Services for information on using Oracle REST Data Services.

See Database Service Names for Autonomous Transaction Processing and Autonomous JSON Database in *Using Oracle Autonomous Database Serverless* for information on the `low` database service.

Access RESTful Services and SODA for REST

Each Autonomous JSON Database includes Oracle REST Data Services (ORDS) that provides HTTPS interfaces for working with the contents of your Oracle Database in REST enabled schemas.

Perform the following prerequisite steps as necessary:

- Open the Oracle Cloud Infrastructure Console by clicking the ☰ next to Oracle Cloud.
- From the Oracle Cloud Infrastructure left navigation menu click **Oracle Database**, and then click Autonomous JSON Database.
- On the Autonomous Databases page select an Autonomous Database from the links under the **Display name** column.

To use Oracle REST Data Services and SODA for REST:

1. From the Autonomous Database details page select **Database Actions** and in the list click **View all database actions**.
2. On the Database Actions launchpad, under **Related Services**, click the **RESTful Services and SODA** card to see the base URL.
3. Click **Copy** to copy the URL.

If you are using Always Free Autonomous Database with Oracle Database 23ai Oracle recommends the following:

- For projects that were started using a database release prior to Oracle Database 21c, explicitly specify the metadata for the default collection as specified in the example in the section SODA Drivers.
- For projects started using release Oracle Database 21c or later, just use the default metadata.

See SODA Drivers for more information.

Use SODA for REST with Autonomous Database

Autonomous JSON Database supports Simple Oracle Document Access (SODA) for REST.

Overview of Using SODA for REST

SODA for REST is a predeployed REST service for managing JSON documents using CRUD operations (create, read, update and delete), and for querying them using NoSQL-style query-by-example (QBE) requests.
To use SODA for REST you need a database schema (user) that is enabled for Oracle REST Data Services (ORDS). With this SQL code a database user with administrator privileges, such as user `ADMIN`, can create such an ORDS-enabled schema (in this case `TEST`). (For information about access using SQL see Oracle Tools for Database Access.)

```
CREATE USER test IDENTIFIED BY <password>;
GRANT DWROLE TO test;
GRANT UNLIMITED TABLESPACE TO test;
EXEC ords.enable_schema(P_SCHEMA => 'TEST');
```

SODA for REST is deployed in ORDS under the following URL pattern, where *schema* corresponds to a REST-enabled database schema.

```
/ords/schema/soda/latest/*
```

The following examples use the cURL command line tool (http://curl.haxx.se/) to submit REST requests to the JSON database. However, other 3rd party REST clients and libraries should

work as well. The examples use database schema `ADMIN`, which is REST-enabled. You can SODA for REST with cURL commands from the Oracle Cloud Shell.

This command creates a new collection named "fruit" in the `ADMIN` schema:

```
> curl -X PUT -u 'ADMIN:<password>' \
"https://example-db.adb.us-phoenix-1.oraclecloudapps.com/ords/admin/soda/latest/fruit"
```

These commands insert three JSON documents into the fruit collection:

```
> curl -X POST -u 'ADMIN:<password>' \
 -H "Content-Type: application/json" --data '{"name":"orange", "count":42}' \
 "https://example-db.adb.us-phoenix-1.oraclecloudapps.com/ords/admin/soda/latest/fruit"

{"items":[{"id":"6F7E5C60197E4C8A83AC7D7654F2E375"...

> curl -X POST -u 'ADMIN:<password>' \
 -H "Content-Type: application/json" --data '{"name":"pear", "count":5}' \
 "https://example-db.adb.us-phoenix-1.oraclecloudapps.com/ords/admin/soda/latest/fruit"

{"items":[{"id":"83714B1E2BBA41F7BA4FA93B109E1E85"...

> curl -X POST -u 'ADMIN:<password>' \
 -H "Content-Type: application/json" \
 --data '{"name":"apple", "count":12, "color":"red"}' \
 "https://example-db.adb.us-phoenix-1.oraclecloudapps.com/ords/admin/soda/latest/fruit"

{"items":[{"id":"BAD7EFA9A2AB49359B8F5251F0B28549"...
```

This example retrieves a stored JSON document from the collection:

```
> curl -X POST -u 'ADMIN:<password>' \
 -H "Content-Type: application/json" --data '{"name":"orange"}' \
 "https://example-db.adb.us-phoenix-1.oraclecloudapps.com/ords/admin/soda/latest/fruit?
action=query"

{
  "items": [
    {
      "id":"6F7E5C60197E4C8A83AC7D7654F2E375",
      "etag":"57215643953D7C858A7CB28E14BB48549178BE307D1247860AFAB2A958400E16",
      "lastModified":"2019-07-12T19:00:28.199666Z",
      "created":"2019-07-12T19:00:28.199666Z",
      "value":{"name":"orange", "count":42}
    }
  ],
  "hasMore":false,
  "count":1
}
```

This SQL query accesses the fruit collection:

```
SELECT
     f.json_document.name,
```

```
    f.json_document.count,
    f.json_document.color
FROM fruit f;
```

The query returns these three rows:

```
name      count     color
--------- --------- -------
orange    42        null
pear      5         null
apple     12        red
```

> **Note:**
>
> If you are using Always Free Autonomous Database with Oracle Database 23ai, Oracle recommends the following:
> For projects that were started using a database release prior to Oracle Database 21c, explicitly specify the metadata for the default collection as specified in the example in the section SODA Drivers. For projects started using release Oracle Database 21c or later, just use the default metadata. See SODA Drivers for more information.

These examples show a subset of the SODA and SQL/JSON features. See the following for more information:

- SODA for REST for complete information on Simple Oracle Document Access (SODA)
- SODA for REST HTTP Operations for information on the SODA for REST HTTP operations

Load Purchase-Order Sample Data Using SODA for REST

Oracle provides a substantial set of JSON purchase-order documents, in plain-text file `POList.json`, as a JSON array of objects, where each such object represents a document.

The following examples use the cURL command line tool (http://curl.haxx.se/) to submit REST requests to the JSON database. However, other 3rd party REST clients and libraries should work as well. The examples use database schema `ADMIN`, which is REST-enabled. You can use SODA for REST with cURL commands from the Oracle Cloud Shell.

You can load this sample purchase-order data set into a collection `purchaseorder` on your Autonomous Database with SODA for REST, using these curl commands:

```
curl -X GET "https://raw.githubusercontent.com/oracle/db-sample-schemas/
master/order_entry/POList.json" -o POList.json

curl -X PUT -u 'ADMIN:password' \
"https://example-db.adb.us-phoenix-1.oraclecloudapps.com/ords/admin/soda/
latest/purchaseorder"

curl -X POST -H -u 'ADMIN:password' 'Content-type: application/json' -d
@POList.json \
```

```
"https://example-db.adb.us-phoenix-1.oraclecloudapps.com/ords/admin/soda/
latest/purchaseorder?action=insert"
```

You can then use this purchase-order data to try out examples in *Oracle Database JSON Developer's Guide*.

For example, the following query selects both the `id` of a JSON document and values from the JSON purchase-order collection stored in column `json_document` of table `purchaseorder`. The values selected are from fields `PONumber`, `Reference`, and `Requestor` of JSON column `json_document`, which are projected from the document as virtual columns (see SQL NESTED Clause Instead of JSON_TABLE for more information).

```
SELECT id, t.*
  FROM purchaseorder
    NESTED json_document COLUMNS(PONumber, Reference, Requestor) t;
```

See the following for more information:

- SODA for REST for complete information on Simple Oracle Document Access (SODA)
- SODA for REST HTTP Operations for information on the SODA for REST HTTP operations

Use SODA for REST with OAuth Client Credentials

You can access SODA for REST on Autonomous Database using OAuth authentication. Depending on your application, accessing SODA for REST with OAuth authentication can improve performance and security.

Perform the following steps to use OAuth authentication to provide limited access to SODA for REST on Autonomous Database:

1. As the ADMIN user, access Database Actions and create a user with the required privileges.
   a. Access Database Actions as ADMIN.

      See Access Database Actions as ADMIN for more information.

   b. In Database Actions, click ☰ to show the available actions.
   c. In Database Actions, under **Administration** select **Database Users**.
   d. Click **Create User**.
   e. In the **Create User** area, on the **User** tab enter **User Name** and a **Password** and confirm the password.
   f. Select **Web Access**.
   g. In the **Create User** area, select the **Granted Roles** tab and grant `DWROLE` to the user.
   h. Click **Create User**.

   See Manage Users and User Roles on Autonomous Database - Connecting with Database Actions in *Using Oracle Autonomous Database Serverless* for more information.
2. Use a SQL worksheet in Database Actions to grant user privileges required to load data.
   a. Access Database Actions as ADMIN.

      See Access Database Actions as ADMIN for more information.

b. In Database Actions, click  to show the available actions.

c. In Database Actions, under **Development** click **SQL** to open a SQL worksheet.

d. Grant user privileges required to load data to the user from Step 1.

```
GRANT UNLIMITED TABLESPACE TO user_name;
```

See Manage User Privileges on Autonomous Database - Connecting with a Client Tool for more information.

3. Sign out as the ADMIN user.
4. Sign in to Database Actions as the user that is setting up to use OAuth authentication.
5. In Database Actions, use a SQL worksheet to register the OAuth client.

   a. Register the OAuth client.

   For example, enter the following commands into the SQL worksheet, where you supply the appropriate values for your user and your client application.

```
BEGIN
  OAUTH.create_client(
    p_name            => 'my_client',
    p_grant_type      => 'client_credentials',
    p_owner           => 'Example Company',
    p_description     => 'A client for my SODA REST resources',
    p_support_email   => 'user_name@example.com',
    p_privilege_names => 'my_priv'
  );

  OAUTH.grant_client_role(
    p_client_name => 'my_client',
    p_role_name   => 'SQL Developer'
  );

  OAUTH.grant_client_role(
    p_client_name => 'my_client',
    p_role_name   => 'SODA Developer'
  );
  COMMIT;
END;
/
```

   b. In the SQL worksheet, click **Run Script** to run the command.

   See OAUTH PL/SQL Package Reference for more information.

   This registers a client named `my_client` to access the `my_priv` privilege using OAuth client credentials.

6. Obtain the `client_id` and `client_secret` required to generate the access token.

   For example, in the SQL worksheet run the following command:

```
SELECT id, name, client_id, client_secret FROM user_ords_clients;
```

7. Obtain the access token. To get an access token you send a `REST GET` request to *database_ORDS_urluser_name*`/oauth/token`.

The *database_ORDS_url* is available from Database Actions, under **Related Services**, on the **RESTful Services and Soda** card. See Access RESTful Services and SODA for REST for more information.

In the following command, use the `client_id` and the `client_secret` you obtained in Step 6.

The following example uses the `cURL` command line tool (http://curl.haxx.se/) to submit REST requests to Autonomous Database. However, other 3rd party REST clients and libraries should work as well.
You can use the `cURL` command line tool to submit the `REST GET` request. For example:

```
> curl -i -k --user SBA-iO9Xe12cdZHYfryBGQ..:vvUQ1AagTqAqdA2oN7afSg.. --
data "grant_type=client_credentials"https://mqssyowmqvgac1y-
doc.adb.region.oraclecloudapps.com/ords/user_name/oauth/token
HTTP/1.1 200 OK
Date: Mon, 22 Jun 2020 15:17:11 GMT
Content-Type: application/jsonTransfer-Encoding: chunked
Connection: keep-alive
X-Frame-Options: SAMEORIGIN

{"access_token":"JbOKtAuDgEh2DXx0QhvPGg","token_type":"bearer","expires_in"
:3600}
```

To specify both the `client_id` and the `client_secret` with the curl `--user` argument, enter a colon to separate the `client_id` and the `client_secret`. If you only specify the user name, `client_id`, curl prompts for a password and you can enter the `client_secret` at the prompt.

8. Use the access token to access the protected resource.

   The token obtained in the previous step is passed in the Authorization header. For example:

```
> curl -i -H "Authorization: Bearer JbOKtAuDgEh2DXx0QhvPGg" -X GET https://
database_id.adb.region.oraclecloudapps.com/ords/user_name/soda/latest
HTTP/1.1 200 OK
Date: Mon, 22 Jun 2020 15:20:58 GMT
Content-Type: application/json
Content-Length: 28
Connection: keep-alive
X-Frame-Options: SAMEORIGIN
Cache-Control: private,must-revalidate,max-age=0


{"items":[],"hasMore":false}
```

See Configuring Secure Access to RESTful Services for complete information on secure access to RESTful Services.

5

Build an Application

Oracle Autonomous JSON Database supports application development in a wide variety of programming languages and platforms.

In general, you follow the same general guidelines and high-level system configuration steps to build an application, regardless of the language or platform. These guidelines and steps are described in The Basics of Building an Application.

For some languages and platforms, you can follow specific step-by-step instructions instead of the general guidelines:

The Basics of Building an Application

Regardless of the language you use to build an application, you follow the same guidelines to build an application that takes advantage of the high-performance features of a JSON database:

- **Connect through an Oracle client.** When you connect to the database through an Oracle client, almost all connection-management operations are performed by the client, permitting you to concentrate on the business logic of your application. Depending on your programming language, you use the Oracle Database JDBC Driver or the Oracle Instant Client.
- **Use connection pools.** When you code your application to use connection pools instead of creating and destroying connections individually, you gain performance improvements. How you code to use connection pools depends on your programming language.
- **Connect to the appropriate database service.** Autonomous JSON Database provides several database services to use when connecting to your database. These database connection services are designed to support different kinds of database operations, as described in Database Service Names for Autonomous Transaction Processing and Autonomous JSON Database in *Using Oracle Autonomous Database Serverless*.

Also regardless of the language you use to build an application, you perform the same basic tasks to configure your system to support application development:

1. Download and install the basic software to develop in the given language. For example, you download and install JDK to develop Java applications.
2. Download and install any extension library or module necessary to permit applications in the given language to connect to an Oracle Database and make SQL calls. For example, you download and install the cx_Oracle extension module to develop Python applications.
3. Download and install the Oracle client appropriate for the given language and extension library or module.
4. Download the client credentials for the database and make them available to Oracle client you installed.

Build a Java Application

To build a Java application that accesses a JSON database, you start by configuring your development system to support database access that can take advantage of the high performance features of Autonomous JSON Database. Then, in your application you code database connections and SQL statements to take advantage of these features.

Topics

Configure Your Java Development System

To configure your development system so that your Java application can take advantage of the high performance features of a JSON database, perform these steps.

To configure your development system so that your Java application can take advantage of the high performance features of a JSON database, perform these steps.

1. Download and install the Java Development Kit (JDK).
2. Download the client credentials for your Autonomous Database.
3. Get the Oracle Java Database Connectivity (JDBC) drivers.

Download and Install the JDK

Go to the Java SE Downloads page. Then, download and install JDK 8u221 or later by following the instructions on the page.

Download the Client Credentials for Your Autonomous Database

1. Download the zip file containing client credentials for your database to a secure directory on your computer.

   This zip file is available for download from the database's Details page in the Oracle Cloud console. If you have an Oracle Cloud user account that permits you to access this page, download the credentials as follows. If you don't have such an account, you need to get the zip file from the administrator of the database, together with the password that was used to protect the zip file.

   a. In your web browser, sign in to Oracle Cloud and navigate to the Details page for the JSON database.

   b. Click **DB Connection**.

   c. On the **Database Connection** page click **Download**.

   d. In the **Download Wallet** dialog, enter a password in the **Password** field and confirm the password in the **Confirm Password** field.

      The password must be at least 8 characters long and must include at least 1 letter and either 1 numeric character or 1 special character.

   e. Click **Download** and unzip, to save the client credentials zip file to a secure directory.

Get the Oracle JDBC Drivers

Get the Oracle JDBC drivers, version 19.6.0.0 or later, from *either* Maven Central *or* the JDBC Downloads page at Oracle Technical Resources. (See the Oracle Technologies JDBC Home page for related videos and other resources.)

To get the JDBC drivers from Maven Central, follow these steps.

1. Get the Oracle JDBC drivers from Central Maven Repository. Choose version **19.6.0.0** or later.

   Provide the driver Maven dependency GAV (GroupID, ArtifactID, VersionID), to pull `ojdbc8.jar`, along with other jars such as `oraclepki.jar`, `osdt_core.jar`, and `osdt_cert.jar`. See Maven Central Guide.

   For `ojdbc8.jar` version 19.6.0.0, provide this GAV:

   ```
   <groupId>com.oracle.database.jdbc</groupId>
   <artifactId>ojdbc8</artifactId>
   <version>19.7.0.0</version>
   ```

   For `ojdbc8.jar` version 19.7.0.0, provide this GAV:

   ```
   <groupId>com.oracle.database.jdbc</groupId>
   <artifactId>ojdbc8-production</artifactId>
   <version>19.7.0.0</version>
   <type>POM</type>
   ```

To get the JDBC drivers from Oracle Technical Resources, follow these steps.

1. Go to the Oracle JDBC Downloads page. Choose the link for version **19.6.0.0** or later, to go to its download page.
2. Download and unzip this archive to the directory where you want to place the JDBC driver: `ojdbc8-full.tar.gz`.
3. Point the connection URL to your Autonomous JSON Database.

   Append `TNS_ADMIN` to the connection URL, setting its value to the full path of the directory where you unzipped the client credentials. For example:

   ```
   // Use TNS alias name plus TNS_ADMIN with JDBC driver 18.3 or higher
   DB_URL="jdbc:oracle:thin:@wallet_dbname?
   TNS_ADMIN=/Users/test/wallet_dbname";

   // For Microsoft Windows, use this for TNS_ADMIN:
   // TNS_ADMIN=C:\\Users\\test\\wallet_dbname”;
   ```

4. Add the paths to the following unzipped JAR files to the `CLASSPATH` environment variable you use when you compile and run Java programs.

   Use DataSourceSample.java or UCPSample.java to verify the connection to your Autonomous JSON Database.

   - `ojdbc8.jar`: the core JDBC driver
   - `oraclepki.jar`, `osdt_core.jar`, and `osdt_cert.jar`: for an Autonomous JSON Database that uses wallet-based authentication

- `ucp.jar`: for Universal Connection Pooling (UCP)

Download, Install, and Configure SODA for Java

Follow these steps to download, install, and configure SODA for Java.

1. Go to the SODA for Java downloads page on GitHub: `https://github.com/oracle/soda-for-java/releases`.
2. Choose the latest release of SODA for Java, and download the following:
   - Jar file `orajsoda-<relno>.jar`, where `<relno>` is the release number
   - The zip or tar.gz source-code archive

> **Note:**
>
> The SODA for Java driver is also available on Central Maven Repository.

3. Extract the source-code archive to the directory where you want to install SODA for Java.
4. Consult the documentation in file `README.md` and the files in folder `doc`, for instructions about building the source code and getting started.

> **Note:**
>
> Autonomous Database does *not* support Metadata builder. To customize collection metadata for a given collection, pass collection metadata strings directly to method createCollection. See SODA Collection Metadata on Autonomous Database for more information.

Set JVM Networking Properties

Autonomous Database uses DNS names that map to multiple IP addresses (multiple load balancers) for better availability and performance. Depending on your application, you may want to configure certain JVM networking properties.

For the Java Virtual Machine (JVM) address cache, any address resolution attempt caches the result whether it was successful or not, so that subsequent identical requests do not have to access the naming service. The address cache properties allow you to tune how the cache operates. In particular, the `networkaddress.cache.ttl` value specifies the number of seconds a successful name lookup is kept in the cache. A value of -1, the default value, indicates a “cache forever” policy, while a value of `0` (zero) means no caching.

If your Java Virtual Machine (JVM) is configured to cache DNS address lookups, your application may be using only one IP address to connect to your Autonomous Database, resulting in lower throughput. To prevent this you can change your JVM's `networkaddress.cache.ttl` value to `0`, so that every connection request does a new DNS lookup. This ensures that different threads in your application are distributed over multiple load balancers.

To change the `networkaddress.cache.ttl` value for all applications, or in your application, do one of the following:

- Configure the security policy to set the value for all applications:

Set `networkaddress.cache.ttl=0` in the file `$JAVA_HOME/jre/lib/security/java.security`

- Set the following property in your application code:

```
java.security.Security.setProperty("networkaddress.cache.ttl" , "0");
```

Code Database Connections and SQL Statements

Follow these guidelines to achieve high performance of your application's connections to the database:

- Use connection pools.
- Use the predefined database service that best matches the operations you will be performing. For most purposes working with JSON data, this is service *tp*, the typical application connection service for transaction processing operations. For information about the available predefined database services see Database Service Names for Autonomous Transaction Processing and Autonomous JSON Database in *Using Oracle Autonomous Database Serverless*.

For example:

```
import java.sql.Connection;
import javax.sql.PooledConnection;
import oracle.jdbc.OracleConnection;
import oracle.jdbc.replay.OracleDataSourceFactory;
import oracle.jdbc.replay.OracleDataSource;
import oracle.jdbc.replay.OracleConnectionPoolDataSource;
...
  PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
  // Set the connection factory first before all other properties

pds.setConnectionFactoryClassName("oracle.jdbc.replay.OracleConnectionPoolData
SourceImpl");
  pds.setURL("jdbc:oracle:thin:@tp?TNS_ADMIN=/users/jdoe/adbcredentials");
  pds.setUser("appuser");
  pds.setPassword("<password>");
  pds.setConnectionPoolName("JDBC_UCP_POOL");

  Connection conn = pds.getConnection();

  // Create an OracleRDBMSClient instance.
  // This is the starting point of the SODA for Java application.
  OracleRDBMSClient cl = new OracleRDBMSClient();

  // Get a database.
  OracleDatabase db = cl.getDatabase(conn);

  // Create a collection with the name "MyJSONCollection".
  OracleCollection col =
    db.admin().createCollection("MyJSONCollection");
```

Additional Resources

For information about SODA for Java, see *Oracle Database SODA for Java Developer's Guide*

For detailed information about the Oracle Database JDBC Driver, see *Oracle Database JDBC Developer's Guide* and *Oracle Database JDBC Java API Reference*.

For detailed information about the Universal Connection Pool, see *Oracle Universal Connection Pool Developer's Guide* and *Oracle Universal Connection Pool API Reference*.

Build a Node.js Application

To build a Node.js application that accesses a JSON database, you start by configuring your development system to support database access that can take advantage of the high performance features of Autonomous JSON Database. Then, in your application you code database connections and SQL statements to take advantage of these features.

Topics

Configure Your Node.js Development System

To configure your development system so that your Node.js application can take advantage of the high performance features of a JSON database, you perform these steps.

1. Download and install Node.js.
2. Download and install Oracle Instant Client.
3. Download and install node-oracledb.
4. Download the client credentials for the database and make them available to Oracle Instant Client.

Download and Install Node.js

Download and install Node.js for your system's OS and architecture:

- **Oracle Linux:**

  Run these commands to download and install the latest version of Node.js:

  ```
  sudo yum install -y oracle-release-el7 oracle-nodejs-release-el7
  sudo yum install -y nodejs
  ```

- **Other OSes and architectures:**

  Go to the Node.js Downloads page, select the latest LTS (Long Term Support) version for your system's OS and architecture, and install it.

Download and Install Oracle Instant Client

You need Oracle Instant Client libraries version 19.6 or later.

Download and install the Oracle Instant Client basic package for your system's OS and architecture:

- **Oracle Linux:**

  Run these commands to download and install the Oracle Instant Client basic package:

  ```
  sudo yum -y install oracle-release-el7
  sudo yum -y install oracle-instantclient19.3-basic
  ```

(If you want to see a list of all Instant Client packages, go to http://yum.oracle.com/repo/OracleLinux/OL7/oracle/instantclient/x86_64/index.html.)

- **Other OSes and architectures:**

  1. Go to the Oracle Instant Client Downloads page and select the download for your system's OS and architecture.
  2. On the download page, accept the Oracle Technology Network License Agreement, download the latest version of the **Basic Package**, and then install it by following the instructions at the bottom of the download page.

Download and Install node-oracledb

Download and install the node-oracledb add-on for Node.js for your system's OS and architecture:

- **Oracle Linux:**

  Run these commands to download and install the latest version of node-oracledb:

  ```
  sudo yum install -y oracle-release-el7 oracle-nodejs-release-el7
  sudo yum install -y node-oracledb-node10
  ```

- **Other OSes and architectures:**

  Go to the Installing node-oracledb page, choose the "My database is on another machine" instructions for your OS and architecture, and then follow the **Install the add-on** instructions.

Download and Install Client Credentials for the Database

1. Download the zip file containing client credentials for your database to a secure directory on your computer.

   This zip file is available for download from the database's Details page in the Oracle Cloud console. If you have an Oracle Cloud user account that permits you to access this page, download the credentials as follows. If you don't have such an account, you need to get the zip file from the administrator of the database, together with the password that was used to protect the zip file.

   a. In your web browser, sign in to Oracle Cloud and navigate to the Details page for the JSON database.

   b. Click **DB Connection**.

   c. On the **Database Connection** page click **Download**.

   d. In the **Download Wallet** dialog, enter a wallet password in the **Password** field and confirm the password in the **Confirm Password** field.

      The password must be at least 8 characters long and must include at least 1 letter and either 1 numeric character or 1 special character.

   e. Click **Download** to save the client credentials zip file to a secure directory.

2. After downloading the zip file, follow these steps:

   a. Unzip the client credentials zip file.

   b. Edit the `sqlnet.ora` file provided in the client credentials, replacing `"?/network/admin"` with the full path of the directory where you unzipped the client credentials; for example, change:

      ```
      (DIRECTORY="?/network/admin")
      ```

to:

```
(DIRECTORY="/users/jdoe/adbcredentials")
```

c. Create the `TNS_ADMIN` environment variable, setting its value to the full path of the directory where you unzipped the client credentials.

Code Database Connections and SQL Statements

Follow these steps to ensure optimal performance of your application's use of the database.

1. Add the dependency on the node-oracledb add-on to your application's `package.json` file.
2. Code connections for high performance.

Add the node-oracledb Dependency to `package.json`

Edit the `dependencies` object in the `package.json` file for your application, adding the `oracledb` package and version. (Use command `npm init` to generate `package.json` if it doesn't exist.) For example:

```
. . .
"dependencies": {
  . . .,
  "oracledb": "^4.0",
  . . .
},
. . .
```

For detailed information about the `dependencies` object, see the npm-package.json page. To display the `oracledb` version installed, you can use the `npm list` command; for example:

```
npm list -g --depth=0
```

Code Connections for High Performance

To achieve high performance, follow these guidelines when making connections to the database.

- Use connection pools.
- Use the predefined database service that best matches the operations you will be performing. For most purposes working with JSON data, this is service *tp*, the typical application connection service for transaction processing operations. For information about the predefined database services, see Database Service Names for Autonomous Transaction Processing and Autonomous JSON Database in *Using Oracle Autonomous Database Serverless*.

For example:

```
var oracledb = require('oracledb');
var config = {
  user: process.env.NODE_ORACLEDB_USER || "ADMIN",
  password: process.env.NODE_ORACLEDB_PASSWORD,
  connectString : process.env.NODE_ORACLEDB_CONNECTIONSTRING || "mydb_tp",
  poolMin: 10,
  poolMax: 10,
  poolIncrement: 0,
}
```

```
async function getCollection() {
  oracledb.autoCommit = true;
  await oracledb.createPool(config);
  var conn = await oracledb.getConnection();
  var soda = conn.getSodaDatabase();
  var collection = await soda.createCollection('myCollection');
conn.close();
}

getCollection();
```

This example creates a pool for connections to database service `tp`.

Additional Resources

For detailed information about node-oracledb, go to the node-oracledb Documentation page, which includes both an API Reference and a User Guide.

For code examples that demonstrate a wide variety of node-oracledb features, go to the node-oracledb examples folder.

Build a Python Application

To build a Python application that accesses a JSON database, you start by configuring your development system to support database access that can take advantage of the high performance features of Autonomous JSON Database. Then, in your application you code database connections and SQL statements to take advantage of these features.

Topics

- Configure Your Python Development System
- Code Database Connections and SQL Statements

Configure Your Python Development System

To configure your development system so that your Python application can take advantage of the high performance features of a JSON database, you perform these steps.

1. Download and install Python.
2. Download and install Oracle Instant Client.
3. Download and install cx_Oracle.
4. Download the client credentials for the database and make them available to Oracle Instant Client.

Download and Install Python

- **Oracle Linux:**

  Oracle Linux 7 includes Python 2.7, so you simply run this command::

  ```
  sudo yum -y install oracle-release-el7
  ```

- **Other OSes and architectures:**

Go to the python.org Downloads page and download and install the latest Python 2.7 or Python 3.5 (or later) version for your OS and architecture.

Download and Install Oracle Instant Client

You need Oracle Instant Client libraries version 19.6 or later.

Download and install the Oracle Instant Client basic package for your system's OS and architecture:

- **Oracle Linux:**

  Run these commands to download and install the Oracle Instant Client basic package:

  ```
  sudo yum -y install oracle-release-el7
  sudo yum -y install oracle-instantclient19.3-basic
  ```

  (If you want to see a list of all Instant Client packages, go to http://yum.oracle.com/repo/OracleLinux/OL7/oracle/instantclient/x86_64/index.html.)

- **Other OSes and architectures:**

  1. Go to the Oracle Instant Client Downloads page and select the download for your system's OS and architecture.
  2. On the download page, accept the Oracle Technology Network License Agreement, download the latest version of the **Basic Package**, and then install it by following the instructions at the bottom of the download page.

Download and Install cx_Oracle

Use Python's `pip` package to install `cx_Oracle` from PyPI (the Python Package Index).:

- **Oracle Linux:**

  Run these commands to download the `pip` package and then use it to install `cx_Oracle`:

  ```
  sudo yum -y install oracle-release-el7
  curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
  sudo python get-pip.py
  python -m pip install cx_Oracle --upgrade
  ```

- **Other OSes and architectures:**

  Run this command:

  ```
  python -m pip install cx_Oracle --upgrade
  ```

Download and Install Client Credentials for the Database

1. Download the zip file containing client credentials for your database to a secure directory on your computer.

   This zip file is available for download from the database's Details page in the Oracle Cloud console. If you have an Oracle Cloud user account that permits you to access this page, download the credentials as follows. If you don't have such an account, you need to get the zip file from the administrator of the database, together with the password that was used to protect the zip file.

   a. In your web browser, sign in to Oracle Cloud and navigate to the Details page for the JSON database.

   b. Click **DB Connection**.

   c. On the **Database Connection** page click **Download**.

d. In the **Download Wallet** dialog, enter a wallet password in the **Password** field and confirm the password in the **Confirm Password** field.

   The password must be at least 8 characters long and must include at least 1 letter and either 1 numeric character or 1 special character.

e. Click **Download** to save the client credentials zip file to a secure directory.

2. After downloading the zip file, follow these steps:

   a. Unzip the client credentials zip file.

   b. Edit the `sqlnet.ora` file provided in the client credentials, replacing `"?/network/admin"` with the full path of the directory where you unzipped the client credentials; for example, change:

   ```
   (DIRECTORY="?/network/admin")
   ```

   to:

   ```
   (DIRECTORY="/users/jdoe/adbcredentials")
   ```

   c. Create the `TNS_ADMIN` environment variable, setting its value to the full path of the directory where you unzipped the client credentials.

Code Database Connections and SQL Statements

Follow these guidelines to achieve high performance for your application's connections to the database.

- Use connection pools.
- Use the predefined database service that best matches the operations you will be performing. For information about the predefined database services, see Database Service Names for Autonomous Transaction Processing and Autonomous JSON Database in *Using Oracle Autonomous Database Serverless*.

For example:

```
pool = cx_Oracle.SessionPool("appuser",
                             SampleEnv.GetMainPassword(),
                             "tp",
                             events=True,
                             threaded=True)
```

This example creates a pool for connections to the `tp` database service.

Additional Resources

For detailed information about cx_Oracle, go to the cx_Oracle Documentation page.

For code examples that demonstrate a wide variety of cx_Oracle features, go to the python-cx_Oracle samples folder.

6

Load JSON

The PL/SQL procedure `DBMS_CLOUD.COPY_COLLECTION` provides support for loading JSON documents into SODA collections. The procedure `DBMS_CLOUD.COPY_DATA` provides support for loading JSON data into an existing table in Autonomous Database.

About Loading JSON Documents

You load SODA collections into Autonomous JSON Database using the PL/SQL procedure `DBMS_CLOUD.COPY_COLLECTION` and you load JSON data into a table using `DBMS_CLOUD.COPY_DATA`.

- `DBMS_CLOUD.COPY_COLLECTION` supports the following typical document loading procedures:
  - Loading line-delimited JSON into a collection. See Load a JSON File of Line-Delimited Documents into a Collection for this procedure.
  - Loading an array of JSON documents into a collection. See Load an Array of JSON Documents into a Collection for this procedure.
- `DBMS_CLOUD.COPY_DATA` supports the following for loading from JSON data in Object Store:
  - Create Credentials and Copy JSON Data into an Existing Table

Load a JSON File of Line-Delimited Documents into a Collection

For loading data from collections in the Cloud, you must first store your object storage credentials in your Autonomous Database and then use the procedure `DBMS_CLOUD.COPY_COLLECTION` to load documents into a collection.

This example loads JSON values from a line-delimited file and uses the JSON file `myCollection.json`. Each value, each line, is loaded into a collection on your JSON database as a single document.

Here's an example of such a file. It has three lines, with one object per line. Each of those objects gets loaded as a separate JSON document.

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

Before loading the data from `myCollection.json` into your database, copy the file to your object store:

- Create a bucket in the object store. For example, create an Oracle Cloud Infrastructure Object Storage bucket from the Oracle Cloud Infrastructure Object Storage link, and then in your selected compartment click **Create Bucket**, or use a command such as the following OCI CLI command to create a bucket:

  ```
  oci os bucket create --name fruit_bucket -c <compartment id>
  ```

- Copy the JSON file to your object store bucket. For example use the following OCI CLI command to copy the JSON file to the `fruit_bucket` on Oracle Cloud Infrastructure Object Storage:

```
oci os object put --bucket-name fruit_bucket \
                  --file "myCollection.json"
```

Load the JSON file from object store into a collection named `fruit` on your JSON database as follows:

1. Store your object store credentials using the procedure `DBMS_CLOUD.CREATE_CREDENTIAL`, as shown in the following example:

```
SET DEFINE OFF
BEGIN
  DBMS_CLOUD.CREATE_CREDENTIAL(
    credential_name => 'DEF_CRED_NAME',
    username => 'ads_user@example.com',
    password => 'password'
  );
END;
/
```

   This operation stores the credentials in the database in an encrypted format. You can use any name for the credential name. Note that this step is required only once unless your object store credentials change. Once you store the credentials, you can use the same credential name for loading all documents.

   Creating a credential to access Oracle Cloud Infrastructure Object Store is not required if you enable resource principal credentials. See Use Resource Principal to Access Oracle Cloud Infrastructure Resources for more information.

   See CREATE_CREDENTIAL Procedure for detailed information about the parameters.

   > **Note:**
   >
   > Some tools like SQL*Plus and SQL Developer use the ampersand character (`&`) as a special character. If you have the ampersand character in your password, then use the `SET DEFINE OFF` command in those tools as shown in the example to disable the special character, and get the credential created properly.

2. Load the data into a collection using the procedure `DBMS_CLOUD.COPY_COLLECTION`.

```
BEGIN
 DBMS_CLOUD.COPY_COLLECTION(
    collection_name => 'fruit',
    credential_name => 'DEF_CRED_NAME',
    file_uri_list   =>
      'https://objectstorage.us-ashburn-1.oraclecloud.com/n/namespace-string/b/
fruit_bucket/o/myCollection.json',
    format          =>
      JSON_OBJECT('recorddelimiter' value '''\n''')  );
```

```
END;
/
```

The parameters are:

- `collection_name`: is the name of the target collection.
- `credential_name`: is the name of the credential created in the previous step. The `credential_name` parameter must conform to Oracle object naming conventions. See Database Object Naming Rules for more information.
- `file_uri_list`: is a comma delimited list of the source files that you want to load.
- `format`: defines the options that you can specify to describe the format of the source file. The format options `characterset, compression, encryption, ignoreblanklines, jsonpath, maxdocsize, recorddelimiter, rejectlimit, type, unpackarrays` are supported while loading JSON data. Any other formats specified will result in an error.

  If the data in your source files is encrypted, decrypt the data by specifying the `format` parameter with the `encryption` option. See Decrypt Data While Importing from Object Storage for more information on decrypting data.

  See DBMS_CLOUD Package Format Options for more information.

Where *`namespace-string`* is the Oracle Cloud Infrastructure object storage namespace and `fruit_bucket` is the bucket name. See Understanding Object Storage Namespaces and Overview of Object Storage for more information.

For detailed information about the parameters, see COPY_COLLECTION Procedure.

The collection `fruit` on your JSON database now contains one document for each line in the file `myCollection.json`.

Load an Array of JSON Documents into a Collection

To load data from collections in the Cloud, you first store your object storage credentials in your Autonomous Database and then use PL/SQL procedure `DBMS_CLOUD.COPY_COLLECTION` to load documents into a collection. This topic explains how to load documents to your database from a JSON array in a file.

> **Note:**
>
> You can also load documents from a JSON array in a file into a collection using SODA for REST. See Load Purchase-Order Sample Data Using SODA for REST.

This example uses the JSON file `fruit_array.json`. The following shows the contents of the file `fruit_array.json`:

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

Before loading data into Autonomous JSON Database, copy the data to your object store as follows:

- Create a bucket in the object store. For example, create an Oracle Cloud Infrastructure Object Store bucket from the Oracle Cloud Infrastructure Object Storage link, in your selected Compartment, by clicking **Create Bucket**, or use a command line tool such as the following OCI CLI command:

  ```
  oci os bucket create -name json_bucket -c <compartment id>
  ```

- Copy the JSON file to the object store. For example, the following OCI CLI command copies the JSON file `fruit_array.json` to the object store:

  ```
  oci os object put --bucket-name json_bucket --file "fruit_array.json"
  ```

Load the JSON file from object store into a SODA collection named `fruit2` on your JSON database:

1. Store your object store credentials using the procedure `DBMS_CLOUD.CREATE_CREDENTIAL`, as shown in the following example:

   ```
   SET DEFINE OFF
   BEGIN
     DBMS_CLOUD.CREATE_CREDENTIAL(
       credential_name => 'DEF_CRED_NAME',
       username => 'ads_user@example.com',
       password => 'password'
     );
   END;
   /
   ```

   This operation stores the credentials in the database in an encrypted format. You can use any name for the credential name. Note that this step is required only once unless your object store credentials change. Once you store the credentials, you can use the same credential name for loading all documents.

   See CREATE_CREDENTIAL Procedure for detailed information about the parameters.

   > **Note:**
   >
   > Some tools like SQL*Plus and SQL Developer use the ampersand character (`&`) as a special character. If you have the ampersand character in your password, then use the `SET DEFINE OFF` command in those tools as shown in the example to disable the special character, and get the credential created properly.

2. Load the data into a collection using the procedure `DBMS_CLOUD.COPY_COLLECTION`.

   ```
   BEGIN
     DBMS_CLOUD.COPY_COLLECTION(
       collection_name => 'fruit2',
       credential_name => 'DEF_CRED_NAME',
       file_uri_list => 'https://objectstorage.us-ashburn-1.oraclecloud.com/n/
   namespace-string/b/json/o/fruit_array.json',
       format => '{"recorddelimiter" : "0x''01''", "unpackarrays" : "TRUE",
   "maxdocsize" : "10240000"}'
     );
   ```

```
END;
/
```

In this example you load a single JSON value which occupies the whole file, so there is no need to specify a record delimiter. To indicate that there is no record delimiter, you can use a character that does not occur in the input file. For this example, to indicate that there is no delimiter, the control character 0x01 (SOH) is set to load the JSON documents into a collection,. Thus, you specify a value for the `recorddelimiter` that does not occur in the JSON file. For example, you can use value `"0x''01''"` because this character does not occur directly in JSON text.

When `unpackarrays` parameter for format value is set to `TRUE`, the array of documents is loaded as individual documents rather than as an entire array. The unpacking of array elements is however limited to single level. If there are nested arrays in the documents, those arrays are not unpacked.

The parameters are:

- `collection_name`: is the name of the target collection.
- `credential_name`: is the name of the credential created in the previous step. The `credential_name` parameter must conform to Oracle object naming conventions. See Database Object Naming Rules for more information.
- `file_uri_list`: is a comma delimited list of the source files that you want to load.
- `format`: defines the options that you can specify to describe the format of the source file. The format options `characterset, compression, encryption, ignoreblanklines, jsonpath, maxdocsize, recorddelimiter, rejectlimit, type, unpackarrays` are supported for loading JSON data. Any other formats specified will result in an error.

  If the data in your source files is encrypted, decrypt the data by specifying the `format` parameter with the `encryption` option. See Decrypt Data While Importing from Object Storage for more information on decrypting data.

  See DBMS_CLOUD Package Format Options for more information.

In this example, *`namespace-string`* is the Oracle Cloud Infrastructure object storage namespace and *`bucketname`* is the bucket name. See Understanding Object Storage Namespaces for more information.

For detailed information about the parameters, see COPY_COLLECTION Procedure.

The load of `fruit_array.json`, with `DBMS_CLOUD.COPY_COLLECTION` using the format option `unpackarrays` recognizes array values in the source and instead of loading the data as a single document, as it would by default, the data is loaded in the collection `fruit2` with each value in the array as a single document.

Create Credentials and Copy JSON Data into an Existing Table

Use `DBMS_CLOUD.COPY_DATA` to load JSON data in the cloud into a table.

The source file in this example is a JSON data file.

1. Store your object store credentials using the procedure `DBMS_CLOUD.CREATE_CREDENTIAL`. For example:

```
SET DEFINE OFF
BEGIN
```

```
  DBMS_CLOUD.CREATE_CREDENTIAL(
    credential_name => 'DEF_CRED_NAME',
    username => 'ads_user@example.com',
    password => 'password'
  );
END;
/
```

This operation stores the credentials in the database in an encrypted format. You can use any name for the credential name. Note that this step is required only once unless your object store credentials change. Once you store the credentials you can then use the same credential name for all data loads.

For detailed information about the parameters, see CREATE CREDENTIAL Procedure.

Creating a credential to access Oracle Cloud Infrastructure Object Store is not required if you enable resource principal credentials. See Use Resource Principal to Access Oracle Cloud Infrastructure Resources for more information.

2. Load JSON data into an existing table using the procedure `DBMS_CLOUD.COPY_DATA`.

   For example:

```
CREATE TABLE WEATHER2
    (WEATHER_STATION_ID VARCHAR2(20),
     WEATHER_STATION_NAME VARCHAR2(50));
/

BEGIN
  DBMS_CLOUD.COPY_DATA(
      table_name      => 'WEATHER2',
      credential_name => 'DEF_CRED_NAME',
      file_uri_list   => 'https://objectstorage.us-
phoenix-1.oraclecloud.com/n/namespace-string/b/bucketname/o/jsonfiles*',
      format          =>  JSON_OBJECT('type' value 'json', 'columnpath'
value '["$.WEATHER_STATION_ID",
          "$.WEATHER_STATION_NAME"]')
    );
END;
/
```

   The parameters are:

   - `table_name`: is the target table's name.
   - `credential_name`: is the name of the credential created in the previous step.
   - `file_uri_list`: is a comma delimited list of the source files you want to load. You can use wildcards in the file names in your URIs. The character "*" can be used as the wildcard for multiple characters, the character "?" can be used as the wildcard for a single character.
   - `format`: for `DBMS_CLOUD.COPY_DATA` with JSON data, the `type` is `json`. Specify other format values to define the options to describe the format of the JSON source file. See DBMS_CLOUD Package Format Options for more information.

   In this example, *`namespace-string`* is the Oracle Cloud Infrastructure object storage namespace and *`bucketname`* is the bucket name. See Understanding Object Storage Namespaces for more information.

For detailed information about the parameters, see COPY_DATA Procedure.

Monitor and Troubleshoot COPY_COLLECTION Loads

All data load operations you perform using the PL/SQL package `DBMS_CLOUD` are logged in the tables `dba_load_operations` and `user_load_operations`. Use these tables to monitor loading with `DBMS_CLOUD.COPY_COLLECTION`.

- `dba_load_operations` shows all load operations
- `user_load_operations` shows the load operations in your schema

You can query these tables to see information about ongoing and completed data loads. For example, the following `SELECT` statement with a `WHERE` clause predicate on the `TYPE` column shows load operations of the type `COPY`:

```
SELECT table_name, owner_name, type, status, start_time, update_time,
logfile_table, badfile_table
   FROM user_load_operations WHERE type = 'COPY';
TABLE_NAME  OWNER_NAME   TYPE    STATUS     START_TIME
UPDATE_TIME           LOGFILE_TABLE   BADFILE_TABLE
FRUIT       ADMIN        COPY    COMPLETED  2020-04-23 22:27:37    2020-04-23
22:27:38   ""             ""
FRUIT       ADMIN        COPY    FAILED     2020-04-23 22:28:36    2020-04-23
22:28:37   COPY$2_LOG      COPY$2_BAD
```

The `LOGFILE_TABLE` column shows the name of the table you can query to look at the log of a load operation. For example, the following query shows the log of the load operation with status `FAILED` and timestamp `2020-04-23 22:28:36`:

```
SELECT * FROM COPY$2_LOG;
```

The column `BADFILE_TABLE` shows the name of the table you can query to review information for the rows reporting errors during loading. For example, the following query shows the rejected records for the load operation:

```
SELECT * FROM COPY$2_BAD;
```

Depending on the errors shown in the log and the rows shown in the `BADFILE_TABLE` table, you might be able to correct errors by specifying different format options with `DBMS_CLOUD.COPY_COLLECTION`.

> **Note:**
>
> The `LOGFILE_TABLE` and `BADFILE_TABLE` tables are stored for two days for each load operation and then removed automatically.

See DELETE_ALL_OPERATIONS Procedure for information on clearing the `user_load_operations` table.

Import SODA Collection Data Using Oracle Data Pump Version 19.6 or Later

Shows the steps to import SODA collections into Autonomous Database with Oracle Data Pump.

You can export and import SODA collections using Oracle Data Pump Utilities starting with version 19.6. Oracle recommends using the latest Oracle Data Pump version for importing data from Data Pump files into your JSON database.
Download the latest version of Oracle Instant Client, which includes Oracle Data Pump, for your platform from Oracle Instant Client Downloads. See the installation instructions on the platform install download page for the installation steps required after you download Oracle Instant Client.

In Oracle Data Pump, if your source files reside on Oracle Cloud Infrastructure Object Storage you can use Oracle Cloud Infrastructure native URIs, Swift URIs, or pre-authenticated URIs. See DBMS_CLOUD Package File URI Formats for details on these file URI formats.

If you are using an Oracle Cloud Infrastructure pre-authenticated URI, you still need to supply a `credential` parameter. However, credentials for a pre-authenticated URL are ignored (and the supplied credentials do not need to be valid). See DBMS_CLOUD Package File URI Formats for information on Oracle Cloud Infrastructure pre-authenticated URIs.

This example shows how to create the SODA collection metadata and import a SODA collection with Data Pump.

1. On the source database, export the SODA collection using the Oracle Data Pump `expdp` command.

   See Export Your Existing Oracle Database to Import into Autonomous JSON Database for more information.

2. Upload the dump file set from Step 1 to Cloud Object Storage.

3. Create a SODA collection with the required SODA collection metadata on your Autonomous Database.

   For example, if you export a collection named *MyCollectionName* from the source database with the following metadata:

   - The content column is a `BLOB` type.
   - The version column uses the `SHA256` method.

   Then on the Autonomous Database where you import the collection, create a new collection:

   - By default on Autonomous Database for a new collection the content column is set to `BLOB` with the `jsonFormat` specified as `OSON`.
   - By default on Autonomous Database for a new collection the `versionColumn.method` is set to `UUID`.

   See SODA Default Collection Metadata on Autonomous Database for details.

   For example:

```
DECLARE
   collection_create SODA_COLLECTION_T;
```

```
BEGIN
   collection_create := DBMS_SODA.CREATE_COLLECTION('MyCollectionName');
END;
/
COMMIT;
```

You can use the PL/SQL function `DBMS_SODA.LIST_COLLECTION_NAMES` to discover existing collections. See LIST_COLLECTION_NAMES Function for more information.

You can view the metadata for the SODA collections by querying the view `USER_SODA_COLLECTIONS`. See USER_SODA_COLLECTIONS for more information.

4. Store your Cloud Object Storage credential using `DBMS_CLOUD.CREATE_CREDENTIAL`.

   For example, to create Oracle Cloud Infrastructure Auth Token credentials:

```
BEGIN
  DBMS_CLOUD.CREATE_CREDENTIAL(
    credential_name => 'DEF_CRED_NAME',
    username => 'ads_user@example.com',
    password => 'password'
  );
END;
/
```

   For more information on Oracle Cloud Infrastructure Auth Token authentication CREATE_CREDENTIAL Procedure.

   For example, to create Oracle Cloud Infrastructure Signing Key based credentials:

```
BEGIN
   DBMS_CLOUD.CREATE_CREDENTIAL (
       credential_name => 'DEF_CRED_NAME',
       user_ocid       =>
'ocid1.user.oc1..aaaaaaaauq54mi7zdyfhw33ozkwuontjceel7fok5nq3bf2vwetkpqsoa'
,
       tenancy_ocid    =>
'ocid1.tenancy.oc1..aabbbbbbaafcue47pqmrf4vigneebgbcmmoy5r7xvoypicjqqge32ew
nrcyx2a',
       private_key     =>
'MIIEogIBAAKCAQEAtUnxbmrekwgVac6FdWeRzoXvIpA9+0r1.....wtnNpESQQQ0QLGPD8NM//
JEBg=',
       fingerprint     =>
'f2:db:f9:18:a4:aa:fc:94:f4:f6:6c:39:96:16:aa:27');
END;
/
```

   For more information on Oracle Cloud Infrastructure Signing Key based credentials see CREATE_CREDENTIAL Procedure.

   Supported credential types:

   - Data Pump Import supports Oracle Cloud Infrastructure Auth Token based credentials and Oracle Cloud Infrastructure Signing Key based credentials.

     For more information on Oracle Cloud Infrastructure Signing Key based credentials see CREATE_CREDENTIAL Procedure.

- Data Pump supports using an Oracle Cloud Infrastructure Object Storage pre-authenticated URL for the `dumpfile` parameter. When you use a pre-authenticated URL, providing the `credential` parameter is required and `impdp` ignores the `credential` parameter. When you use a pre-authenticated URL for the `dumpfile`, you can use a `NULL` value for the `credential` in the next step. See Using Pre-Authenticated Requests for more information.

5. Run Data Pump Import with the *dumpfile* parameter set to the list of file URLs on your Cloud Object Storage and the *credential* parameter set to the name of the credential you created in the previous step.

> **Note:**
>
> Import the collection data using the option `CONTENT=DATA_ONLY`.

Specify the collection you want to import using the `INCLUDE` parameter. This is useful if a data file set contains the entire schema and the SODA collection you need to import is included as part of the dump file set.

Use `REMAP_DATA` to change any of the columns during import. This example shows using `REMAP_DATA` to change the version column method from `SHA256` to `UUID`.

```
impdp admin/password@ADS1_high \
     directory=data_pump_dir \
     credential=def_cred_name \
     dumpfile= https://namespace-string.objectstorage.us-ashburn-1.oci.customer-
oci.com/n/namespace-string/b/bucketname/o/export%l.dmp \

     encryption_pwd_prompt=yes \
     SCHEMA=my_schema \
     INCLUDE=TABLE:\"= \'MyCollectionName\'\"  \
     CONTENT=DATA_ONLY \
     REMAP_DATA=my_schema.'\"MyCollectionName\"'.VERSION:SYS.DBMS_SODA.TO_UUID
```

Notes for Data Pump parameters:

- If during the export with `expdp` you used the `encryption_pwd_prompt=yes` parameter then use `encryption_pwd_prompt=yes` and input the same password at the `impdp` prompt that you specified during the export.
- The `dumpfile` parameter supports the `%L` and `%l` wildcards in addition to the legacy `%U` and `%u` wildcards. For example, `dumpfile=export%L.dmp`. Use the `%L` or `%l` wildcard for exports from Oracle Database Release 12.2 and higher. This wildcard expands the dumpfile file name into a 3-digit to 10-digit, variable-width incrementing integer, starting at 100 and ending at 2147483646.

  Use the legacy `%U` or `%u` wildcard for exports from Oracle Database prior to Release 12.2. If you use this option and more than 99 dump files are needed, you must specify multiple dumpfile names, each with the `%U` or `%u` parameter.

For `dumpfile`, this example uses the recommended URI format using OCI Dedicated Endpoints for commercial realm (OC1). The *`namespace-string`* is the Oracle Cloud Infrastructure object storage namespace and *`bucketname`* is the bucket name. See Object Storage Dedicated Endpoints, Regions and Availability Domains, and Understanding Object Storage Namespaces for more information.

In Oracle Data Pump version 19.6 and later, the *credential* argument authenticates Oracle Data Pump to the Cloud Object Storage service you are using for your source files. The `credential` parameter cannot be an Azure service principal, Amazon Resource Name (ARN), or a Google service account. See Accessing Cloud Resources by Configuring Policies and Roles for more information on resource principal based authentication.

The *dumpfile* argument is a comma delimited list of URLs for your Data Pump files.

For the best import performance use the `HIGH` database service for your import connection and set the `parallel` parameter to one quarter the number of ECPUs (.25 x **ECPU count**). If you are using OCPU compute model, set the parallel parameter to the number of OCPUs (1 x **OCPU count**).

For information on which database service name to connect to run Data Pump Import, see Manage Concurrency and Priorities on Autonomous JSON Database.

For the dump file URL format for different Cloud Object Storage services, see DBMS_CLOUD Package File URI Formats.

> **Note:**
>
> To perform a full import or to import objects that are owned by other users, you need the `DATAPUMP_CLOUD_IMP` role.

For information on disallowed objects in Autonomous JSON Database, see SQL Commands.

In this import example, the specification for the `REMAP_DATA` parameter uses the function `DBMS_SODA.TO_UUID` to generate UUID values. By default, for on-premise databases, the version column of a SODA collection is computed using SHA-256 hash of the document's content. On Autonomous Database the version column uses UUID generated values, which are independent of the document's content.

In this example the `REMAP_DATA` parameter uses the `DBMS_SODA.TO_UUID` function to replace the source collection version type with UUID versioning. If in the export dump file set that you are importing the `versionColumn.method` is already set to UUID, then the `REMAP_DATA` for this field is not required.

For detailed information on Oracle Data Pump Import parameters see *Oracle Database Utilities*.

The log files for Data Pump Import operations are stored in the directory you specify with the Data Pump Import `DIRECTORY` parameter. See Accessing the Log File for Data Pump Import for more information.

Textual JSON Objects That Represent Extended Scalar Values

Native binary JSON data (OSON format) extends the JSON language by adding scalar types, such as date, that correspond to SQL types and are not part of the JSON standard. Oracle Database also supports the use of textual JSON *objects* that *represent* JSON scalar values, including such nonstandard values.

When you create native binary JSON data from textual JSON data that contains such **extended objects**, they can optionally be *replaced* with corresponding (native binary) JSON scalar values.

An example of an extended object is `{"$numberDecimal":31}`. It represents a JSON scalar value of the nonstandard type *decimal number*, and when interpreted as such it is replaced by a decimal number in native binary format.

For example, when you use the JSON data type constructor, `JSON`, if you use keyword `EXTENDED` then recognized extended objects in the textual input are replaced with corresponding scalar values in the native binary JSON result. If you do not include keyword `EXTENDED` then no such replacement occurs; the textual extended JSON objects are simply converted as-is to JSON objects in the native binary format.

In the opposite direction, when you use SQL/JSON function `json_serialize` to serialize binary JSON data as textual JSON data (`VARCHAR2`, `CLOB`, or `BLOB`), you can use keyword `EXTENDED` to replace (native binary) JSON scalar values with corresponding textual extended JSON objects.

> **Note:**
>
> If the database you use is an Oracle Autonomous Database then you can use PL/SQL procedure `DBMS_CLOUD.copy_collection` to create a JSON document collection from a file of JSON data such as that produced by common NoSQL databases, including Oracle NoSQL Database.
>
> If you use **`ejson`** as the value of the `type` parameter of the procedure, then recognized extended JSON objects in the input file are replaced with corresponding scalar values in the resulting native binary JSON collection. In the other direction, you can use function `json_serialize` with keyword `EXTENDED` to replace scalar values with extended JSON objects in the resulting textual JSON data.

These are the two main use cases for extended objects:

- *Exchange* (import/export):
  - Ingest existing JSON data (from somewhere) that contains extended objects.
  - Serialize native binary JSON data as textual JSON data with extended objects, for some use outside the database.
- *Inspection* of native binary JSON data: see what you have by looking at corresponding extended objects.

For exchange purposes, you can ingest JSON data from a file produced by common NoSQL databases, including Oracle NoSQL Database, converting extended objects to native binary JSON scalars. In the other direction, you can export native binary JSON data as textual data, replacing Oracle-specific scalar JSON values with corresponding textual extended JSON objects.

> **Tip:**
>
> As an example of inspection, consider an object such as `{"dob" : "2000-01-02T00:00:00"}` as the result of serializing native JSON data. Is `"2000-01-02T00:00:00"` the result of serializing a native binary value of type date, or is the native binary value just a string? Using `json_serialize` with keyword `EXTENDED` lets you know.

The mapping of extended object fields to scalar JSON types is, in general, many-to-one: more than one kind of extended JSON object can be mapped to a given scalar value. For example, the extended JSON objects `{"$numberDecimal":"31"}` and `{"$numberLong:"31"}` are both translated as the value 31 of JSON-language scalar type number, and item method `type()` returns `"number"` for each of those JSON scalars.

Item method `type()` reports the JSON-language scalar type of its targeted value (as a JSON string). Some scalar values are distinguishable internally, even when they have the same scalar type. This generally allows function `json_serialize` (with keyword `EXTENDED`) to reconstruct the original extended JSON object. Such scalar values are distinguished internally either by using *different SQL types* to implement them or by *tagging them with the kind of extended JSON object* from which they were derived.

When `json_serialize` reconstructs the original extended JSON object the result is not always *textually* identical to the original, but it is always *semantically* equivalent. For example, `{"$numberDecimal":"31"}` and `{"$numberDecimal":31}` are semantically equivalent, even though the field values differ in type (string and number). They are translated to the same internal value, and each is tagged as being derived from a `$numberDecimal` extended object (same tag). But when serialized, the *result for both* is `{"$numberDecimal":31}`. Oracle always uses the most directly relevant type for the field value, which in this case is the JSON-language value `31`, of scalar type number.

Table 6-1 presents correspondences among the various types used. It maps across (1) types of extended objects used as input, (2) types reported by item method `type()`, (3) SQL types used internally, (4) standard JSON-language types used as output by function `json_serialize`, and (5) types of extended objects output by `json_serialize` when keyword `EXTENDED` is specified.

Table 6-1 Extended JSON Object Type Relations

Extended Object Type (Input)	Oracle JSON Scalar Type (Reported by type())	SQL Scalar Type	Standard JSON Scalar Type (Output)	Extended Object Type (Output)
`$numberDouble` with value a JSON number, a string representing the number, or one of these strings: `"Infinity"`, `"-Infinity"`, `"Inf"`, `"-Inf"`, `"Nan"`[1]	double	`BINARY_DOUBLE`	number	`$numberDouble` with value a JSON number or one of these strings: `"Inf"`, `"-Inf"`, `"Nan"`[2]
`$numberFloat` with value the same as for `$numberDouble`	float	`BINARY_FLOAT`	number	`$numberFloat` with value the same as for `$numberDouble`
`$numberDecimal` with value the same as for `$numberDouble`	number	`NUMBER`	number	`$numberDecimal` with value the same as for `$numberDouble`
`$numberInt` with value a signed 32-bit integer or a string representing the number	number	`NUMBER`	number	`$numberInt` with value the same as for `$numberDouble`
`$numberLong` with value a JSON number or a string representing the number	number	`NUMBER`	number	`$numberLong` with value the same as for `$numberDouble`

Table 6-1 (Cont.) Extended JSON Object Type Relations

Extended Object Type (Input)	Oracle JSON Scalar Type (Reported by type())	SQL Scalar Type	Standard JSON Scalar Type (Output)	Extended Object Type (Output)
`$binary` with value one of these: • a string of base-64 characters • An object with fields `base64` and `subType`, whose values are a string of base-64 characters and the number `0` (arbitrary binary) or `4` (UUID), respectively When the value is a string of base-64 characters, the extended object can also have field `$subtype` with value 0 or 4, expressed as a one-byte integer (0-255) or a 2-character hexadecimal string. representing such an integer	binary	`BLOB` or `RAW`	string Conversion is equivalent to the use of SQL function `rawtohex`.	One of the following: • `$binary` with value a string of base-64 characters • `$rawid` with value a string of 32 hexadecimal characters, if input had a `subType` value of `4` (UUID)
`$oid` with value a string of 24 hexadecimal characters	binary	`RAW(12)`	string Conversion is equivalent to the use of SQL function `rawtohex`.	`$rawid` with value a string of 24 hexadecimal characters
`$rawhex` with value a string with an even number of hexadecimal characters	binary	`RAW`	string Conversion is equivalent to the use of SQL function `rawtohex`.	`$binary` with value a string of base-64 characters, right-padded with = characters
`$rawid` with value a string of 24 or 32 hexadecimal characters	binary	`RAW`	string Conversion is equivalent to the use of SQL function `rawtohex`.	`$rawid`
`$oracleDate` with value an ISO 8601 date string	date	`DATE`	string	`$oracleDate` with value an ISO 8601 date string
`$oracleTimestamp` with value an ISO 8601 timestamp string	timestamp	`TIMESTAMP`	string	`$oracleTimestamp` with value an ISO 8601 timestamp string
`$oracleTimestampTZ` with value an ISO 8601 timestamp string with a numeric time zone offset or with `Z`	timestamp with time zone	`TIMESTAMP WITH TIME ZONE`	string	`$oracleTimestampTZ` with value an ISO 8601 timestamp string with a numeric time zone offset or with `Z`

Table 6-1 (Cont.) Extended JSON Object Type Relations

Extended Object Type (Input)	Oracle JSON Scalar Type (Reported by type())	SQL Scalar Type	Standard JSON Scalar Type (Output)	Extended Object Type (Output)
`$date` with value one of the following: • An integer millisecond count since January 1, 1990 • An ISO 8601 timestamp string • An object with field `numberLong` with value an integer millisecond count since January 1, 1990	timestamp with time zone	`TIMESTAMP WITH TIME ZONE`	string	`$oracleTimestampTZ` with value an ISO 8601 timestamp string with a numeric time zone offset or with `Z`
`$intervalDaySecond` with value an ISO 8601 interval string as specified for SQL function `to_dsinterval`	daysecondInterval	`INTERVAL DAY TO SECOND`	string	`$intervalDaySecond` with value an ISO 8601 interval string as specified for SQL function `to_dsinterval`
`$intervalYearMonth` with value an ISO 8601 interval string as specified for SQL function `to_yminterval`	yearmonthInterval	`INTERVAL YEAR TO MONTH`	string	`$intervalYearMonth` with value an ISO 8601 interval string as specified for SQL function `to_yminterval`
Two fields: • Field `$vector` with value an array whose elements are numbers or the strings `"Nan"`, `"Inf"`, and `"-Inf"` (representing not-a-number and infinite values). • Field `$vectorElementType` with string value either`"float32"` or `"float64"`. These correspond to IEEE 32-bit and IEEE 64-bit numbers, respectively.	vector	`VECTOR`	array of numbers	Two fields: • Field `$vector` with value an array whose elements are numbers or the strings `"Nan"`, `"Inf"`, and `"-Inf"` (representing not-a-number and infinite values). • Field `$vectorElementType` with string value either`"float32"` or `"float64"`.

1 The string values are interpreted case-insensitively. For example, `"NAN"` `"nan"`, and `"nAn"` are accepted and equivalent, and similarly `"INF"`, `"inFinity"`, and `"iNf"`. Infinitely large (`"Infinity"` or `"Inf"`) and small (`"-Infinity"` or `"-Inf"`) numbers are accepted with either the full word or the abbreviation.

2 On output, only these string values are used — no full-word *Infinity* or letter-case variants.

> **See Also:**
>
> IEEE Standard for Floating-Point Arithmetic (IEEE 754)

7

Oracle Tools for Database Access

For many database operations you need SQL or PL/SQL access. Oracle Database tools like SQL Developer, SQLcl and SQL*Plus provide such access to JSON databases.

The following sections provide step-by-step instructions for setting up these tools.

Topics

Connect with Built-In Oracle Database Actions

You can access Database Actions from Autonomous JSON Database. Database Actions provides development tools, data tools, administration, and monitoring features for Autonomous JSON Database. Using Database Actions you can run SQL statements, queries, and scripts in a worksheet.

Access Database Actions as ADMIN

Database Actions (also known as SQL Developer Web) is bundled with each Autonomous Database instance.

Database Actions runs in Oracle REST Data Services and access is provided through schema-based authentication. To use Database Actions, you must sign in as a database user whose schema is enabled for Database Actions. By default the ADMIN user is enabled to access Database Actions.

See Provide Database Actions Access to Database Users to enable another database user's schema to access Database Actions.

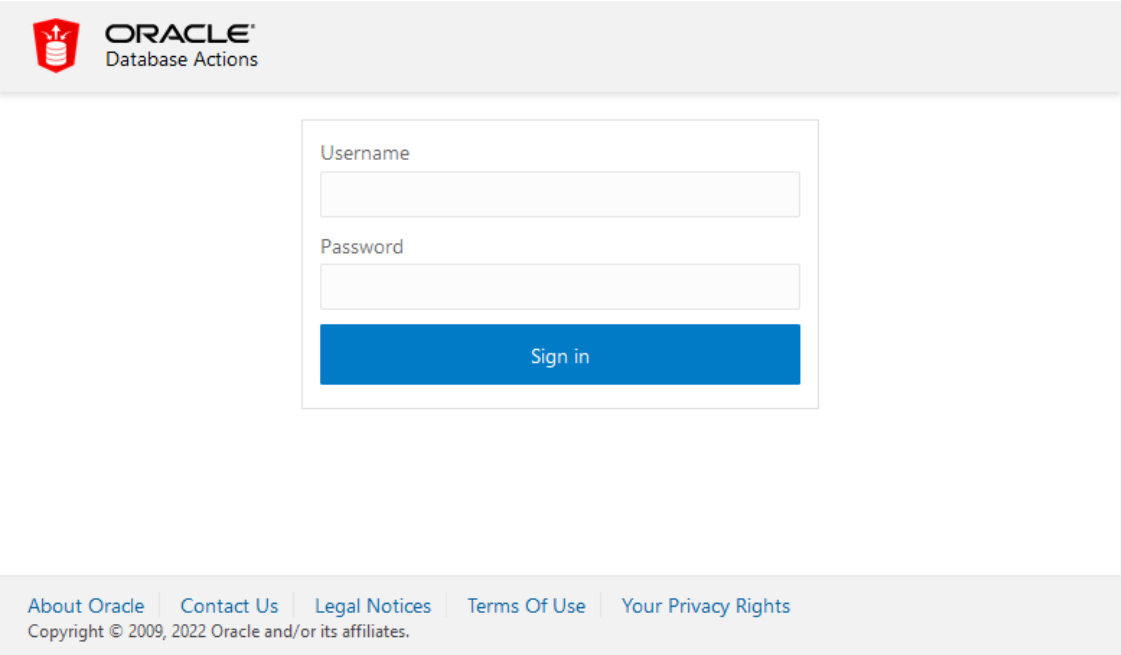
> **Note:**
>
> If your Autonomous JSON Database is configured to use a Private Endpoint, then you can only access Database Actions from clients in the same Virtual Cloud Network (VCN).

To access Database Actions from the Oracle Cloud Infrastructure Console:

1. On the Autonomous Database Details page click the **Database actions** dropdown list.
2. Select a quick link to go directly to a quick link action or select **View all database actions** to access the full Database Actions Launchpad.

For example, click **SQL** to use a SQL Worksheet. On the SQL Worksheet you can use the **Consumer Group** drop-down list to select the consumer group to run your SQL or PL/SQL code. See Executing SQL Statements in the Worksheet Editor for more information.

Depending on your browser, if the Console cannot access the database as ADMIN you will be prompted for your database ADMIN username and password.



Provide Database Actions Access to Database Users

The ADMIN user provides access to Database Actions for other database users.

Database users who are not service administrators do not have access to the Oracle Cloud Infrastructure Console. The ADMIN user provides access to Database Actions as follows:

- Use Database Actions to create a user and assign roles to the user. If the user already exists, check that Web Access is selected for the schema (with Web Access selected, the user's card shows **REST Enabled**).

  See Create Users on Autonomous Database for information on adding database users.

  See Required Roles to Access Tools from Database Actions for information on required roles for Database Actions.

- Provide the user with a URL to access Database Actions.

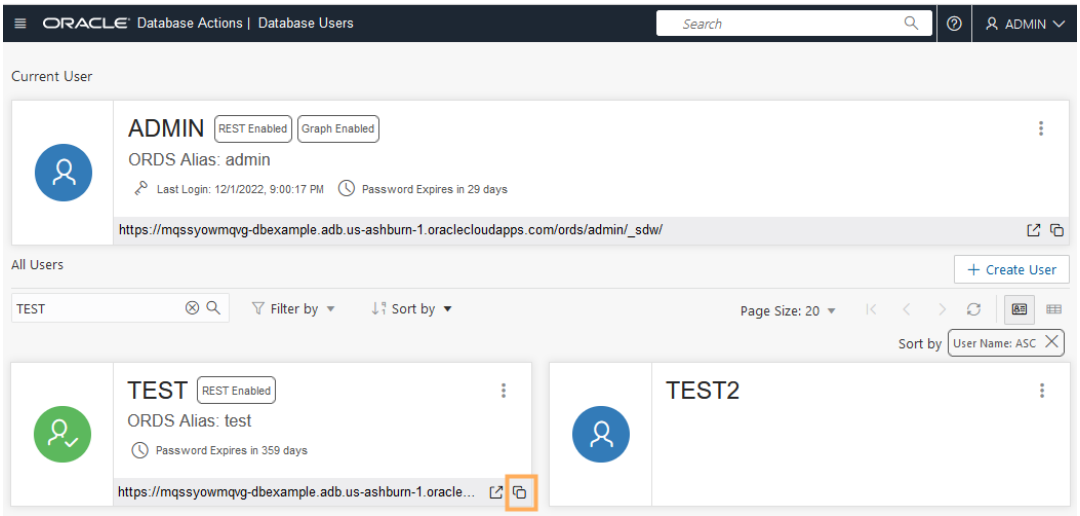
After adding a user and selecting Web Access, the ADMIN provides a user with the URL to access Database Actions:

1. In Database Actions, click ☰ to show the available actions.
2. Under **Administration** select **Database Users**.

   This displays information about users, such as user names, whether a user is REST Enabled, and the last login date and time. On a user's card, the icon on the left displays the user status with one of the following colors: green (Open), blue (Locked), or red (Expired).

The default view is Card View. You can select either grid view or card view by clicking the **Card View** or **Grid View** icons.

3. A URL is displayed in the user's card only if the user is REST Enabled. It provides the URL to access Database Actions. Click to copy the URL to the clipboard.



4. Provide the user with the URL you copied.

After you provide the URL to a user, to access Database Actions the user pastes the URL into their browser and then enters their Username and Password in the Sign-in dialog.

See Manage Users and User Roles on Autonomous Database - Connecting with Database Actions for more information.

Connect Oracle SQL Developer with a Wallet (mTLS)

Oracle SQL Developer is a free integrated development environment that simplifies the development and management of Autonomous Database.

SQL Developer can connect to Autonomous Database and contains enhancements for key Autonomous Database features. You can download the latest version of Oracle SQL Developer for your platform from the **Download** link on this page: Oracle SQL Developer.

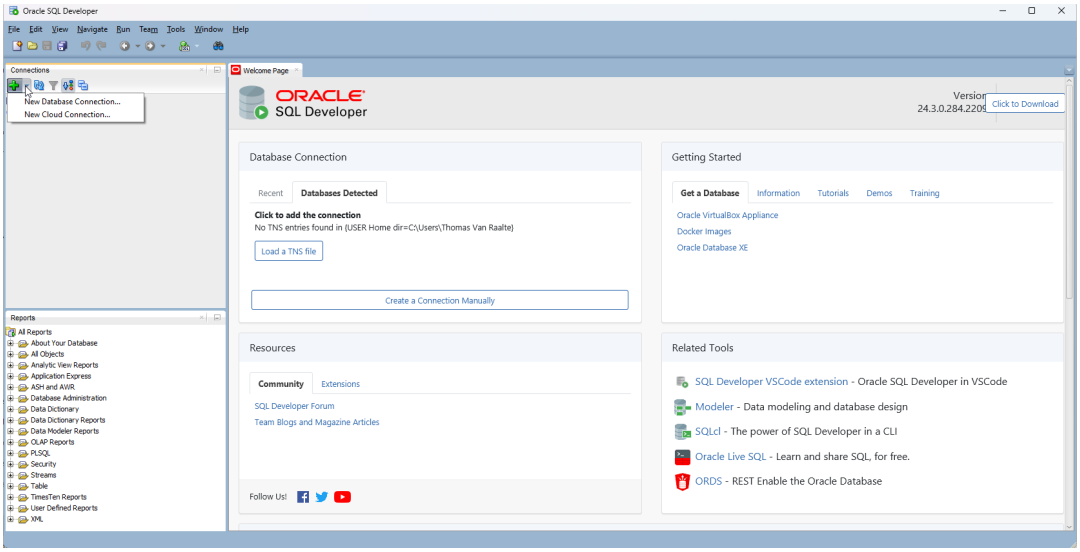
For connecting with mTLS authentication, Oracle SQL Developer provides support for wallet files using the **Cloud Wallet** Connection Type. Oracle recommends that you use version 18.2 (or later); however, earlier versions of SQL Developer will work with Autonomous Database using an Oracle Wallet.

For connecting with TLS authentication, Oracle SQL Developer provides support using the **Custom JDBC** Connection Type. See Connect with Oracle SQL Developer with TLS Authentication for details on connecting using TLS authentication.

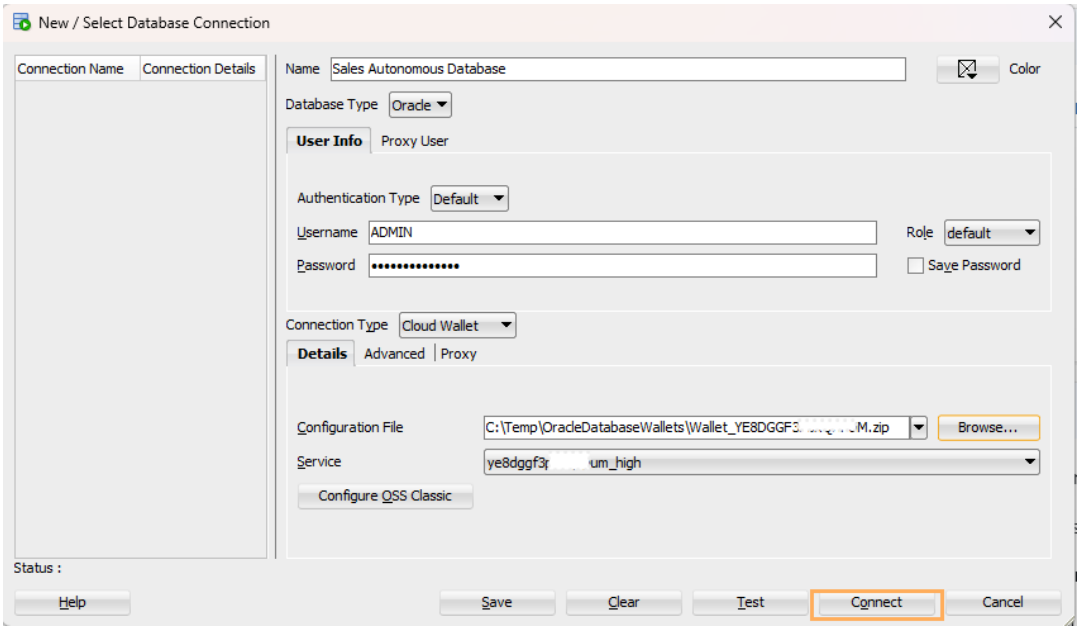
To create a new mTLS connection to Autonomous JSON Database, do the following:

Obtain your credentials to access Autonomous JSON Database. For more information, see Download Client Credentials (Wallets).

1. Start Oracle SQL Developer and in the connections panel, right-click **Connections** and select **New Database Connection...**.



2. Choose the Connection Type **Cloud Wallet**.
3. Enter the following information:
   - **Connection Name**: Enter the name for this connection.
   - **Username**: Enter the database username. You can either use the default administrator database account (`ADMIN`) provided as part of the service or create a new schema, and use it.
   - **Password**: Enter the password for the database user.
   - **Connection Type**: Select **Cloud Wallet**
   - **Configuration File** : Click **Browse**, and select the client credentials zip file.
   - **Service**: Enter the database TNS name. The client credentials file includes a `tnsnames.ora` file that provides database TNS names with corresponding services.



4. Click **Connect** to connect to the database.

> **Note:**
>
> If you are using Microsoft Active Directory, then for **Username** enter the Active Directory "*AD_domain\AD_username*" (you may include double quotes), and for the **Password**, enter the password for the Active Directory user. See Use Microsoft Active Directory with Autonomous Database for more information.

Connect Oracle SQLcl Cloud with a Wallet (mTLS)

SQLcl is a command-line interface used to enter SQL commands. You can use SQLcl to connect to an Autonomous Database with client credentials configured (mTLS).

You can use SQLcl version 4.2 or later with Autonomous Database. Download SQLcl from oracle.com.

SQLcl can connect to an Autonomous Database instance using either an Oracle Call Interface (OCI) or a JDBC thin connection.

- If you use Oracle Call Interface (OCI), prepare for OCI, ODBC and JDBC OCI Connections. See Prepare for Oracle Call Interface (OCI), ODBC, and JDBC OCI Connections.
- If you use JDBC Thin, prepare for JDBC Thin Connections. See Prepare for JDBC Thin Connections.

SQLcl with Oracle Call Interface

To connect using Oracle Call Interface, use the `-oci` option, supply the database user name, a password, and the database service name provided in the `tnsnames.ora` file. For example:

```
sql -oci

SQLcl: Release 22.1 Production on Fri May 06 16:07:46 2022

Copyright (c) 1982, 2022, Oracle.  All rights reserved.

Username? (''?) ads_user@adsc_medium
Password? (**********?) ***************
Connected.
SQL>
```

When connecting using Oracle Call Interface, the Oracle Wallet is transparent to SQLcl.

SQLcl with a JDBC Thin Connection

To connect using a JDBC Thin connection, first configure the SQLcl cloud configuration and then connect to the database.

1. Start SQLcl with the /nolog option.

   ```
   sql /nolog
   ```

2. Configure the SQLcl session to use your Oracle Wallet:

   ```
   SQL> set cloudconfig directory/client_credentials.zip
   ```

3. Connect to the database:

```
SQL> connect username@servicename
password
```

To avoid the prompt, connect and supply the password inline:

```
SQL> connect username/password@servicename
```

For example:

```
sql /nolog

SQLcl: Release 22.1 Production on Fri May 06 14:48:26 2022

Copyright (c) 1982, 2022, Oracle.  All rights reserved.

SQL> set cloudconfig /home/adb/Wallet_db2022ADB.zip

SQL> connect ads_user@adsc_medium

Password? (**********?) ***************
Connected.
SQL>
```

SQLcl with a JDBC Thin Connection with an HTTP Proxy

1. Start SQLcl with the /nolog option.

```
sql /nolog
```

2. Configure the SQLcl session to use a proxy host and your Oracle Wallet:

```
SQL> set cloudconfig -proxy=proxyhost:port directory/client_credentials.zip
```

3. Connect to the database.

```
SQL> connect username@servicename
password
```

To avoid the prompt, connect and supply the password inline:

```
SQL> connect username/password@servicename
```

For example:

```
sql /nolog

SQLcl: Release 22.1 Production on Fri May 06 11:59:38 2022

Copyright (c) 1982, 2022, Oracle.  All rights reserved.
SQL> set cloudconfig -proxy=http://myproxyhost.com:80 /home/adb/
Wallet_db2022.zip

SQL> connect ads_user@adsc_medium

Password? (**********?) ***************
Connected.
SQL>
```

> **Note:**
>
> If you are connecting to Autonomous Database using Microsoft Active Directory credentials, then connect using an Active Directory user name in the form of "*AD_domain\AD_username*" (double quotes must be included), and Active Directory user password. See Use Microsoft Active Directory with Autonomous Database for more information.

For more information, on the connection types specified in `tnsnames.ora`, see Database Service Names for Autonomous Transaction Processing and Autonomous JSON Database.

For information on SQLcl, see Oracle SQLcl.

Connect SQL*Plus with a Wallet (mTLS)

SQL*Plus is a command-line interface used to enter SQL commands. SQL*Plus connects to an Oracle database.

To install and configure the client and connect to the Autonomous JSON Database using SQL*Plus with client credentials (mTLS), do the following:

1. Prepare for Oracle Call Interface (OCI), ODBC and JDBC OCI Connections. See Prepare for Oracle Call Interface (OCI), ODBC, and JDBC OCI Connections with Wallets (mTLS).
2. Connect using a database user, *password*, and database TNS name provided in the `tnsnames.ora` file.

   For example:

```
sqlplus ads_user@adsc_medium

SQL*Plus: Release 19.0.0.0.0 - Production on Mon Nov 23 15:08:48 2020
Version 19.8.0.0.0

Copyright (c) 1982, 2020, Oracle.  All rights reserved.

Enter password:
Last Successful login time: Wed Nov 18 2020 12:36:56 -08:00

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.5.0.0.0

SQL>
```

Notes:

- The Oracle Wallet is transparent to SQL*Plus because the wallet location is specified in the `sqlnet.ora` file. This is true for any Oracle Call Interface (OCI), ODBC, or JDBC OCI connection.
- If you are connecting to a JSON database using Microsoft Active Directory credentials, then connect using an Active Directory user name in the form of "*AD_domain\AD_username*" (double quotes must be included), and Active Directory user password. See Use Microsoft Active Directory with Autonomous Database for more information.

8

Oracle Extensions for IDEs

Oracle extensions let developers connect to, browse, and manage Autonomous Databases directly from common IDEs.

Use Oracle Cloud Infrastructure Toolkit for Eclipse

Oracle Cloud Infrastructure Toolkit for Eclipse is a plugin that enables Java developers to easily connect to Oracle Autonomous Database through their IDE. The plugin is free and is available for Linux, UNIX, Microsoft Windows, and Apple Mac OS.

You can use the plugin to perform cloud and database operations right from Eclipse, such as creating Autonomous Databases, stopping and starting, scaling up and down, and so on. You can also use the plugin to easily connect to the databases to browse the schema, access tables, execute SQL statements, and perform other development tasks.

Users with permissions to manage the databases can perform a number of actions, including those listed below. For detailed information about permissions, see Toolkit for Eclipse in the Oracle Cloud Infrastructure documentation. You can:

- Create Autonomous Databases
- Start, stop, terminate, clone, and restore Autonomous Databases
- Scale up and down
- Download the client credentials zip file (database wallet)
- Connect to Autonomous Databases
- Browse the schema
- Choose compartments and regions
- Change the administrator password, and so on

Download the latest version of the plugin from GitHub (`com.oracle.oci.eclipse-`*`version`*`.zip`, where *`version`* is the latest version, for instance `1.2.0`):

`https://github.com/oracle/oci-toolkit-eclipse/releases`

Then follow the installation instructions and details about how to get started in this step-by-step walkthrough:

New Eclipse Plugin for Accessing Autonomous Database (ATP/ADW)

Use Oracle Developer Tools for Visual Studio

Oracle Developer Tools for Visual Studio is a tightly integrated extension for Microsoft Visual Studio and Oracle Autonomous Database. The extension is free and supports Visual Studio 2019 and Visual Studio 2017 on Microsoft Windows.

You can use the extension to perform database management operations right from Visual Studio, such as creating Autonomous Databases, stopping and starting, scaling up and down, and so on. You can also use the extension to easily connect to the databases and perform

development tasks, such as browsing your Oracle schema and launching integrated Oracle designers and wizards to create and alter schema objects.

Users with permissions to manage the databases can perform a number of actions, including the following:

- Sign up for Oracle Cloud
- Connect to a cloud account using a simple auto-generated config file and key file
- Create new or clone existing Always Free Autonomous Database, Autonomous Database Dedicated, and Autonomous Database Serverless databases
- Automatically download credentials files (including wallets) and quickly connect, browse, and operate on Autonomous Database schemas
- Change compartments and regions without reconnecting
- Start, stop, or terminate Autonomous Database
- Scale up/down Autonomous Database resources
- Restore from backup
- Update instance credentials
- Rotate wallets
- Convert Always Free Autonomous Database to paid:

> **Note:**
>
> Promotion of Always Free to a paid Autonomous Database is supported only if the database version for the Always Free Autonomous Database is Oracle Database 19c or Oracle Database 23ai.

Download the extension from Visual Studio Marketplace:

- Oracle Developer Tools for Visual Studio 2019
- Oracle Developer Tools for Visual Studio 2017

You'll find lots of information about the extension on those Marketplace pages.

Then follow the installation instructions and details about how to get started in this step-by-step walkthrough:

New Release: Visual Studio Integration with Oracle Autonomous Database

For detailed information about how to use the extension, see the online documentation that's optionally installed with Oracle Developer Tools for Visual Studio. Press the F1 key to display the context-sensitive help for each dialog.

Use Oracle Developer Tools for VS Code

Oracle Developer Tools for VS Code is a tightly integrated extension for Microsoft Visual Studio Code (VS Code) and Oracle Autonomous Database. The extension is free and is available for Linux, Microsoft Windows, and Apple Mac OS.

You can use the extension to connect to Autonomous Databases right from Visual Studio Code and easily explore database schema, view table data, and edit and execute SQL and PL/SQL.

Download the extension from Visual Studio Marketplace:

Oracle Developer Tools for VS Code

Installation instructions and information about how to get started can be found in this quick start guide:

Getting Started Using Oracle Developer Tools for VS Code

9

Code for High Performance

Oracle Autonomous JSON Database includes several features that automatically monitor, analyze and optimize the performance of your JSON database.

How your application connects to your database and how you code SQL calls to the database determine the overall performance of your application's transaction processing and reporting operations.

To ensure optimal performance of your application's use of the database, you need to make sure it:

- Connects to the database based on the kind of database operation being performed, as described in Connect for High Performance.
- Uses efficient SQL calls to perform the operation, as described in Code for High Performance.

Oracle provides several tools to help you monitor performance, diagnose performance problems, and tune the performance of your SQL code and the database. See Tools for Monitoring and Tuning Performance.

Connect for High Performance

When making connections to your JSON database, two factors have great impact on the performance of your application's interaction with the database:

- Which database service you connect to: connect to the database service that best matches the database operations you are performing. (For most applications that use JSON documents you use the typical connection service for transaction processing, **tp**.) For a list of the database services and their characteristics, see Database Service Names for Autonomous Transaction Processing and Autonomous JSON Database in *Using Oracle Autonomous Database Serverless*.
- Whether you use connection pools: use connection pools to reduce the performance overhead of repeatedly creating and destroying individual connections. For more information, see Use Connection Pools.

Use Connection Pools

The use of connection pools instead of individual connections can benefit almost every transaction processing application. A connection pool provides the following benefits.

- Reduces the number of times new connection objects are created.
- Promotes connection object reuse.
- Quickens the process of getting a connection.
- Controls the amount of resources spent on maintaining connections.
- Controls the amount of resources spent on maintaining connections.
- Reduces the amount of coding effort required to manually manage connection objects.

Special-Purpose Connection Features

Oracle Net Services (previous called SQL*Net) provides a variety of connection features that improve performance in specific connection scenarios. These features are described in *Oracle Database Net Services Administrator's Guide*.

- **Colocation tagging** is one such feature that is useful in certain transaction processing applications. If your application repeatedly makes connections to the same database service, colocation tagging permits all such connections to be directed to the same database instance, bypassing the load-balancing processing normally done on the database side of connections. For more information, see COLOCATION_TAG of Client Connections.
- **Shared Server Configuration** is another feature supported by Oracle Autonomous JSON Database for maintaining legacy applications designed without connection pooling. The shared server architecture enables the database server to allow many client processes to share very few server processes. This increases the number of users that can be supported by the application. Using the shared server architecture for such legacy applications enables them to scale up without making any changes to the application itself.

  By default, the shared server mode is disabled for Autonomous JSON Database. To enable it, submit an SR in My Oracle Support requesting support operations to assist you with shared server configuration for the required Exadata Infrastructure OCIDs.

  See also Oracle Database Net Services Administrator's Guide for more detailed information about shared server, including features such as session multiplexing.

  The client that wants to use the shared server configuration must configure `(SERVER=shared)` in the `CONNECT_DATA` section of the connect descriptor. For example:

```
sales=
(DESCRIPTION=
(ADDRESS=(PROTOCOL=tcp)(HOST=sales-server)(PORT=1521))
(CONNECT_DATA=
(SERVICE_NAME=sales.us.example.com)
(SERVER=shared))
```

> **Tip:**
>
> You can disable Shared Server for a specific Autonomous JSON Database created under a Shared Server enabled Autonomous Container Database by setting its `SHARED_SERVERS` value to `0`. To re-enable Shared Servers for that Autonomous JSON Database, run the `ALTER SYSTEM RESET SHARED_SERVERS` command.

Code for High Performance

Great applications begin with well written SQL. Oracle Autonomous Database provides numerous features that enable you to build high performance applications and validate your SQL and PL/SQL code. Some of these features are new in Release 19c; for example:

- Automatic Indexing
- Automatic resolution of SQL plan regressions
- Automatic quarantine of runaway SQL statements

- SQL Plan comparison function

Others such features have been available in Oracle Database and used by developers for years; for example:

- SQL Plan Management
- SQL Tuning sets
- SQL Tuning Advisor
- SQL Access Advisor

As you develop your application, you can quickly learn how these features are affecting the SQL code you write and so improve your code by using the Worksheet tool provided by both Oracle Database Actions (which is built into your JSON database) and Oracle SQL Developer (a free application you install on your development system). For more information about these tools, see Tools for Monitoring and Tuning Performance.

Tools for Monitoring and Tuning Performance

Several situations can give rise to application performance issues: changing workloads, resource limitations on application and database servers, or simply network bottlenecks.

Oracle provides a wide range of tools to help you monitor performance, diagnose performance issues, and tune your application or the database to resolve the issue.

A readily available feature-rich tool is the Performance Hub, which is available in each of the following:

- The Oracle Cloud Infrastructure console — see Monitor Autonomous Database with Performance Hub in *Using Oracle Autonomous Database Serverless*.
- Oracle Database Actions — see Connect with Built-In Oracle Database Actions
- Oracle Management Cloud

You can monitor the performance of SQL statements by choosing SQL Monitoring from the Performance Hub. See Monitor SQL Statements in *Using Oracle Autonomous Database Serverless*.

To turn monitoring on or off for a given SQL statement add the hint `MONITOR` or `NO MONITOR`, respectively, to the statement. See MONITOR and NO_MONITOR Hints in *Oracle Database SQL Tuning Guide*.

With SODA for Java you can use the same hints to monitor the SQL statements that underlie SODA operations. See the following topics in *Oracle Database SODA for Java Developer's Guide*:

- SODA for Java Read and Write Operations
- Inserting Documents into Collections with SODA for Java
- Saving Documents into Collections with SODA for Java

Two other commonly used tools are the Automatic Workload Repository (AWR) and the Automatic Database Diagnostic Monitor (ADDM). AWR stores performance related statics for an Oracle database, and ADDM is a diagnostic tool that analyzes the AWR data on a regular basis, locates root causes of any performance problems, provides recommendations for correcting the problems, and identifies non-problem areas of the system. Because AWR is a repository of historical performance data, ADDM can analyze performance issues after the event, often saving time and resources in reproducing a problem. For instructions on using these tools, as well as detailed information about database performance monitoring and tuning,

see *Oracle Database Performance Tuning Guide*. For a quick introduction to database performance monitoring and tuning, see *Oracle Database Get Started with Performance Tuning*.

For a complete list of the SQL tuning and performance management features of Oracle Autonomous Database, and instructions on how to use them, see *Oracle Database SQL Tuning Guide*.

A

Autonomous JSON Database for Experienced Oracle Database Users

This appendix provides information on using Autonomous JSON Database for experienced Oracle Database users with Autonomous Database Serverless.

For equivalent information about using Oracle Database features and options with Autonomous Database on dedicated Exadata infrastructure, see Oracle Database Features in Dedicated Autonomous Database Deployments.

Autonomous Database – Oracle Database Features

Describes Oracle Database features available with Autonomous Database.

Autonomous JSON Database includes features that:

- Automate index management tasks, such as creating, rebuilding, and dropping indexes based on changes in the application workload.

  See Manage Automatic Indexing on Autonomous Database in *Using Oracle Autonomous Database Serverless* for more information.

  > **Note:**
  >
  > There are restrictions for Automatic Indexing when you use JSON data with Autonomous JSON Database. See SODA Notes for more information.

- Gather real-time statistics automatically while a conventional DML workload is running. Because statistics can go stale between stats gathering jobs, online statistics gathering for conventional DML helps the optimizer generate more optimal plans. Online statistics aim to reduce the possibility of the optimizer being misled by stale statistics.

  See Real-Time Statistics for more information.

- Gather statistics automatically on a more frequent basis. High-Frequency Automatic Optimizer Statistics Collection complements the standard statistics collection job. By default, the collection occurs every 15 minutes, meaning that statistics have less time in which to be stale. High-Frequency Automatic Optimizer Statistics Collection is enabled by default.

  See Configuring High-Frequency Automatic Optimizer Statistics Collection for more information.

- Quarantine execution plans for SQL statements, for example, statements that are terminated by the Resource Manager for consuming excessive system resources in an Oracle Database. Automatic SQL Quarantine based on Resource Manager consumption limit violations is disabled by default but any manually quarantined SQL statement will be honored.

  See Quarantine for Execution Plans for SQL Statements Consuming Excessive System Resources for more information.

- Automatically assess the opportunity for SQL plan changes to improve the performance for known statements.

  See Managing the SPM Evolve Advisor Task for more information.

- Apache ORC format is supported in Autonomous Database for loading and querying data in object store.

  See Create Credentials and Load Data Pump Dump Files into an Existing Table and Query External Data with ORC, Parquet, or Avro Source Files in *Using Oracle Autonomous Database Serverless* for more information.

- Complex types are supported in Autonomous Database for ORC, Avro, and Parquet structured files.

  See DBMS_CLOUD Package ORC, Parquet and Avro Complex Types in *Using Oracle Autonomous Database Serverless* for more information.

SODA Notes

When you use SODA with Autonomous Database the following restrictions apply:

- Automatic indexing is not supported for SQL and PL/SQL code that uses the SQL/JSON function `json_exists`. See SQL/JSON Condition JSON_EXISTS for more information.
- Automatic indexing is not supported for SODA query-by-example (QBE).

About Autonomous Database for Experienced Oracle Database Users

Autonomous JSON Database configures and optimizes your database for you. You do not need to perform administration operations for configuring the database. SQL commands used for database administration such as `CREATE TABLESPACE` are not available. Similarly, other administrative interfaces and utilities such as `RMAN` are not available.

See Transaction Processing and JSON Database Workloads with Autonomous Database

Transaction Processing and JSON Database Workloads with Autonomous Database

Autonomous JSON Database configures and optimizes your database for you, based on your workload.

Characteristics of Autonomous Database with Transaction Processing or JSON Database workloads:

- The default data and temporary tablespaces for the database are configured automatically. Adding, removing, or modifying tablespaces is not allowed. Autonomous Database creates one tablespace or multiple tablespaces automatically depending on the storage size.
- The database character set is Unicode `AL32UTF8`. See Choose a Character Set for Autonomous Database for more information.
- Compression is not enabled by default but Autonomous JSON Database honors a compression clause if compression is specified on a table.

Accessing a JSON database:

- You do not have direct access to the database node. You can create and drop directories with `CREATE DIRECTORY` and `DROP DIRECTORY`, as described in Create and Manage Directories.

  You can use `DBMS_CLOUD` procedures such as `DBMS_CLOUD.DELETE_FILE`, `DBMS_CLOUD.GET_OBJECT`, and `DBMS_CLOUD.PUT_OBJECT` with files and objects. You do not have direct access to the local file system.

Parallel Execution with Transaction Processing or JSON Database workloads:

- Parallelism is determined by the database service you use. See Database Service Names for Autonomous Database for details about parallelism support for each database service.
- When you want to run DML operations in parallel and the database service you are using allows this, you can enable parallel DML in your session using the following SQL command:

  ```
  ALTER SESSION ENABLE PARALLEL DML;
  ```

  See *VLDB and Partitioning Guide* for more information on parallel DML operations.
- If you create an index manually and specify the `PARALLEL` clause, the `PARALLEL` attribute remains after the index is created. In this case SQL statements can run in parallel unbeknownst to the end user.

  To specify serial execution, change the `INDEX` parallel clause to `NOPARALLEL` or set the `PARALLEL` degree attribute to 1 to specify serial execution:

  ```
  ALTER INDEX index_name NOPARALLEL;
  ```

  or

  ```
  ALTER INDEX index_name PARALLEL 1;
  ```

Autonomous Database Views

Autonomous Database provides several views that are not available in Oracle Database 19c. This topic lists the Autonomous Database specific views.

Track Table and Partition Scan Access with Autonomous Database Views

Oracle Autonomous Database tracks the scan count for tables and partitions. Use the table access stats data dictionary and dynamic views to retrieve scan count information.

GV$TABLE_ACCESS_STATS and V$TABLE_ACCESS_STATS Views

The `GV$TABLE_ACCESS_STATS` and `V$TABLE_ACCESS_STATS` views list the scan count for tables and partitions. The scan data collection begins at instance startup time.

Column	Datatype	Description
`READ_COUNT`	`NUMBER`	Aggregated scan count since instance startup
`OBJECT_ID`	`NUMBER`	Object ID of the table or partition

Column	Datatype	Description
INST_ID	NUMBER	Instance number where table/partition was scanned This column (INST_ID) is only shown in GV$TABLE_ACCESS_STATS
CON_ID	NUMBER	Container ID of the database

ALL_TABLE_ACCESS_STATS and DBA_TABLE_ACCESS_STATS Views

The ALL_TABLE_ACCESS_STATS and DBA_TABLE_ACCESS_STATS views list the scan count for tables and partitions. The scan data collection begins at instance startup time.

> **Note:**
>
> The ALL_TABLE_ACCESS_STATS and DBA_TABLE_ACCESS_STATS views do not list scan count information for Oracle-maintained schemas.

Column	Datatype	Description
TABLE_OWNER	VARCAR2(128)	Owner of the table
TABLE_NAME	VARCAR2(128)	Name of the table
PARTITION_NAME	VARCAR2(128)	Name of the partition A NULL value specifies a non-partitioned table
INSTANCE_ID	NUMBER	Instance number where table or partition was scanned
READ_COUNT	NUMBER	Aggregated scan count since instance startup

USER_TABLE_ACCESS_STATS View

The USER_TABLE_ACCESS_STATS view lists the scan count for the user's tables and partitions. The scan data collection begins at instance startup time.

Column	Datatype	Description
TABLE_NAME	VARCAR2(128)	Name of the table
PARTITION_NAME	VARCAR2(128)	Name of the partition A NULL value specifies a non-partitioned table
INSTANCE_ID	NUMBER	Instance number where table/partition was scanned
READ_COUNT	NUMBER	Aggregated scan count since instance startup

Track Oracle Cloud Infrastructure Resources, Cost and Usage Reports with Autonomous Database Views

Oracle Autonomous Database tracks the Oracle Cloud Infrastructure resources, cost and usage reports. You can access these reports using the OCI views.

Prerequisite Steps to Use OCI Resource Views

Describes the prerequisite steps you must perform to use OCI resource views on Autonomous Database.

> **Note:**
>
> Only ADMIN user has access to the OCI resource views by default. To access these views as another user, the ADMIN must grant READ privileges.

To query an OCI resource view, do the following:

1. Create a dynamic group that includes your Autonomous Database instance and define the required policies to access a view.

   For example, the Autonomous Database instance is specified in the `resource.id` parameter with an OCID:

   ```
   resource.id = '<your_Autonomous_Database_instance_OCID>'
   ```

   Each view shows the details for the policy that you must define to query the view.

   See Perform Prerequisites to Use Resource Principal with Autonomous Database for details on creating a dynamic group and defining policies.

   For example, to access all of the views, define the following policy:

   ```
   Define tenancy usage-report as
   ocid1.tenancy.oc1..aaaaaaaaned4fkpkisbwjlr56u7cj63lf3wffbilvqknstgtvzub7vhq
   kggq
   Endorse dynamic-group <group-name> to read objects in tenancy usage-report
   Allow dynamic-group <group-name> to read buckets in tenancy
   Allow dynamic-group <group-name> to read autonomous-database in tenancy
   Allow dynamic-group <group-name> to read usage-budgets in tenancy
   ```

   > **Note:**
   >
   > Do not replace the OCID in this policy with another OCID. This `usage-report` OCID provides the Oracle Cloud Infrastructure usage data for your tenancy.

2. Verify that resource principal is enabled for the ADMIN user on the Autonomous Database instance.

   ```
   SELECT owner, credential_name FROM dba_credentials
      WHERE credential_name = 'OCI$RESOURCE_PRINCIPAL' AND owner = 'ADMIN';

   OWNER CREDENTIAL_NAME
   ----- --------------------
   ADMIN OCI$RESOURCE_PRINCIPAL
   ```

If the resource principal is not enabled, then enable the resource principal:

```
EXEC DBMS_CLOUD_ADMIN.ENABLE_RESOURCE_PRINCIPAL();
```

See Use Resource Principal to Access Oracle Cloud Infrastructure Resources for more information.

3. Run a query on an OCI resource view.

   For example:

```
SELECT NAME, APPROXIMATESIZE FROM OCI_OBJECTSTORAGE_BUCKETS;
SELECT * FROM OCI_USAGE_DATA;
```

OCI_AUTONOMOUS_DATABASES View

`OCI_AUTONOMOUS_DATABASES` describes all the Oracle Cloud Infrastructure Autonomous Databases in the Oracle Cloud Infrastructure tenancy obtained from the current Autonomous Database instance.

To query this view you need a dynamic group that includes your Autonomous Database instance and the following policy defined on that dynamic group:

```
Allow dynamic-group <group-name> to read autonomous-database in tenancy
```

This policy lets you list all Autonomous Databases in your tenancy. Optionally you can restrict it to list Autonomous Databases in a given compartment:

```
Allow dynamic-group <group-name> to read autonomous-database in compartment
<compartment-name>
```

Column	Datatype	Description
DISPLAYNAME	VARCHAR2	The user friendly name for the Autonomous Database
REGION	VARCHAR2	Region Name
COMPARTMENTID	VARCHAR2	The OCID of the compartment
ID	VARCHAR2	The OCID of the Autonomous Database
DBNAME	VARCHAR2	The database name
LIFECYCLESTATE	VARCHAR2	The current state of the Autonomous Database
TIMECREATED	VARCHAR2	The date and time the Autonomous Database was created
DATASTORAGESIZEINTBS	VARCHAR2	The quantity of data in the database in terabytes
LICENSEMODEL	VARCHAR2	The Oracle license model that applies to the Autonomous Database
SERVICECONSOLEURL	VARCHAR2	The URL of the Service Console for the Autonomous Database
APEXDETAILS	CLOB	Information about Oracle APEX Application Development
AREPRIMARYWHITELISTEDIPSUSED	VARCHAR2	Primary White Listed IPs
AUTONOMOUSCONTAINERDATABASEID	VARCHAR2	The Autonomous Container Database OCID
AUTONOMOUSMAINTENANCESCHEDULETYPE	VARCHAR2	Maintenance Schedule Type

Column	Datatype	Description
AVAILABLEUPGRADEVERSIONS	VARCHAR2	List of Oracle Database versions available for a database upgrade
BACKUPCONFIG	CLOB	Autonomous Database Backup Config
CONNECTIONSTRINGS	CLOB	Autonomous Database Connection Strings
CONNECTIONURLS	CLOB	Autonomous Database Connection URLs
CPUCORECOUNT	NUMBER	The number of OCPU cores to be made available to the database
CUSTOMERCONTACTS	CLOB	The Customer Contacts
DATASAFESTATUS	VARCHAR2	Status of the Data Safe registration for this Autonomous Database
DATASTORAGESIZEINGBS	NUMBER	The quantity of data in the database in gigabytes
DBVERSION	VARCHAR2	The Oracle Database version for the Autonomous Database
DATAGUARDREGIONTYPE	VARCHAR2	The Autonomous Data Guard region type of the Autonomous Database
DBWORKLOAD	VARCHAR2	The Autonomous Database workload type
DEFINEDTAGS	CLOB	Defined tags for the resource
FAILEDDATARECOVERYINSECONDS	NUMBER	Indicates the number of seconds of data loss for an Autonomous Data Guard failover
FREEFORMTAGS	CLOB	Free form tags for the resource
INFRASTRUCTURETYPE	VARCHAR2	The infrastructure type this resource belongs to
ISACCESSCONTROLENABLED	VARCHAR2	Indicates if the database level access control is enabled
ISAUTOSCALINGENABLED	VARCHAR2	Indicates if auto scaling is enabled for the Autonomous Database
ISDATAGUARDENABLED	VARCHAR2	Indicates whether the Autonomous Database has a local Autonomous Data Guard enabled
ISDEDICATED	VARCHAR2	True if the database uses dedicated Exadata infrastructure
ISFREETIER	VARCHAR2	Indicates if this is an Always Free resource
ISMTLSCONNECTIONREQUIRED	VARCHAR2	Indicates whether the Autonomous Database requires mTLS connections
ISPREVIEW	VARCHAR2	Indicates if the Autonomous Database version is a preview version
ISREFRESHABLECLONE	VARCHAR2	Indicates whether the Autonomous Database is a refreshable clone
KEYHISTORYENTRY	CLOB	Key History Entry
KEYSTOREID	VARCHAR2	The OCID of the key store
KEYSTOREWALLETNAME	VARCHAR2	The wallet name for Oracle Cloud Infrastructure Vault
KMSKEYID	VARCHAR2	The OCID of the key container that is used as the master encryption key
KMSKEYLIFECYCLEDETAILS	VARCHAR2	Customer managed key lifecycle details
LIFECYCLEDETAILS	VARCHAR2	Information about the current lifecycle state
NSGIDS	CLOB	A list of the OCIDs of the network security groups NSGs
OCPUCOUNT	NUMBER	The number of OCPU cores to be made available to the database
OPENMODE	VARCHAR2	The Autonomous Database open mode
OPERATIONSINSIGHTSSTATUS	VARCHAR2	Status of OCI Ops Insights for this Autonomous Database

Column	Datatype	Description
PEERDBIDS	VARCHAR2	The list of OCIDs of standby databases located in Autonomous Data Guard
PERMISSIONLEVEL	CLOB	The Autonomous Database permission level
PRIVATEENDPOINT	VARCHAR2	The private endpoint for the resource
PRIVATEENDPOINTIP	VARCHAR2	The private endpoint IP address for the resource
PRIVATEENDPOINTLABEL	VARCHAR2	The private endpoint label for the resource
REFRESHABLEMODE	VARCHAR2	The refresh mode of the clone
REFRESHABLESTATUS	VARCHAR2	The refresh status of the clone
ROLE	VARCHAR2	The Autonomous Data Guard role
SOURCEID	VARCHAR2	The OCID of the source Autonomous Database that was cloned
SQLWEBDEVELOPERURL	VARCHAR2	The Database Actions (SQL Developer Web) URL for the Autonomous Database
STANDBYDB	CLOB	Autonomous Database Standby Summary
STANDBYWHITELISTEDIPS	CLOB	The client IP access control list
SUBNETID	VARCHAR2	The OCID of the subnet the resource is associated with
SUPPORTEDREGIONSTOCLONETO	CLOB	The list of regions that support the creation of Autonomous Data Guard
SYSTEMTAGS	CLOB	System tags for this resource
TIMEDATAGUARDROLECHANGED	VARCHAR2	The date and time the Autonomous Data Guard role was switched
TIMEDELETIONOFFREEAUTONOMOUSDATABASE	NUMBER	Time deletion of Free Autonomous Database
TIMELOCALDATAGUARDENABLED	VARCHAR2	The date and time that Autonomous Data Guard was enabled for the Autonomous Database
TIMEMAINTENANCEBEGIN	VARCHAR2	The date and time when maintenance will begin
TIMEMAINTENANCEEND	VARCHAR2	The date and time when maintenance will end
TIMEOFLASTFAILOVER	VARCHAR2	The timestamp of the last failover operation
TIMEOFLASTREFRESH	VARCHAR2	The date and time of the last refresh
TIMEOFLASTREFRESHPOINT	VARCHAR2	The refresh point timestamp
TIMEOFLASTSWITCHOVER	VARCHAR2	The timestamp of the last switchover operation for the Autonomous Database
TIMEOFNEXTREFRESH	VARCHAR2	The date and time of next refresh
TIMERECLAMATIONOFFREEAUTONOMOUSDATABASE	VARCHAR2	The date and time the Always Free database
USEDDATASTORAGESIZEINTBS	NUMBER	The amount of storage that has been used in terabytes
VAULTID	VARCHAR2	The OCID of the Oracle Cloud Infrastructure Vault
WHITELISTEDIPS	CLOB	The client IP access control list

OCI_BUDGET_ALERT_RULES View

OCI_BUDGET_ALERT_RULES describes all the Oracle Cloud Infrastructure budget alert rules in the Oracle Cloud Infrastructure tenancy obtained from the current Autonomous Database instance.

Queries against this view return results only if you have budgets and budget alerts created in your tenancy.

See Budgets Overview for more information.

To query this view you need a dynamic group that includes your Autonomous Database instance and the following policy defined on that dynamic group:

```
Allow dynamic-group <group-name> to read usage-budgets in tenancy
```

This policy lets you list budget summary and budget alerts in your tenancy (if you created a budget and a budget alert). Optionally you can restrict the result returned by querying the view to a given compartment:

```
Allow dynamic-group <group-name> to read usage-budgets in compartment <compartment-name>
```

Column	Datatype	Description
BUDGETID	VARCHAR2	The OCID of the budget
REGION	VARCHAR2	Region name
COMPARTMENTID	VARCHAR2	The compartment ID in which the bucket is authorized
DEFINEDTAGS	CLOB	Defined tags for the resource
DESCRIPTION	VARCHAR2	The description of the alert rule
DISPLAYNAME	VARCHAR2	The name of the alert rule
FREEFORMTAGS	CLOB	Free-form tags for the resource
ID	VARCHAR2	The OCID of the alert rule
LIFECYCLESTATE	VARCHAR2	The current state of the alert rule
MESSAGE	VARCHAR2	The custom message that will be sent when the alert is triggered
RECIPIENTS	VARCHAR2	The audience that receives the alert when it triggers
THRESHOLD	NUMBER	The threshold for triggering the alert
THRESHOLDTYPE	VARCHAR2	The type of threshold
TIMECREATED	VARCHAR2	The time when the budget was created
TIMEUPDATED	VARCHAR2	The time when the budget was updated
TYPE	VARCHAR2	ACTUAL or FORECAST types of alert triggers
VERSION	NUMBER	The version of the alert rule

OCI_BUDGET_SUMMARY View

OCI_BUDGET_SUMMARY describes all the Oracle Cloud Infrastructure budget summaries in the Oracle Cloud Infrastructure tenancy obtained from the current Autonomous Database instance.

Queries against this view return results only if you have budgets created in your tenancy.

See Budgets Overview for more information.

To query this view you need a dynamic group that includes your Autonomous Database instance and the following policy defined on that dynamic group:

```
Allow dynamic-group <group-name> to read usage-budgets in tenancy
```

This policy lets you list budget summary and budget alerts in your tenancy (if you created a budget and a budget alert). Optionally you can restrict the result returned by querying the view to a given compartment:

```
Allow dynamic-group <group-name> to read usage-budgets in compartment <compartment-name>
```

Column	Datatype	Description
REGION	VARCHAR2	Region name
COMPARTMENTID	VARCHAR2	The OCID of the compartment
AMOUNT	NUMBER	The amount of the budget, expressed in the currency of a rate card
DEFINEDTAGS	CLOB	Defined tags for the resource
FREEFORMTAGS	CLOB	Free-form tags for the resource
DISPLAYNAME	VARCHAR2	The display name of the budget
LIFECYCLESTATE	VARCHAR2	The current state of the budget
ACTUALSPEND	NUMBER	The actual spend in currency for the current budget cycle
ALERTRULECOUNT	NUMBER	The total number of alert rules in the budget
BUDGETPROCESSINGPERIODSTARTOFFSET	NUMBER	The number of days offset from the first day of the month, at which the budget processing period starts
DESCRIPTION	VARCHAR2	The description of the budget
FORECASTEDSPEND	NUMBER	The forecasted spend in currency by the end of the current budget cycle
ID	VARCHAR2	The OCID of the budget
RESETPERIOD	VARCHAR2	The reset period for the budget
TARGETS	CLOB	The list of targets on which the budget is applied
TARGETCOMPARTMENTID	VARCHAR2	Target compartment OCID
TARGETTYPE	VARCHAR2	The type of target on which the budget is applied
TIMECREATED	VARCHAR2	The time the budget was created
TIMESPENDCOMPUTED	VARCHAR2	The time the budget spend was last computed
TIMEUPDATED	VARCHAR2	The time the budget was updated
VERSION	VARCHAR2	The version of the budget

OCI_COST_DATA View

`OCI_COST_DATA` describes all the Oracle Cloud Infrastructure cost data for the Oracle Cloud Infrastructure tenancy obtained from the current Autonomous Database instance.

To query this view you need a dynamic group that includes your Autonomous Database instance and the following policy defined on that dynamic group:

```
Define tenancy usage-report as
ocid1.tenancy.oc1..aaaaaaaaned4fkpkisbwjlr56u7cj63lf3wffbilvqknstgtvzub7vhqkggq
Endorse dynamic-group <group-name> to read objects in tenancy usage-report
```

> **Note:**
>
> Do not replace the OCID in this policy with another OCID. This `usage-report` OCID provides the Oracle Cloud Infrastructure usage data for your tenancy.

Column	Datatype	Description
REFERENCE_NUMBER	VARCHAR2	Reference Number/Line identifier used for debugging and corrections

Column	Datatype	Description
TENANT_ID	VARCHAR2	The identifier (OCID) for the Oracle Cloud Infrastructure tenant
INTERVAL_USAGE_START	TIMESTAMP	The start time of the usage interval for the resource in UTC
INTERVAL_USAGE_END	TIMESTAMP	The end time of the usage interval for the resource in UTC
SERVICE_NAME	VARCHAR2	The service that the resource is in
COMPARTMENT_ID	VARCHAR2	The ID of the compartment that contains the resource
COMPARTMENT_NAME	VARCHAR2	The name of the compartment that contains the resource
REGION	VARCHAR2	The region that contains the resource
AVAILABILITY_DOMAIN	VARCHAR2	The availability domain that contains the resource
RESOURCE_ID	VARCHAR2	The identifier for the resource
BILLED_QUANTITY	VARCHAR2	The quantity of the resource that has been billed over the usage interval
BILLED_QUANTITY_OVERAGE	VARCHAR2	The usage quantity for which you were billed
SUBSCRIPTION_ID	VARCHAR2	A unique identifier associated with your commitment or subscription
PRODUCT_SKU	VARCHAR2	The Part Number for the resource in the line
PRODUCT_DESCRIPTION	VARCHAR2	The product description for the resource in the line
UNIT_PRICE	VARCHAR2	The cost billed to you for each unit of the resource used
UNIT_PRICE_OVERAGE	VARCHAR2	The cost per unit of usage for overage usage of a resource
MY_COST	VARCHAR2	The cost charged for this line of usage
MY_COST_OVERAGE	VARCHAR2	The cost billed for overage usage of a resource
CURRENCY_CODE	VARCHAR2	The currency code for your tenancy
BILLING_UNIT_READABLE	VARCHAR2	The unit measure associated with the usage/ billedQuantity in the line
SKU_UNIT_DESCRIPTION	VARCHAR2	The unit used for measuring billed quantity
OVERAGE_FLAG	CHAR	Flag used for overage usage
IS_CORRECTION	VARCHAR2	Used if the current line is a correction
BACK_REFERENCE_NUMBER	VARCHAR2	Data amendments and corrections reference
CREATED_BY	VARCHAR2	The user who created the service
CREATED_ON	TIMESTAMP	The time when the service was created
FREE_TIER_RETAINED	VARCHAR2	Is the service retained on free tier

OCI_OBJECTSTORAGE_BUCKETS View

OCI_OBJECTSTORAGE_BUCKETS describes all the Oracle Cloud Infrastructure object storage buckets in the Oracle Cloud Infrastructure tenancy obtained from the current Autonomous Database instance.

To query this view you need a dynamic group that includes your Autonomous Database instance and the following policy defined on that dynamic group:

```
Allow dynamic-group <group-name> to read buckets in tenancy
```

This policy lets you list object storage buckets in your tenancy. Optionally you can restrict the result returned by querying this view to a given compartment:

```
Allow dynamic-group <group-name> to read buckets in compartment <compartment-name>
```

Column	Datatype	Description
REGION	VARCHAR2	Region name
COMPARTMENTID	VARCHAR2	The compartment ID in which the bucket is authorized
NAMESPACE	VARCHAR2	The Object Storage namespace in which the bucket resides
APPROXIMATECOUNT	NUMBER	The approximate number of objects in the bucket
APPROXIMATESIZE	NUMBER	The approximate total size in bytes of all objects in the bucket
AUTOTIERING	VARCHAR2	The auto tiering status on the bucket
CREATEDBY	VARCHAR2	The OCID of the user who created the bucket
DEFINEDTAGS	CLOB	Defined tags for the resource
FREEFORMTAGS	CLOB	Free-form tags for the resource
ETAG	VARCHAR2	The entity tag (ETag) for the bucket
ID	VARCHAR2	The OCID of the bucket
ISREADONLY	VARCHAR2	Whether or not this bucket is read only
KMSKEYID	VARCHAR2	The OCID of a master encryption key
METADATA	VARCHAR2	Arbitrary string keys and values for user-defined metadata
NAME	VARCHAR2	The name of the bucket
OBJECTEVENTSENABLED	VARCHAR2	Whether or not events are emitted for object state changes in this bucket
OBJECTLIFECYCLEPOLICYETAG	VARCHAR2	The entity tag (ETag) for the live object lifecycle policy on the bucket
PUBLICACCESSTYPE	VARCHAR2	The type of public access enabled on this bucket
REPLICATIONENABLED	VARCHAR2	Whether or not this bucket is a replication source
STORAGETIER	VARCHAR2	The storage tier type assigned to the bucket
TIMECREATED	VARCHAR2	The date and time the bucket was created
VERSIONING	VARCHAR2	The versioning status on the bucket

OCI_USAGE_DATA View

OCI_USAGE_DATA describes all the Oracle Cloud Infrastructure usage data for the Oracle Cloud Infrastructure tenancy obtained from the current Autonomous Database instance.

To query this view you need a dynamic group that includes your Autonomous Database instance and the following policy defined on that dynamic group:

```
Define tenancy usage-report as
ocid1.tenancy.oc1..aaaaaaaaned4fkpkisbwjlr56u7cj63lf3wffbilvqknstgtvzub7vhqkggq
Endorse dynamic-group <group-name> to read objects in tenancy usage-report
```

> **Note:**
>
> Do not replace the OCID in this policy with another OCID. This `usage-report` OCID provides the Oracle Cloud Infrastructure cost and usage data for your tenancy.

Column	Datatype	Description
REFERENCE_NUMBER	VARCHAR2	Reference Number/Line identifier used for debugging and corrections
TENANT_ID	VARCHAR2	The identifier (OCID) for the Oracle Cloud Infrastructure tenant
INTERVAL_USAGE_START	TIMESTAMP	The start time of the usage interval for the resource in UTC
INTERVAL_USAGE_END	TIMESTAMP	The end time of the usage interval for the resource in UTC
SERVICE_NAME	VARCHAR2	The service that the resource is in
RESOURCE_NAME	VARCHAR2	The resource name used by the metering system
COMPARTMENT_ID	VARCHAR2	The ID of the compartment that contains the resource
COMPARTMENT_NAME	VARCHAR2	The name of the compartment that contains the resource
REGION	VARCHAR2	The region that contains the resource
AVAILABILITY_DOMAIN	VARCHAR2	The availability domain that contains the resource
RESOURCE_ID	VARCHAR2	The identifier for the resource
CONSUMED_QUANTITY	VARCHAR2	The quantity of the resource that has been consumed over the usage interval
BILLED_QUANTITY	VARCHAR2	The quantity of the resource that has been billed over the usage interval
CONSUMED_QUANTITY_UNITS	VARCHAR2	The unit for the consumed quantity and billed quantity
CONSUMED_QUANTITY_MEASURE	VARCHAR2	The measure for the consumed quantity and billed quantity
IS_CORRECTION	VARCHAR2	Used if the current line is a correction
BACK_REFERENCE_NUMBER	VARCHAR2	Data amendments and corrections reference
CREATED_BY	VARCHAR2	The user who created the service
CREATED_ON	TIMESTAMP	The time when the service was created
FREE_TIER_RETAINED	VARCHAR2	Is the service retained on free tier

Always Free Autonomous Database – Oracle Database 21c

When you provision Always Free Autonomous Database you can select either Oracle Database 19c or Oracle Database 23ai.

Always Free Autonomous Database Oracle Database 21c Features

When you provision Always Free Autonomous Database you can select either Oracle Database 19c or Oracle Database 23ai.

> **Note:**
>
> With the availability of Always Free Autonomous Database Oracle Database 23ai, Oracle Database 21c is no longer available as a provisioning or cloning option. Existing Always Free Autonomous Databases running with Oracle Database 21c continue as Always Free Autonomous Database.

Always Free Autonomous Database running with Oracle Database 21c offers many new innovative autonomous and developer-oriented functionality, including but not limited to the following:

Performance Features

- **Automatic Zone Maps**

  Automatic zone maps are created and maintained for any user table without any customer intervention. Zone maps allow the pruning of block ranges and partitions based on the predicates in the queries. Automatic zone maps are maintained for direct loads, and are maintained and refreshed for any other DML operation incrementally and periodically in the background.

  The feature is enabled as follows:

  ```
  exec dbms_auto_zonemap.configure('AUTO_ZONEMAP_MODE','ON');
  ```

  The feature is disabled as follows:

  ```
  exec dbms_auto_zonemap.configure('AUTO_ZONEMAP_MODE','OFF');
  ```

  See Summary of DBMS_AUTO_ZONEMAP Subprograms for more information.

- **Object Activity Tracking System**

  Object Activity Tracking System (OATS) tracks the usage of various types of database objects. Usage includes operations such as access, data manipulation, or refresh.

  No manual intervention is required to enable OATS, and zero or minimal configuration is required. See PL/SQL procedure DBMS_ACTIVITY.CONFIGURE and database dictionary views DBA_ACTIVITY_CONFIG for details.

Application Development: Advanced Analytical SQL Capabilities

- **SQL Macros**

  SQL Macros, the capability to factor out common SQL constructs supports scalar expressions, increasing developer productivity, simplify collaborative code development, and improve code quality. See SQL Macros for more information.

- **Enhanced Analytic Functions**

  Window functions support the full ANSI Standard, including the support of EXCLUDE options and the WINDOW clause. Supporting the full ANSI standard enables easier migration of applications that were developed with other standard-compliant database systems. See Windowing Functions for more information.

- **New Analytical and Statistical Aggregate Functions**

Several new analytical and statistical aggregate functions are available in SQL in Oracle Database 21c. With these additional SQL aggregation functions, you can write more efficient code and benefit from faster in-database processing.

- `CHECKSUM` computes the checksum of the input values or expression.

  Supports the keywords `ALL` and `DISTINCT`.

- `KURTOSIS` functions `KURTOSIS_POP` and `KURTOSIS_SAMP` measure the tailedness of a data set where a higher value means more of the variance within the data set is the result of infrequent extreme deviations as opposed to frequent modestly sized deviations. Note that a normal distribution has a kurtosis of zero.

  Supports the keywords `ALL`, `DISTINCT`, and `UNIQUE`.

- `SKEWNESS` functions `SKEWNESS_POP` and `SKEWNESS_SAMP` are measures of asymmetry in data. A positive skewness is means the data skews to the right of the center point. A negative skewness means the data skews to the left.

  Supports the keywords `ALL`, `DISTINCT`, and `UNIQUE`.

- `ANY_VALUE`, a function to simplify and optimize the performance of `GROUP BY` statements, returns a random value in a group and is optimized to return the first value in the group. It ensures that there are no comparisons for any incoming row and eliminates the necessity to specify every column as part of the `GROUP BY` clause.

See Oracle Database 21c SQL Language Reference Guide for more information.

- **Bitwise Aggregate Functions**

  With the new bitwise type processing functions `BIT_AND_AGG`, `BIT_OR_AGG`, and`BIT_XOR_AGG`, native bitwise type processing is provided by Oracle Database 21c. These functions enable a type of processing inside the database for new types of application processing, improving the overall performance, avoiding unnecessary data movement, and natively taking advantage of core database functionality such as parallel processing. See Oracle Database 21c SQL Language Reference Guide. for more information.

JavaScript Execution using DBMS_MLE

The `DBMS_MLE` package allows users to execute JavaScript code inside the Oracle Database and exchange data seamlessly between PL/SQL and JavaScript. The JavaScript code itself can execute PL/SQL and SQL through built-in JavaScript modules. JavaScript data types are automatically mapped to Oracle Database data types and vice versa.

With the `DBMS_MLE` package, developers can write their data processing logic in JavaScript. JavaScript is a widely-used and popular programming language that can now also be used for writing programs that need to execute close to the data.

See DBMS_MLE for more information.

Blockchain Table

Blockchain tables are append-only tables in which only insert operations are allowed. Deleting rows is either prohibited or restricted based on time. Rows in a blockchain table are made tamper-resistant by special sequencing and chaining algorithms. Users can verify that rows have not been tampered. A hash value that is part of the row metadata is used to chain and validate rows.

Blockchain tables enable you to implement a centralized ledger model where all participants in the blockchain network have access to the same tamper-resistant ledger.

A centralized ledger model reduces administrative overheads of setting up a decentralized ledger network, leads to a relatively lower latency compared to decentralized ledgers, enhances developer productivity, reduces the time to market, and leads to significant savings for the organization. Database users can continue to use the same tools and practices that they would use for other database application development.

See Managing Blockchain Tables for more information.

JSON Document Store Enhancements

- **Enhancements to Data Guide**

  Enhances development flexibility and allows for materialized views, which may improve query performance with a trade-off against DML performance.

  – `JSON_DATAGUIDE` now gathers statistic information if you specify `DBMS_JSON.GATHER_STATS` in the third argument. They are computed dynamically (up-to-date) at the time of the function call.

  – `DBMS_JSON.CREATE_VIEW` now gives you the option to create a materialized view instead of a standard view. It also gives you the option to specify a particular path so the view can be created on a subset of the data. Both `CREATE_VIEW` and `ADD_VIRTUAL_COLUMN` are enhanced to allow automatic resolution of column naming conflicts, to provide a prefix to be applied to column names, and to specify the case-sensitivity of column names.

  See JSON Data Guide for more information.

- **Multivalue Index for JSON DataType**

  A new create index syntax `CREATE MULTIVALUE INDEX` allows you to create a functional index on arrays of strings or numbers within a JSON datatype column. Each unique value within the array will become a searchable index entry. This avoids the need for full JSON scans to find values within arrays in JSON columns, when searched using the `JSON_EXISTS` or `JSON_VALUE` operators. It provides similar benefits to conventional functional indexes when searching JSON, but conventional functional indexes are limited to a single indexed value per row.

  See Creating Multivalue Function-Based Indexes for JSON_EXISTS and Using a Multivalue Function-Based Index for more information.

- **New JSON Data Type**

  JSON is a new SQL and PL/SQL data type for JSON data. Using this type provides a substantial increase in query and update performance. JSON data type uses binary format OSON that is optimized for SQL/JSON query and DML processing. Using the binary format can yield database performance improvements for processing JSON data.

  You can use JSON data type and its instances in most places where a SQL data type is allowed, including:

  – As the column type for table or view DDL

  – With SQL/JSON functions and conditions, and with PL/SQL procedures and functions

  – In Oracle dot-notation query syntax

  – For creation of functional and search indexes

  Oracle Call Interface and Java Database Connectivity (JDBC) clients now provide APIs that can work directly with binary JSON datatype OSON format, significantly saving network costs and server CPU cycles. Going forward, Oracle recommends using JSON datatype to store and process JSON data.

The Oracle Autonomous JSON Database uses OSON format to store and process JSON data.

See Creating a Table With a JSON Column for more information.

- **New Oracle SQL Function JSON_TRANSFORM**

  You can use SQL function `JSON_TRANSFORM` to update parts of a JSON document. You specify which parts to modify, the modifications, and any new values. `JSON_TRANSFORM` is optimized by doing partial updates at OSON format level to achieve better JSON datatype update performance.

  `JSON_TRANSFORM` makes it easier for an application to modify a JSON document, without having to parse and rebuild it. In most cases, it also avoids a round-trip between the server and client for the whole document.

  See Oracle SQL Function JSON_TRANSFORM for more information.

- **SQL/JSON Syntax Improvements**

  You can now express more complex SQL/JSON queries and express some queries more succinctly:

  – New SQL function `JSON_SCALAR` accepts a scalar instance of a SQL data type and returns a scalar JSON value as an instance of JSON data type.

  – New JSON path-language item methods support `JSON_SCALAR: float()`, `double()`, `binary()`, `ymInterval()`, and `dsInterval()`.

  – The JSON path-language and dot-notation syntax support the aggregate item methods: `avg()`, `count()`, `minNumber()`, `maxNumber()`, `minString()`, `maxString()`, `sum()`.

    See Simple Dot-Notation Access to JSON Data and SQL/JSON Path Expression Item Methods for more information.

SODA Enhancements: New JSON Data Type

The default collection storage changes to the JSON data type. See Creating a Document Collection with SODA for PL/SQL for more information.

PL/SQL Enhancements

- PL/SQL is enhanced to help you program iteration controls using new iterators in loops and in qualified expressions.

  The new iterator constructs are clear, simple, understandable, and efficient.

  See PL/SQL Extended Iterators for more information.

Gradual Database Password Rollover for Applications

An application can change its database passwords without an administrator having to schedule downtime.

To accomplish this, a database administrator can associate a profile having a non-zero limit for the `PASSWORD_ROLLOVER_TIME` password profile parameter, with an application schema. This allows the database password of the application user to be altered while allowing the older password to remain valid for the time specified by the `PASSWORD_ROLLOVER_TIME` limit. During the rollover period of time, the application instance can use either the old password or the new password to connect to the database server. When the rollover time expires, only the new password is allowed.

In addition to the clause `PASSWORD_ROLLOVER_TIME` in the `CREATE PROFILE` and `ALTER PROFILE` statements, the `ALTER USER` statement has a clause, `EXPIRE PASSWORD ROLLOVER PERIOD`. The `ACCOUNT_STATUS` column of the `DBA_USERS` and `USER_USERS` data dictionary views have several statuses indicating values to indicate rollover status.

See Managing Gradual Database Password Rollover for Applications for more information.

Always Free Autonomous Database Oracle Database 21c Notes

If you are using Always Free Autonomous Database with Oracle Database 21c, the following Oracle Database 21c functionality is not supported:

- Automatic Materialized Views
- Autonomous Database only supports Cloud Links when your database version is Oracle Database 19c. Cloud Links are not supported with database version Oracle Database 21c. See Use Cloud Links for Read Only Data Access on Autonomous Database for more information.

Autonomous Database RMAN Recovery Catalog

You can use Oracle Autonomous Database as a Recovery Manager (RMAN) recovery catalog. A recovery catalog is a database schema that RMAN uses to store metadata about one or more Oracle databases.

Use Autonomous Database as an RMAN Recovery Catalog

Recovery Manager (RMAN) recovery catalog is preinstalled in Autonomous Database in schema `RMAN$CATALOG`. The preinstalled catalog version is based on the latest version of Oracle Database and is compatible with all supported Oracle database versions.

The recovery catalog contains metadata about RMAN operations for each registered target database. When RMAN is connected to a recovery catalog, RMAN obtains its metadata exclusively from the catalog.

> **Note:**
>
> Autonomous Database is not supported as an RMAN target database. An RMAN target database is an Oracle Database to which RMAN is connected with the `TARGET` keyword. A target database is a database on which RMAN is performing backup and recovery operations. See Backup and Restore Autonomous Database Instances for information on Autonomous Database backup and recovery operations.

Access to RMAN Recovery Catalog

Access to the recovery catalog is provided through predefined user `RMAN$CATALOG` with the appropriate access to the recovery catalog only. The `RMAN$CATALOG` user is locked by default.

You can either proxy to the predefined user `RMAN$CATALOG` through the ADMIN user or explicitly unlock the preinstalled schema:

- ADMIN user proxy into `RMAN$CATALOG` using ADMIN user's password:

```
connect admin[rman$catalog]/password@connect_string
```

- ADMIN user can set a password for `RMAN$CATALOG`. Then the `RMAN$CATALOG` user can directly connect:

```
connect admin/password@connect_string
alter user rman$catalog identified by password account unlock;
connect rman$catalog/password@connect_string
```

Use the RMAN Recovery Catalog

You can use the RMAN recovery catalog by connecting RMAN to the preinstalled recovery catalog. Registering a target database in the recovery catalog maintains the database's records in the recovery catalog. For example, to register a target database:

```
RMAN> connect catalog rman$catalog/password@connect_string;

connected to recovery catalog database
recovery catalog schema version 21.01.00.00. is newer than RMAN version

RMAN> register database;
database registered in recovery catalog
starting full resync of recovery catalog
```

To use your Autonomous Database as a recovery catalog, it is recommended to connect with the `LOW` service.

See Registering a Database in the Recovery Catalog for more details about using the RMAN recovery catalog.

Notes for Users Migrating from Other Oracle Databases

Describes information that is useful when you are migrating from other Oracle Databases to Oracle Autonomous Database.

Initialization Parameters

Autonomous Database configures database initialization parameters automatically when you provision a database. You do not need to set any initialization parameters to start using your service. But, you can modify some parameters if you need to.

Modifiable Initialization Parameters

The following table shows the initialization parameters that are only modifiable with `ALTER SESSION`.

Only Modifiable with ALTER SESSION	More Information
CONSTRAINTS	
CONTAINER	
CURRENT_SCHEMA	
CURSOR_INVALIDATION	CURSOR_INVALIDATION
DEFAULT_COLLATION	
DEFAULT_CREDENTIAL	
EDITION	

Only Modifiable with ALTER SESSION	More Information
ISOLATION_LEVEL	
JSON_BEHAVIOR	This parameter is only applicable with Oracle Database 23ai. See JSON_BEHAVIOR for more information.
JSON_EXPRESSION_CHECK	JSON_EXPRESSION_CHECK
OPTIMIZER_SESSION_TYPE	OPTIMIZER_SESSION_TYPE
OPTIMIZER_USE_INVISIBLE_INDEXES	OPTIMIZER_USE_INVISIBLE_INDEXES
READ_ONLY	
SQL_TRACE	See Perform SQL Tracing on Autonomous Database for details
SQL_TRANSLATION_PROFILE	
SQL_TRANSPILER	This parameter is only applicable with Oracle Database 23ai. See SQL_TRANSPILER for more information.
STATISTICS_LEVEL	STATISTICS_LEVEL
TIME_ZONE	For more information on TIME_ZONE, see *Oracle Database SQL Language Reference*.
XML_PARAMS	This parameter is only applicable with Oracle Database 23ai. See XML_PARAMS for more information.

The following table shows the initialization parameters that are only modifiable with ALTER SYSTEM.

Only Modifiable with ALTER SYSTEM	More Information
BLANK_TRIMMING	BLANK_TRIMMING
FIXED_DATE	FIXED_DATE
JOB_QUEUE_PROCESSES	JOB_QUEUE_PROCESSES
LOCKDOWN_ERRORS	See LOCKDOWN_ERRORS for details
MAX_IDLE_BLOCKER_TIME	MAX_IDLE_BLOCKER_TIME With a value higher than 5, the parameter acts as if it was set to 5
MAX_IDLE_TIME	MAX_IDLE_TIME
SESSION_EXIT_ON_PACKAGE_STATE_ERROR	SESSION_EXIT_ON_PACKAGE_STATE_ERROR

The following table shows the initialization parameters that are modifiable with either ALTER SESSION or ALTER SYSTEM.

Modifiable with ALTER SESSION or ALTER SYSTEM	More Information
APPROX_FOR_AGGREGATION	APPROX_FOR_AGGREGATION
APPROX_FOR_COUNT_DISTINCT	APPROX_FOR_COUNT_DISTINCT
APPROX_FOR_PERCENTILE	APPROX_FOR_PERCENTILE
CLIENT_PREFETCH_ROWS	See CLIENT_PREFETCH_ROWS
CONTAINER_DATA	CONTAINER_DATA
CURSOR_SHARING	CURSOR_SHARING
DDL_LOCK_TIMEOUT	DDL_LOCK_TIMEOUT

Modifiable with ALTER SESSION or ALTER SYSTEM	More Information
GROUP_BY_POSITION	
GROUP_BY_POSITION_ENABLED	This parameter is only applicable with Oracle Database 23ai. See GROUP_BY_POSITION_ENABLED for more information
HEAT_MAP	HEAT_MAP
IGNORE_SESSION_SET_PARAM_ERRORS	IGNORE_SESSION_SET_PARAM_ERRORS
LDAP_DIRECTORY_ACCESS	LDAP_DIRECTORY_ACCESS
LOAD_WITHOUT_COMPILE	
MAX_STRING_SIZE	See Data Types for details
NLS_CALENDAR	NLS_CALENDAR
NLS_COMP	NLS_COMP
NLS_CURRENCY	NLS_CURRENCY
NLS_DATE_FORMAT	NLS_DATE_FORMAT
NLS_DATE_LANGUAGE	NLS_DATE_LANGUAGE
NLS_DUAL_CURRENCY	NLS_DUAL_CURRENCY
NLS_ISO_CURRENCY	NLS_ISO_CURRENCY
NLS_LANGUAGE	NLS_LANGUAGE
NLS_LENGTH_SEMANTICS	NLS_LENGTH_SEMANTICS
NLS_NCHAR_CONV_EXCP	NLS_NCHAR_CONV_EXCP
NLS_NUMERIC_CHARACTERS	NLS_NUMERIC_CHARACTERS
NLS_SORT	NLS_SORT
NLS_TERRITORY	NLS_TERRITORY
NLS_TIME_FORMAT	
NLS_TIME_TZ_FORMAT	
NLS_TIMESTAMP_FORMAT	NLS_TIMESTAMP_FORMAT
NLS_TIMESTAMP_TZ_FORMAT	NLS_TIMESTAMP_TZ_FORMAT
OPTIMIZER_CAPTURE_SQL_QUARANTINE	OPTIMIZER_CAPTURE_SQL_QUARANTINE
OPTIMIZER_IGNORE_HINTS	For more information on OPTIMIZER_IGNORE_HINTS, see Manage Optimizer Statistics on Autonomous Database.
OPTIMIZER_IGNORE_PARALLEL_HINTS	For more information on OPTIMIZER_IGNORE_PARALLEL_HINTS, see Manage Optimizer Statistics on Autonomous Database.
OPTIMIZER_MODE	OPTIMIZER_MODE
OPTIMIZER_REAL_TIME_STATISTICS	OPTIMIZER_REAL_TIME_STATISTICS
OPTIMIZER_USE_SQL_QUARANTINE	OPTIMIZER_USE_SQL_QUARANTINE
PLSCOPE_SETTINGS	PLSCOPE_SETTINGS
PLSQL_CCFLAGS	PLSQL_CCFLAGS
PLSQL_DEBUG	PLSQL_DEBUG
PLSQL_IMPLICIT_CONVERSION_BOOL	This parameter is only applicable with Oracle Database 23ai. See PLSQL_IMPLICIT_CONVERSION_BOOL
PLSQL_OPTIMIZE_LEVEL	PLSQL_OPTIMIZE_LEVEL
PLSQL_WARNINGS	PLSQL_WARNINGS

Modifiable with ALTER SESSION or ALTER SYSTEM	More Information
QUERY_REWRITE_INTEGRITY	QUERY_REWRITE_INTEGRITY
RECYCLEBIN	RECYCLEBIN
REMOTE_DEPENDENCIES_MODE	REMOTE_DEPENDENCIES_MODE
RESULT_CACHE_INTEGRITY	See RESULT_CACHE_INTEGRITY
RESULT_CACHE_MODE	See RESULT_CACHE_MODE
SKIP_UNUSABLE_INEDEXES	SKIP_UNUSABLE_INDEXES
SYSDATE_AT_DBTIMEZONE	See SYSDATE_AT_DBTIMEZONE Select a Time Zone for SYSDATE on Autonomous Database
XML_CLIENT_SIDE_DECODING	See XML_CLIENT_SIDE_DECODING

For more information on initialization parameters see *Oracle Database Reference*.

SESSION_EXIT_ON_PACKAGE_STATE_ERROR

SESSION_EXIT_ON_PACKAGE_STATE_ERROR enables or disables special handling for stateful PL/SQL packages running in a session.

Property	Description
Parameter type	Boolean
Default Value	FALSE
Modifiable	ALTER SYSTEM
Range of values	TRUE | FALSE

SESSION_EXIT_ON_PACKAGE_STATE_ERROR specifies the handling for a stateful PL/SQL package running in a session. When such a package undergoes modification, such as during planned maintenance for Oracle-supplied objects, the sessions that have an active instantiation of the package receive the following error when they attempt to run the package:

```
ORA-4068 existing state of package has been discarded
```

However, the application code that receives the ORA-4068 error may not be equipped to handle this error with its retry logic.

Setting SESSION_EXIT_ON_PACKAGE_STATE_ERROR to TRUE provides different handling for this case. When SESSION_EXIT_ON_PACKAGE_STATE_ERROR is TRUE, instead of just raising the ORA-4068 error when the package state is discarded, the session immediately exits. This can be advantageous because many applications are able to handle session termination by automatically and transparently re-establishing the connection.

SYSDATE_AT_DBTIMEZONE Select a Time Zone for SYSDATE on Autonomous Database

SYSDATE_AT_DBTIMEZONE enables special handling in a session for the date and time value returned in calls to SYSDATE and SYSTIMESTAMP.

Depending on the value of SYSDATE_AT_DBTIMEZONE, you see either the date and time based on the default Autonomous Database time zone, Coordinated Universal Time (UTC), or based on the time zone that you set in your database.

Property	Description
Parameter type	Boolean
Default Value	FALSE
Modifiable	ALTER SESSION, ALTER SYSTEM
Range of values	TRUE | FALSE

Default Autonomous Database Time Zone

The default Autonomous Database time zone is Coordinated Universal Time (UTC) and by default calls to SYSDATE and SYSTIMESTAMP return the date and time in UTC.

In order to change database time zone, you can run the following statement. This example sets the database time zone to UTC-5.

```
ALTER DATABASE SET TIME_ZONE='-05:00';
```

> **Note:**
>
> You must restart the Autonomous Database instance for the change to take effect.

After you set the database time zone, by default SYSDATE and SYSTIMESTAMP continue to return date and time in UTC (SYSDATE_AT_DBTIMEZONE is FALSE by default). If you set SYSDATE_AT_DBTIMEZONE to TRUE in a session, SYSDATE and SYSTIMESTAMP return the database time zone.

See Setting the Database Time Zone for more information on using the SET TIME_ZONE clause with ALTER DATABASE.

Using SYSDATE_AT_DBTIMEZONE in a Session

When SYSDATE_AT_DBTIMEZONE is FALSE in a session, calls to SYSDATE and SYSTIMESTAMP return values based on the default Autonomous Database time zone, Coordinated Universal Time (UTC).

When SYSDATE_AT_DBTIMEZONE is TRUE in a session, calls to SYSDATE or SYSTIMESTAMP return the date and time based on the database time zone.

> **Note:**
>
> Setting SYSDATE_AT_DBTIMEZONE to TRUE only affects the use of SYSDATE and SYSTIMESTAMP as operators in application SQL (for example, in queries, DML, and CTAS operations). When using this parameter, it is recommended that your client/session timezone matches your database timezone.

Example

The following example returns dates and times for two different time zones, based on the `SYSDATE_AT_DBTIMEZONE` parameter value:

```
SQL> SELECT DBTIMEZONE FROM DUAL;

DBTIMEZONE
_____________
-05:00

SQL> ALTER SESSION SET SYSDATE_AT_DBTIMEZONE=FALSE;

Session altered.

SQL> SELECT SYSTIMESTAMP FROM DUAL;

SYSTIMESTAMP
_____________________________________
27-JAN-22 06.59.45.708082000 PM GMT

SQL> ALTER SESSION SET SYSDATE_AT_DBTIMEZONE=TRUE;

Session altered.

SQL> SELECT SYSTIMESTAMP FROM DUAL;

SYSTIMESTAMP
_____________________________________
27-JAN-22 02.14.47.578946000 PM -05:00
```

> **Note:**
>
> When a `SYSDATE` or `SYSTIMESTAMP` query is executed in SQL Worksheet of Database Actions, the time and date value that is returned is in UTC (when `SYSDATE_AT_DBTIMEZONE` parameter is set to `TRUE` or `FALSE`). To obtain the database time zone when working in Database Actions, use `TO_CHAR()` as follows:
>
> ```
> SQL> SELECT TO_CHAR(SYSTIMESTAMP,'YYYY-MM-DD"T"HH24:MI:SS TZH":"TZM')
> FROM DUAL;
>
> TO_CHAR(SYSTIMESTAMP,'YYYY-MM-DD"T"HH24:MI:SSTZH":"TZM')
> ____________________________________________________________
> 2022-01-27T14:15:00 -05:00
> ```

SQL Commands

Autonomous Database allows most of the SQL commands available in Oracle Database. To ensure the security and the performance of Autonomous Database, some SQL commands are restricted.

This section provides a list of SQL command limitations that are required to protect security and for the performance integrity of Autonomous Databases. Most of the standard SQL and PL/SQL syntax and constructs available with Oracle Database work in Autonomous Databases.

> **Note:**
>
> If you try to use a restricted SQL command the system reports:
>
> ```
> ORA-01031: insufficient privileges
> ```
>
> This error indicates that you are not allowed to run the SQL command in Autonomous JSON Database.

The following SQL statements are not available in Autonomous Database:

- `ADMINISTER KEY MANAGEMENT`: By default Autonomous Database uses Oracle-managed encryption keys. Using Oracle-managed keys, Autonomous Database creates and manages the encryption keys that protect your data and Oracle handles rotation of the TDE master key.

  If you want customer-managed keys, a master encryption key in the Oracle Cloud Infrastructure Vault is used to generate the TDE master key on Autonomous Database. See Managing Encryption Keys on Autonomous Database for more information.
- `CREATE TABLESPACE`, `ALTER TABLESPACE`, and `DROP TABLESPACE`: Autonomous Database automatically configures default data and temporary tablespaces for the database. Adding, removing, or modifying tablespaces is not allowed. Autonomous Database creates one tablespace or multiple tablespaces automatically depending on the storage size.
- `CREATE DATABASE LINK`

  Use `DBMS_CLOUD_ADMIN.CREATE_DATABASE_LINK` to create database links in Autonomous Database. See Use Database Links with Autonomous Database for more information.
- `CREATE LIBRARY`
- `DROP DATABASE LINK`

  Use `DBMS_CLOUD_ADMIN.DROP_DATABASE_LINK` to drop database links in Autonomous Database. See Use Database Links with Autonomous Database for more information.

Roles and Views Restrictions for Data Dictionary

Granting `SELECT ANY DICTIONARY` does not provide access to the `SYS`/`SYSTEM` schemas. You can grant `SELECT_CATALOG_ROLE` to allow `SELECT` privileges on all data dictionary views, if needed.

SQL Statements with Restrictions in Autonomous Database

The following DDL statements are available in Autonomous Database with some restrictions:

SQL Command	Restrictions
ALTER PLUGGABLE DATABASE and ALTER DATABASE	Only the following clauses are allowed: DATAFILE AUTOEXTEND ON DATAFILE AUTOEXTEND OFF DEFAULT EDITION SET TIME_ZONE SET CMU_WALLET
ALTER PROFILE	Using ALTER PROFILE, there are restrictions for a user defined PASSWORD_VERIFY_FUNCTION. See Manage Password Complexity on Autonomous Database for more information. Using ALTER PROFILE, the optional CONTAINER clause is ignored if specified. See Create Users on Autonomous Database for information on the password parameter values defined in the default profile.
ALTER SESSION	Only the following clauses are allowed: ADVISE COMMIT, ADVISE ROLLBACK, ADVISE NOTHING CLOSE DATABASE LINK ENABLE COMMIT IN PROCEDURE, DISABLE COMMIT IN PROCEDURE ENABLE PARALLEL <QUERY|DDL|DML>, DISABLE PARALLEL <QUERY|DDL|DML>, FORCE PARALLEL <QUERY|DDL|DML> ENABLE RESUMABLE, DISABLE RESUMABLE SET CONSTRAINTS SET CURRENT_SCHEMA SET DEFAULT_COLLATION SET EDITION SET ISOLATION_LEVEL SET OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES SET ROW ARCHIVAL VISIBILITY SET STATISTICS_LEVEL SET TIME_ZONE
ALTER SYSTEM	ALTER SYSTEM is not allowed except ALTER SYSTEM SET and ALTER SYSTEM KILL SESSION SET can only be used to set parameters listed in Initialization Parameters.
ALTER USER	The following clause is ignored: DEFAULT TABLESPACE The IDENTIFIED with the EXTERNALLY clause is not supported. The IDENTIFIED BY VALUES clause is not allowed.
ALTER TABLE	For restrictions, see ALTER TABLE Restrictions.

SQL Command	Restrictions
CREATE PROFILE	PASSWORD_VERIFY_FUNCTION See Manage Password Complexity on Autonomous Database for more information. Using ALTER PROFILE, the optional CONTAINER clause is ignored if specified. See Create Users on Autonomous Database for information on the password parameter values defined in the default profile.
CREATE TABLE	For restrictions, see CREATE TABLE Restrictions.
CREATE OR REPLACE TRIGGER	The AFTER STARTUP and BEFORE SHUTDOWN events are not supported with CREATE OR REPLACE TRIGGER.
CREATE USER	The following clause is ignored: • DEFAULT TABLESPACE IDENTIFIED with the EXTERNALLY clause is not supported. The IDENTIFIED BY VALUES clause is not allowed.

CREATE TABLE Restrictions

XMLType tables using XML schema-based storage are not allowed. See Oracle XML DB for more information.

The clauses not in this list are allowed.

Clause	Comment
cluster	Ignored
ilm_clause	Ignored
inmemory_table_clause	Ignored
LOB_storage_clause	The LOB_compression_clause is recognized. Other LOB_storage_clause parameters are ignored. See LOB_compression_clause for more information.
logging_clause	Ignored
organization external	Ignored
organization index	Creates a regular table with a primary key. Using the organization index clause does not create an index-organized table. You should test and verify the performance of the generated table for your application.
physical_properties	Ignored

> **Note:**
>
> For more information on CREATE TABLE, see *Database SQL Language Reference*.

ALTER TABLE Restrictions

The clauses not in this list are allowed.

Clause	Comment
allocate_extent_clause	Ignored
alter_iot_clauses	Ignored
deallocate_unused_clause	Ignored
ilm_clause	Ignored
inmemory_table_clause	Ignored
logging_clause	Ignored
modify_LOB_storage_clause	Ignored
physical_attributes_clause	Ignored
shrink_clause	Ignored

> **Note:**
>
> For more information on ALTER TABLE, see *Database SQL Language Reference*.

Data Types

Autonomous Database allows most of the data types available in Oracle Database. To ensure the security and the performance of Autonomous Database, some data types are restricted.

The following data types are not supported or have limited support in Autonomous Database:

- Large Object (LOB) data types: only SecureFiles LOB storage is supported. BasicFiles LOBs are automatically converted to SecureFiles LOBs.
- Media types are not supported (Oracle Multimedia is desupported)

Checking and Setting MAX_STRING_SIZE

By default Autonomous JSON Database uses extended data types and the value of `MAX_STRING_SIZE` is set to the value `EXTENDED`. With this setting you can specify a maximum size of 32767 bytes for the `VARCHAR2`, `NVARCHAR2`, and `RAW` data types. The default, `EXTENDED`, is the recommended setting and allows Autonomous Database to take full advantage of database capabilities.

Use `DBMS_MAX_STRING_SIZE` subprograms to check usage of extended data types and to change the database to revert to the older style `STANDARD`, supporting a maximum size of 4000 bytes for `VARCHAR2`, `NVARCHAR2`, and `RAW` data types.

> **Note:**
>
> Using `DBMS_MAX_STRING_SIZE.MODIFY_MAX_STRING_SIZE` is a one-way change that cannot be reverted. After a database is switched back to the `STANDARD` style of supporting a maximum length of 4000 bytes for the `VARCHAR2`, `NVARCHAR2`, and `RAW` data types, you cannot re-enable `EXTENDED` data types.
>
> The ADMIN user is granted `EXECUTE` privilege `WITH GRANT OPTION` clause on `DBMS_MAX_STRING_SIZE`. Oracle recommends that you do not `GRANT EXECUTE` on this package to other users.

1. Check whether your environment can be reverted to the old style, `STANDARD` behavior:

   ```
   SELECT * FROM
   TABLE(DBMS_MAX_STRING_SIZE.CHECK_MAX_STRING_SIZE('STANDARD'));
   ```

   See CHECK_MAX_STRING_SIZE Function in *Using Oracle Autonomous Database Serverless* for more information.
2. Check and correct all reported violations from Step 1, if applicable.
3. After fixing any reported violations found in Step 1, if you want to revert to a maximum length of 4000 bytes for `VARCHAR2`, `NVARCHAR2`, and `RAW` data types, use `DBMS_MAX_STRING_SIZE.MODIFY_MAX_STRING_SIZE` as follows:

   ```
   EXEC DBMS_MAX_STRING_SIZE.MODIFY_MAX_STRING_SIZE('STANDARD');
   ```

   See MODIFY_MAX_STRING_SIZE Procedure in *Using Oracle Autonomous Database Serverless* for more information.

See MAX_STRING_SIZE for more information.

See Extended Data Types for details on extended data types.

For a list of Oracle data types see *Oracle Database SQL Language Reference*.

PL/SQL Package Notes for Autonomous Database

Notes for Oracle Database PL/SQL packages in Autonomous Database.

Unavailable PL/SQL Packages

- `DBMS_DEBUG_JDWP`
- `DBMS_DEBUG_JDWP_CUSTOM`

DBMS_LDAP PL/SQL Package Notes

Provides notes for the `DBMS_LDAP` package:

- Specifying an IP address in the host name is not allowed.
- The only allowed port is 636.

- The SSLWRL and SSLWALLETPASSWD arguments to the OPEN_SSL procedure are ignored. The default value for the SSLWRL property is set to the wallet that is used by UTL_HTTP and DBMS_CLOUD for making outbound web requests on Autonomous Database.
- The DBMS_LDAP.SIMPLE_BIND_S and DBMS_LDAP.BIND_S subprograms perform authentication to the directory server.

  The DBMS_LDAP.SIMPLE_BIND_S and DBMS_LDAP.BIND_S subprograms are modified to accept credential objects as an argument.

  Following are the usage notes and examples of these modified subprograms:

  – The modified SIMPLE_BIND_S and BIND_S subprograms enable you to pass credential objects to set directory server authentication. Credential objects are schema objects, hence they can be accessed only by privileged users and enable you to configure schema-level privileges to access control the credentials. Passing scheduler credentials is an appropriate and secure way to store and manage username/password/keys for authentication.

  – The modified SIMPLE_BIND_S and BIND_S subprograms are a secure and convenient alternative to previously existed SIMPLE_BIND_S and BIND_S subprogram.

    See FUNCTION simple_bind_s and FUNCTION bind_s for more information.

  – The CREDENTIAL argument of the SIMPLE_BIND_S and BIND_S functions is used to perform credential based authentication to the directory server.

  – For example:

    * Create a credential object:

```
BEGIN DBMS_CLOUD.CREATE_CREDENTIAL (
    credential_name => 'LDAP_CRED',
    username        => 'web_app_user',
    password        => 'password' );
END;
```

      This creates a credential object which creates a stored username/password pair.

      See CREATE_CREDENTIAL Procedure for more information.

      See Specifying Scheduler Job Credentials for more information.

    * Invoke DBMS_LDAP.SIMPLE_BIND_S:

```
DECLARE
      l_mail_conn DBMS_LDAP.INIT;
    BEGIN
      l_ldap_conn   := DBMS_LDAP.INIT('ldap.example.com', 636);
      l_auth_result := DBMS_LDAP.SIMPLE_BIND_S(l_ldap_conn,
'LDAP_CRED');
      ...
END;
```

      The code in this example first invokes the DBMS_LDAP.INIT function which initializes a session with an LDAP server and establishes a connection with the LDAP server ldap.example.com at port number 636. The value l_ldap_conn in the SIMPLE_BIND_S function is the LDAP session handle and LDAP_CRED is the credentials name.

* The `function bind_s` performs complex authentication to the directory server. For example:

```
DECLARE
      l_mail_conn DBMS_LDAP.INIT;
    BEGIN
      l_ldap_conn   := DBMS_LDAP.INIT('ldap.example.com', 636);
      l_auth_result := DBMS_LDAP.BIND_S(l_ldap_conn, 'LDAP_CRED',
METH => DBMS_LDAP.AUTH_SIMPLE);
      ...
END;
```

The code in this example first invokes the `DBMS_LDAP.INIT` function which initializes a session with an LDAP server and establishes a connection with the LDAP server `ldap.example.com` at port number `636`. The value `l_ldap_conn` in the `BIND_S` function is the LDAP session handle and `LDAP_CRED` is the credentials name. `METH` is the authentication method. The only valid value is `DBMS_LDAP_UTL.AUTH_SIMPLE`.

– The `EXECUTE` privileges on `DBMS_CLOUD` or `DWROLE` is required to create scheduler credentials.

– The passed credentials must be present in the current user schema and be in the enabled state.

– A public or private synonym that points to a credential in a different user schema can be supplied as a value for the `CREDENTIAL` parameter provided you have the `EXECUTE` privilege on the base credential object pointed to by the synonym. See Overview of Synonyms for more information.

- SSL/TLS is enforced for all communication happening between LDAP server and Autonomous Database.
- When your Autonomous Database instance is configured with a private endpoint, set the `ROUTE_OUTBOUND_CONNECTIONS` database parameter to '`PRIVATE_ENDPOINT`' to specify that all outgoing LDAP connections are subject to the Autonomous Database instance private endpoint VCN's egress rules. See Enhanced Security for Outbound Connections with Private Endpoints for more information.
- To use `DBMS_LDAP` for a connection on a private endpoint, use `DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE` and specify the `private_target` parameter with value `TRUE`.

> **Note:**
>
> If you set `ROUTE_OUTBOUND_CONNECTIONS` to `PRIVATE_ENDPOINT`, setting the `private_target` parameter to `TRUE` is not required in this API. See Enhanced Security for Outbound Connections with Private Endpoints for more information.

- **`DBMS_LDAP` Error**

Error Message	Potential Cause
`ORA-31400: Missing or invalid scheduler credential`	Passed credentials are NULL or invalid.

See DBMS_LDAP in *PL/SQL Packages and Types Reference* for more information.

DBMS_NETWORK_ACL_ADMIN PL/SQL Package Notes

Provides notes for the `DBMS_NETWORK_ACL_ADMIN` package:

- Granting ACL privileges on IP addresses is not allowed.
- The `HTTP_PROXY` ACL is allowed on private endpoints.

See DBMS_NETWORK_ACL_ADMIN in *PL/SQL Packages and Types Reference* for more information.

UTL_HTTP PL/SQL Package Notes

Provides notes for the `UTL_HTTP` package:

- Connections through IP addresses are not allowed.
- Only `HTTPS` is allowed when the Autonomous Database instance is on a public endpoint. When the Autonomous Database instance is on a private endpoint, both `HTTPS` and `HTTP_PROXY` connections are allowed (`HTTP` connections are disallowed for both public endpoints and private endpoints).
- The `UTL_HTTP.set_proxy` API is allowed when the Autonomous Database instance is on a private endpoint.
- When the Autonomous Database instance is on a private endpoint and you use `HTTP_PROXY` or the `UTL_HTTP.SET_PROXY` API:
  - `DBMS_CLOUD` requests do not honor the proxy server you set with `UTL_HTTP.SET_PROXY`. This includes `DBMS_CLOUD.SEND_REQUEST` and all object storage access for `DBMS_CLOUD` external tables that you define with `DBMS_CLOUD.CREATE_EXTERNAL_TABLE`, `DBMS_CLOUD.CREATE_EXTERNAL_PART_TABLE`, or `DBMS_CLOUD.CREATE_HYBRID_PART_TABLE`.
  - `APEX_WEB_SERVICE` requests do not honor the proxy server you set with `UTL_HTTP.SET_PROXY`.
- All web services must be secured. The only allowed port is 443 when the Autonomous Database instance is on a public endpoint. When the Autonomous Database instance is on a private endpoint this restriction does not apply.

  Your Autonomous Database instance is preconfigured with an Oracle Wallet that contains more than 90 of the most commonly trusted root and intermediate SSL certificates. The Oracle Wallet is centrally managed. You can configure `UTL_HTTP` to use a wallet for a site that is protected using self-signed SSL certificates. See Use a Customer-Managed Wallet for External Calls with UTL_HTTP for more information.
- The `SET_AUTHENTICATION_FROM_WALLET` procedure is disallowed.
- The `WALLET_PATH` and `WALLET_PASSWORD` arguments for the `CREATE_REQUEST_CONTEXT`, `REQUEST`, and `REQUEST_PIECES` procedures are ignored.
- The `CREDENTIAL` argument of the `SET_CREDENTIAL` procedure is used to pass the credential object as an input to the procedure. See Specifying Scheduler Job Credentials and CREATE_CREDENTIAL Procedure for more information.
- The `EXECUTE` privileges on `DBMS_CLOUD` or `DWROLE` is required to create credential objects.
- The passed credentials must be present in the current user schema and be in the enabled state.

- A public or private synonym that points to a credential in a different user schema can be supplied as a value for the `CREDENTIAL` parameter provided you have the `EXECUTE` privilege on the base credential object pointed to by the synonym. See Overview of Synonyms for more information.
- Oracle Wallet configuration cannot be altered. All arguments for `SET_WALLET` procedure are ignored.
- When your Autonomous Database instance is configured with a private endpoint, set the `ROUTE_OUTBOUND_CONNECTIONS` database parameter to '`PRIVATE_ENDPOINT`' to specify that all outgoing `UTL_HTTP` connections are subject to the Autonomous Database instance private endpoint VCN's egress rules. See Enhanced Security for Outbound Connections with Private Endpoints for more information.
- **`UTL_HTTP` Errors**

  The following table shows error messages and possible causes for these error messages when using `UTL_HTTP`:

Error Message	Potential Cause
`ORA-12545: Connect failed because target host or object does not exist`	Target host or object does not exist or it is private.
`ORA-24247: network access denied by access control list (ACL)`	Access control list (ACL) for the specified host could not be found.
`ORA-29024: Certificate validation failure`	Certificate of the host does not exist or is not among the supported certificates.
`ORA-29261: Bad argument`	Passed credentials are invalid or disabled or the user does not have sufficient privileges on the credential.

See UTL_HTTP in *PL/SQL Packages and Types Reference* for more information.

UTL_INADDR PL/SQL Package Notes

Provides notes for the `UTL_INADDR` package:

- The `UTL_INADDR` package is available for use on an Autonomous Database instance with a private endpoint (PE).
- The **`GET_HOST_ADDRESS`** function is available.
- The **`GET_HOST_NAME`** function is not available.

See UTL_INADDR in *PL/SQL Packages and Types Reference* for more information.

UTL_SMTP PL/SQL Package Notes

Provides notes for the `UTL_SMTP` package:

- The only supported email provider is Oracle Cloud Infrastructure Email Delivery service. See Overview of the Email Delivery Service for more information.
- Mail with an IP address in the host name is not allowed.
- The only allowed ports are 25 and 587.
- The `CREDENTIAL` argument of the `SET_CREDENTIAL` function is used to pass the scheduler credentials object as an input to the function. See Specifying Scheduler Job Credentials and CREATE_CREDENTIAL Procedure for more information.

- The `EXECUTE` privileges on `DBMS_CLOUD` or `DWROLE` is required to create credential objects.
- The `CREDENTIAL` argument of the `SET_CREDENTIAL` procedure is used to pass the credential objects object as an input to the procedure. See Specifying Scheduler Job Credentials for more information.
- The passed credentials must be present in the current user schema and be in the enabled state.
- A public or private synonym that points to a credential in a different user schema can be supplied as a value for the `CREDENTIAL` parameter provided you have the `EXECUTE` privilege on the base credential object pointed to by the synonym. See Overview of Synonyms for more information.
- When your Autonomous Database instance is configured with a private endpoint, set the `ROUTE_OUTBOUND_CONNECTIONS` database parameter to '`PRIVATE_ENDPOINT`' to specify that all outgoing `UTL_SMTP` connections are subject to the Autonomous Database instance private endpoint VCN's egress rules. See Enhanced Security for Outbound Connections with Private Endpoints for more information.
- **`UTL_SMTP` Error**

Error Message	Potential Cause
`ORA-29261: Bad argument`	Passed credentials are invalid or disabled or the user does not have sufficient privileges on the credential.

See UTL_SMTP in *PL/SQL Packages and Types Reference* for more information.

UTL_TCP PL/SQL Package Notes

Provides notes for the `UTL_TCP` package:

- The IP address is not allowed in the host name.
- The only allowed ports are: 443 (HTTP) 25 and 587 (SMTP).
- For port 443, only HTTPS URLs are allowed.
- The `WALLET_PATH` and `WALLET_PASSWORD` arguments for the `OPEN_CONNECTION` procedure are ignored. The default value for the `WALLET_PATH` and `WALLET_PASSWORD` property are set to the wallet that is used by `UTL_HTTP` and `DBMS_CLOUD` for making outbound web requests on Autonomous Database.
- SSL/TLS is enforced for all communication happening over TCP/IP connections.
- When your Autonomous Database instance is configured with a private endpoint, set the `ROUTE_OUTBOUND_CONNECTIONS` database parameter to '`PRIVATE_ENDPOINT`' to specify that all outgoing `UTL_TCP` connections are subject to the Autonomous Database instance private endpoint VCN's egress rules. See Enhanced Security for Outbound Connections with Private Endpoints for more information.

See UTL_TCP in *PL/SQL Packages and Types Reference* for more information.

Oracle XML DB

Describes Autonomous Database support for Oracle XML DB features. To ensure the security and the performance of your Autonomous Database, some Oracle XML DB features are restricted.

The following is supported, in addition to the features listed:

- Full support for XMLQuery, XMLTable, and other SQL/XML standard functions
- Indexing schema including functional indexes using SQL/XML expressions, Structured XMLIndex and XQuery Full Text Index

> **Note:**
>
> If you migrate tables containing `XMLType` columns to Autonomous JSON Database using Oracle Data Pump, you need to convert to Non-Schema Binary XML prior to using Oracle Data Pump Export (expdp).

Area	XML DB Feature	Supported in Autonomous Database	More Information
Repository	XML DB Protocol	No	Repository Access Using Protocols
Repository	XML DB Resources	No	Oracle XML DB Repository Resources
Repository	XML DB ACLs	No	Repository Access Control
Storage	XML Schema Registration	No [1]	XML Schema Registration with Oracle XML DB
Storage	CLOB	No	Deprecated
Storage	Object Relational	No	XML Schema and Object-Relational XMLType
Storage	Binary XML	Yes (Non schema-based only)	XMLType Storage Models
Index	Structured XML Index	Yes	XMLIndex Structured Component
Index	XQuery Full Text Index	Yes	Indexing XML Data for Full-Text Queries
Index	Unstructured XMLIndex	No	XMLIndex Unstructured Component
Packages	XML DOM package	Yes	PL/SQL DOM API for XMLType (DBMS_XMLDOM)
Packages	XML Parser Package	Yes	PL/SQL Parser API for XMLType (DBMS_XMLPARSER)
Packages	XSL Processor (DBMS_XSLPROCESSOR)	Yes	PL/SQL XSLT Processor for XMLType (DBMS_XSLPROCESSOR)

[1] While you cannot do XML schema registration, runtime validation of XML documents is available through DBMS_XMLSCHEMA_UTIL.

For details on Oracle XML DB, see Oracle XML DB Developer's Guide.

Oracle Text

Describes Autonomous Database support for Oracle Text features. To ensure the security and the performance of your Autonomous Database, some Oracle Text features are restricted.

Oracle Text Feature	Supported in Autonomous Database	More Information
All logging, and APIs which perform logging such as `ctx_report.query_log_summary`	Not Supported	QUERY_LOG_SUMMARY

Oracle Text Feature	Supported in Autonomous Database	More Information
File Datastore	Not Supported (see replacement `DIRECTORY_DATASTORE`)	Datastore Types
URL Datastore	Not Supported (see replacement `NETWORK_DATASTORE`)	Datastore Types
`CREATE INDEX` with `BIG_IO` option	Supported if you grant the privilege to create a trigger to the user (`GRANT CREATE TRIGGER`).	Improved Response Time Using the BIG_IO Option of CONTEXT Index
`OPTIMIZE_INDEX` in rebuild mode	Supported if you grant the privilege to create a trigger to the user (`GRANT CREATE TRIGGER`).	OPTIMIZE_INDEX

For details on Oracle Text, see Oracle Text Application Developer's Guide.

Oracle Flashback

Oracle Flashback Technology is a group of Oracle Database features that let you view past states of database objects or to return database objects to a previous state without using point-in-time media recovery.

To restore and recover your database to a point in time, see Restore and Recover your Autonomous JSON Database Database.

Oracle Flashback Feature	Supported in Autonomous Database
`DBMS_FLASHBACK`	Yes except the procedure: `DBMS_FLASHBACK.TRANSACTION_BACKOUT`
Flashback Data Archive	Yes
Flashback Drop	Yes
Flashback Query	Yes
Flashback Table	Yes
Flashback Transaction	No
Flashback Transaction Query	Yes
Flashback Version Query	Yes

Oracle Database Real Application Security

Oracle Database Real Application Security is a database authorization model that: supports declarative security policies, enables end-to-end security for multitier applications, provides an integrated solution to secure database and application resources, and advances the security

architecture of Oracle Database to meet existing and emerging demands of applications developed for the Internet.

See Introducing Oracle Database Real Application Security more information.

Real Application Security works the same on Autonomous Database as on an on-premises Oracle Database except you need to perform the following ADMIN tasks before using Real Application Security on Autonomous Database:

- To create Real Application Security users/roles, you need the `PROVISION` system privilege. As the ADMIN user run the following command to grant this privilege to a database user:

  ```
  SQL> EXEC
  XS_ADMIN_CLOUD_UTIL.GRANT_SYSTEM_PRIVILEGE('PROVISION','DB_USER');
  ```

  In this example, *DB_USER* is a database user.

  Running this command on Autonomous Database replaces the following on-premise database command (note the `_CLOUD_` is not in the following package name):

  ```
  SQL> EXEC SYS.XS_ADMIN_UTIL.GRANT_SYSTEM_PRIVILEGE('PROVISION', 'DB_USER',
  XS_ADMIN_UTIL.PTYPE_XS);
  ```

  See General Procedures for Creating Application User Accounts for more information.

- To create Real Application Security data controls, you need the `ADMIN_ANY_SEC_POLICY` privilege. As the ADMIN user run the following command to grant this privilege:

  ```
  EXEC
  XS_ADMIN_CLOUD_UTIL.GRANT_SYSTEM_PRIVILEGE('ADMIN_ANY_SEC_POLICY','DB_USER'
  );
  ```

  In this example, *DB_USER* is a database user.

  Running this command on Autonomous Database replaces the following on-premise database command (note the `_CLOUD_` is not in the following package name):

  ```
  SQL> EXEC
  SYS.XS_ADMIN_UTIL.GRANT_SYSTEM_PRIVILEGE('ADMIN_ANY_SEC_POLICY','DB_USER');
  ```

  See Creating Roles and Application Users for more information.

Oracle LogMiner

Describes restrictions for Oracle LogMiner on Autonomous Database.

Archived Log Retention Maximum is 48 Hours

Autonomous Database archived log files are kept for up to 48 hours. LogMiner can only access up to 48 hours of archived log files.

If you attempt to mine log files older than 48 hours, LogMiner reports `ORA-1285: "error reading file"`.

Choose a Character Set for Autonomous Database

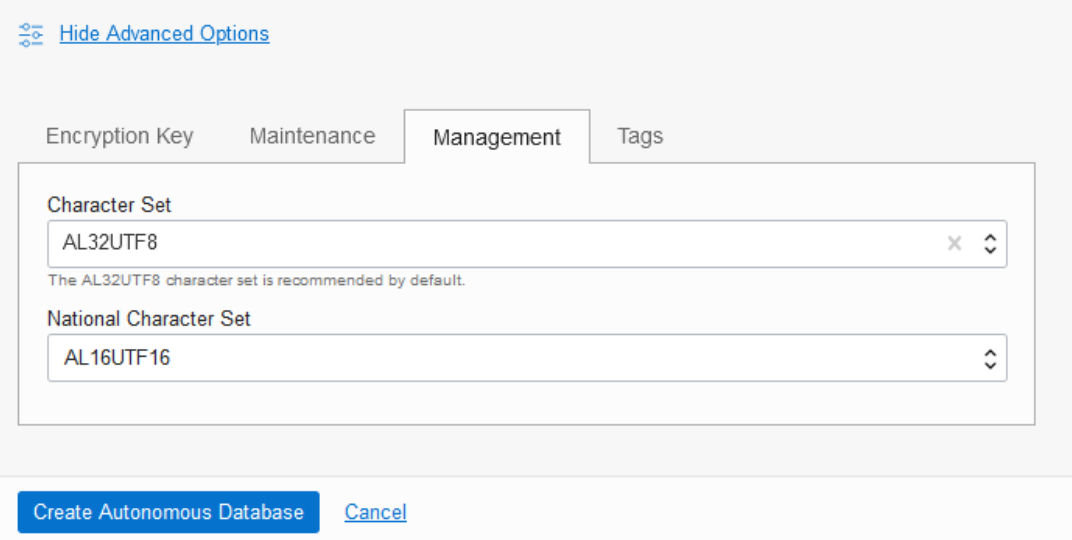
The Autonomous Database default database character set is Unicode AL32UTF8 and the default national character set is AL16UTF16. When you provision a database, depending on the workload type, you can select a database character set and a national character set.

> **Note:**
>
> Oracle recommends using the default Unicode database character set (AL32UTF8) for its universality and compatibility with contemporary and future technologies and language requirements.

If you are using an on-premises database with a non-Unicode character set, migrating to the default Unicode character set can be a convoluted process requiring complex data analysis. Thus, Autonomous Database lets you choose a character set when you provision an Autonomous Database instance.

1. Create an Autonomous Database following the provisioning steps.

   See Provision an Autonomous JSON Database Instance for more information.
2. On the **Create Autonomous Database** page expand **Show Advanced Options**.
3. In the advanced options area, select the **Management** tab.



   - In the **Character Set** field, use the selector to choose a character set.
   - In the **National Character Set** field, use the selector to choose a national character set.

   To make selecting a value easier, when you type in the text area this filters the list. For example, if you type JA you see only the options containing JA, including: JA16EUC, JA16EUCTILDE, JA16SJIS, JA16JA16SJISTILDE, and JA16VMS.
4. Click **Create Autonomous Database** to provision the Autonomous Database.

See the following for more information:

- Choosing an Oracle Database Character Set

- Character Set Migration
- Support Note 788156.1

Notes for Character Set Selection

Provides notes and limitations for selecting a character set and a national character set on Autonomous Database.

- When you provision an Autonomous Database instance you can only select a character set for **Data Warehouse**, **Transaction Processing**, or **APEX** workload types.

  The **JSON Database** workload type only supports the default character set and you cannot select a different character set.
- Always Free Autonomous Database does not support character set selection.
- You cannot select a different character set when you clone an instance. A cloned Autonomous Database instance has the same character set as the source database.
- You cannot change the character set of an existing Autonomous Database instance.
- APEX developer and administration pages are supported in English only when the database character set is different from the default (AL32UTF8).

  Translation of user applications into other languages or setting an application's primary language to a value other than English is not supported. If you use APEX in a language other than English, you may experience issues such as illegible, garbage text. You can, however, use your APEX applications to process non-English user data, as long as the languages of the data are supported by the selected database character set.

  If you need full globalization support in APEX, migrate your applications and data to AL32UTF8, the universal Unicode character set, which is the default and recommended character set for Autonomous Database.

  When you choose a character set other than the default Unicode character set (AL32UTF8) on Autonomous Database, the APEX language selector only shows languages that are supported in that database character set. For example, if you choose the database character set WE8ISO8859P1 (ISO 8859-1 West European), the language selector does not show Japanese or Korean.
- While you are provisioning an instance the **Character Set** selector on the **Management** tab lists the supported character set names. If you want to see a list of supported database character sets before you provision a database, refer to the following:
  - See Table A-4 in Recommended Database Character Sets
  - See Table A-6 in Other Character Sets

Database Features Unavailable in Autonomous Database

Lists the Oracle Database features that are not available in Autonomous Database. Additionally, database features designed for administration are not available.

List of Unavailable Oracle Features

- Oracle Real Application Security Administration Console (RASADM)
- Oracle Industry Data Models
- Oracle Database Lifecycle Management Pack

- Oracle Data Masking and Subsetting Pack

  > **Note:**
  >
  > Oracle Data Safe, available with Autonomous Database provides Data Masking. See Use Oracle Data Safe with Autonomous Database for more information.

- Oracle Cloud Management Pack for Oracle Database
- Oracle Multimedia: Not available in Autonomous Database and deprecated in Oracle Database 18c.
- Oracle Sharding
- Oracle XStream
- Custom locale objects, including: language, territory, character set, and collation (linguistic sort) are not supported. Custom locale definitions created using Oracle Locale Builder cannot be deployed on Autonomous Database. See Customizing Locale Data for more information.

B

SODA Collection Metadata on Autonomous Database

Describes default and customized collection metadata on Autonomous Database.

SODA Default Collection Metadata on Autonomous Database

Describes the default collection metadata on Autonomous Database, that is the metadata for a collection that is added when custom metadata is not supplied.

Each SODA implementation provides a way to create a default collection when you supply a collection name. For example, in SODA for Java you use the `createCollection` method and supply just a collection name parameter:

```
db.admin().createCollection("myCol");
```

This creates a collection with default collection metadata. When you create a default collection on your JSON database, the collection metadata includes the following information (regardless of which SODA implementation you use to create the default collection):

```
{
   "keyColumn" :
   {
      "name" : "ID",
      "sqlType" : "VARCHAR2",
      "maxLength" : 255,
      "assignmentMethod" : "UUID"
   },

   "contentColumn" :
   {
      "name" : "JSON_DOCUMENT",
      "sqlType" : "BLOB",
      "jsonFormat" : "OSON"
   },
   "versionColumn" :
   {
     "name" : "VERSION",
     "method" : "UUID"
   },

   "lastModifiedColumn" :
   {
     "name" : "LAST_MODIFIED"
   },

   "creationTimeColumn" :
   {
```

```
      "name" : "CREATED_ON"
   },

   "readOnly" : false
}
```

> **Note:**
>
> Using Always Free Autonomous Database with Oracle Database 23ai, the default metadata changes as follows.

```
{
   "keyColumn" :
   {
      "name" : "ID",
      "sqlType" : "VARCHAR2",
      "maxLength" : 255,
      "assignmentMethod" : "UUID"
   },

   "contentColumn" :
   {
      "name" : "JSON_DOCUMENT",
      "sqlType" : "JSON",
   },
   "versionColumn" :
   {
     "name" : "VERSION",
     "method" : "UUID"
   },

   "lastModifiedColumn" :
   {
     "name" : "LAST_MODIFIED"
   },

   "creationTimeColumn" :
   {
      "name" : "CREATED_ON"
   },

   "readOnly" : false
}
```

SODA Customized Collection Metadata on Autonomous Database

Describes SODA collection custom metadata on Autonomous Database.

Each SODA implementation provides a way to customize the collection metadata during collection creation. For example, in SODA for Java, you can use the following command:

```
OracleDocument metadata = db.createDocumentFromString("metadata_string");
OracleCollection col = db.admin().createCollection("myCustomColl", metadata);
```

In this example, for *metadata_string* you can use the default metadata as the starting point, and customize the following:

- Change `keyColumn.assignmentMethod` to `CLIENT`: Change the value of the `assignmentMethod` under `keyColumn` in the metadata to `CLIENT` (instead of `UUID`).

  Valid values for `keyColumn.assignmentMethod` on Autonomous Database:

  - **UUID** (default): Keys are generated by SODA, based on the `UUID`.
  - **CLIENT**: Keys are assigned by the client application.

The following example specifies client-assigned keys. Otherwise, the default settings are used.

```
{
   "keyColumn" :
   {
      "name" : "ID",
      "sqlType" : "VARCHAR2",
      "maxLength" : 255,
      "assignmentMethod" : "CLIENT"
   },

   "contentColumn" :
   {
      "name" : "JSON_DOCUMENT",
      "sqlType" : "BLOB",
      "jsonFormat" : "OSON"
   },

   "versionColumn" :
   {
     "name" : "VERSION",
     "method" : "UUID"
   },

   "lastModifiedColumn" :
   {
     "name" : "LAST_MODIFIED"
   },

   "creationTimeColumn" :
   {
      "name" : "CREATED_ON"
```

```
    },

    "readOnly" : false
}
```