ORACLE

Back Up Oracle Fusion Data Intelligence Data with Oracle Data Pump

Use Oracle Data Pump to back up selective custom schemas and custom objects.

March 27, 2026, version 5.0

Disclaimer

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle. Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

Revision History

The following revisions have been made to this document since its initial publication.

DATE	REVISION
January 2026	Initial publications
February 2026	Minor editorial changes
February 2026	Publish to 26.R1 library
May 2026	Publish to 26.R2 library

Authors: Jay Pearson and Wojciech Burzynski

Table of Contents

Purpose

This document outlines the process for leveraging Oracle Data Pump to selectively back up custom Autonomous Data Warehouse objects and their corresponding data to Oracle Object Storage. Additionally, it details the steps necessary for restoring these specific objects and data as needed.

The provided recommendations, links, and code snippets are designed to help you implement a robust backup and restore strategy for your custom database schemas and data. This process is exclusively for custom objects, as all pre-built Autonomous Data Warehouse objects and their data are automatically updated daily and don't require manual backup. See Automated Backup for Oracle Autonomous Data Warehouse.

For strategies on backing up other components of Fusion Data Intelligence, see Back Up and Restore.

Overview

Use Oracle Data Pump to back up selective custom schemas and custom objects. You can create fine-grained backups using Oracle Data Pump and securely save to a resilient file store such as Oracle Storage Service. In the event of a data loss, you can use these backups to import and recover database objects and data. See Overview of Oracle Data Pump.

Note: The Oracle Cloud Infrastructure Object Storage URIs differ depending on the location of your realm. See Oracle Cloud Infrastructure Object Storage Native URI Format.

Back up Data to an Oracle Object Storage using Data Pump

Follow these steps to back up your Autonomous Data Warehouse (ADW) objects and data to an Oracle Object Storage using Data Pump

1. Log in to Oracle Cloud Infrastructure, navigate to **Oracle Database**, then navigate to **Autonomous Data Warehouse**, and then obtain the Autonomous Data Warehouse OCID. See Database OCID.

   Alternatively, connect to the database using a SQL query tool, and obtain the Autonomous Data Warehouse OCID by querying the CLOUD_IDENTITY column of the V$PDBS view. See Obtain Tenancy Details.

   ```
   SELECT cloud_identity FROM v$pdbs;
   ```

2. Create and configure a private Object Storage bucket for secure database backup storage (see Creating an Object Storage Bucket), enable Object Versioning (see Object Storage Versioning), and create a Lifecycle Policy Rule to automatically delete files that are older than a specific number of days. See Creating the Object Lifecycle Policy in Object Storage.

3. Create a Dynamic Group and add a matching rule based on your ADW OCID. See Managing Dynamic Groups.

   ```
   any { resource.id = '<<FDI ADW OCID>>'}
   ```

4. Create a policy to authorize the Dynamic Group for API interactions with Oracle Cloud Infrastructure services. Then enable the Autonomous Data Warehouse to list the Object Storage buckets within the compartment. Then upload files to Object Storage and issue notification messages. See Writing Policies for Dynamic Groups.

   ```
   Allow dynamic-group <<DYNAMIC_GROUP>> to read buckets in compartment <<COMPARTMENT>>
   Allow dynamic-group <<DYNAMIC_GROUP>> to manage objects in compartment <<COMPARTMENT>> where
   all
   ```

```
{target.bucket.name='<<BUCKET_NAME>>', any
{request.permission='OBJECT_CREATE',request.permission='OBJECT_READ',request.permission='OBJEC
T_OVERWRITE',request.permission='OBJECT_INSPECT'}}
Allow dynamic-group <<DYNAMIC_GROUP>> to use ons-family in compartment <<COMPARTMENT>>
```

5. Enable Resource Principal on Autonomous Data Warehouse using the OCI$RESOURCE_PRINCIPAL credential to secure access to Oracle Cloud Infrastructure resources. See Use Resource Principal to Access Oracle Cloud Infrastructure resources.

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

   As an alternative, create and use your own credentials. See CREATE_CREDENTIAL Procedure.

```
DBMS_CLOUD.CREATE_CREDENTIAL (
credential_name IN VARCHAR2,
user_ocid       IN VARCHAR2,
tenancy_ocid    IN VARCHAR2,
private_key     IN VARCHAR2,
fingerprint     IN VARCHAR2);
```

6. Validate the bucket access from Autonomous Data Warehouse using `dbms_cloud.list_objects`. See LIST_OBJECTS Function.

```
select * from
dbms_cloud.list_objects('OCI$RESOURCE_PRINCIPAL','<<BUCKET_NATIVE_URI>>');
```

7. Optional: Under Notifications, create a topic (see Creating a Topic) and add a subscription (see Creating a Subscription) to it.

8. Optional: Create a non-Administrator user (see Create Users on Autonomous Database), grant the required privileges to manage the backups, (see Manage User Roles and Privileges on Autonomous Database), and enable resource principal for the user (see Use Resource Principal to Access Oracle Cloud Infrastructure Resources).

```
CREATE USER custombackup IDENTIFIED BY <<password>>;
```

```
GRANT RESOURCE, CONNECT TO custombackup;
GRANT UNLIMITED TABLESPACE TO custombackup;
GRANT CREATE TABLE TO custombackup;
GRANT EXECUTE ON DBMS_CLOUD TO custombackup;
GRANT READ, WRITE ON DIRECTORY DATA_PUMP_DIR TO custombackup;
GRANT EXECUTE ON DBMS_CLOUD TO custombackup;
GRANT EXECUTE ON DBMS_CLOUD_OCI_ONS_MESSAGE_DETAILS_T TO custombackup;
GRANT EXECUTE ON DBMS_CLOUD_OCI_ONS_NOTIFICATION_DATA_PLANE_PUBLISH_MESSAGE_RESPONSE_T TO
custombackup;
GRANT EXECUTE ON DBMS_CLOUD_OCI_ONS_NOTIFICATION_DATA_PLANE TO custombackup;
GRANT EXECUTE ON DBMS_CLOUD_OCI_DB_DATABASE_GET_AUTONOMOUS_DATABASE_RESPONSE_T TO
custombackup;
GRANT EXECUTE ON DBMS_CLOUD_OCI_DB_DATABASE TO custombackup;
GRANT SELECT ON v$pdbs TO custombackup;
```

```
EXEC DBMS_CLOUD_ADMIN.ENABLE_RESOURCE_PRINCIPAL(username => 'custombackup');
```

9. Create a Stored Procedure that exports the custom schema to the object storage bucket and perform the following steps:

a) Export the schema directly to the `DATA_PUMP_DIR` database directory, using the `DBMS_DATAPUMP` package. See DBMS_DATAPUMP. You don't need an external server.

b) Use the `DBMS_CLOUD` package (see DBMS_CLOUD Package) to upload (see LIST_FILES Function and PUT_OBJECT Procedure) all the files located within the `DATA_PUMP_DIR` database directory to Object Storage.

c) Remove the outdated files from Object Storage. See DELETE_FILE Procedure.

d) Issue a notification message. See Overview of Notifications.

Suggested procedures and functions to utilize within the stored procedure include the following:

- Use `dbms_datapump.open` to declare a new job using the Data Pump API. See OPEN Function.
- Use `dbms_datapump.add_file` to add the files to the dump file set for an Export, Import, or `SQL_FILE` operation, or specify the log file or the output file for a `SQL_FILE` operation. See ADD_FILE Procedure.
- Use `dbms_datapump.metadata_filter` to add filters to the job. See METADATA_FILTER Procedure.
- Use `dbms_datapump.set_parallel` to adjust the degree of parallelism. See SET_PARALLEL Procedure.
- Use `dbms_datapump.set_parameter` to specify the job processing options. See SET_PARAMETER Procedures.
- Use `dbms_datapump.start_job` to begin or resume executing the job. See START_JOB Procedure.
- Use `dbms_output.put_line` to display the debugging information. See DBMS_OUTPUT.
- Use `dbms_datapump.get_status` to monitor the status of the job. See GET_STATUS Procedure.
- Use `dbms_datapump.detach` to conclude using the handle. See DETACH Procedure.

  Use `dbms_cloud_oci_ons_notification_data_plane.publish_message` to publish a message to the specified topic. See PUBLISH_MESSAGE Function.
- Use `dbms_cloud_oci_db_database.get_autonomous_database` to get details of the specified Autonomous Database. See GET_AUTONOMOUS_CONTAINER_DATABASE Function)
- Use `dbms_cloud.put_object` to upload the dump file to the Object Storage. See PUT_OBJECT Procedure.
- Use `dbms_cloud.list_files` to list the files in the specified directory. See LIST_FILES Function.
- Use `dbms_cloud.delete_file` to clean up the specified directory. See DELETE_FILE Procedure.

10. Schedule a database job to run the stored procedure using Database Actions Jobs (see Create or Edit Job) or DBMS_SCHEDULER (see DBMS_SCHEDULER).
11. Validate the job from Oracle Database Actions, navigate to Scheduling, locate the job created in the previous step, and review the history page for log run details. See History.
12. Validate the backup by navigating to the bucket and reviewing the objects created by the job. See Listing Object Storage Objects in a Bucket.

Restore your database objects from Object Storage using Data Pump

Follow these steps to restore your Autonomous Data Warehouse objects and data from Oracle Object Storage using Data Pump:

1. Log in to Oracle Cloud Infrastructure, navigate to Oracle Database, Autonomous Data Warehouse, and then obtain the Autonomous Data Warehouse OCID. See Database OCID.

   Alternatively, connect to the database using a SQL query tool, and obtain the Autonomous Data Warehouse OCID by querying the `CLOUD_IDENTITY` column of the `V$PDBS` view. See Obtain Tenancy Details.

   ```
   SELECT cloud_identity FROM v$pdbs;
   ```

2. Create and configure a private Object Storage bucket to archive log files. See Creating an Object Storage Bucket). Enable Object Versioning (see Object Storage Versioning), and create a Lifecycle Policy Rule to automatically delete files after they are older than a specific number of days. See Creating the Object Lifecycle Policy in Object Storage.

3. Create a Dynamic Group and add a matching rule based on your ADW OCID. See Managing Dynamic Groups.

   ```
   any {resource.id = '<<FDI ADW OCID>>'}
   ```

4. Create a policy to upload the log files to the bucket, enable the Autonomous Data Warehouse to list the Object Storage buckets within the compartment, upload files to Object Storage, and issue notification messages. See Writing Policies for Dynamic Groups.

   ```
   Allow dynamic-group <<DYNAMIC_GROUP>> to read buckets in compartment <<COMPARTMENT>>
   Allow dynamic-group <<DYNAMIC_GROUP>> to manage objects in compartment <<COMPARTMENT>> where
   all
   {target.bucket.name='<<BUCKET_NAME>>', any
   {request.permission='OBJECT_CREATE',request.permission='OBJECT_READ',request.permission='OBJEC
   T_OVERWRITE',request.permission='OBJECT_INSPECT'}}
   ```

5. Create a policy to read the backup files from Object Storage. See Writing Policies for Dynamic Groups.

   ```
   Allow dynamic-group <<DYNAMIC_GROUP> to read objects in compartment <<COMPARTMENT>> where all
   {target.bucket.name='<<BUCKET_NAME>>'}
   ```

6. Enable Resource Principal on Autonomous Data Warehouse, using the OCI$RESOURCE_PRINCIPAL credentials to secure access to Oracle Cloud Infrastructure resources. See Use Resource Principal to Access Oracle Cloud Infrastructure resources.

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

7. Validate the log bucket access from Autonomous Data Warehouse using `dbms_cloud.list_objects`. See LIST_OBJECTS Function.

   ```
   select * from
   dbms_cloud.list_objects('OCI$RESOURCE_PRINCIPAL','<<BUCKET_NATIVE_URI>>');
   ```

8. Validate the backup bucket access from Autonomous Data Warehouse using `dbms_cloud.list_objects`. See LIST_OBJECTS Function.

   ```
   select * from
   dbms_cloud.list_objects('OCI$RESOURCE_PRINCIPAL','<<BUCKET_NATIVE_URI>>');
   ```

9. Create a Stored Procedure (see CREATE PROCEDURE Statement) that will import the custom schema from the Object Storage bucket into Autonomous Data Warehouse and perform the following steps:

   a. Download the backup files from Object Storage to the `DATA_PUMP_DIR` database directory by using the `DBMS_CLOUD` package.

   b. Import the schema directly from the `DATA_PUMP_DIR` database directory by using the `DBMS_DATAPUMP` package. See DBMS_DATAPUMP. No external server is required.

c. Invoke `DBMS_CLOUD` to upload all logs from the `DATA_PUMP_DIR` database directory to the log Object Storage.

d. Remove the outdated files from Object Storage. See DELETE_FILE Procedure.

Suggested procedures and functions to utilize within the stored procedure include the following:

- Use `dbms_datapump.open` to declare a new job using the Data Pump API. See OPEN Function.
- Use `dbms_datapump.add_file` to add the files to the dump file set for an Export, Import, or `SQL_FILE` operation, or specify the log file or the output file for a `SQL_FILE` operation. See ADD_FILE Procedure.
- Use `dbms_datapump.metadata_filter` to add filters to the job. See METADATA_FILTER Procedure.
- Use `dbms_datapump.set_parallel` to adjust the degree of parallelism. See SET_PARALLEL Procedure.
- Use `dbms_datapump.set_parameter` to specify the job processing options. See SET_PARAMETER Procedures.
- Use `dbms_datapump.start_job` to begin or resume the job execution. See START_JOB Procedure.
- Use `dbms_output.put_line` to display the debugging information. See DBMS_OUTPUT.
- Use `dbms_datapump.get_status` to monitor the status of the job. See GET_STATUS Procedure.
- Use `dbms_datapump.detach` to conclude using the handle. See DETACH Procedure.

Use `dbms_cloud_oci_ons_notification_data_plane.publish_message` to publish a message to the specified topic. See PUBLISH_MESSAGE Function.

- Use `dbms_cloud_oci_db_database.get_autonomous_database` to get details of the specified Autonomous Database. See GET_AUTONOMOUS_CONTAINER_DATABASE Function.
- Use `dbms_cloud.list_objects` to lists objects from Object Storage. See LIST_OBJECTS Function.
- Use `dbms_cloud.get_object` to download an object from Object Storage and copy it to the database directory. See GET_OBJECT Procedure and Function.
- Use `dbms_cloud.put_object` to upload the dump file to the Object Storage. See PUT_OBJECT Procedure.
- Use `dbms_cloud.list_files` to list the files in the specified directory. See LIST_FILES Function.
- Use `dbms_cloud.delete_file` to cleanup the specified directory. See DELETE_FILE Procedure.

10. Call the stored procedure once using `EXEC` (see EXEC section in CREATE PROCEDURE Statement) or schedule a database job to run the stored procedure either using Database Actions Jobs (see Create or Edit Job) or `DBMS_SCHEDULER` (see DBMS_SCHEDULER).
11. Validate the job from Oracle Database Actions, navigate to **Scheduling**, locate the job created in the previous step, and review history page for log run details. See History.
12. Validate that the logs are written to the log bucket and review the objects the job has created. See Listing Object Storage Objects in a Bucket.

Backup Sample SQL

```
CREATE OR REPLACE PROCEDURE CUSTOMDATA_BACKUP
-----------------------------------------------------------------------
-- Overview
-- This stored procedure implements, a mechanism to create backup of
-- custom schema, upload backup files to Object Storage and performing
```

```
-- post backup maintenance steps such as cleanup.
-------------------------------------------------------------------------
AS
    -- MANDATORY: Object storage uri for export dumps
    v_object_storage_uri  VARCHAR2(120) :=
'https://objectstorage.{region}.oraclecloud.com/n/{namespace}/b/{bucket_name}/o/';
    -- MANDATORY: Database schema to export
    v_schema VARCHAR2(30)     := '{DB_Schema_Name}';
    -- MANDATORY: Export filename. The substitution variable _U% is added by default to the filename.
    v_filename VARCHAR2(30)   := '{File_Name}';
    -- MANDATORY: Log filename.
    v_log_filename VARCHAR2(30)   := '{Export_Log}.log';
       -- MANDATORY: Maximum size of each dump file
       v_filesize VARCHAR2(10)   := '5G';
    -- OPTIONAL: Use Notifications service to get notified when backup is completed
    -- Set to null to do NOT use Notifications
    v_topic_ocid VARCHAR2(100) :=
'{ocid1.onstopic.oc1.abc.aaaaaaaa11111111111111111111111111111111111111111111}';
    -- OPTIONAL: Notification level
    --   0 - Notification are disabled
    --   1 - Send notification only when expectation is raised
    --   2 - Always send notification
    v_notify_level NUMBER := 2;

    -- MANDATORY: Data pump directory (use default value)
    v_directory VARCHAR2(30)  := 'DATA_PUMP_DIR';
    -- MANDATORY: Cloud service credentials for accessing object storage
    v_credentials VARCHAR2(30) := 'OCI$RESOURCE_PRINCIPAL';

    h1 NUMBER;               -- Data Pump job handle
    v_job_name  VARCHAR2(60);
    v_timeout NUMBER := -1;
    v_index NUMBER := 0;
    job_state VARCHAR2(30);
    job_status ku$_Status;
    TYPE files_t IS TABLE OF VARCHAR2(60);
    files_array files_t :=  files_t();
    ex_incorrect_file EXCEPTION;

    -- Procedure triggers OCI notification service
    PROCEDURE send_notification(p_topic_ocid VARCHAR2
                               ,p_credential_name VARCHAR2
                               ,p_notify_level NUMBER
                               ,p_exception BOOLEAN DEFAULT FALSE)
    IS
        l_msg_details DBMS_CLOUD_OCI_ONS_MESSAGE_DETAILS_T;
        l_msg_response dbms_cloud_oci_ons_notification_data_plane_publish_message_response_t;
        l_msg_title VARCHAR2(100);
        l_msg_body VARCHAR2(4000);
        l_lines_arr DBMSOUTPUT_LINESARRAY;
        l_num_line NUMBER;
        l_region VARCHAR2(100);
        l_db_response dbms_cloud_oci_db_database_get_autonomous_database_response_t;
        l_cloud_identity VARCHAR2(4000);
    BEGIN
        IF (p_topic_ocid is not null and p_notify_level > 0) THEN
```

```
            dbms_output.get_lines(
                  lines => l_lines_arr
                , numlines => l_num_line);
            FOR i In 1 .. l_num_line
            LOOP
               -- Message to display in output
               dbms_output.put_line(l_lines_arr(i));
               -- Message to send using notifications service
               l_msg_body := l_msg_body || l_lines_arr(i) || CHR(10);
            END LOOP;

            SELECT CLOUD_IDENTITY
            into l_cloud_identity
            FROM v$pdbs;
            l_region := lower(json_value(l_cloud_identity,'$.REGION'));
            l_db_response := dbms_cloud_oci_db_database.get_autonomous_database(
                        autonomous_database_id =>
lower(json_value(l_cloud_identity,'$.DATABASE_OCID'))
                      , opc_request_id => null
                      , region => l_region
                      , endpoint => null
                      , credential_name => v_credentials
             );

            IF (p_notify_level >= 1 and p_exception = TRUE ) THEN
                l_msg_body := l_msg_body || DBMS_UTILITY.FORMAT_ERROR_STACK() || CHR(10);
                l_msg_body := l_msg_body || DBMS_UTILITY.FORMAT_ERROR_BACKTRACE() || CHR(10);
                l_msg_title := l_db_response.response_body.display_name || ' - schema '|| v_schema ||
' backup FAILED';
                l_msg_details := DBMS_CLOUD_OCI_ONS_MESSAGE_DETAILS_T(
                          title => l_msg_title
                        , body => l_msg_body
                     );
                l_msg_response := dbms_cloud_oci_ons_notification_data_plane.publish_message(
                          topic_id => p_topic_ocid
                        , message_details => l_msg_details
                        , opc_request_id => null
                        , message_type => null
                        , region => l_region
                        , endpoint => null
                        , credential_name => p_credential_name
                    );
            ELSIF(p_notify_level = 2 ) THEN
                l_msg_title := l_db_response.response_body.display_name || ' - schema '|| v_schema ||
' backup COMPLETED';
                l_msg_details := DBMS_CLOUD_OCI_ONS_MESSAGE_DETAILS_T(
                          title => l_msg_title
                        , body => l_msg_body
                     );
                l_msg_response := dbms_cloud_oci_ons_notification_data_plane.publish_message(
                          topic_id => p_topic_ocid
                        , message_details => l_msg_details
                        , opc_request_id => null
                        , message_type => null
                        , region => l_region
                        , endpoint => null
```

```
                    , credential_name => p_credential_name
                );
            END IF;
        END IF;
    END;
BEGIN
    -- Clear output cache
    DBMS_OUTPUT.DISABLE();
    DBMS_OUTPUT.ENABLE();
    dbms_output.put_line('Start export process ' || to_char(sysdate,'YYYY-MM-DD HH24:MI:SS'));
    -- Create a (schema-named) Data Pump job to esport a schema.
    v_job_name := 'EXP_'||v_schema||to_char(sysdate,'YYYYMMDD_hh24miss');
    h1 := dbms_datapump.open (operation => 'EXPORT'
                            , job_mode => 'SCHEMA'
                            , job_name => v_job_name
                            , version => 'COMPATIBLE');
    -- Create export data file set
    BEGIN
        dbms_datapump.add_file(handle => h1
                    , filename => v_filename||'_%U.dmp'
                    , directory => v_directory
                    , filesize => v_filesize
                    , filetype => dbms_datapump.KU$_FILE_TYPE_DUMP_FILE
                    , reusefile => 1);
    EXCEPTION
        WHEN OTHERS THEN
            RAISE ex_incorrect_file;
    END;
    -- Add log file
    dbms_datapump.add_file(handle => h1
                        , filename => v_log_filename
                        , directory => v_directory
                        , filesize => '500M'
                        , filetype => dbms_datapump.KU$_FILE_TYPE_LOG_FILE
                        , reusefile => 1);

    -- A metadata filter is used to specify the schema that will be exported.
    dbms_datapump.metadata_filter(handle => h1
                                , name => 'SCHEMA_EXPR'
                                , value => 'IN('''|| v_schema ||''')');

    -- Set parallelizm
    dbms_datapump.set_parallel(handle => h1, degree => 1);

    -- Add expdp parameters
    dbms_datapump.set_parameter(handle => h1, name => 'COMPRESSION', value => 'ALL');
    dbms_datapump.set_parameter(handle => h1, name => 'INCLUDE_METADATA', value => 1);
    dbms_datapump.set_parameter(handle => h1, name => 'DATA_ACCESS_METHOD', value => 'AUTOMATIC');
    dbms_datapump.set_parameter(handle => h1, name => 'ESTIMATE', value => 'BLOCKS');

    -- Start the job. An exception will be generated if something is not set up properly.
    DBMS_DATAPUMP.START_JOB(h1);

    -- The export job should now be running. In the following loop, the job
    -- is monitored until it completes.
    job_state := 'UNDEFINED';
```

```
    WHILE (job_state != 'COMPLETED') and (job_state != 'STOPPED')
    LOOP
        dbms_datapump.get_status(handle => h1
                            ,mask => dbms_datapump.ku$_status_job_error +
                                     dbms_datapump.ku$_status_job_status +
                                     dbms_datapump.ku$_status_wip
                            ,timeout => -1
                            ,job_state => job_state
                            ,status => job_status);
    END LOOP;
    dbms_output.put_line('Dump files created');
    IF (job_status.job_status.files is not null) THEN
        FOR i IN 1..job_status.job_status.files.COUNT
        LOOP
            -- file_Type 0 - disk
            -- file_Type 3 - template
            IF (job_status.job_status.files(i).file_type = 0) THEN
                v_index := v_index + 1;
                files_array.extend();
                files_array(v_index) := REGEXP_SUBSTR(job_status.job_status.files(i).file_name,
'[^/]+$');
                dbms_output.put_line(files_array(v_index));
            END IF;
        END LOOP;
    END IF;
    -- Indicate that the job finished and detach from it.
    dbms_output.put_line('Export job has completed');
    dbms_output.put_line('Export final job state = ' || job_state);
    dbms_datapump.detach(h1);

    -- Upload files to the object storage
    dbms_output.put_line('Starting upload to object storage');
    FOR i IN files_array.FIRST .. files_array.LAST
    LOOP
        dbms_output.put_line('Uploading file: ' || files_array(i));
        DBMS_CLOUD.PUT_OBJECT(
            credential_name => v_credentials,
            object_uri => v_object_storage_uri || files_array(i),
            directory_name => v_directory,
            file_name => files_array(i));
    END LOOP;
    FOR r IN
        (SELECT object_name
        FROM DBMS_CLOUD.LIST_FILES(v_directory)
        WHERE object_name = v_log_filename)
    LOOP
        dbms_output.put_line('Uploading file: '||r.object_name);
        DBMS_CLOUD.PUT_OBJECT(
            credential_name => v_credentials,
            object_uri => v_object_storage_uri || r.object_name,
            directory_name => v_directory,
            file_name => r.object_name);
    END LOOP;
    dbms_output.put_line('Upload completed');

    -- Delete all uploaded files from the specified directory
```

```
    dbms_output.put_line('Starting cleanup on directory ' || v_directory);
    FOR i IN files_array.FIRST .. files_array.LAST
    LOOP
        dbms_output.put_line('Deleting file: ' || files_array(i));
        DBMS_CLOUD.DELETE_FILE(v_directory,files_array(i));
    END LOOP;
    FOR r IN
        (SELECT object_name
        FROM DBMS_CLOUD.LIST_FILES(v_directory)
        WHERE object_name = v_log_filename)
    LOOP
        dbms_output.put_line('Deleting file: '||r.object_name);
        DBMS_CLOUD.DELETE_FILE(v_directory,r.object_name);
    END LOOP;
    dbms_output.put_line('Cleanup completed');
    dbms_output.put_line('End export process ' || to_char(sysdate,'YYYY-MM-DD HH24:MI:SS'));

    -- Send output to notification service
    send_notification(p_topic_ocid => v_topic_ocid
                      ,p_credential_name => v_credentials
                      ,p_notify_level => v_notify_level);
    EXCEPTION
        WHEN ex_incorrect_file THEN
            v_timeout := 30;
            dbms_datapump.get_status(handle => h1
                            ,mask => dbms_datapump.ku$_status_job_error +
                                     dbms_datapump.ku$_status_wip
                            ,timeout => v_timeout
                            ,job_state => job_state
                            ,status => job_status);
            -- Print job error message
            IF (job_status.error is not null) THEN
                FOR i IN 1..job_status.error.COUNT
                LOOP
                    dbms_output.put_line(job_status.error(i).errornumber || ': ' ||
job_status.error(i).logtext);
                END LOOP;
            END IF;
            -- Remove job
            IF ( h1 is not null ) THEN
                dbms_datapump.detach(h1);
            END IF;
            -- Send output to notification service
            send_notification(p_topic_ocid => v_topic_ocid
                              ,p_notify_level => v_notify_level
                              ,p_credential_name => v_credentials
                               ,p_exception => TRUE);
        WHEN others THEN
            -- Send output to notification service
            send_notification(p_topic_ocid => v_topic_ocid
                              ,p_credential_name => v_credentials
                              ,p_notify_level => v_notify_level
                              ,p_exception => TRUE);
            RAISE;
END;
/
```

Import Sample SQL

```
CREATE OR REPLACE PROCEDURE CUSTOMDATA_IMPORT
-----------------------------------------------------------------------
-- Overview
-- This stored procedure implements a mechanism to download backup
-- files from Object Storage, restore custom schema in ADW
-- and performing post restore maintenance steps such as cleanup
-- and upload logs to object storage.

AS
    -- Object storage uri with backup dump files
    v_bkp_object_storage_uri  VARCHAR2(120) :=
'https://objectstorage.{region}.oraclecloud.com/n/{namespace}/b/{bucket name}/o/';
    -- Object storage uri for logs
    v_log_object_storage_uri  VARCHAR2(120) :=
'https://objectstorage.{region}.oraclecloud.com/n/{namespace}/b/{bucket name}/o/';
    -- Database target schema to import
    v_schema VARCHAR2(30)     := '{DB Schema}';
    -- Backup dump filename. Can include SQL wildcard characters.
    v_filename VARCHAR2(30)   := '{DB Schema}.dmp';
    -- Log filename.
    v_log_filename VARCHAR2(30)   := '{log file}.log';

    -- Data pump directory (use default value)
    v_directory VARCHAR2(30)  := 'DATA_PUMP_DIR';
    -- Cloud service credentials for accessing object storage
    v_credentials VARCHAR2(30) := 'OCI$RESOURCE_PRINCIPAL';

    h1 NUMBER;               -- Data Pump job handle
    v_job_name  VARCHAR2(60);
    v_job_phase NUMBER := 0;
    v_timeout NUMBER := -1;
    v_index NUMBER := 0;
    job_state VARCHAR2(30);
    job_status ku$_Status;
    TYPE files_t IS TABLE OF VARCHAR2(60);
    files_array files_t :=  files_t();
    ex_no_files EXCEPTION;
    ex_incorrect_file EXCEPTION;
BEGIN
    dbms_output.put_line('Start import process ' || to_char(sysdate,'YYYY-MM-DD HH24:MI:SS'));

    -- Download dmp files from Object Storage
    dbms_output.put_line('Starting download files from object storage');
    FOR r IN
        (SELECT object_name
        FROM dbms_cloud.list_objects(v_credentials,v_bkp_object_storage_uri)
        WHERE object_name like v_filename
        ORDER BY 1)
    LOOP
        dbms_output.put_line('Downloading file: '||r.object_name);
        DBMS_CLOUD.GET_OBJECT (
           credential_name => v_credentials
          ,object_uri => v_bkp_object_storage_uri || r.object_name
          ,directory_name => v_directory);
```

```
    v_index := v_index + 1;
    files_array.extend();
    files_array(v_index) := r.object_name;
END LOOP;

-- When no files found exit program.
IF (v_index = 0) THEN
    RAISE ex_no_files;
END IF;
dbms_output.put_line('Download completed');

-- Create a (schema-named) Data Pump job to do a schema import.
v_job_name := 'IMP_'||v_schema||to_char(sysdate,'YYYYMMDD_hh24miss');
h1 := dbms_datapump.open (operation => 'IMPORT'
                        , job_mode => 'SCHEMA'
                        , job_name => v_job_name
                        , version => 'COMPATIBLE');

-- Create import data file set
dbms_output.put_line('Creating import dump file set');
BEGIN
    FOR i IN files_array.FIRST .. files_array.LAST
    LOOP
        dbms_output.put_line('Adding file: ' || files_array(i));
        dbms_datapump.add_file(handle => h1
                    , filename => files_array(i)
                    , directory => v_directory
                    , filetype => dbms_datapump.KU$_FILE_TYPE_DUMP_FILE);
    END LOOP;
EXCEPTION
    WHEN OTHERS THEN
        RAISE ex_incorrect_file;
END;

-- Add log file
dbms_datapump.add_file(handle => h1
                    , filename => v_log_filename
                    , directory => v_directory
                    , filesize => '500M'
                    , filetype => dbms_datapump.KU$_FILE_TYPE_LOG_FILE);

-- A metadata filter is used to specify target schema.
dbms_datapump.metadata_filter(handle => h1
                            , name => 'SCHEMA_EXPR'
                            , value => 'IN('''|| v_schema ||''')');
-- Set parallelizm
dbms_datapump.set_parallel(handle => h1, degree => 1);
-- Add impdp parameters
dbms_datapump.set_parameter(handle => h1, name => 'INCLUDE_METADATA', value => 1);
dbms_datapump.set_parameter(handle => h1, name => 'DATA_ACCESS_METHOD', value => 'AUTOMATIC');
dbms_datapump.set_parameter(handle => h1, name => 'TABLE_EXISTS_ACTION', value => 'REPLACE' );
dbms_datapump.set_parameter(handle => h1, name => 'SKIP_UNUSABLE_INDEXES', value => 0 );

-- Start the job. An exception will be generated if something is not set up properly.
dbms_output.put_line('Starting import job');
```

```
    BEGIN
        DBMS_DATAPUMP.START_JOB(handle => h1, skip_current => 0, abort_step => 0 );
        v_job_phase := 1;
    EXCEPTION
      WHEN OTHERS THEN
      v_job_phase := 2;
      dbms_output.put_line('DBMS_DATAPUMP.START_JOB failed. Please check '||v_log_filename);
    END;

    -- The import job should now be running. In the following loop, the job
    -- is monitored until it completes.
    job_state := 'UNDEFINED';
    -- If START_JOB failed, wait 30 second to capture error to the log file
    IF ( v_job_phase = 2 ) THEN
        DBMS_SESSION.SLEEP(30);
        v_timeout := 30;
    END IF;
    WHILE (job_state != 'COMPLETED') and (job_state != 'STOPPED')
    LOOP
        dbms_datapump.get_status(handle => h1
                            ,mask => dbms_datapump.ku$_status_job_error +
                                     dbms_datapump.ku$_status_wip
                            ,timeout => v_timeout
                            ,job_state => job_state
                            ,status => job_status);
        -- Print job error message
        IF (job_status.error is not null) THEN
            FOR i IN 1..job_status.error.COUNT
            LOOP
                dbms_output.put_line(job_status.error(i).errornumber || ': ' ||
job_status.error(i).logtext);
            END LOOP;
        END IF;
        EXIT when (v_job_phase = 2);
    END LOOP;
    -- Indicate that the job finished and detach from it.
    dbms_output.put_line('Import job completed');
    dbms_output.put_line('Import job final state = ' || job_state);
    dbms_datapump.detach(h1);

    -- Upload log file to the object storage
    dbms_output.put_line('Starting upload logs to object storage');
    FOR r IN
        (SELECT object_name
        FROM DBMS_CLOUD.LIST_FILES(v_directory)
        WHERE object_name = v_log_filename)
    LOOP
        dbms_output.put_line('Uploading file: '||r.object_name);
        DBMS_CLOUD.PUT_OBJECT(
            credential_name => v_credentials,
            object_uri => v_log_object_storage_uri || r.object_name,
            directory_name => v_directory,
            file_name => r.object_name);
    END LOOP;
    dbms_output.put_line('Upload completed');
```

```
    -- Delete all imported files from the specified directory
    dbms_output.put_line('Starting cleanup on directory ' || v_directory);
    FOR i IN files_array.FIRST .. files_array.LAST
    LOOP
        dbms_output.put_line('Deleting file: ' || files_array(i));
        DBMS_CLOUD.DELETE_FILE(v_directory,files_array(i));
    END LOOP;
    FOR r IN
        (SELECT object_name
        FROM DBMS_CLOUD.LIST_FILES(v_directory)
        WHERE object_name = v_log_filename)
    LOOP
        dbms_output.put_line('Deleting file: '||r.object_name);
        DBMS_CLOUD.DELETE_FILE(v_directory,r.object_name);
    END LOOP;
    dbms_output.put_line('Cleanup completed');
    dbms_output.put_line('End import process ' || to_char(sysdate,'YYYY-MM-DD HH24:MI:SS'));

    EXCEPTION
        WHEN ex_no_files THEN
            dbms_output.put_line('No files found in object storage');
        WHEN ex_incorrect_file THEN
            dbms_datapump.get_status(handle => h1
                            ,mask => dbms_datapump.ku$_status_job_error +
                                     dbms_datapump.ku$_status_wip
                            ,timeout => v_timeout
                            ,job_state => job_state
                            ,status => job_status);
            -- Print job error message
            IF (job_status.error is not null) THEN
                FOR i IN 1..job_status.error.COUNT
                LOOP
                    dbms_output.put_line(job_status.error(i).errornumber || ': ' ||
job_status.error(i).logtext);
                END LOOP;
            END IF;

            -- Remove job
            IF ( h1 is not null ) THEN
                dbms_datapump.detach(h1);
            END IF;

        WHEN others THEN
            IF ( h1 is not null ) THEN
                dbms_datapump.detach(h1);
            END IF;
            -- Drop import master table
            FOR r IN
                (SELECT table_name
                FROM DBA_TABLES WHERE TABLE_NAME = v_job_name)
            LOOP
                EXECUTE IMMEDIATE
                    'DROP TABLE' || DBMS_ASSERT.ENQUOTE_NAME(r.table_name, FALSE );
            END LOOP;
            RAISE;
END;
```

```
/
```

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle