51 DBMS_CLOUD_REPO

The `DBMS_CLOUD_REPO` package provides for use of and management of cloud hosted code repositories from Oracle Database. Supported cloud code repositories include GitHub, AWS CodeCommit , and Azure Repos.

DBMS_CLOUD_REPO Overview

The `DBMS_CLOUD_REPO` package provides easy access to files in Cloud Code (Git) Repositories, including: GitHub, AWS CodeCommit, and Azure Repos.

This package is a single interface for access to Multicloud Code repositories and allows you to upload SQL files to Git repositories or install SQL scripts directly from Cloud Code Repositories. This package also allows you to use a Cloud Code Repository to manage code versions for SQL scripts and to install or patch application code from Git repositories.

Concepts

- Git Version Control System: Git is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows.
- Git Repository: A Git repository is a virtual storage of your project. It allows you to save versions of your code, which you can access when needed.

Architecture

`DBMS_CLOUD_REPO` package provides four feature areas:

- Repository Initialization with Generic Cloud Code Repository Handle
  - Initialize a GitHub Code Repository
  - Initialize an AWS CodeCommit Code Repository
  - Initialize an Azure Repos Code Repository
- Repository Management Operations
  - Create a repository
  - Update a repository
  - List repositories
  - Delete a repository
- Repository File Management Operations
  - Upload a file to Code Repository from Oracle Database.
  - Download a file from Code Repository to Oracle Database.
  - Delete files from Code Repository.
  - List files from Code Repository.
- SQL Install Operations

- Export Database object metadata DDL to repository.
- Install SQL statements from a file in the Code Repository in Oracle Database.
- Install SQL statements from a buffer.

DBMS_CLOUD_REPO Data Structures

The `DBMS_CLOUD_REPO` package defines record types and a generic JSON object type `repo`.

REPO JSON Object

A `DBMS_CLOUD_REPO REPO` is an opaque JSON object to represent a Cloud Code Repository of a specific cloud provider. A REPO object can be passed to different `DBMS_CLOUD_REPO` APIs. This opaque object ensures that `DBMS_CLOUD_REPO` procedures and functions are multicloud compatible; you do not have to change any code when you migrate from one Cloud Code Repository provider to another Cloud Code Repository.

DBMS_CLOUD_REPO Subprogram Groups

The `DBMS_CLOUD_REPO` package subprograms can be grouped into four categories: Initialization Operations, Repository Management Operations, File Operations, and SQL Install Operations.

DBMS_CLOUD_REPO Initialization Operations

Lists the subprograms for initialization operations within the `DBMS_CLOUD_REPO` package.

Subprogram	Description
INIT_AWS_REPO Function	This function initializes an AWS repository handle and returns an opaque type.
INIT_AZURE_REPO Function	This function initializes an Azure repository handle and returns an opaque type.
INIT_GITHUB_REPO Function	This function initializes a GitHub repository handle and returns an opaque type.
INIT_REPO Function	This function initializes a Cloud Code Repository handle and returns an opaque JSON object.

DBMS_CLOUD_REPO Repository Management Operations

Shows the subprograms for repository management operations within the `DBMS_CLOUD_REPO` package.

Subprogram	Description
CREATE_REPOSITORY Procedure	This procedure creates a Cloud Code Repository identified by the `repo` handle argument.
DELETE_REPOSITORY Procedure	This procedure deletes the Cloud Code Repository identified by the `repo` handle argument.
LIST_REPOSITORIES Function	This function lists all the Cloud Code Repositories identified by the `repo` handle argument.
UPDATE_REPOSITORY Procedure	This procedure updates a Cloud Code repository identified by the `repo` handle argument. The procedure supports updating the name, description, or private visibility status, as supported by the Cloud Code repository.

DBMS_CLOUD_REPO Repository Branch Management Operations

Lists the subprograms for repository branch management operations within the `DBMS_CLOUD_REPO` package.

Subprogram	Description
CREATE_BRANCH Procedure	This procedure creates a branch in a Cloud Code Repository identified by the `repo` handle argument.
DELETE_BRANCH Procedure	This procedure deletes a branch in a Cloud Code Repository identified by the `repo` handle argument.
LIST_BRANCHES Function	This function lists all the Cloud Code Repository branches identified by the `repo` handle argument.
LIST_COMMITS Function	This function lists all the commits in a Cloud Code Repository branch identified by the `repo` handle argument.
MERGE_BRANCH Procedure	This procedure merges a Cloud Code Repository branch into another specified branch in a Cloud Code Repository identified by the `repo` handle argument.

DBMS_CLOUD_REPO File Operations

Lists the subprograms for file operations within the `DBMS_CLOUD_REPO` package.

Subprogram	Description
DELETE_FILE Procedure	This procedure deletes a file from the Cloud Code repository identified by the `repo` handle argument.
GET_FILE Procedure and Function	The function downloads the contents of a file from the Cloud Code repository. The procedure allows you to download the contents of a file from the Cloud Code repository and save the file in a directory.
LIST_FILES Function	This function downloads a file from Cloud Code repository. Optionally, file content can be accessed from either a specific branch, tag or commit name. By default, the file is accessed from the default repository branch.
PUT_FILE Procedure	This procedure uploads a file to the Cloud Code repository identified by the `repo` handle argument. The procedure is overloaded to support either uploading a file from a directory object or uploading the contents from a CLOB to the repository file.

DBMS_CLOUD_REPO SQL Install Operations

Lists the subprograms for SQL install operations within the `DBMS_CLOUD_REPO` package.

Subprogram	Description
EXPORT_OBJECT Procedure	This procedure uploads the DDL metadata of a database object to the Cloud Code repository identified by the `repo` handle argument.
EXPORT_SCHEMA Procedure	This procedure exports metadata of all objects in a schema to a Cloud Code Repository branch identified by the `repo` handle argument.
INSTALL_FILE Procedure	This procedure installs SQL statements from a file in the Cloud Code repository identified by the `repo` handle argument.
INSTALL_SQL Procedure	This procedure installs SQL statements from a buffer given as input.

Summary of DBMS_CLOUD_REPO Subprograms

This section covers the `DBMS_CLOUD_REPO` subprograms provided with Oracle Database.

The `DBMS_CLOUD_REPO` package is made up of the following:

- DBMS_CLOUD_REPO Initialization Operations
- DBMS_CLOUD_REPO Repository Management Operations
- DBMS_CLOUD_REPO File Operations
- DBMS_CLOUD_REPO SQL Install Operations

CREATE_BRANCH Procedure

This procedure creates a branch in the Cloud Code Repository identified by the `repo` handle argument.

Syntax

```
PROCEDURE DBMS_CLOUD_REPO.CREATE_BRANCH(
    repo               IN   CLOB,
    branch_name        IN   VARCHAR2,
    parent_branch_name IN   VARCHAR2 DEFAULT NULL,
    parent_commit_id   IN   VARCHAR2 DEFAULT NULL
 );
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle. This parameter is mandatory and supported for all cloud providers.
`branch_name`	Specifies the repository branch name. This parameter is mandatory and supported for all cloud providers.
`parent_branch_name`	Creates the new branch using the head commit of the specified parent branch. This parameter is supported for all cloud providers. If you do not supply a `parent_branch_name` value, the `parent_branch_name` is set to `main`.
`parent_commit_id`	Creates the new branch using the specified repository commit. This parameter is supported for all cloud providers. If you do not supply a `parent_commit_id` value, the `parent_commit_id` is set to a NULL value.

Note:

To create a branch in a Cloud Code repository, you must specify the parent branch or the parent commit id.

Example

```
BEGIN
  DBMS_CLOUD_REPO.CREATE_BRANCH (
    repo                  => l_repo,
    branch_name           => 'test_branch',
    parent_branch_name    => 'main'
    );
END;
/
```

Usage Note

To run the `DBMS_CLOUD_REPO.CREATE_BRANCH` procedure, you must be logged in as the ADMIN user or have `EXECUTE` privilege on `DBMS_CLOUD_REPO`.

CREATE_REPOSITORY Procedure

This procedure creates a Cloud Code Repository identified by the `repo` handle argument.

Syntax

```
PROCEDURE DBMS_CLOUD_REPO.CREATE_REPOSITORY(
      repo                 IN   CLOB,
      description          IN   CLOB      DEFAULT NULL,
      private              IN   BOOLEAN   DEFAULT TRUE
);
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle. This parameter is supported for all cloud providers.
`description`	A short text description for the repository. This parameter is supported for GITHUB and AWS cloud provider.
`private`	Repository is private and only accessible with valid credentials This parameter is only supported for the GITHUB cloud provider.

Example

```
BEGIN
  DBMS_CLOUD_REPO.CREATE_REPOSITORY(
    repo        => l_repo,
    description => 'My test repo',
    private => TRUE
    );
END;
/
```

DELETE_BRANCH Procedure

This procedure deletes a branch in the Cloud Code repository identified by the `repo` handle argument.

Syntax

```
PROCEDURE DBMS_CLOUD_REPO.DELETE_BRANCH (
      repo                 IN   CLOB,
      branch_name          IN   VARCHAR2   DEFAULT NULL
 );
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle. This parameter is mandatory and supported for all cloud providers.
`branch_name`	Delete branch from a specific repository. This parameter is mandatory and supported for all cloud providers.

Example

```
BEGIN
  DBMS_CLOUD_REPO.DELETE_BRANCH (
      repo        => l_repo,
      branch_name => 'test_branch'
  );
END;
/
```

Usage Note

To run the `DBMS_CLOUD_REPO.DELETE_BRANCH` procedure, you must be logged in as the ADMIN user or have `EXECUTE` privilege on `DBMS_CLOUD_REPO`.

DELETE_FILE Procedure

This procedure deletes a file from the Cloud Code repository identified by the `repo` handle argument.

Syntax

```
PROCEDURE DBMS_CLOUD_REPO.DELETE_FILE(
      repo                 IN   CLOB,
      file_path            IN   VARCHAR2,
      branch_name          IN   VARCHAR2   DEFAULT NULL,
      commit_details       IN   CLOB       DEFAULT NULL
);
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle.
`file_path`	File path to delete file in the repository.
`branch_name`	Delete file from a specific branch.
`commit_details`	Commit Details as a JSON document `{"message": "Commit message", "author": {"name": "Committing user name", "email": "Email of committing user" } }`

Example

```
BEGIN
  DBMS_CLOUD_REPO.DELETE_FILE(
      repo        => l_repo,
      file_path   => 'scripts/test3.sql',
      branch_name => 'test_branch'
  );
END;
/
```

DELETE_REPOSITORY Procedure

This procedure deletes the Cloud Code Repository identified by the `repo` handle argument.

Syntax

```
PROCEDURE DBMS_CLOUD_REPO.DELETE_REPOSITORY(
      repo           IN    CLOB
);
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle.

Example

```
BEGIN
  DBMS_CLOUD_REPO.DELETE_REPOSITORY(
       repo => l_repo
  );
END;
/
```

EXPORT_OBJECT Procedure

This procedure uploads the DDL metadata of a database object to the Cloud Code repository identified by the `repo` handle argument. This procedure is an easy way to upload the metadata definition of a database object in single step.

Syntax

```
PROCEDURE DBMS_CLOUD_REPO.EXPORT_OBJECT(
      repo              IN  CLOB,
      file_path         IN  VARCHAR2,
      object_type       IN  VARCHAR2,
      object_name       IN  VARCHAR2 DEFAULT NULL,
      object_schema     IN  VARCHAR2 DEFAULT NULL,
      branch_name       IN  VARCHAR2 DEFAULT NULL,
      commit_details    IN  CLOB     DEFAULT NULL,
      append            IN  BOOLEAN  DEFAULT FALSE
);
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle.
`file_path`	File path to upload object metadata in the repository.
`object_type`	Object type supported by DBMS_METADATA.
`object_name`	Name of the database object to retrieve metadata.
`object_schema`	Owning schema of the database object.
`branch_name`	Put file to a specific branch.
`commit_details`	Commit Details as a JSON document:`{"message": "Commit message", "author": {"name": "Committing user name", "email": "Email of committing user" } }`
`append`	Append metadata DDL to existing file.

Usage Note

For customized control on the object DDL, you can use `DBMS_METADATA.GET_DDL` along with `DBMS_CLOUD_REPO.PUT_FILE`. In order to get metadata definition of the object, the current user must be privileged to retrieve the object metadata. See DBMS_METADATA for the security requirements of the package.

Example

```
BEGIN
  DBMS_CLOUD_REPO.EXPORT_OBJECT(
     repo        => l_repo,
     object_type => 'PACKAGE',
     object_name => 'MYPACK',
     file_path   => 'mypack.sql'
  );
```

```
END;
/
```

EXPORT_SCHEMA Procedure

This procedure exports metadata of all objects in a schema to the Cloud Code Repository branch identified by the `repo` handle argument.

Syntax

```
PROCEDURE DBMS_CLOUD_REPO.EXPORT_SCHEMA(
     repo            IN CLOB,
     file_path       IN VARCHAR2,
     schema_name     IN VARCHAR2,
     filter_list     IN CLOB          DEFAULT NULL,
     branch_name     IN VARCHAR2      DEFAULT NULL,
     commit_details  IN CLOB          DEFAULT NULL
);
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle. This parameter is mandatory and supported for all cloud providers.
`file_path`	Specifies the name of the schema file to upload to the repo. This parameter is mandatory and supported for all cloud providers.
`schema_name`	Specifies the name of the schema for which a DDL script is to be uploaded to the Cloud Code Repository branch. This parameter is mandatory and supported for all cloud providers.
`filter_list`	Specifies the CLOB of the JSON array that defines the filter conditions to include or exclude the objects whose metadata needs to be exported. This parameter is supported for all cloud providers. JSON parameters for `filter_list` are: • `match_type`: Specifies the type of filter to be applied to the object types or object names. Valid `match_type` values are: – `in/not_in` – `like/not_like` – `equal/not_equal` • `type`: Specifies the type of object by which to filter. • `name`: Specifies the name of the object by which to filter.
`branch_name`	Specifies the repository branch name. This parameter is supported for all cloud providers. If you do not supply a `branch_name` value, the `branch_name` is set to the default repository branch.

Parameter	Description
`commit_details`	Commit Details as a JSON document `{"message": "Commit message", "author": {"name": "Committing user name", "email": "Email of committing user" } }` This parameter is supported for all cloud providers. If you do not supply a `commit_details` value, the `commit_details` is set to the default commit message that includes the information about the current database session user and database name performing the commit.

Example

```
BEGIN
  DBMS_CLOUD_REPO.EXPORT_SCHEMA(
    repo          => l_repo,
    schema_name   => 'USER1',
    file_path     => 'myschema_ddl.sql'
    filter_list   =>
        to_clob('[
             { "match_type":"equal",
                 "type":"table"
             },
             { "match_type":"not_equal",
                 "type":"view"
             },
             { "match_type":"in",
                 "type":"table",
                 "name": " ''EMPLOYEE_SALARY'',''EMPLOYEE_ADDRESS'' "
             },
             { "match_type":"equal",
                 "type":"sequence",
                 "name": "EMPLOYEE_RECORD_SEQ"
             },
             { "match_type":"like",
                 "type":"table",
                 "name": "%OFFICE%"
             }
        ]'
     );
  );
END;
/
```

Usage Note

To run the `DBMS_CLOUD_REPO.EXPORT_SCHEMA` procedure, you must be logged in as the ADMIN user or have `EXECUTE` privilege on `DBMS_CLOUD_REPO`.

GET_FILE Procedure and Function

The function downloads the contents of a file from the Cloud Code repository. The procedure allows you to download the contents of a file from the Cloud Code repository and save the file in a directory.

Syntax

```
FUNCTION DBMS_CLOUD_REPO.GET_FILE(
      repo              IN  CLOB,
      file_path         IN  VARCHAR2,
      branch_name       IN  VARCHAR2  DEFAULT NULL,
      tag_name          IN  VARCHAR2  DEFAULT NULL,
      commit_name       IN  VARCHAR2  DEFAULT NULL
) RETURN CLOB;

PROCEDURE DBMS_CLOUD_REPO.GET_FILE(
      repo              IN  CLOB,
      file_path         IN  VARCHAR2,
      directory_name    IN  VARCHAR2,
      target_file_name  IN  VARCHAR2  DEFAULT NULL,
      branch_name       IN  VARCHAR2  DEFAULT NULL,
      tag_name          IN  VARCHAR2  DEFAULT NULL,
      commit_name       IN  VARCHAR2  DEFAULT NULL
);
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle.
`file_path`	File path in the repository.
`directory_name`	Directory object name to save the file contents.
`target_file_name`	Target file name to save contents in directory.
`branch_name`	Get file from a specific branch.
`tag_name`	Get file from a specific Tag.
`commit_name`	Get file from a specific commit.

Example

```
BEGIN
  DBMS_CLOUD_REPO.GET_FILE(
     repo             =>  l_repo,
     file_path        => 'test3.sql',
     directory_name   => 'DATA_PUMP_DIR',
     target_file_name => 'test2.sql'
  );
END;
/
```

INIT_AWS_REPO Function

This function initializes an AWS repository handle and returns an opaque type.

Syntax

```
FUNCTION DBMS_CLOUD_REPO.INIT_AWS_REPO(
      credential_name IN  VARCHAR2,
      repo_name       IN  VARCHAR2,
      region          IN  VARCHAR2
)  RETURN repo;
```

Parameters

Parameter	Description
`credential_name`	Credential object specifying AWS CodeCommit accesskey/secretkey.
`repo_name`	Specifies the repository name.
`region`	Specifies the AWS region for the CodeCommit repository.

Example

```
BEGIN
  :repo := DBMS_CLOUD_REPO.INIT_AWS_REPO(
                 credential_name => 'AWS_CRED',
                 repo_name       => 'my_repo',
                 region          => 'us-east-1'
           );
END;
/
```

INIT_AZURE_REPO Function

This function initializes an Azure repository handle and returns an opaque type. This function is only supported for Azure cloud provider.

Syntax

```
FUNCTION DBMS_CLOUD_REPO.INIT_AZURE_REPO(
      credential_name IN  VARCHAR2,
      repo_name       IN  VARCHAR2,
      organization    IN  VARCHAR2,
      project         IN  VARCHAR2
)  RETURN repo;
```

Parameters

Parameter	Description
`credential_name`	Credential object specifying Azure, with a Username and Personal Access Token (PAT).

Parameter	Description
`repo_name`	Specifies the repository name.
`organization`	Specifies the Azure DevOps Organization.
`project`	Azure Team Project name.

Example

```
BEGIN
  :repo := DBMS_CLOUD_REPO.INIT_AZURE_REPO(
                 credential_name => 'AZURE_CRED',
                 repo_name       => 'my_repo',
                 organization    => 'myorg',
                 project         => 'myproj',
           );
END;
/
```

INIT_GITHUB_REPO Function

This function initializes a GitHub repository handle and returns an opaque type.

Syntax

```
FUNCTION DBMS_CLOUD_REPO.INIT_GITHUB_REPO(
      credential_name IN  VARCHAR2  DEFAULT NULL,
      repo_name       IN  VARCHAR2,
      owner           IN  VARCHAR2)
RETURN repo;
```

Parameters

Parameter	Description
`credential_name`	Credential object specifying GitHub. User Email and Personal Access Token (PAT).
`repo_name`	Specifies the repository name.
`owner`	Specifies the repository owner.

Example

```
BEGIN
  :repo := DBMS_CLOUD_REPO.INIT_GITHUB_REPO(
                 credential_name => 'GITHUB_CRED',
                 repo_name       => 'my_repo',
                 owner           => 'foo'
           );
END;
/
```

INIT_REPO Function

This function initializes a Cloud Code Repository handle and returns an opaque JSON object. This function is a generic interface to accept a JSON document, and avoids having to change code, you only need to change a JSON document, when moving a code repository from one Cloud Code repository to another Cloud Code repository.

Syntax

```
FUNCTION DBMS_CLOUD_REPO.INIT_REPO(
     params       IN    CLOB)
RETURN CLOB;
```

Parameters

JSON Parameter	Description

`provider`	Cloud code repository provider from the following:
	`DBMS_CLOUD_REPO.GITHUB_REPO ('GITHUB')`
	`DBMS_CLOUD_REPO.AWS_REPO ('AWS')`
	`DBMS_CLOUD_REPO.AZURE_REPO ('AZURE')`

`repo_name`	Specifies the repository name. `DBMS_CLOUD_REPO.PARAM_REPO_NAME`
`owner`	GitHub Repository Owner. `DBMS_CLOUD_REPO.PARAM_OWNER` This parameter is only applicable for GitHub cloud provider.
`region`	AWS Repository Region `DBMS_CLOUD_REPO_PARAM_REGION` This parameter is only applicable for AWS cloud provider.
`organization`	Azure Organization `DBMS_CLOUD_REPO_PARAM_ORGANIZATION` This parameter is only applicable for Azure cloud provider.
`project`	Azure Team Project `DBMS_CLOUD_REPO_PARAM_PROJECT` This parameter is only applicable for Azure cloud provider

Example

```
BEGIN
  :repo := DBMS_CLOUD_REPO.INIT_REPO(
          params => JSON_OBJECT('credential_name' value 'mycred',
                        'repo_name'       value 'myrepo',
                        'repo_owner'      value 'foo')
       );
END;
/
```

INSTALL_FILE Procedure

This procedure installs SQL statements from a file in the Cloud Code repository identified by the `repo` handle argument.

Syntax

```
PROCEDURE DBMS_CLOUD_REPO.INSTALL_FILE(
      repo              IN   CLOB,
      file_path          IN   VARCHAR2,
      branch_name        IN   VARCHAR2   DEFAULT NULL,
      tag_name           IN   VARCHAR2   DEFAULT NULL,
      commit_name        IN   VARCHAR2   DEFAULT NULL,
      stop_on_error      IN   BOOLEAN    DEFAULT TRUE
);
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle.
`file_path`	File path in the repository.
`branch_name`	Branch to install file from a specific branch.
`tag_name`	Tag to install file from a specific Tag.
`commit_name`	Commit ID to install file from a specific commit.
`stop_on_error`	Stop executing the SQL statements on first error.

Usage Notes

- You can install SQL statements containing nested SQL from a Cloud Code repository file using the following:
  - `@`: includes a SQL file with a relative path to the ROOT of the repository.
  - `@@`: includes a SQL file with a path relative to the current file.
- The scripts are intended as schema install scripts and not as generic SQL scripts:
  - Scripts cannot contain SQL*Plus client specific commands.
  - Scripts cannot contain bind variables or parameterized scripts.
  - SQL statements must be terminated with a slash on a new line (/).
  - Scripts can contain DDL, DML PLSQL statements, but direct `SELECT` statements are not supported. Using SELECT within a PL/SQL block is supported.

  Any SQL statement that can be run using `EXECUTE IMMEDIATE` will work if it does not contain bind variables or defines.

Example

```
BEGIN
     DBMS_CLOUD_REPO.INSTALL_FILE(
         repo          => l_repo,
         file_path     => 'test3.sql',
         stop_on_error => FALSE
     );
END;
/
```

INSTALL_SQL Procedure

This procedure installs SQL statements from a buffer given as input.

Syntax

```
PROCEDURE DBMS_CLOUD_REPO.INSTALL_SQL(
      content           IN  CLOB,
      stop_on_error     IN  BOOLEAN  DEFAULT TRUE
);
```

Parameters

Parameter	Descriptions
`content`	Is the CLOB containing the SQL statements to run.
`stop_on_error`	Stop executing the SQL statements on first error.

Usage Notes

- The scripts are intended as schema install scripts and not as generic SQL scripts:
  - Scripts cannot contain SQL*Plus client specific commands.
  - Scripts cannot contain bind variables or parameterized scripts.
  - SQL statements must be terminated with a slash on a new line (/).
  - Scripts can contain DDL, DML PLSQL statements, but direct `SELECT` statements are not supported. Using SELECT within a PL/SQL block is supported.

  Any SQL statement that can be run using `EXECUTE IMMEDIATE` will work if it does not contain bind variables or defines.

Example

```
BEGIN
  DBMS_CLOUD_REPO.INSTALL_SQL(
      content    => 'create table t1 (x varchar2(30))' || CHR(10) || '/',
      stop_on_error => FALSE
  );
END;
/
```

LIST_BRANCHES Function

This function lists branches in the Cloud Code Repository branch identified by the `repo` handle argument.

Syntax

```
FUNCTION DBMS_CLOUD_REPO.LIST_BRANCHES(
    repo           IN   CLOB
) RETURN list_branch_ret_tab PIPELINED PARALLEL_ENABLE;
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle. This parameter is mandatory and supported for all cloud providers.

Example

```
SELECT * FROM DBMS_CLOUD_REPO.LIST_BRANCHES (repo => l_repo);
```

Usage Notes

- This is a pipelined table function with return type as `list_branch_ret_tab`.
- `DBMS_CLOUD_REPO.LIST_BRANCHES` returns the column: `name`, which indicates the name of the Cloud Code Repository branch.

LIST_COMMITS Function

This function lists commits in the Cloud Code Repository branch identified by the `repo` handle argument.

Syntax

```
FUNCTION DBMS_CLOUD_REPO.LIST_COMMITS(
      repo              IN  CLOB,
      branch_name       IN  VARCHAR2  DEFAULT NULL,
      file_path         IN  VARCHAR2 DEFAULT NULL,
      commit_id         IN  VARCHAR2  DEFAULT NULL
) RETURN list_commit_ret_tab PIPELINED PARALLEL_ENABLE;
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle. This parameter is mandatory and supported for all cloud providers.
`branch_name`	List commits from a specific branch. This parameter is supported for all cloud providers. If you do not supply a `branch_name` value, the `branch_name` is set to `main`.
`file_path`	List files under the specified subfolder path in the repository. This parameter is only supported for Git and Azure cloud providers. If you do not supply a `file_path` value, the `file_path` is set to a NULL value.
`commit_id`	List files starting from the specified `sha/id` This parameter is supported for all cloud providers. If you do not supply a `commit_id` value, the `commit_id` is set to a NULL value.

Example

```
SELECT name FROM DBMS_CLOUD_REPO.LIST_COMMITS(repo => l_repo);
```

Example

```
SELECT name FROM DBMS_CLOUD_REPO.LIST_COMMITS (
    repo         => l_repo,
    commit_id    => '66dd2b23b74cd0afabd11af66c6aa9c550540ba6',
    file_path    => 'sub_dir/test11.sql'
);
```

Usage Notes

- This is a pipelined table function with a return type as `list_commit_ret_tab`.
- `DBMS_CLOUD_REPO.LIST_COMMITS` returns the column: `commit_id`.

LIST_FILES Function

This function downloads a file from Cloud Code repository. Optionally, file content can be accessed from either a specific branch, tag or commit name. By default, the file is accessed from the default repository branch. The results include the file names and additional metadata about the files.

Syntax

```
FUNCTION DBMS_CLOUD_REPO.LIST_FILES(
      repo              IN  CLOB,
      path              IN  VARCHAR2  DEFAULT NULL,
      branch_name       IN  VARCHAR2  DEFAULT NULL,
      tag_name          IN  VARCHAR2  DEFAULT NULL,
      commit_id         IN  VARCHAR2  DEFAULT NULL
) RETURN list_file_ret_tab PIPELINED PARALLEL_ENABLE;
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle.
`path`	List files under the specified subfolder path in the repository.
`branch_name`	List files from a specific branch.
`tag_name`	List files from a specific Tag.
`commit_name`	List files from a specific commit.

Usage Notes

- This is a pipelined table function with return type as `list_file_ret_tab`.
- `DBMS_CLOUD_REPO.LIST_FILES` returns the columns: `id`, `name`, `url`, and `bytes`.

Example

```
SELECT name FROM DBMS_CLOUD_REPO.LIST_FILES(repo => l_repo);

NAME
-------------------------
test3.sql
```

LIST_REPOSITORIES Function

This function lists all Cloud Code Repositories identified by the `repo` handle argument. If the `repo` handle has a repository name specified, the function does not restrict the listing to the specified repository name; it lists all repositories accessible to the user.

Syntax

```
FUNCTION DBMS_CLOUD_REPO.LIST_REPOSITORIES(
      repo                 IN   CLOB
) RETURN list_repo_ret_tab PIPELINED PARALLEL_ENABLE;
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle. This parameter is supported by all cloud providers.

Usage Notes

- This is a pipelined table function with return type as `list_repo_ret_tab`.
- `DBMS_CLOUD_REPO.LIST_REPOSITORIES` returns the columns: `id`, `name`, `owner`, `description`, `private`, `url`, `bytes`, `created`, and `last_modified`.

Example

```
SELECT name description FROM DBMS_CLOUD_REPO.LIST_REPOSITORIES(:repo);

NAME                   DESCRIPTION
---------------------- ---------------
TestRepo1              My test repo
```

MERGE_BRANCH Procedure

This procedure merges a repository branch into another specified branch in the Cloud Code Repository identified by the `repo` handle argument. The `MERGE_BRANCH` procedure is currently not supported in Azure.

Syntax

```
PROCEDURE DBMS_CLOUD_REPO.MERGE_BRANCH (
    repo               IN   CLOB,
    branch_name        IN   VARCHAR2,
    parent_branch_name IN   VARCHAR2 DEFAULT NULL,
    commit_details     IN   CLOB      DEFAULT NULL
 );
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle. This parameter is mandatory and supported for GITHUB and AWS cloud providers.
`branch_name`	Specifies the Git branch name to merge. This parameter is mandatory and supported for all cloud providers.
`target_branch_name`	Specifies the target branch name to merge into. This parameter is mandatory and supported for all cloud providers.
`commit_details`	Commit Details as a JSON document `{"message": "Commit message", "author": {"name": "Committing user name", "email": "Email of committing user" } }` If you do not supply a `commit_details`value, the `commit_details` is set to the default commit message that includes the information about current database session user and database name performing the commit.

Example

```
BEGIN
  DBMS_CLOUD_REPO.MERGE_BRANCH (
    repo               => l_repo,
    branch_name        => 'test_branch',
    target_branch_name => 'main'
    );
END;
/
```

Usage Note

To run the `DBMS_CLOUD_REPO.MERGE_BRANCH` procedure, you must be logged in as the ADMIN user or have `EXECUTE` privilege on `DBMS_CLOUD_REPO`.

PUT_FILE Procedure

This procedure uploads a file to the Cloud Code repository identified by the `repo` handle argument. The procedure is overloaded to support either uploading a file from a directory object or uploading the contents from a BLOB to the repository file.

Syntax

```
PROCEDURE DBMS_CLOUD_REPO.PUT_FILE(
      repo              IN  CLOB,
      file_path         IN  VARCHAR2,
      contents          IN  BLOB,
      branch_name        IN   VARCHAR2   DEFAULT NULL,
      commit_details     IN   CLOB       DEFAULT NULL
);

PROCEDURE DBMS_CLOUD_REPO.PUT_FILE(
      repo               IN   CLOB,
      file_path          IN   VARCHAR2,
      directory_name     IN   VARCHAR2,
      source_file_name   IN   VARCHAR2   DEFAULT NULL,
      branch_name        IN   VARCHAR2   DEFAULT NULL,
      commit_details     IN   CLOB       DEFAULT NULL
);
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle.
`file_path`	File path to upload file in the repository.
`contents`	BLOB containing the file contents.
`directory_name`	Directory object name containing the file name.
`source_file_name`	Source file name to upload to repository.
`branch_name`	Put file to a specific branch.
`commit_details`	Commit Details as a JSON document: `{"message": "Commit message", "author": {"name": "Committing user name", "email": "Email of committing user" } }`

Example

```
BEGIN
  DBMS_CLOUD_REPO.PUT_FILE(
     repo    => l_repo,
  );
END;
/
```

UPDATE_REPOSITORY Procedure

This procedure updates a Cloud Code repository identified by the `repo` handle argument. UPDATE_REPOSITORY supports updating the name, description, or private visibility status, as supported by the Cloud Code repository.

Syntax

```
PROCEDURE DBMS_CLOUD_REPO.UPDATE_REPOSITORY(
      repo                    IN OUT  CLOB,
      new_name                IN      VARCHAR2 DEFAULT NULL,
      description             IN      CLOB     DEFAULT NULL,
      private                 IN      BOOLEAN  DEFAULT NULL
);
```

Parameters

Parameter	Description
`repo`	Specifies the repository handle. This parameter is supported for all cloud providers.
`new_name`	New name for repository. This parameter is supported for all cloud providers.
`description`	A short text description for the repository. This parameter is supported for GITHUB and AWS cloud providers.
`private`	Repository is private and only accessible with valid credentials. This parameter is supported for the GITHUB cloud provider.

Example

```
BEGIN
  DBMS_CLOUD_REPO.UPDATE_REPOSITORY(
     repo         => l_repo,
     new_name     => 'repo2'
  );
END;
/
```