REPSTANCE

Prepare to be Amazed

Repstance User Guide

Version: 5.1.0022

2023

Collabcloud Limited
47 St Pauls Road, Staines-upon-Thames, Surrey, TW18 3HQ, England

TABLE OF CONTENTS

1. ABOUT REPSTANCE

1.1 What it is

Repstance is an entirely new real-time data propagation product for homogeneous and heterogeneous environments that supports various replication topologies. It is available in two versions: “Repstance” and “Repstance Advanced Edition”.

“Repstance” supports *Oracle*, *MS SQL Server* (with the ability to enable CDC) as a Source Database and *Oracle*, *MS SQL Server* as a Target Database.

“Repstance Advanced Edition” is an extended version of “Repstance”, which in additional supports *PostgreSQL*, *Amazon Redshift*, *MySQL*, *Snowflake*, *Amazon Aurora* (*PostgreSQL*, *MySQL*) as a Target Database.

Repstance includes the functionality to support DML and DDL operations, both of which can be automatically included in the replication stream and if desired uses of sophisticated transformation abilities. Fast initial data loading and the ability to synchronize data from any desired timestamp.

Repstance supports Multi-Master replication in all forms of topology with ability the to ignore data produced by itself (loopback control), thus providing "True" Bi/Multi-Directional replication, therefore ensuring all instances are equal and continuously synchronized, any changes made in one node will be propagated (replicated) into the other nodes.

Flexible third-party product integration (APIs, CLI, custom integration). Ability to easily configure and monitor replication process using either Web UI or CLI tool.

1.2 How it works

The Repstance Service is delivered via a “Virtual Machine Image”, which is Linux based instance with preconfigured Repstance software. It is available in the both **AWS** and **Azure** marketplaces. Repstance Server can also be deployed out off the cloud infrastructure. Please reach out Repstance support at support@repstance.com for the details.

Repstance has to have access to both Source and Target Databases, in addition it needs a set of minimal Database Objects to function. This functionality is added during the "Prepare Database" stage (see chapter 7.1 Prepare Source and Target Databases or 8.2.1 Database Configuration in case of using Web UI).

Repstance uses the running Daemon process to "Multi-task" – that is to say, by using "threads" within the Daemon it is capable of running Multiple Capture and Apply Processes concurrently.

Both of the Processes, while they depend on each other to supply or insert the necessary data they, nonetheless, run as independent threads.

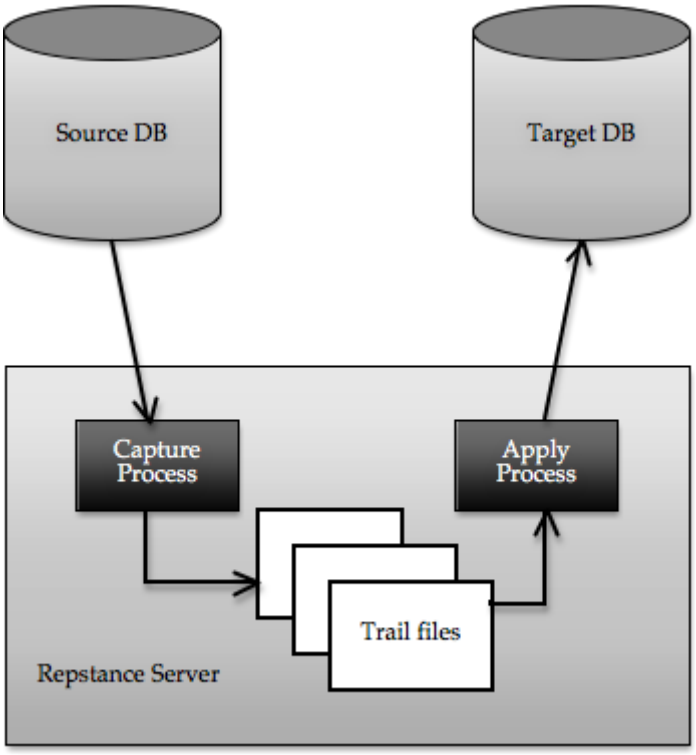
The Capture Process is the means by which the data to be transferred is extracted from the Source Database. It puts this information into locally stored Trail Files, which the Apply Processes can use. The data from a single Capture Process can therefore be used by many Apply Processes and, as a result, this data can therefore be propagated into multiple Target Databases.

The data extracted by the Capture Process is written to these Trail Files in the same sequential order as the transactions occurred in the Source Database, which in turn allows the Apply Process to insert this data into the Target Database in the same order they were executed in the Source Database. This means that both Source and Target Databases will be synchronized, that is to say, in a "Consistent State".

The Capture Process does not need to have a running or configured Apply Process as it will quite simply continue extracting data using the criteria supplied, conversely the Apply Process only needs valid Trail Files from a Capture Process to consume.

Both Capture and Apply Processes insert Checkpoints in the form of LSN/SCN – this in turn means that there is the possibility for Repstance to be restarted from any specified point in the circumstances where any unexpected event occurs that may lead to data loss. This provides a robust form of data security and ensures that the data is, again, always "Consistent".

The diagram below demonstrates the flow of data from Source to Target Databases.


Source DB
Target DB
Capture Process
Apply Process
Trail files
Repstance Server

2. GETTING STARTED WITH REPSTANCE

The Repstance Server is delivered as an AMI paid instance. It is accessible from the **AWS** Marketplace, using the "1 Click" download process here:

https://aws.amazon.com/marketplace/seller-profile?id=90d50a1a-60d0-47ca-898b-90b953f89b34

For the **Azure** the Repstance Server is accessible by the following link:

https://azuremarketplace.microsoft.com/en-us/marketplace/apps?exp=ubp8&search=collabcloud

It requires you as a user to have your own Cloud Account and be subscribed to the Repstance, and then select the instance of sufficient size and processing power to manage your database requirements (see vendor recommended instance size).

During the installation process you must ensure that TCP port "22" is opened for access, as a minimum, and in the case that you want remote access, that port "8797" is also opened. In order to use Web UI port "3000" should be opened as well.

To connect to the operating system, use SSH protocol. In **AWS** the only username that can be used is "**ec2-user**" along with the key that you were provided with during the "launch" of the installation process. See the following link for help:

https://docs.aws.amazon.com/en_us/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html

For the **Azure** instances the user credentials are used, that were provided during the instance deployment process.

Once it is installed in your environment, it is ready to run.

3. INTERACTING WITH A REPSTANCE SERVER

There are three possible ways to engage with a running Repstance Server Instance, they are, either via the *Command Line Interface (CLI)* or via the *Rest API* or via *Web User Interface (Web UI)*.

3.1 Command Line Interface (CLI)

The Repstance Command Line Interface utility (**repcli**) is the interface with the Repstance server, and it is used to configure and manage the server as well as providing a reporting mechanism.

The **repcli** utility can be invoked directly within this environment, and the user can start to configure Processes without the need for any further configuration.

Use SSH protocol to connect to the Repstance Server and run the **repcli** command:

```
$ repcli
$ repcli>help

Commands:
  alter         This command is used to change both the Capture and Apply
process parameters

  clear         Clear The Screen

  exit          Exit The Program

  help          Display Help
```

etc.

The **repcli** utility can be configured to communicate with remote Repstance Server using the HTTPS protocol (see chapter 3.1.3 Configure repcli for Remote Access).

The **repcli** utility can operate in two modes either "Command Mode" or "Fully Interactive Mode". The two different ways of using Repstance have been developed to provide alternate methods of controlling the product.

If **repcli** is executed without any parameters it starts functioning in "Fully Interactive Mode", by default, which means that it will wait for an input command, until user issues the **"exit"** command at which point it returns to the console.

The "Fully Interactive Mode" is useful for a user to construct and/or configure Repstance, as it provides, at each stage of the process construction, a list of the possible commands and their applicable parameters, see the "Visuals" below.

"Command mode" is more likely to be used when there is a need to "Embed" Repstance functions into a deployment Script for Automation processes within the user's own environment.

3.1.1 Fully Interactive Mode

This is the utility's default behavior. As the user starts to provide a command for any Repstance Process and then "presses" the "Tab" key, **repcli** utility will supply the user with the expected possible parameters.

The following example shows the **repcli** interface displaying the list of all possible commands:

```
repcli>
status    exit    help    prepare    remove    alter    reset    clear    validate
show    run    stop
```

This example shows all commands starting with "s":

```
repcli>s
status    show    stop
```

And the next one shows all the possible parameters for the "**alter**" command:

```
repcli>alter process=capture id=1
server=    port=    user=    password=    dbname=    name=    autostart=
debuglevel=    ddlinclude=    ddlexclude=    dmlinclude=    dmlexclude=    map=
```

The **repcli** has built-in help facility, which can be invoked to provide detailed "Help" for each of the commands, and is shown below:

```
repcli>alter help

This command is used to change the Capture or Apply process parameters.

The Syntax is:

      alter process=capture|apply id=processID [parameter=value, ...]

Note: The 'id' parameter can not be changed.
      The 'name' parameter for a Capture process as well as the 'capturename'
parameter for an Apply process can be changed only in the case that it has
not been provided before - eg. it must be "Unique".
```

```
The process has to exist otherwise the command will fail.

To see available possible parameters, start to use the command.
```

3.1.2 Command Mode

This is the basic interface format, when used in this mode it will simply supply the result of the process defined and return to the console prompt:

```
$repcli status process=capture

Capture Process 1, Name: czdt, DB: zdt, Server: source.repstance.com
  RUN (In Process), Waiting for Transactions
  Last Source DB Change : 2019/02/15 15:47:00.410 UTC (0x000006830000319C0004)
  Last Captured DB Change  :  2019/02/15 15:47:00.410 UTC (0x000006830000319C0004)
Total since  2019/02/25 13:30:00.973 UTC (0x00006830000025750004):
  Processing the Transactions :  00:00:58.505
  Waiting for Transactions    :  00:45:22.859
  Average Speed (ops)     :  4.683
  Transactions: 274 (DDL: 0,  Delete: 137,  Insert: 137,  Update: 0)

Capture Process 2, Name: newcap, DB: mydb, Server: local
  RUN (Failed), Waiting for the Next Command, [lookup local: no such host]
$
```

3.1.3 Configure repcli for Remote Access

The **repcli** tool can be used as standalone application to manage remote Repstance Server through the HTTPS protocol.

The tool can be found in the following directories, once the product is installed:

- `/opt/repstance/cli/macos` – For MacOS Operating System
- `/opt/repstance/cli/win` – For Windows
- `/opt/repstance/cli/linux` – For Linux

Or they can be downloaded from here:

- https://repstance.com/download/cli/macos/repcli.zip
- https://repstance.com/download/cli/win/repcli.zip
- https://repstance.com/download/cli/linux/repcli.zip

These remote clients are only able to connect using the HTTPS protocol and are supplied with the following default configuration:

- HTTPS port is 8797
- For HTTPS initial use a Self-Signed Certificate is provided

- HTTPS authorisation "Token" in the form of the Repstance AWS Instance ID

In order to use the **repcli** utility to communicate with the remote Repstance Server it has to be configured using the **repcli** command "**configure**".

The following example demonstrates the command syntax:

```
$ repcli configure url=https://hostname:8797/ token=aws-instance-id
verifyhttps=0
```

Note – The **`verifyhttps`** parameter should be set to `0` to avoid the Self-signed certificate error.

As the product is initially supplied with a Self-Signed Certificate it is highly recommended that the user replaces this with a security certificate of the required Security level themselves. The new certificate and key needs to be supplied to the Repstance Server and the appropriate entries in the `/opt/repstance/conf/repstance.conf` file need to be edited to reflect these changes:

- **`key`** – path to the key
- **`crt`** – path to the certificate

Using this method, the default HTTPS port can also be changed by modifying the **`port`** parameter.

The default token used to access the server can also be changed by modifying the **`token`** parameter in the same file.

If the user wishes to prevent any external access, the **`port`** parameter should be set to value `0`.

3.2 Web User Interface (Web UI)

Repstance Instance is delivered with Web UI Server, which allows to configure, manage and monitor Repstance Processes using a Web browser.

By default, Web UI Server is configured to use HTTPS protocol through the ports number 3000 and 8797, so these ports must be accessible from the client machine.

The Web UI Server is initially supplied with a Self-Signed Certificate it is highly recommended that the user replaces this with a security certificate of the required Security level themselves. The new certificate and key needs to be supplied to the Repstance Server and the appropriate entries in the `/opt/repstance/conf/repstance.conf` file need to be edited to reflect these changes:

- **`key`** – path to the key
- **`crt`** – path to the certificate

Using this method, the default HTTPS port can also be changed by modifying the **`port`** parameter.

After successful deployment of Repstance instance find its Public DNS. If the Public DNS is not configured, use the Private IP address.

To enter Repstance Web UI, use your web browser to access to the following address:

```
https://<Public DNS or Private IP>:3000
```

Use "repstance" as username. The default password in the AWS environment is the **Instance ID** and the **VM Id** in the Azure cloud. The password can also be found and changed in the `/opt/repstance/conf/repstance.conf` file (see "token" section).

3.3 REST API

This is the detailed command structure for communicating with Repstance Services, it sends an HTTP/HTTPS request in the "JSON Format" to the services and adheres to the established protocols for the JSON format. List of the available Repstance commands and their specification is provided in chapter 7. Commands to be used.

The following example demonstrates how to use REST API to call "**status"** command:

```
$curl --header "Content-Type: application/json" \
 --request POST \
 --data '{"command": "status"}' http://localhost:8796/control/process
```

4. SUPPORTED DATABASE REQUIREMENTS

4.1 MS SQL Server Database

Currently the product supports versions of MS SQL Server that permit the use of CDC (Change Data Capture) as a **Source Database** with the following limitations:

- System databases are not supported,
- No Contained databases are supported,
- No databases enabled with In-Memory Optimization (2014/2016 feature) are currently supported,
- Database Compatibility level must be 100 or higher,
- Asynchronous AlwaysOn databases are not supported,
- The SQL Server Agent must be running.

Any SQL Server databases starting 2008, including RDS and Azure SQL Server instances, are supported as a **Target Database**.

The *RDS* and *Azure SQL Server* instances are supported for **Source** and/or **Target Databases**.

4.2 Oracle Database

Currently Repstance supports Oracle versions 10g through 19c including RAC configuration as **Source** and/or **Target Database**. Both *EC2* and *RDS* instances are supported.

The Oracle Source Database must be configured to run in ARCHIVELOG mode prior to use. It is recommended practice to ensure that the archive logs are not "cleaned-up" until they are processed by the Repstance Server. At least the "minimal supplemental logging" needs to be enabled either prior to use or as a part of "**prepare database**" command.

4.3 MySQL Database

Currently the product supports MySQL as a **Target Database** only. The following versions are supported:

- MySQL versions 5.5-5.7
- MySQL 8.0

- Aurora MySQL
- MariaDB versions 10.0 – 10.3

4.4 PostgreSQL Database

Currently the product supports PostgreSQL as a **Target Database only**. The following versions are supported:

- PostgreSQL versions 10.1-11.6
- Aurora PostgreSQL compatible with PostgreSQL 10.x.

4.5 Redshift Database

Currently the product supports Redshift as a **Target Database only**.

4.6 Snowflake Database

Currently the product supports Snowflake as a **Target Database only**.

5. HOW TO USE REPSTANCE

To replicate data Repstance uses Capture and Apply Processes. The Capture Process extracts data from the Source Database and puts it into locally stored Trail Files in the same sequential order as the transactions occurred in the database. The Trial Files are consumed by Apply Process to insert the captured data into the Target Database. The Apply Process can also modify data or/and data definition if the appropriable transformation rules are configured.

Before running the Capture and Apply Processes both Source and Target Databases need to be prepared by meaning that the minimal set of the Repstance's objects must be installed. Repstance has built-in functionality that allows it to prepare databases by simply running the CLI or REST API command (see chapter 7.1 Prepare Source and Target Databases) or via Web UI (see chapter 8.2.1 Database Configuration).

Once it is done, we recommend validate both databases by running the "**validate**" command (see chapter 7.3 Validate Source and Target Database if using CLI or REST API, or chapter 8.2.1 Database Configuration for Web UI) to make sure that the necessary functionality is installed.

The next step is to configure the replication processes.

This is to be done in two parts, the first is to configure the Capture Process, which is extracting the data from the specified Source Database, and the second is to configure the corresponding Apply Process, which will insert the "captured" data into the specified Target Database. The both Processes can be configured either using CLI or REST API (see chapter 7.4 Prepare Capture Process and 7.9 Prepare Apply Process) or through the Web UI (see chapter 8.2.2 Processes Configuration).

The data to be replicated will be defined during the Capture Process configuration. Validation of correctly constructed Capture or Apply Processes can be checked using the "**validate**" command. See chapters 7.7 Validate Capture Process, 7.12 Validate Apply Process or 8.2.2 Process Configuration for the details.

For MS SQL Server the Capture Process will enable CDC on the specified tables only the first time it is run. If the Capture Process is "altered" at any point, during the next "run" it will disable CDC on the unwanted tables and initialize CDC on any new tables that are specified.

The first time Capture Process is run, is the first point and the only point at which data will start to be captured by this Process.

Assuming that the Capture Process has been properly configured and run, only at this point it will start to produce Trail Files, which can be used by Apply Process.

In order for an Apply Process to succeed, of necessity the Trail Files of the corresponding Capture Process must exist.

In order to ensure that an Apply Process will start from the specified LSN/SCN, this LSN/SCN must exist in the Trail Files and the "**reset"** command (see chapter 7.13 Reset Apply Process) can be used to point this Apply Process at the required LSN/SCN. If the Apply Process has not been reset, it will start processing data from the first available transaction found in the Trail File.

Once the Apply Process has been configured to use and the proper LSN/SCN (if needed) has been specified, the Apply Process can be run (see chapter 7.15.1 Run any Capture or Apply Processes and 8.2.2 Process Configuration).

At the point when both the Capture and Apply Processes are running, the "**status**" command (see chapter 7.15.3 Status of any Capture or Apply Process and 8.2.2.4 Processes Monitoring and Maintenance) can be used to monitor them.

In the event it is necessary to "stop" any currently running processes use the "**stop**" command (see chapter 7.15.2 Stop any Capture or Apply Processes and 8.2.2.4 Processes Monitoring and Maintenance).

The following example demonstrates how to setup simple DML and DDL replications between two MS SQL Server databases for all objects located in the **dbo** schema using **repcli** tool:

```
repcli>prepare database=source dbtype=mssql server=source.repstance.com port=1433 user=ds password=secret dbname=sdemo
[Completed]: Database [source.repstance.com:1433,sdemo] has been prepared

repcli>prepare database=target dbtype=mssql dbtype=mssql server=target.repstance.com port=1433 user=ds password=secret dbname=tdemo
[Completed]: Database [target.repstance.com:1433,tdemo] has been prepared

repcli>validate database=source dbtype=mssql server=source.repstance.com port=1433 user=ds password=secret dbname=sdemo
[Valid]: Database SDEMO is valid
```

```
repcli>validate database=target dbtype=mssql server=target.repstance.com port=1433 user=ds password=secret dbname=tdemo
[Valid]: Database TDEMO is valid

repcli>prepare process=capture dbtype=mssql id=1 name=rep1 server=source.repstance.com port=1433 user=ds password=secret dbname=sdemo dmlinclude=dbo.% ddlinclude=dbo.%
[Completed]: Capture 1 has been added

repcli>validate process=capture id=1
[Valid]: Capture process 1 is valid

repcli>run process=capture id=1
[In Process]: Command [RUN] has been sent. Use [status] command to monitor the Capture 1 process

repcli>prepare process=apply id=1 dbtype=mssql capturename=rep1 server=target.repstance.com port=1433 user=ds password=secret dbname=tdemo repuser=1
[Completed]: Apply 1 has been added

repcli>validate process=apply id=1
[Valid]: Apply process 1 is valid

repcli>run process=apply id=1
[In Process]: Command [RUN] has been sent. Use [status] command to monitor the Apply 1 process

repcli>status
Capture Process 1, Name: rep1, DB: sdemo, Server: source.repstance.com
  RUN (In Process), Waiting for Transactions
    Last Source DB Change : 2019/02/25 21:13:00.173 UTC (0x00000044000002AE0004)
    Last Captured DB Change : 2019/02/25 21:13:00.173 UTC (0x00000044000002AE0004)
  Total since  2019/02/25 21:10:00.983 UTC (0x000000440000020A0004):
    Processing the Transactions :  00:00:05.289
    Waiting for Transactions    :  00:02:44.396
    Average Speed (ops)  :  1.134
    Transactions: 6 (DDL: 0,  Delete: 3,  Insert: 3,  Update: 0)

Apply Process 1, Transactions Provider (Capture): rep1, DB: tdemo, Server: target.repstance.com
  RUN (In Process), Waiting for Transactions
     Last Source DB Change  : 2019/02/25 21:13:00.173 UTC (0x00000044000002AE0004)
     Last Applied DB Change : 2019/02/25 21:13:00.173 UTC (0x00000044000002AE0004)
     Lag                    :  00:00:00.000
  Total since  2019/02/25 21:13:00.150 UTC (0x00000000000000000001):
    Processing the Transactions :  00:00:02.065
    Waiting for Transactions    :  00:00:08.284
    Average Speed (ops)  :  2.905
    Transactions: 6 (DDL: 0,  Delete: 3,  Insert: 3,  Update: 0)
```

6. HOW TO REMOVE REPSTANCE

The steps involved in removing Repstance are detailed below.

6.1 How to remove Apply Process

To remove an Apply Process use the "**remove**" command (see chapter 7.14 Remove Apply Process). The details on how to remove Apply Process through the Web UI are provided in chapter 8.2.2.3 Modify, Delete and Validate Process.

After using this command, it will remove **all** information about the Apply Process, namely configuration files, and the checkpoint details, together with historical information. If the Target Database **is** reachable, the "**remove**" command will clean-up all the checkpoint information on the Target Database.

As the "**remove"** command can only accept a single "Apply parameter", this action can only be used for one Apply Process at a time.

6.2 How to remove Capture Process

To remove a Capture Process use the "**remove**" command (see chapter 7.8 Remove Capture Process). The details on how to remove Capture Process through the Web UI are provided in chapter 8.2.2.3 Modify, Delete and Validate Processes.

After using this command, it will remove **all** information about the Capture Process, namely configuration files, checkpoint details, and the Trail Files. If the Source Database is MS SQL Server and it **is** reachable the "**remove**" command will disable CDC on the tables that were used by this Capture Process only.

As the "**remove**" command can only accept a single "Capture parameter", this action can only be used for one Capture Process at a time.

Note – If there are still any Apply Processes using Trail Files from this Capture Process, they will fail.

The steps involved in removing Repstance Server Database Objects are detailed below.

6.3 Remove Target Database Objects

To remove Target Database Objects use the "**remove**" command (see chapter 7.2 Remove Repstance Database Objects) or Web UI (see chapter 8.2.1 Database Configuration).

After using this command, it will remove all the Data Objects that were required by the Apply Process. If there are still Apply Processes inserting data, using these objects they will immediately fail.

Note – It is strongly recommended that before running this command you run the "**status**" command and make a record the LSN/SCN in the "Last Applied DB Change" section for each Apply Process running on this database, in this way it will be possible to reverse this action for this database at a later point in time, assuming that the correct Trail Files still exist and can be accessed.

6.4 Remove Source Database Objects

To remove Source Database Objects use the "**remove**" command (see chapter 7.2 Remove Repstance Database Objects) or Web UI (see chapter 8.2.1 Database Configuration).

Note – When using this command it MUST be understood that there will be no way to "Recover" from its effects.

After using this command, it will remove all the Data Objects that were required by the Capture Processes. If there are still Capture Processes extracting data, using these objects they will immediately fail. For MS SQL Server it will disable CDC on this database. At this point Repstance will no longer be able to record or extract data from this database.

7. COMMANDS TO BE USED

7.1 Prepare Source and Target Databases

Both Source and Target Databases must be prepared for replication before running any replication processes. The "**prepare**" database command is used to configure databases. The command must be executed on both Source and Target Databases.

7.1.1 Prepare MS SQL Server Database as Source Database

The "**prepare**" database command is used to configure MS SQL Server database in order to use it as a Source. It enables CDC and creates the Database Objects necessary for any Capture Processes.

Note – In order to use the "**prepare**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://***repstance url***/configure/database
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command":"prepare",
   "parameters":[
      ["database","source"],
      ["dbtype","mssql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"],
      ["housekeeping","0|1|2"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body:**

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=source dbtype=mssql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName housekeeping={0|1|2}
```

The input parameters are:

- **database** – Database role, the only possible value is *source* – to insert Capture Process objects and enable CDC
- **dbtype** – Type of the Source Database, the only acceptable value is *mssql*
- **server** – Host name or IP address of the database server
- **port** – Database port number
- **dbname** – Database name to be connected to
- **user** – Database User name
- **password** – Database User password
- **housekeeping** – This parameter determines how the CDC data will be cleaned up. The possible values are:
  - ➢ *0* – data will be cleaned up by an SQL Server job only,
  - ➢ *1* – install Repstance job to clean up data only after extraction and keep SQL Server cleanup job, which is a time based CDC setting,
  - ➢ *2* – install Repstance job and remove SQL Server cleanup job, data will be cleaned up only by the Repstance job.

7.1.2 Prepare Oracle Database as Source Database

The "**prepare**" database command is used to configure Oracle database in order to use it as a Source. It enables the necessary level of supplemental logging and creates the database objects necessary for any Capture Processes.

Note – In order to use the "**prepare**" database command the database user must have sufficient privileges.

There are two possible connection types – *EZCONNECT* **and** *TNS*. **Depending on which one is chosen, there will be a different set of possible parameters.**

REST API:

- **Endpoint:** https://***repstance url***/configure/database
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command":"prepare",
   "parameters":[
      ["database","source"],
      ["dbtype","oracle"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"],
      ["createlogdirs","0|1"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=source dbtype=oracle \
      connectiontype=tns|ezconnect tnsname=tns_alias \
      server=databaseHost port=databasePort \
      servicename=service_or_SID \
      user=username password=password \
      dbname=databaseName createlogdirs=0|1
```

The input parameters are:

- **database** – Database role, the possible value is *source* – to insert Capture Process Objects
- **dbtype** – Type of RDBMS, the only possible value is *oracle*
- **connectiontype** – Specifies the Oracle connection method, the possible values are:

    - ➢ *tns* – Local Naming Method to be used
    - ➢ *ezconnect* – EZCONNECT to be used
- **tnsname** – Name of the TNS alias, valid **only** if `connectiontype=tns`
- **server** – Host name or IP address of the database server, valid **only** if `connectiontype=ezconnect`
- **port** – Database port number, valid **only** if `connectiontype=ezconnect`
- **servicename** – Database service name or SID, valid **only** if `connectiontype=ezconnect`
- **dbname** – Name of either container or pluggable database, valid **only** if `database=source` and for any Oracle versions 12c-19c but excluding RDS instances
- **user** – Database User name
- **password** – Database User password
- **createlogdirs** – The parameter determines if the "ONLINELOG_DIR" and "ARCHIVELOG_DIR" Oracle directories need to be created. The possible values are:
    - ➢ *0* – do not run create the directories (default value),
    - ➢ *1* – create the directories.

Note – The directories are used if the Capture Process is configured to extract changes in the "DirectLog" mode (see chapter 7.4.2 Prepare Capture Process for Oracle Database for the details).

7.1.3 Prepare MS SQL Server Database as Target Database

The "**prepare**" database command is used to configure MS SQL Server database as a Target. The command must be executed before running Apply Process. It creates the necessary database objects for any Apply Processes.

Note – In order to use the "**prepare**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://***repstance url***/configure/database
- **Method:** *POST*
- **Header:**
    - ➢ Content-Type: *application/json*
    - ➢ X-Token: *token*
- **Body:**

```
{
   "command":"prepare",
   "parameters":[
      ["database","target"],
      ["dbtype","mssql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=target dbtype=mssql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role, the only possible value is *target* – to insert Apply Process Objects
- **dbtype** – Type of RDBMS, the only appropriate value is *mssql*
- **server** – Host name or IP address of the database server
- **port** – Database port number
- **dbname** – Database name to be connected to
- **user** – Database User name
- **password** – Database User password

7.1.4 Prepare Oracle Database as Target Database

The "**prepare**" database command is used to configure Oracle database as a Target. The command must be executed before running Apply Process. It creates the necessary database objects for any Apply Processes.

Note – In order to use the "**prepare**" database command the database user must have sufficient privileges.

There are two possible connection types – *EZCONNECT* and *TNS*. Depending on which one is chosen, there will be a different set of possible parameters.

REST API:

- **Endpoint:** https://***repstance url***/configure/database
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command":"prepare",
   "parameters":[
      ["database","target"],
      ["dbtype","oracle"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["user","db_user_name"],
      ["password","db_user_password"],
      ["tablespace","user_tablespace"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, the possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=target dbtype=oracle \
      connectiontype=tns|ezconnect tnsname=tns_alias \
      server=databaseHost port=databasePort \
      servicename=service_or_SID \
      user=username password=password \
      tablespace=user_tablespace
```

The input parameters are:

- **database** – Database role, the only possible value is *target* – to insert Apply Process Objects
- **dbtype** – Type of RDBMS, the only possible value is *oracle*
- **connectiontype** – Specifies the Oracle connection method, the possible values are:
  - ➢ *tns* – Local Naming Method to be used
  - ➢ *ezconnect* – EZCONNECT to be used
- **tnsname** – Name of the TNS alias, valid **only** if connectiontype=*tns*
- **server** – Host name or IP address of the database server, valid **only** if connectiontype=*ezconnect*
- **port** – Database port number, valid **only** if connectiontype=*ezconnect*
- **servicename** – Database service name or SID, valid **only** if connectiontype=*ezconnect*
- **user** – Database User name
- **password** – Database User password
- **tablespace** – Name of the tablespace that the Repstance's objects are to be installed in. The default is the *USERS* tablespace if no alternative has been specified.

7.1.5 Prepare PostgreSQL and Aurora PostgreSQL Databases as Target Database

The "**prepare**" database command is used to configure PostgreSQL and Aurora PostgreSQL databases as a Target. The command must be executed before running Apply Process. It creates the necessary database objects for any Apply Processes.

Note – In order to use the "**prepare**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://***repstance_url***/configure/database
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command":"prepare",
   "parameters":[
      ["database","target"],
```

```
        ["dbtype","postgresql"],
        ["server","host_name"],
        ["port","port_number"],
        ["dbname","database_name"],
        ["user","db_user_name"],
        ["password","db_user_password"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=target dbtype=postgresql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role, the only possible value is *target* – to insert Apply Process Objects
- **dbtype** – Type of RDBMS, the only possible value is *postgresql*
- **server** – Host name or IP address of the database server
- **port** – Database port number
- **dbname** – Database name to be connected to
- **user** – Database User name
- **password** – Database User password

7.1.6 Prepare Redshift Database as Target Database

The "**prepare**" database command is used to configure Redshift database as a Target. The command must be executed before running Apply Process. It creates the necessary database objects for any Apply Processes.

Note – In order to use the "**prepare**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://***repstance_url***/configure/database
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command":"prepare",
   "parameters":[
      ["database","target"],
      ["dbtype","redshift"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=target dbtype=redshift \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role, the only possible value is *target* – to insert Apply Process Objects
- **dbtype** – Type of RDBMS, the only possible value is *redshift*
- **server** – Host name or IP address of the database server

- **port** – Database port number
- **dbname** – Database name to be connected to
- **user** – Database User name
- **password** – Database User password

7.1.7 Prepare MySQL and Aurora MySQL Databases as Target Database

The "**prepare**" database command is used to configure MySQL, MariaDB and Aurora MySQL databases as a Target. The command must be executed before running Apply Process. It creates the necessary database objects for any Apply Processes.

Note – In order to use the "**prepare**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://***repstance_url***/configure/database
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command":"prepare",
   "parameters":[
      ["database","target"],
      ["dbtype","mysql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=target dbtype=mysql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role, the possible value is *target* – to insert Apply Process Objects
- **dbtype** – Type of RDBMS, the possible value is *mysql*
- **server** – Host name or IP address of the database server
- **port** – Database port number
- **dbname** – Database name to be connected to
- **user** – Database User name
- **password** – Database User password

7.1.8 Prepare Snowflake Database as Target Database

The "**prepare**" database command is used to configure Snowflake database as a Target. The command must be executed before running Apply Process. It creates the necessary database objects for any Apply Processes.

Note – In order to use the "**prepare**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://***repstance_url***/configure/database
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command":"prepare",
   "parameters":[
      ["database","target"],
      ["dbtype","snowflake"],
      ["account","account"],
      ["region","region"],
      ["warehouse","warehouse"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"]
   ]
```

```
}
```

Server response:

- **`HTTP Status`** – status of the command, possible codes are:
  - ➢ *`200`* – if no error occurs
  - ➢ *`422`* – if error occurs
- **`Body`**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

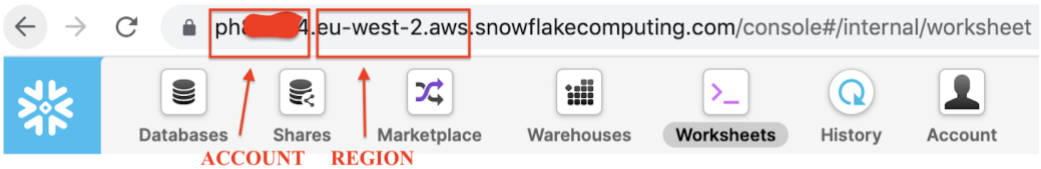
CLI Syntax:

```
repcli prepare database=target dbtype=snowflake \
      account=account region=region warehouse=warehouse \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **`database`** – Database role, the only possible value is *`target`* – to insert Apply Process Objects
- **`dbtype`** – Type of RDBMS, the correct value is *`snowflake`*
- **`account`** – Snowflake Account name
- **`region`** – Snowflake Region name
- **`warehouse`** – Warehouse name
- **`dbname`** – Database name to be connected to
- **`user`** – Database User name
- **`password`** – Database User password

Note – Account name and Region can be easily taken from the Snowflake URL:



7.2 Remove Repstance Database Objects

The "**remove**" database objects command is used to remove Repstance database replication objects.

The successful result of using this command will be:

- ❑ For the Source and Target Database, it will remove the Repstance Database Objects that were created by the "**prepare**" command;
- ❑ For MS SQL Server Database configured as a Source, it will also disable CDC on the tables used by Repstance.

After successful completion of the command any remaining running Processes will fail.

7.2.1 Remove Repstance Database Objects in MS SQL Server

Note – In order to use the "**remove**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://***repstance url***/configure/database
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command":"remove",
   "parameters":[
      ["database","source|target"],
      ["dbtype","mssql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
   "Status":"{Completed|Failed}",
   "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli remove database=source|target dbtype=mssql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role, the possible values are:
  - *source* – to remove Capture Process Objects and disable CDC
  - *target* – to remove Apply Process Objects
- **dbtype** – Type of RDBMS, the possible value is *mssql*
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password

7.2.2 Remove Repstance Database Objects in Oracle

Note – In order to use the "**remove**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://***repstance url***/configure/database
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command":"remove",
   "parameters":[
      ["database","source|target"],
      ["dbtype","oracle"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
     ]
}
```

Server response:

- **`HTTP Status`** – status of the command, possible codes are:
  - ➢ *`200`* – if no error occurs
  - ➢ *`422`* – if error occurs
- **`Body`**:

```
{
   "Status":"{Completed|Failed}",
   "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli remove database=source|target dbtype=oracle \
      connectiontype=tns|ezconnect tnsname=tns_alias \
      server=databaseHost port=databasePort \
      servicename=service_or_SID \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **`database`** – Database role, the possible values are:
  - ➢ *`source`* – to remove Capture Process Objects
  - ➢ *`target`* – to remove Apply Process Objects
- **`dbtype`** – Type of RDBMS, the only possible value is *`oracle`*
- **`connectiontype`** – Specifies the method of connection to Oracle, possible values are:
  - ➢ *`tns`* – Local Naming Method to be used
  - ➢ *`ezconnect`* – EZCONNECT to be used
- **`tnsname`** – Name of the TNS alias, valid **only** if `connectiontype=tns`
- **`server`** – Host name or IP address of the database server, valid **only** if `connectiontype=ezconnect`
- **`port`** – Database port number, valid **only** if `connectiontype=ezconnect`
- **`servicename`** – Database service name or SID, valid **only** if `connectiontype=ezconnect`
- **`dbname`** – Name of either container or pluggable database. Valid **only** if `database=source` and for any Oracle versions 12c-19c but excluding RDS instances
- **`user`** – Database User name
- **`password`** – Database User password

7.2.3 Remove Repstance Database Objects in PostgreSQL and Aurora PostgreSQL Databases

Note – In order to use the "**remove**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://***repstance_url***/configure/database
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command":"remove",
   "parameters":[
      ["database","target"],
      ["dbtype","postgresql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
   "Status":"{Completed|Failed}",
   "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli remove database=target dbtype=postgresql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role, the only possible value is *target* – to remove Apply Process Objects
- **dbtype** – Type of RDBMS, the possible value is *postgresql*
- **server** – Host name or IP address of the database server

- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password

7.2.4 Remove Repstance Database Objects in Redshift

Note – In order to use the "**remove**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://***repstance_url***/configure/database
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command":"remove",
   "parameters":[
      ["database","target"],
      ["dbtype","redshift"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
   "Status":"{Completed|Failed}",
   "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli remove database=target dbtype=redshift \
     server=databaseHost port=databasePort \
     user=username password=password \
```

dbname=*databaseName*

The input parameters are:

- **database** – Database role, the only possible value is *target* – to remove Apply Process Objects
- **dbtype** – Type of RDBMS, the possible value is *redshift*
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password

7.2.5 Remove Repstance Database Objects in MySQL and Aurora MySQL Databases

Note – In order to use the "**remove**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://***repstance url***/configure/database
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
    "command":"remove",
    "parameters":[
        ["database","target"],
        ["dbtype","mysql"],
        ["server","host_name"],
        ["port","port_number"],
        ["dbname","database_name"],
        ["user","db_user_name"],
        ["password","db_user_password"]
     ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs

- **Body**:

```
{
   "Status":"{Completed|Failed}",
   "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli remove database=target dbtype=mysql \
     server=databaseHost port=databasePort \
     user=username password=password \
     dbname=databaseName
```

The input parameters are:

- **database** – Database role, the only possible value is *target* – to remove Apply Process Objects
- **dbtype** – Type of RDBMS, the possible value is *mysql*
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password

7.2.6 Remove Repstance Database Objects in Snowflake

Note – In order to use the "**remove**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://***repstance url***/configure/database
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command":"remove",
   "parameters":[
      ["database","target"],
      ["dbtype","snowflake"],
      ["account","account"],
      ["region","region"],
      ["warehouse","warehouse"],
      ["user","username"],
      ["password","password"],
```

```
        ["dbname","databaseName"]
      ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
   "Status":"{Completed|Failed}",
   "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli remove database=target dbtype=snowflake \
      account=account region=region warehouse=warehouse \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role, the possible value is *target* – to remove Apply Process Objects
- **dbtype** – Type of RDBMS, the possible value is *snowflake*
- **account** – Snowflake Account name
- **region** – Snowflake Region name
- **warehouse** – Warehouse name
- **user** – Database User name
- **password** – Database User password
- **dbname** – Database name

How to find "Snowflake Account" and "Snowflake Region" is described in chapter 7.1.8 Prepare Snowflake Database as Target Database.

7.3 Validate Source and Target Databases

The "**validate**" database objects command is used to validate that the correctly initialized Database Objects exist, and Database is ready for replication.

It is considered to be "Good Practice" to run this command after either the initial configuration has been completed or if changes have been made to either a Capture or Apply Process.

7.3.1 Validate MS SQL Server Database

REST API:

- **Endpoint:** https://***repstance url***/configure/database
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command":"validate",
   "parameters":[
      ["database","source|target"],
      ["dbtype","mssql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli validate database=source|target dbtype=mssql \
     server=databaseHost port=databasePort \
     user=username password=password \
     dbname=databaseName
```

The input parameters are:

- **database** – Database role, the possible values are:
  - *source* – to validate Capture Process Objects

    - ➢ *target* – to validate Apply Process Objects
- **dbtype** – Type of RDBMS, the possible value is *mssql*
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password

7.3.2 Validate Oracle Database

REST API:

- **Endpoint:** https://***repstance url***/configure/database
- **Method:** *POST*
- **Header:**
    - ➢ Content-Type: *application/json*
    - ➢ X-Token: *token*
- **Body:**

```
{
   "command":"validate",
   "parameters":[
      ["database","source|target"],
      ["dbtype","oracle"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
    - ➢ *200* – if no error occurs
    - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli validate database=source|target dbtype=oracle \
     connectiontype=tns|ezconnect tnsname=tns_alias \
```

```
server=databaseHost port=databasePort \
servicename=service_or_SID dbname=databaseName \
user=username password=password \
```

The input parameters are:

- **database** – Database role, the possible values are:
  - *source* – to validate Capture Process Objects
  - *target* – to validate Apply Process Objects
- **dbtype** – Type of RDBMS, the possible value is *oracle*
- **connectiontype** – Specifies connection method to Oracle, possible values are:
  - *tns* – Local Naming Method to be used
  - *ezconnect* – EZCONNECT to be used
- **tnsname** – Name of the TNS alias, valid **only** if connectiontype=*tns*
- **server** – Host name or IP address of the database server, valid **only** if connectiontype=*ezconnect*
- **port** – Database port number, valid **only** if connectiontype=*ezconnect*
- **servicename** – Database service name or SID, valid **only** if connectiontype=*ezconnect*
- **dbname** – Name of either container or pluggable database. Valid **only** if database=*source* and for any Oracle versions 12c-19c but excluding RDS instances
- **user** – Database User name
- **password** – Database User password

7.3.3 Validate PostgreSQL and Aurora PostgreSQL Databases

REST API:

- **Endpoint:** https://***repstance_url***/configure/database
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command":"validate",
   "parameters":[
      ["database","target"],
      ["dbtype","postgresql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
   ]
```

```
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli validate database=target dbtype=postgresql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role, the possible value is *target* – to validate Apply Process Objects
- **dbtype** – Type of RDBMS, the value is *postgresql*
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password

7.3.4 Validate Redshift Database

REST API:

- **Endpoint:** https://***repstance url***/configure/database
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command":"validate",
   "parameters":[
      ["database","target"],
      ["dbtype","redshift"],
      ["server","host_name"],
      ["port","port_number"],
```

```
        ["dbname","database_name"],
        ["user","db_user_name"],
        ["password","db_user_password"]
    ]
}
```

Server response:

- **`HTTP Status`** – status of the command, possible codes are:
  - ➢ *`200`* – if no error occurs
  - ➢ *`422`* – if error occurs
- **`Body`**:

```
{
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli validate database=target dbtype=redshift \
     server=databaseHost port=databasePort \
     user=username password=password \
     dbname=databaseName
```

The input parameters are:

- **`database`** – Database role, the possible value is *`target`* – to validate Apply Process Objects
- **`dbtype`** – Type of RDBMS, the value is *`redshift`*
- **`server`** – Host name or IP address of the database server
- **`port`** – Database port
- **`dbname`** – Database name
- **`user`** – Database User name
- **`password`** – Database User password

7.3.5 Validate MySQL Database and Aurora MySQL Databases

REST API:

- **`Endpoint:`** https://***repstance_url***/configure/database
- **`Method:`** *`POST`*
- **`Header:`**
  - ➢ `Content-Type:` *`application/json`*
  - ➢ `X-Token:` *`token`*
- **`Body:`**

```
{
   "command":"validate",
   "parameters":[
```

```
        ["database","target"],
        ["dbtype","mysql"],
        ["server","host_name"],
        ["port","port_number"],
        ["dbname","database_name"],
        ["user","db_user_name"],
        ["password","db_user_password"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli validate database=target dbtype=mysql \
     server=databaseHost port=databasePort \
     user=username password=password \
     dbname=databaseName
```

The input parameters are:

- **database** – Database role, the only possible value is *target* – to validate Apply Process Objects
- **dbtype** – Type of RDBMS, the value is *mysql*
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password

7.3.6 Validate Snowflake Database

REST API:

- **Endpoint:** https://***repstance url***/configure/database
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*

- **Body:**

```
{
   "command":"validate",
   "parameters":[
      ["database","target"],
      ["dbtype","snowflake"],
      ["account","account"],
      ["region","region"],
      ["warehouse","warehouse"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli validate database=target dbtype=snowflake \
      account=account region=region warehouse=warehouse \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role, the only possible value is *target* – to validate Apply Process Objects
- **dbtype** – Type of RDBMS, the value is *snowflake*
- **account** – Snowflake Account name
- **region** – Snowflake Region name
- **warehouse** – Warehouse name
- **user** – Database User name
- **password** – Database User password
- **dbname** – Database name

How to find "Snowflake Account" and "Snowflake Region" is described in chapter 7.1.8 Prepare Snowflake Database as Target Database.

7.4 Prepare Capture Process

The "**prepare**" Capture Process command is used to create new Capture Process.

The "**prepare**" command does not enable the Database to start capturing data on the configured objects, until the specified Capture Process runs (see chapters 7.15.1 Run any Capture or Apply Processes and 8.2.2.4 Processes Monitoring and Maintenance for the details).

Parameter **id** is mandatory for "**prepare**" command. All other parameters are optional. All parameters except **id** can be changed using "**alter**" Capture Process command (see chapter 7.5 Alter Capture Process and 8.2.2.3 Modify, Delete and Validate Processes). The **name** parameter can be altered only in the case if it has not been provided earlier. If there is another Capture Process, which has the same **id** or the same **name** then the "**prepare**" command will fail.

7.4.1 Prepare Capture Process for MS SQL Server Database

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command":"prepare",
   "parameters":[
      ["process","capture"],
      ["id","captureID"],
      ["dbtype","mssql"],
      ["name","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlinclude","objectsMask"],
      ["ddlexclude","objectsMask"],
      ["dmlinclude","objectsMask"],
      ["dmlexclude","objectsMask"],
      ["loadinclude","objectsMask"],
      ["loadexclude","objectsMask"],
      ["map=mapID","mappingClause"],
```

```
        ["skipapply","skipapplyMask"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=capture id=captureID \
      dbtype=mssql name=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} debuglevel={0-15} \
      ddlinclude=objectsMask ddlexclude=objectsMask \
      dmlinclude=objectsMask dmlexclude=objectsMask \
      loadinclude=objectsMask loadexclude=objectsMask \
      map=mapID,mappingClause \
      skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only appropriate value is *capture*
- **id** – The Capture Process id
- **dbtype** – The appropriate value is *mssql*
- **name** – Name of the Capture Process
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Process must be run automatically, the possible values are:
  - ➢ *0* – do not run the Process automatically (default value)
  - ➢ *1* – to run the Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **ddlinclude** – Mask of the DDL objects to be captured (see chapter 7.4.4 Capture Objects Specification for the details)

- **ddlexclude** – Mask of the DDL objects to be skipped by the Capture Process (see chapter 7.4.4 Capture Objects Specification for the details)
- **dmlinclude** – Mask of the DML objects to be captured (see chapter 7.4.4 Capture Objects Specification for the details)
- **dmlexclude** – Mask of the DML objects to be skipped by the Capture Process (see chapter 7.4.4 Capture Objects Specification for the details)
- **loadinclude** – Mask of the objects to be included into the Initial Load (see chapter 7.4.5 Initial Load for the details)
- **loadexclude** – Mask of the objects to be skipped during Initial Load (see chapter 7.4.5 Initial Load for the details)
- **map** – The set of the parameters to determine objects' transformation (see chapter 7.4.6 Objects Mapping and Possible Transforms for the details)
- **skipapply** – Used to define the behavior of a Capture Process when extracting data, which was produced by Apply Processes as part of the "loop-back" control function, in that, this Capture Process will extract or ignore the data. The Apply Processes to be skipped are specified by **id***s* separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*all*".

7.4.2 Prepare Capture Process for Oracle Database

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command": "prepare",
   "parameters": [
      ["process","capture"],
      ["id","captureID"],
      ["dbtype","oracle"],
      ["name","captureName"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["dbname","databaseName"],
      ["user","username"],
      ["password","password"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
```

```
        ["ddlinclude","objectsMask"],
        ["ddlexclude","objectsMask"],
        ["dmlinclude","objectsMask"],
        ["dmlexclude","objectsMask"],
        ["loadinclude","objectsMask"],
        ["loadexclude","objectsMask"],
        ["map=mapID","mappingClause"],
        ["skipapply","skipapplyMask"],
        ["loadconsistent","{0|1}"],
        ["directlogmode","0|1|2"],
        ["asmuser","asm_user"],
        ["asmpassword","asm_password"],
        ["asmservicename","asm_service_or_SID"],
        ["minearchonly","0|1"],
        ["locallogdir","path_to_log_folder"],
        ["poolingdelay","pooling_delay_in_sec"],
        ["preview","0|1"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, the possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=capture id=captureID \
      dbtype=oracle name=captureName \
      connectiontype=tns|ezconnect tnsname=tns_alias \
      server=databaseHost port=databasePort \
      servicename=service_or_SID \
      user=username password=password \
      dbname=databaseName autostart={0|1} debuglevel={0-15} \
      ddlinclude=objectsMask ddlexclude=objectsMask \
      dmlinclude=objectsMask dmlexclude=objectsMask \
      loadinclude=objectsMask loadexclude=objectsMask \
      loadconsistent={0|1} map=mapID,mappingClause \
      skipapply=skipapplyMask directlogmode={0|1|2} \
      asmuser=asm_username asmpassword=asm_password \
      asmservicename=asm_service_name \
      minearchonly={0|1} locallogdir=path_to_log_directory \
      poolingdelay={0-99999} preview={0|1}
```

The input parameters are:

- **process** – The only appropriate value is *capture*
- **id** – Capture Process id

- **dbtype** – Type of RDBMS, the only possible value is *oracle*
- **name** – Name of the Capture Process
- **connectiontype** – Specifies the Oracle connection method, the possible values are:
  - *tns* – Local Naming Method to be used
  - *ezconnect* – EZCONNECT to be used
- **tnsname** – Name of the TNS alias, valid **only** if connectiontype=*tns*
- **server** – Host name or IP address of the database server, valid **only** if connectiontype=*ezconnect*
- **port** – Database port number, valid **only** if connectiontype=*ezconnect*
- **servicename** – Database service name or SID, valid **only** if connectiontype=*ezconnect*
- **dbname** – Name of either container or pluggable database. Valid **only** if database=*source* and for any Oracle versions 12c-19c but excluding RDS instances
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Process must be run automatically, the possible values are:
  - *0* – do not run the Process automatically (default value)
  - *1* – to run the Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **ddlinclude** – Mask of the DDL objects to be captured (see the 7.4.4 Capture Objects Specification chapter for the details)
- **ddlexclude** – Mask of the DDL objects to be skipped by the Capture Process (see the 7.4.4 Capture Objects Specification chapter for the details)
- **dmlinclude** – Mask of the DML objects to be captured (see the 7.4.4 Capture Objects Specification chapter for the details)
- **dmlexclude** – Mask of the DML objects to be skipped by the Capture Process (see the 7.4.4 Capture Objects Specification chapter for the details)
- **loadinclude** – Mask of the objects to be included into the Initial Load (see chapter 7.4.5 Initial Load for the details)
- **loadexclude** – Mask of the objects to be skipped during Initial Load (see chapter 7.4.5 Initial Load for the details)
- **loadconsistent** – Determines if all the tables included into Initial Load must be consistent to a single point in time. The possible values are:
  - *0* – do not export all the tables at the same SCN (default)
  - *1* – export all the tables at the SCN

- **map** – The set of the parameters to determine objects' transformation (see 7.4.6 Objects Mapping and Possible Transforms chapter for the details)
- **skipapply** – Used to define the behavior of a Capture Process when extracting data, which was produced by Apply Processes as part of the "loop-back" control function, in that, this Capture Process will extract or ignore the data. The Apply Processes to be skipped are specified by **id***s* separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*all*".
- **directlogmode** – Used to define Data Capturing Method (see chapter 7.4.3 Overview of Data Capturing Methods for Oracle Database for the details). The possible values are:
  - *0* – use LogMiner method to capture the data (default)
  - *1* – use Direct Log Mining method to capture the data. If the "locallogdir" parameter is in use the Capture Process will process the Redo Logs from the local directory, otherwise the Capture Process will read the Logs from the Source Database as Oracle directory objects
  - *2* – use Direct Log Mining for Redo Logs located on the ASM device
- **asmuser** – ASM user name
- **asmpassword** – ASM user password
- **asmservicename** – ASM service name or SID
- **minearchonly** – Determines if the Capture Process must process Archived Redo Logs only. The possible values are:
  - *0* – process Online and Archived Redo Logs (default)
  - *1* – process Archived Redo Logs only
- **locallogdir** – Path to the local directory to process archived logs from (see chapter 7.4.3 Overview of Data Capturing Methods for Oracle Database for details)
- **poolingdelay** – The parameter is used to specify number of seconds the Capture Process will wait after each database changes gathering
- **preview** – This parameter is used to validate list of the tables to be included into the replication along with the transformation rules. The possible values are:
  - *0* – do not use "preview" mode (default)
  - *1* – do not apply the configuration to the Capture Process but provide the list of the tables to be included into the replication along with the transformation rules.

7.4.3 Overview of Data Capturing Methods for Oracle Database

Capture Process can extract Oracle database changes either using "LogMiner" functionality or by getting them directly from "Redo" or "Archived Redo" logs (Direct Log Mining).

LogMiner is built-in Oracle method, which is to analyze Redo Logs and extract the databases changes. LogMiner method can be considered in the following cases:

- Small number of changes (less than 5000 DML operations per second) should be replicated
- Possible CPU extra load (up to 15% at peak) is acceptable for the Source Database
- The replication of the LOB columns contain data size more than 4000 bytes is not required
- Size of the changes to be extracted is less than 5% of the Redo log size
- Clustered tables need to be replicated
- Access to Redo/Archived Logs is not possible to configure

In Direct Log Mining mode the Capture Process retrieves Redo Logs and use them to extract the database changes.

Direct Log Mining method best fits for most databases and used to meet the following replications requirements:

- High replication speed
- Minimal impact on the Source Database
- Support LOB columns
- Replication of Clustered Tables is not required
- Replication of Partitioned Index-Organized Tables is not required

Capture Process configuration varies depending on the Redo Log location. It supports Redo Logs located as either on file system or ASM device. Capture Process can also be configured to extract the changes from Redo Log files, which are available in the Repstance instance.

File System Location

For the Redo Logs located on the file system the "ONLINELOG_DIR" and "ARCHIVELOG_DIR" directories should be created (see chapter 7.1.2 Prepare Oracle Database as Source Database for the details). The "`directlogmode`" parameter of the Capture Process should be set to "*1*".

ASM Device

Capture Process requires ASM connection to be configured to access Redo Logs. The following parameters should be provided:

- *asmuser*
- *asmpassword*
- *asmservicename*

The "*directlogmode*" parameter should be set to "*2*".

Note – The ASM User has to have "sysdba" role assigned.

Repstance Local File System

Capture Process can be configured to extract the changes from Redo Logs delivered into Repstance instance by any external process. This configuration may be used to reduce load to the Oracle Net Services or if the access to Redo Logs through the Oracle Instance is hard to implement. In order to configure the "`locallogdir`" parameter should be provided and the "`directlogmode`" parameter should be set to "*1*".

7.4.4 Capture Objects Specification

Repstance allows flexibility to determine a set of tables, which are to be captured. The tables for DML and DDL operations are specified by the different parameters.

The following parameters are used for DML replication:

- `dmlinclude` – list of the tables to be included into the DML replication
- `dmlexclude` – list of the tables to be excluded from the DML replication

The following parameters are used for DDL replication:

- `ddlinclude` – list of the tables to be included into the DDL replication
- `ddlexclude` – list of the tables to be excluded from the DDL replication

Note – These criteria are mutually exclusive. The table will only be replicated in the case that the name matches the "include" criteria and does not match the "exclude" criteria.

The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The "**%**" symbol may be used in order to match any number of characters. It may also be used in both *schema_name* and *table_name* parts.

The following examples show various ways of defining tables:

- **Customers** and **employees** tables, which are in the **dbo** schema:

```
dbo.customers,dbo.employees
```

- Any tables in the **dbo** schema:

```
dbo.%
```

- Any tables having name started from **rep** or **tmp**:

```
%.rep%,%.tmp%
```

- Any table in report schema ends with **#:**

```
report.%#
```

The following examples show how to use the **DML/DDL** parameters:

- To capture DML changes for all tables in **dbo** schema except where tables start with **rep**:

```
dmlinclude=dbo.%
dmlexclude=dbo.rep%
```

- To capture DML changes for all tables in the **cap1**, **cap2** and **cap3** schemas except the **report1** table from the **cap1** schema:

```
dmlinclude=cap1.%,cap2.%,cap3.%
dmlexclude=cap1.report1
```

- To capture DML and DDL operations for **any** tables in the **dbo** schema:

```
dmlinclude=dbo.%
ddlinclude=dbo.%
```

- To capture only DDL operations for objects in the **report** schema:

```
ddlinclude=report.%
```

These parameters influence the newly created tables as well. The description below shows the capture behavior of the tables created:

Tables that match DML criteria	Tables that match DDL criteria	Capture Behavior
true	true	This create statement is captured. Any further DML

		statements will be captured as well
true	false	This create statement is not captured. No DML statements will be captured
false	true	This create statement is captured. DML statements are not captured. However, any further DDL statements are captured
false	false	This create statement is not captured. Neither DML or DDL statements will be captured

7.4.5 Initial Load

Repstance has built-in functionality to load objects' data and start the replication process from the timestamp at which the data has been created. Initial Loading is performing by the Capture Process at the "Run" stage (see chapter 7.15.1 Run any Capture or Apply Processes for the details).

This functionality is used when it is necessary to synchronize data between the Source and Target Databases before starting the replication process. The Initial Load parameters can be configured to either clean up or preserve data in the Target Database before loading from the Source Database. Two further options are to create table in the Target Database if it doesn't exist and recreate this table in the Target Database if it does exist.

The following parameters are used to configure the objects for Initial Loading:

- `loadinclude` – list of the tables to be included into the Initial Load
- `loadexclude` – list of the tables to be excluded from the Initial Load

Note – These parameters MUST be used in conjunction with the *`dmlinclude`*/ *`dmlexclude`* object definitions as well, otherwise the Capture Process will skip them.

The format to be specified takes the following form:

```
schema_name1.table_name1:[loadOption1],
schema_name2.table_name2:[loadOption2], ...
```

In order to define the list of the tables to be part of this process, and where there are more than one tables mask to be defined each of these must be separated by comma.

The `%` symbol may be used in order to match any number of characters. It may also be used in both *`schema_name`* and *`table_name`* parts.

The *`loadOption`* is an optional parameter, and it used to perform the action detailed below on the data before insertion into the Target Database, and can have the following values:

- `A` – preserve the data which is the **DEFAULT** value
- `T` – use **truncate** statement to clean up the data
- `D` – use **delete** statement to clean up the data
- `C` – create table if it doesn't exist
- `R` – recreate table if it already exists

Note – Once the Initial Load has been completed by Capture Process the values of *`loadinclude`* and *`loadexclude`* parameters will be removed automatically. The details of Initial Load criteria can be found in the Capture log files.

The following examples show various ways of defining the Initial Loading parameters:

- To reload data for the **emp** and **emp_audit_trail** tables, which are in the **dbo** schema and **truncate** these tables before insertion:

```
loadinclude=dbo.emp:T,dbo.emp_audit_trail:T
```

- Any data from tables starting with the rep_ prefix in the dbo schema which need to be **deleted** before insertion:

```
loadinclude=dbo.rep_%:D
```

- To reload all tables started with emp prefix except the employee_archive tables:

```
loadinclude=dbo.emp%:T
loadexclude=dbo.employee_archive
```

- To reload **customer** table and create it if it doesn't exist in the Target Database:

```
loadinclude=dbo.customer:C
```

7.4.6 Objects Mapping and Possible Transforms

Transforms are primarily used where there is a need to reformat any statements. The transformation can only be configured for a Capture Process, as the Apply function is

inherent in the use of transformation. Therefore, it is not possible nor is it necessary to construct an Apply Process for a Transform.

Repstance supports transformation rules, which can be applied to the following objects:

- ❑ schema name
- ❑ table name
- ❑ column name
- ❑ "identity" attribute (valid only for SQL Server DB)
- ❑ "primary key" attribute
- ❑ data type
- ❑ data type length
- ❑ data

Transformation rules can be triggered based on:

- ❑ schema name mask
- ❑ table name mask
- ❑ column name mask
- ❑ "identity" attribute (valid only for SQL Server DB)
- ❑ reference "identity" column (valid only for SQL Server DB)
- ❑ "primary key" attribute
- ❑ data type mask
- ❑ data type length

Transformation rules can be used to exclude the following objects:

- ❑ table
- ❑ column
- ❑ "identity" attribute (valid only for SQL Server DB)
- ❑ reference "identity" column (valid only for SQL Server DB)
- ❑ "primary key" attribute

Transformation rules can be used to add the following objects:

- ❑ column
- ❑ "identity" attribute (valid only for SQL Server DB)
- ❑ "primary key" attribute

Transformation rules can be used to change the following objects:

- ❑ schema name
- ❑ table name
- ❑ column name

- ❑ "identity" attribute (valid only for SQL Server DB)
- ❑ "primary key" attribute
- ❑ data type
- ❑ data type length
- ❑ data

Note – Data can be changed to the :

- ❑ constant value
- ❑ original value of another column from **this** table
- ❑ result of SQL Server function execution. The original values of any columns from this table can be passed to the Function as parameters. **The Function must exist in the Target Database, where the Apply Process is to be run.**

There are normally three separate parts that comprise a Transformation. They are:

- ❑ **`id`** – the id, which is unique to each Transform has the purpose of determining the order in which the Transform is executed by the Process
- ❑ **`rule`** – specifies the object to be transformed and specifies how it is to be transformed
- ❑ **`description`** – a meaningful description of this Transformation

The syntax used is:

`map=`*`id`*`,rule=(`*`CaptureCriteria:TransformCriteria`*`),description="`*`desc`*`"`

where:

`CaptureCriteria=`*`schema_mask.table_mask.column_mask.[attr_spec]`*

`attr_spec=PK=`*`PK`*`;TYPE=`*`type`*`;ID=`*`id`*`;LEN=`*`len`*

The `CaptureCriteria` consists of:

- `schema_mask` – the mask is used to specify schema name
- `table_mask` – the mask is used to specify table name
- `column_mask` – the mask is used to specify column name

The `attr_spec` is:

- `PK` – the specification of PK attribute, it can accept the following values:
  - ➢ *`N`* – to determine column without PK
  - ➢ *`U`* – to include PK column

- `ID` – determine if the column should or should not be handled as an "identity_insert", the possible values are:
  - ➢ *0* – to process as "non identity_insert" column
  - ➢ *1* – to process as "identity_insert" column
  - ➢ *2* – to process as reference "identity" column
- `TYPE` – name of the data type
- `LEN` – length of data type if applicable

`TransformCriteria=`*`schema_mask.table_mask.column_mask.`*`[`*`attr_spec`*`]`

`attr_spec=PK=`*`PK`*`;TYPE=`*`type`*`;ID=`*`id`*`;LEN=`*`len`*`;DATA="`*`data_specification`*`"`

`data_specification=DATA="`*`SQL_function_or_operation`*`"`

The `TransformCriteria` consists of:

- `schema_mask` – the mask is used to specify schema name
- `table_mask` – the mask is used to specify table name
- `column_mask` – the mask is used to specify column name

The `attr_spec` is:

- `PK` – the specification of PK attribute, it can accept the following values:
  - ➢ *N* – to determine column without PK
  - ➢ *U* – to include PK column
- `ID` – determine if the column should or should not be handled as an "identity_insert", the possible values are:
  - ➢ *0* – to process as "non identity_insert" column
  - ➢ *1* – to process as "identity_insert" column
  - ➢ *2* – to process as reference "identity" column
- `TYPE` – name of the data type
- `LEN` – length of data type if applicable
- `DATA` – this is the data that will result from the transformation specified on the captured data. This can be:
  - ➢ predefined value in format "*predefined_value*"
  - ➢ any operations based on the values of this column. In order to use the values found in the column the **%** needs to be used
  - ➢ any operations based on the values of any columns in this table. In order to use the values of any column the following format is used "`C`*`:number_of_column`*". Table name can be passed as parameter as well. The format is "*T:0*".

To **exclude columns** the following syntax is used:

*schema_mask.table_mask.*NULL

where:

- schema_mask – the mask is used to specify schema name
- table_mask – the mask is used to specify table name

To **include any columns that were not represented in the source table** the following syntax is used:

schema_mask.table_mask.column_name.[TYPE=*type*;LEN=*len*;
PK=*pk*;ID=*id*;DATA="*data_specification*"]

where:

- schema_mask – the mask is used to specify schema name
- table_mask – the mask is used to specify table name
- column_name – column name (this must be included and can not use the **%** symbol)
- TYPE – data type definition of the column to be created (this must be included and can not use the **%** symbol)
- LEN – length of data type, where necessary must be specified here
- PK – the specification of PK attribute. It accepts the following values:
  - *N* – to determine column without PK
  - *U* – to include PK column
- ID – determine if the column should or should not be handled as an "identity_insert". The possible values are:
  - *0* – to process as "non identity_insert" column
  - *1* – to process as "identity_insert" column
- DATA – this is the data that will result from the transformation specified on the captured data. This can be:
  - predefined value in format "*predefined_value*"
  - any operations based on the values of this column. In order to use the values found in the column the **%** needs to be used
  - any operations based on the values of any columns in this table. In order to use the values of any column the following format is used "C*:number_of_column*". Table name can be passed as parameter as well. The format is "*T:0*".

The following examples show how to use the Transformation Rules:

❑ For all tables located in the **dbo** schema change the schema name to **report**:

```
map=1,rule=(dbo.%:report.%),description="sample 1"
```

❑ Rename **dbo.emp** table to **report.employees** table:

```
map=2,rule=(dbo.emp:report.employees),description="sample 2"
```

❑ Exclude **info** column in the **dbo.customer** table from the replication:

```
map=3,rule=(dbo.customer.info:dbo.customer.NULL),description="sample 3"
```

❑ For any tables in the **dbo** schema having name starting with **rep_** replace the prefix to the **report_**:

```
map=4,rule=(dbo.rep_%:dbo.report_%),description="sample 4"
```

❑ For all tables in the **dbo** schema having **ident** column change the column name to **id**:

```
map=5,rule=(dbo.%.ident:dbo.%.id),description="sample 5"
```

❑ Exclude from the replication any columns having name **id** and defined as primary key:

```
map=6,rule=(dbo.%.id.[PK=U]:dbo.%.id.NULL),description="sample 6"
```

❑ For any tables having column **id** of **int** type handle this column as primary key. Any update and delete statements will be built to use **id** column as primary key or the part of composite primary key:

```
map=7,rule=(dbo.%.id.[TYPE=int]:dbo.%.id.[PK=P]),description="sample 7"
```

❑ Put the value of the **desc** column in the **branch** table into the uppercase:

```
map=8,rule=(dbo.branch.desc:dbo.branch.desc.[DATA=\"upper(%)\"])
```

❑ Add new column named **tag** of the **nvarchar(100)** type into the **customer** table contains predefined value **rep**:

```
map=9,rule=(dbo.customer.NULL:dbo.customer.tag.[TYPE=nvarchar;
                    LEN=100;DATA=\"'rep'\"])
```

❑ Add new column named **name** of the **nvarchar(255)** type to the **person** table contains concatenation of the **firstname** and **lastname** columns (in the table firstname has the position 2 and lastname has the position 3):

```
map=10,rule=(dbo.person.NULL:dbo.person.name.[TYPE=nvarchar;
                LEN=255;DATA=\"C:2+C:3\"])
```

❑ For any tables in the **dbo** schema having name starting with **rep_** and column **id**, which is defined as primary key and **identity_insert**, rename the table into the **report** schema and without **rep_** prefix and use custom function **report.generateID** to reproduce the value of this column. This column will still be used as a primary key, but not as **identity_insert**:

```
map=11,rule=(dbo.rep_%.id.[PK=U;ID=1]:report.%.id.[PK=U;ID=0;
                DATA=\"report.generateID()\"])
```

7.4.7 Transformation Rules and Triggering Order

It is imperative that you understand that the order in which the rules are applied can significantly impact on the expected results. The following logic is used to handle rule order processing. The specified Capture Process will go and look for any possible data that meets the first of the configured rules, to apply. In the case where there is more than one rule to be processed, the rule **which will** "take precedence" will be the rule which best fulfills the data found. If the objects found match the criteria given in the **first rule** to be processed any subsequent rules will be ignored because the previous criteria have been met.

The logic used in transformation does not necessarily follow the rules for human logic, in that, the least significant amount of data to be processed should be the **first rule**, and the rule which affects the largest amount of data should be the **last rule**. In this way the user will be able to apply the requirements of the transform in the desired way.

The following example demonstrates this program logic. Assume that you need to rename all tables in the **dbo** schema having a name starting with **emp** into the schema **hr** and keep the original table name. Another requirement is to rename the **ident** columns to be the **id** for any tables in the **dbo** schema, in the case where we have an **employee** table that contains the **ident** column, this matches both rules. The examples demonstrate the replication behavior depending on the order that the rules are presented:

❑ In this case you can't expect the **ident** column to be renamed into **id** column together with renaming **dbo** schema in **hr** schema. The **dbo** schema for **employee** table will be changed to **hr** schema but the **ident** column will not be changed:

```
map=1,rule=(dbo.emp%:hr.emp%),description="sample 1"
map=2,rule=(dbo.%.ident:dbo.%.id),description="sample 2"
```

❑ In this case the **dbo** schema will not be changed into the **hr** schema with renaming the **ident** column to the **id** column. The **dbo** schema for **employee** table will not be changed but the **ident** column will be changed to **id**:

```
map=1,rule=(dbo.%.ident:dbo.%.id),description="sample 3"
map=2,rule=(dbo.emp%:hr.emp%),description="sample 4"
```

❑ In this case the **dbo** schema for **employee** table will be changed and the **ident** column will be changed to **id**:

```
map=1,rule=(dbo.emp%.ident:hr.emp%.id)
map=2,rule=(dbo.emp%:hr.emp%)
map=3,rule=(dbo.%.ident:dbo.%.id)
```

7.5 Alter Capture Process

The "**alter**" Capture Process command is used to change the Capture Process parameters. The **id** parameter can not be changed. The **name** parameter can be changed only in the case that it has not been provided before – eg. it must be "Unique".

The Capture Process has to exist otherwise the command will fail.

Note – You can alter the Capture Process even if it is running, however these changes will only be implemented after the Capture Process has been stopped and rerun.

7.5.1 Alter Capture Process for MS SQL Server Database

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","capture"],
      ["id","captureID"],
      ["dbtype","mssql"],
      ["name","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
```

```
        ["password","password"],
        ["dbname","databaseName"],
        ["autostart","{0|1}"],
        ["debuglevel","{0-15}"],
        ["ddlinclude","objectsMask"],
        ["ddlexclude","objectsMask"],
        ["dmlinclude","objectsMask"],
        ["dmlexclude","objectsMask"],
        ["loadinclude","objectsMask"],
        ["loadexclude","objectsMask"],
        ["map=mapID","mappingClause"],
        ["skipapply","skipapplyMask"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=capture id=captureID dbtype=mssql \
      name=captureName server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} debuglevel={0-15} \
      ddlinclude=objectsMask ddlexclude=objectsMask \
      dmlinclude=objectsMask dmlexclude=objectsMask \
      loadinclude=objectsMask loadexclude=objectsMask \
      map=mapID,mappingClause skipapply=skipapplyMask
```

The input parameters are:

- **process** – Process type, the only appropriate value is *capture*
- **id** – Capture Process identifier
- **dbtype** – Type of RDBMS, the possible value is *mssql*
- **name** – Name of the Capture Process
- **server** – Host name or IP address of the database server
- **port** – Database port number
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password

- **autostart** – Determines if the Capture Process must be run automatically. Possible values are:
  - ➢ *0* – do not run the Capture Process automatically (default value)
  - ➢ *1* – to run the Capture Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **ddlinclude** – Mask of the DDL objects to be captured (see chapter 7.4.4 Capture Objects Specification for the details)
- **ddlexclude** – Mask of the DDL objects to be skipped by the Capture Process (see chapter 7.4.4 Capture Objects Specification for the details)
- **dmlinclude** – Mask of the DML objects to be captured (see chapter 7.4.4 Capture Objects Specification for the details)
- **dmlexclude** – Mask of the DML objects to be skipped by the Capture Process (see chapter 7.4.4 Capture Objects Specification for the details)
- **loadinclude** – Mask of the objects to be included into the Initial Load (see chapter 7.4.5 Initial Load for the details)
- **loadexclude** – Mask of the objects to be skipped during Initial Load (see chapter 7.4.5 Initial Load for the details)
- **map** – The set of the parameters to determine objects' transformation (see chapter 7.4.6 Objects Mapping and Possible Transforms for the details)
- **skipapply** – Used to define the behavior of a Capture Process when extracting data, which was produced by Apply Processes as part of the "loop-back" control function, in that, this Capture Process will extract or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*all*".

7.5.2 Alter Capture Process for Oracle Database

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","capture"],
      ["id","captureID"],
      ["dbtype","oracle"],
      ["name","captureName"],
```

```
        ["connectiontype","tns|ezconnect"],
        ["tnsname","tns_alias"],
        ["server","host_name"],
        ["port","port_number"],
        ["servicename","service_or_SID"],
        ["user","username"],
        ["password","password"],
        ["dbname","databaseName"],
        ["autostart","{0|1}"],
        ["debuglevel","{0-15}"],
        ["ddlinclude","objectsMask"],
        ["ddlexclude","objectsMask"],
        ["dmlinclude","objectsMask"],
        ["dmlexclude","objectsMask"],
        ["loadinclude","objectsMask"],
        ["loadexclude","objectsMask"],
        ["map=mapID","mappingClause"],
        ["skipapply","skipapplyMask"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=capture id=captureID \
      dbtype=oracle name=captureName \
      connectiontype=tns|ezconnect \
      tnsname=tns_alias \
      server=databaseHost port=databasePort \
      servicename=service_or_SID
      user=username password=password \
      dbname=databaseName autostart={0|1} debuglevel={0-15} \
      ddlinclude=objectsMask ddlexclude=objectsMask \
      dmlinclude=objectsMask dmlexclude=objectsMask \
      loadinclude=objectsMask loadexclude=objectsMask \
      map=mapID,mappingClause skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *capture*
- **id** – Capture Process id
- **dbtype** – Type of RDBMS, the possible value is *oracle*
- **connectiontype** – Specifies the Oracle connection method, possible values are:

    - ➢ *tns* – Local Naming Method to be used
    - ➢ *ezconnect* – EZCONNECT to be used
- **tnsname** – Name of the TNS alias, valid **only** if connectiontype=*tns*
- **server** – Host name or IP address of the database server, valid **only** if connectiontype=*ezconnect*
- **port** – Database port number, valid **only** if connectiontype=*ezconnect*
- **servicename** – Database service name or SID, valid **only** if connectiontype=*ezconnect*
- **dbname** – Name of either container or pluggable database. Valid **only** if database=*source* and for any Oracle versions 12c-19c but excluding RDS instances
- **name** – Name of the Capture Process
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Capture Process must be run automatically, possible values are:
    - ➢ *0* – do not run the Capture Process automatically (default value)
    - ➢ *1* – to run the Capture Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **ddlinclude** – Mask of the DDL objects to be captured (see 7.4.4 Capture Objects Specification chapter for the details)
- **ddlexclude** – Mask of the DDL objects to be skipped by the Capture Process (see 7.4.4 Capture Objects Specification chapter for the details)
- **dmlinclude** – Mask of the DML objects to be captured (see 7.4.4 Capture Objects Specification chapter for the details)
- **dmlexclude** – Mask of the DML objects to be skipped by the Capture Process (see 7.4.4 Capture Objects Specification chapter for the details)
- **loadinclude** – Mask of the objects to be included into the Initial Load (see chapter 7.4.5 Initial Load for the details)
- **loadexclude** – Mask of the objects to be skipped during Initial Load (see chapter 7.4.5 Initial Load for the details)
- **map** – The set of the parameters to determine objects' transformation (see chapter 7.4.6 Objects Mapping and Possible Transforms for the details)
- **skipapply** – Used to define the behavior of a Capture Process when extracting data, which was produced by Apply Processes as part of the "loop-back" control function, in that, this Capture Process will extract or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*all*".

7.6 Show Capture Process

The "**show**" Capture command is used to show a Capture Process configuration. This command will show the latest "Set" of configured processes, regardless of whether or not the Capture Process has been "stopped and re-applied", this will not necessarily be the configuration of the currently running Capture Process.

Note – If the configuration has been changed when the Capture Process is running, the "**show**" command will still display the currently configured process, not the "running" one.

7.6.1 Show Capture Process for MS SQL Server Database

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command": "show",
   "parameters": [
      ["process","capture"],
      ["id","captureID"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details",
  "parameters":[
      ["dbtype","mssql"],
      ["name","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
```

```
        ["dbname","databaseName"],
        ["autostart","{0|1}"],
        ["debuglevel","{0-15}"],
        ["ddlinclude","objectsMask"],
        ["ddlexclude","objectsMask"],
        ["loadinclude","objectsMask"],
        ["loadexclude","objectsMask"],
        ["dmlinclude","objectsMask"],
        ["dmlexclude","objectsMask"],
        ["map=mapID","mappingClause"],
        ["skipapply","skipapplyMask"]
    ]
}
```

CLI Syntax:

```
repcli show process=capture id=captureID
```

The input parameters are:

- **process** – The only possible value is *capture*
- **id** – The Capture Process identifier

Note – Only if the command completes successfully will the response contain all the "parameter" values and at the same time this information will be displayed in Plain Text in the "Message" display.

7.6.2 Show Capture Process for Oracle Database

Note – If the configuration has been changed when the Capture Process is running, the "**show**" command will still display the currently configured process, not the "running" one.

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
    "command": "show",
    "parameters": [
        ["process","capture"],
        ["id","captureID"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details",
  "parameters":[
      ["id","captureID"],
      ["dbtype","oracle"],
      ["name","captureName"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlinclude","objectsMask"],
      ["ddlexclude","objectsMask"],
      ["dmlinclude","objectsMask"],
      ["dmlexclude","objectsMask"],
      ["loadinclude","objectsMask"],
      ["loadexclude","objectsMask"],
      ["map=mapID","mappingClause"],
      ["skipapply","skipapplyMask"],
      ["loadconsistent","{0|1}"],
      ["directlogmode","0|1|2"],
      ["asmuser","asm_user"],
      ["asmpassword","asm_password"],
      ["asmservicename","asm_service_or_SID"],
      ["minearchonly","0|1"],
      ["locallogdir","path_to_log_folder"],
      ["poolingdelay","pooling_delay_in_sec"],
      ["preview","0|1"]
   ]
}
```

CLI Syntax:

```
repcli show process=capture id=captureID
```

The input parameters are:

- **process** – The only possible value is *capture*
- **id** – The Capture Process identifier

Note – Only if the command completes successfully will the response contain all the "parameter" values and at the same time this information will be displayed in Plain Text in the "Message" display.

7.7 Validate Capture Process

The "**validate**" Capture Process command is used to perform a set background checks to ensure that all the necessary parameters have been supplied to enable the functionality of a Capture configuration.

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command": "validate",
   "parameters": [
      ["process","capture"],
      ["id","captureID"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details",
}
```

CLI Syntax:

```
repcli validate process=capture id=captureID
```

The input parameters are:

- **process** – The only possible value is *capture*
- **id** – The Capture Process identifier

In case the **Status** is *Invalid* the **Message** will contain detailed information about the validation error.

7.8 Remove Capture Process

The "**remove**" Capture Process command is used to remove **all** of the Capture Process configuration files, Trail Files and the checkpoint information.

For MS SQL Server Database if the Source Database **is** reachable the "**remove**" command will disable CDC on the tables, which are configured to be captured by this Process only.

The command can only be executed if the specified Capture Process is not running.

It is strongly recommended that you backup all Trail and configuration files before running this command (see chapter 9.4 Backup Repstance Files).

Note – After removing the Capture Process it is not possible to reverse this action.

Any Apply Processes using data from this Capture will cease to work.

This command cannot be applied to a group of "Configures" at the same time, it must be used individually for each of the configured Capture Processes.

REST API:

- **Endpoint:** https://***repstance_url***/configure/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command": "remove",
   "parameters": [
      ["process","capture"],
      ["id","captureID"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:

- ➢ *200* – if no error occurs
- ➢ *422* – if error occurs

- **Body**:

```
{
  "Status":"{Completed|Error|Warning}",
  "Message":"The command execution details",
}
```

CLI Syntax:

```
repcli remove process=capture id=captureID
```

The input parameters are:

- **process** – The only possible value is *capture*
- **id** – The identifier of the Capture Process

The output parameters are:

- **Status** – Status of the command, possible values are:
  - ➢ *Error* – the command has failed to run
  - ➢ *Warning* – the process has failed to remove all the Capture objects
  - ➢ *Completed* – the command has completed successfully
- **Message** – The details of command execution

The following example demonstrates how to remove Capture 2:

```
repcli remove process=capture id=2
```

7.9 Prepare Apply Process

The "**prepare**" Apply command is used to add a new Apply Process. The "**prepare**" command can not be used to validate the database connection, and availability of the Trail Files or the presence of the necessary database objects to be used by the Apply Process.

In order to validate that the Apply Process is configured properly and it is able to run on the Target Database, the "**validate**" command must be used (see chapter 7.12. Validate Apply Process).

The "**prepare**" command does not enable the Apply Process to write data into the Target Database until the specified Apply Process is run (see chapter 7.15.1 Run any Capture or Apply Processes).

Parameter **id** is mandatory. All other parameters are optional. All parameters except **id** can be changed using "**alter**" Apply Process command (see chapter 7.10 Alter Apply Process for the details).

7.9.1 Prepare Apply Process for MS SQL Server Database

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command": "prepare",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","mssql"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["repuser","0|1"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=apply id=applyID dbtype=mssql \
```

```
capturename=captureName \
server=databaseHost port=databasePort \
user=username password=password \
dbname=databaseName autostart={0|1} \
debuglevel={0-15} repuser={0|1} \
ddlcreate=objectMask:options \
ddldrop=objectMask:options \
ddlprocessing=native|dictionary \
skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process id
- **dbtype** – Type of RDBMS, the only possible value is *mssql*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – do not run the Apply Process automatically (default value)
  - *1* – to run the Apply Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **repuser** – Determines if the Apply Process must represent itself as SQL Server replication process. The possible values are:
  - *0* – do not represent itself as SQL Server replication process (default value)
  - *1* – to represent itself as SQL Server replication process
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – do not run DDL create command if the table already exists
  - *recreate* – to drop the existing table and recreate it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each

table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following option:

  - ➢ *skip* – do not run DDL drop command if the table doesn't exist

- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - ➢ *native* – to process DDL based on the user statement (default value)
  - ➢ *dictionary* – to generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

You should be aware that if the **repuser** parameter is set to "1" and is invoked, then when delivering data to a SQL Server Target Database, any changes of objects marked as 'Not for Replication' (Identity columns, table triggers, foreign keys and check constraints) will be handled in exactly the same way, as they would be produced by an SQL Server replication agent. As an example, any triggers which are created with the "Not for Replication" option will never be fired.

7.9.2 Prepare Apply Process for Oracle Database

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command": "prepare",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","oracle"],
      ["capturename","captureName"],
```

```
        ["connectiontype","tns|ezconnect"],
        ["tnsname","tns_alias"],
        ["server","host_name"],
        ["port","port_number"],
        ["servicename","service_or_SID"],
        ["user","username"],
        ["password","password"],
        ["autostart","{0|1}"],
        ["ddlcreate","objectMask:options"],
        ["ddldrop","objectMask:options"],
        ["ddlprocessing","native|dictionary"],
        ["debuglevel","{0-15}"],
        ["skipapply","skipapplyMask"],
        ["ldtxsize","{0-9999999999}"],
        ["batchmode","{0|1}"],
        ["filter","filterID,filteringClause"],
        ["transformload","{0|1}"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=apply id=applyID dbtype=oracle \
      capturename=captureName \
      connectiontype="tns|ezconnect" tnsname=tns_alias \
      server=databaseHost port=databasePort \
      user=username password=password \
      servicename=service_or_SID autostart={0|1} \
      ddlcreate=objectMask:options ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      debuglevel={0-15} skipapply=skipapplyMask \
      ldtxsize={0-9999999999} batchmode={0|1} \
      filter=filterID,filteringClause transformload={0|1}
```

The input parameters are:

- **process** – The only appropriate value is *apply*
- **id** – The Apply Process id
- **dbtype** – Type of RDBMS, the only possible value is *oracle*
- **connectiontype** – Specifies the Oracle connection method, the possible values are:

- ➢ *tns* – Local Naming Method to be used
- ➢ *ezconnect* – EZCONNECT to be used

- **tnsname** – Name of the TNS alias, alid **only** if connectiontype=*tns*
- **server** – Host name or IP address of the database server, valid **only** if connectiontype=*ezconnect*
- **port** – Database port number, valid **only** if connectiontype=*ezconnect*
- **servicename** – Database service name or SID, valid **only** if connectiontype=*ezconnect*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Apply Process must be run automatically. The possible values are:
  - ➢ *0* – do not run the Apply Process automatically
  - ➢ *1* – to run the Apply Process automatically
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - ➢ *skip* – do not run DDL create command if the table already exists
  - ➢ *recreate* – drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - ➢ *skip* – do not run DDL drop command if the table doesn't exist
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - ➢ *native* – to process DDL based on the user statement (default value)
  - ➢ *dictionary* – to generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*all*" value to be used.

In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

- **ldtxsize** – Determines the number of records to be inserted in one transaction at the Initial Load stage. If the value is ***0*** (default) the table is loaded in a single transaction.
- **batchmode** – This parameter is to specify transaction processing method. If the parameter is enabled the Apply Process combines and processes set of transactions in a single transaction. The possible values are:
  - ➢ *0* – do not apply transactions in batch mode (default)
  - ➢ *1* – apply transactions in batch mode
- **filter** – The set of the parameters to determine objects' filtering (see chapter 7.9.7 Objects Filtering for the details).
- **transformload** – Determines if the transformation rules must be applied to the Initial Load. The possible values are:
  - ➢ *0* – do not use transformation rules for the Initial Load (default)
  - ➢ *1* – use transformation rules for the Initial Load.

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.9.3 Prepare Apply Process for PostgreSQL and Aurora PostgreSQL Databases

REST API:

- **Endpoint:** https://***repstance_url***/configure/process
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command": "prepare",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","postgresql"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
```

```
        ["ddlcreate","objectMask:options"],
        ["ddldrop","objectMask:options"],
        ["ddlprocessing","native|dictionary"],
        ["skipapply","skipapplyMask"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=apply id=applyID dbtype=postresql \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password dbname=databaseName \
      autostart={0|1} debuglevel={0-15} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process id
- **dbtype** – Type of RDBMS, the only possible value is *postgresql*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Apply Process must be run automatically, possible values are:
  - ➢ *0* – do not run the Apply Process automatically (default value)
  - ➢ *1* – to run the Apply Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)

- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - ➢ *skip* – do not run DDL create command if the table already exists
  - ➢ *recreate* – drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - ➢ *skip* – do not run DDL drop command if the table doesn't exist
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - ➢ *native* – to process DDL based on the user statement (default value)
  - ➢ *dictionary* – to generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.9.4 Prepare Apply Process for Redshift Database

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
```

```
   "command": "prepare",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","redshift"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=apply id=applyID dbtype=redshift \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password dbname=databaseName \
      autostart={0|1} debuglevel={0-15} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process id
- **dbtype** – Type of RDBMS, the only possible value is *redshift*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server

- **`port`** – Database port
- **`dbname`** – Database name
- **`user`** – Database User name
- **`password`** – Database User password
- **`autostart`** – Determines if the Apply Process must be run automatically, possible values are:
  - ➢ *`0`* – do not run the Apply Process automatically (default value)
  - ➢ *`1`* – to run the Apply Process automatically
- **`debuglevel`** – Level of debugging, possible values are *`0-15`* (default value is *`0`*)
- **`ddlcreate`** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *`schema_name.table_name`* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - ➢ *`skip`* – do not run DDL create command if the table already exists
  - ➢ *`recreate`* – drops the existing table and recreates it
- **`ddldrop`** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *`schema_name.table_name`* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - ➢ *`skip`* – do not run DDL drop command if the table doesn't exist
- **`ddlprocessing`** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - ➢ *`native`* – to process DDL based on the user statement (default value)
  - ➢ *`dictionary`* – to generate DDL based on the dictionary changes
- **`skipapply`** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*`all`*" value to be used. In order to avoid skipping any Apply Processes the "*`none`*" value to be used. The default value is "*`none`*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.9.5 Prepare Apply Process for MySQL and Aurora MySQL Databases

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command": "prepare",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","mysql"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=apply id=applyID dbtype=mysql \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} debuglevel={0-15} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
```

```
skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS, the only possible value is *mysql*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – do not run the Apply Process automatically (default value)
  - *1* – to run the Apply Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – do not run DDL create command if the table already exists
  - *recreate* – drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – do not run DDL drop command if the table doesn't exist
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *native* – to process DDL based on the user statement (default value)
  - *dictionary* – to generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by id*s*

separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.9.6 Prepare Apply Process for Snowflake Database

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command": "prepare",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","snowflake"],
      ["capturename","captureName"],
      ["account","account"],
      ["region","region"],
      ["warehouse","warehouse"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"],
      ["ldtxsize","{0-9999999999}"],
      ["filter","filterID,filteringClause"],
      ["transformload","{0|1}"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs

- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=apply id=applyID dbtype=snowflake \
      capturename=captureName \
      account=account region=region warehouse=warehouse \
      user=username password=password \
      dbname=databaseName autostart={0|1} debuglevel={0-15} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      skipapply=skipapplyMask ldtxsize={0-9999999999} \
      filter=filterID,filteringClause transformload={0|1}
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS, the only possible value is *snowflake*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **account** – Snowflake Account name
- **region** – Snowflake Region name
- **warehouse** – Warehouse name
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – do not run the Apply Process automatically (default value)
  - *1* – to run the Apply Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – do not run DDL create command if the table already exists
  - *recreate* – drops the existing table and recreates it

- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – do not run DDL drop command if the table doesn't exist
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *native* – to process DDL based on the user statement (default value)
  - *dictionary* – to generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".
- **ldtxsize** – Determines the number of records to be inserted in one transaction at the Initial Load stage. If the value is *0* (default) the table is loaded in a single transaction.
- **filter** – The set of the parameters to determine objects' filtering (see chapter 7.9.7 Objects Filtering for the details).
- **transformload** – Determines if the transformation rules must be applied to the Initial Load. The possible values are:
  - *0* – do not use transformation rules for the Initial Load (default)
  - *1* – use transformation rules for the Initial Load.

How to find "Snowflake Account" and "Snowflake Region" is described in chapter 7.1.8 Prepare Snowflake Database as Target Database.

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.9.7 Objects Filtering

Filters are used by Apply Process to filter out data while processing DML operations. The filter should be configured for either single table or group of the tables and at least for

one DML operation. The filtering statement might also be provided, which is to be used to construct "**where**" condition of the DML statement. If the filtering statement is not provided, then the Apply Process will ignore this DML operation.

The syntax used is:

```
filter=id,rule=(table_mask;dml_operations;"filter_statement")
```

where:

- *id* – The id, which is unique to each Filter has the purpose of determining the order in which the Filter is used by the Apply Process.
- *table_mask* – It is used to specify list of the tables. The format is *schema_name.table_name* and the **%** symbol may be used in order to match any number of characters.
- *dml_operations* – List of the DML operations separated by comma. The valid DML operations are: *INSERT*, *UPDATE*, *DELETE*.
- *filter_statement* – The statement to be used to build the "where" clause of DML operation.

The following examples show various ways of defining the Filter parameters:

❑ Skip any "delete" operations for "**logs"** table in "scott" schema:

```
filter=1,rule=(scott.logs;DELETE;"")
```

❑ Skip any "**delete**" operations for the **dbo.orders** table if **processed** column equal to "1":

```
filter=2,rule=(dbo.orders;DELETE;"processed=1")
```

❑ Skip any DML operations for the **dbo.emp** table if the **info** column has "do_not_replicate" value:

```
filter=3,rule=(dbo.emp;INSERT,UPDATE,DELETE;"info='do_not_replicate'")
```

❑ Do not insert record into the **dbo.acct** table if the record with the same **id** already exists:

```
filter=4,rule=(dbo.acct;INSERT;"id in (select id from dbo.acct)")
```

7.10 Alter Apply Process

The "**alter**" Apply command is used to change the Apply Process parameters. The **id** parameter can not be changed. The Apply Process has to exist otherwise the command

will fail. The **capturename** parameter can be altered only in the case that it has not been used earlier.

Note – You can alter the Apply Process even if it is running, however **these changes will only be implemented after this Apply Process has been stopped and rerun**.

7.10.1 Alter Apply Process for MS SQL Server Database

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","mssql"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["repuser","0|1"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=apply id=applyID dbtype=mssql \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} \
      debuglevel={0-15} repuser={0|1} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS, the only possible value is *mssql*
- **capturename** – Name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – do not run the Apply Process automatically (default value)
  - *1* – to run the Apply Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **repuser** – Determines if the Apply Process must represent itself as MS SQL Server replication process. The possible values are:
  - *0* – do not represent itself as SQL Server replication process (default value)
  - *1* – to represent itself as SQL Server replication process
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – do not run DDL create command if the table already exists
  - *recreate* – drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in

schema_name.table_name format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has following option:

  - ➢ *skip* – do not run DDL drop command if the table doesn't exist

- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - ➢ *native* – to process DDL based on the user statement (default value)
  - ➢ *dictionary* – to generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.10.2 Alter Apply Process for Oracle Database

REST API:

- **Endpoint:** https://***repstance_url***/configure/process
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","oracle"],
      ["capturename","captureName"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["user","username"],
      ["password","password"],
      ["autostart","{0|1}"],
```

```
        ["ddlcreate","objectMask:options"],
        ["ddldrop","objectMask:options"],
        ["debuglevel", "{0-15}"],
        ["ddlprocessing","native|dictionary"],
        ["skipapply","skipapplyMask"],
        ["ldtxsize","{0-9999999999}"],
        ["batchmode","{0|1}"],
        ["filter","filterID,filteringClause"],
        ["transformload","{0|1}"]
    ]
}
```

Server response:

- **HTTP Status** – status of the command, the possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=apply id=applyID dbtype=oracle \
      capturename=captureName \
      connectiontype=tns|ezconnect tnsname=tns_alias \
      servicename=service_or_SID \
      server=databaseHost port=databasePort \
      user=username password=password \
      autostart={0|1} \
      ddlcreate=objectMask:options ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      debuglevel={0-15} skipapply=skipapplyMask \
      ldtxsize={0-9999999999} batchmode={0|1} \
      filter=filterID,filteringClause transformload={0|1}
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS, the only possible value is *oracle*
- **connectiontype** – Specifies the Oracle connection method, possible values are:
  - ➢ *tns* – Local Naming Method to be used
  - ➢ *ezconnect* – EZCONNECT to be used
- **tnsname** – Name of the TNS alias, valid only if connectiontype=*tns*
- **server** – Host name or IP address of the database server, valid only if connectiontype=*ezconnect*
- **port** – Database port number, valid only if connectiontype=*ezconnect*

- **servicename** – Database service name or SID, valid **only** if `connectiontype=ezconnect`
- **capturename** – Name of the Capture Process which is providing the data to be used
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Apply Process must be run automatically or not. Possible values are:
  - *0* – do not run the Apply Process automatically (default value)
  - *1* – to run the Apply Process automatically
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – do not run DDL create command if the table already exists
  - *recreate* – drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has following option:
  - *skip* – do not run DDL drop command if the table doesn't exist
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *native* – to process DDL based on the user statement (default value)
  - *dictionary* – to generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".
- **ldtxsize** – Determines the number of records to be inserted in one transaction at the Initial Load stage. If the value is *0* (default) the table is loaded in a single transaction.

- **batchmode** – This parameter is to specify transaction processing method. If the parameter is enabled the Apply Process combines and processes set of transactions in a single transaction. The possible values are:
  - *0* – do not apply transactions in batch mode (default)
  - *1* – apply transactions in batch mode
- **filter** – The set of the parameters to determine objects' filtering (see chapter 7.9.7 Objects Filtering for the details).
- **transformload** – Determines if the transformation rules must be applied to the Initial Load. The possible values are:
  - *0* – do not use transformation rules for the Initial Load (default)
  - *1* – use transformation rules for the Initial Load.

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.10.3 Alter Apply Process for PostgreSQL and Aurora PostgreSQL Databases

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","postgresql"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – Status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=apply id=applyID dbtype=postgresql \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} \
      debuglevel={0-15} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS, the only possible value is *postgresql*
- **capturename** – Name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - ➢ *0* – do not run the Apply Process automatically (default value)
  - ➢ *1* – to run the Apply Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each

table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:

  - *skip* – do not run DDL create command if the table already exists
  - *recreate* – drops the existing table and recreates it

- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – do not run DDL drop command if the table doesn't exist
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *native* – to process DDL based on the user statement (default value)
  - *dictionary* – to generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.10.4 Alter Apply Process for Redshift Database

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","redshift"],
```

```
        ["capturename","captureName"],
        ["server","databaseHost"],
        ["port","databasePort"],
        ["user","username"],
        ["password","password"],
        ["dbname","databaseName"],
        ["autostart","{0|1}"],
        ["debuglevel","{0-15}"],
        ["ddlcreate","objectMask:options"],
        ["ddldrop","objectMask:options"],
        ["ddlprocessing","native|dictionary"],
        ["skipapply","skipapplyMask"]
    ]
}
```

Server response:

- **HTTP Status** – Status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=apply id=applyID dbtype=redshift \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} \
      debuglevel={0-15} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS, the only possible value is *redshift*
- **capturename** – Name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name

- **password** – Database User password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - ➢ *0* – do not run the Apply Process automatically (default value)
  - ➢ *1* – to run the Apply Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - ➢ *skip* – do not run DDL create command if the table already exists
  - ➢ *recreate* – drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - ➢ *skip* – do not run DDL drop command if the table doesn't exist
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - ➢ *native* – to process DDL based on the user statement (default value)
  - ➢ *dictionary* – to generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.10.5 Alter Apply Process for MySQL and Aurora MySQL Databases

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/*json
  - X-Token: *token*
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","mysql"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – Status of the command, the possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=apply id=applyID \
      dbtype=mysql capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} \
      debuglevel={0-15} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS, the possible value is *mysql*
- **capturename** – Name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – do not run the Apply Process automatically (default value)
  - *1* – to run the Apply Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – do not run DDL create command if the table already exists
  - *recreate* – drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – do not run DDL drop command if the table doesn't exist
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *native* – to process DDL based on the user statement (default value)
  - *dictionary* – to generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*all*" value to be used.

In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.10.6 Alter Apply Process for Snowflake Database

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/*json
  - X-Token: *token*
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","snowflake"],
      ["capturename","captureName"],
      ["account","account"],
      ["region","region"],
      ["warehouse","warehouse"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"],
      ["ldtxsize","{0-9999999999}"],
      ["filter","filterID,filteringClause"],
      ["transformload","{0|1}"]
   ]
}
```

Server response:

- **HTTP Status** – Status of the command, the possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
```

```
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=apply id=applyID \
      dbtype=snowflake capturename=captureName \
      account=account region=region warehouse=warehouse \
      user=username password=password \
      dbname=databaseName autostart={0|1} debuglevel={0-15} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      skipapply=skipapplyMask ldtxsize={0-9999999999} \
      filter=filterID,filteringClause transformload={0|1}
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS, the possible value is *snowflake*
- **capturename** – Name of the Capture Process which is providing the data to be used
- **account** – Snowflake Account name
- **region** – Snowflake Region name
- **warehouse** – Warehouse name
- **dbname** – Database name
- **user** – Database User name
- **password** – Database User password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – do not run the Apply Process automatically (default value)
  - *1* – to run the Apply Process automatically
- **debuglevel** – Level of debugging, possible values are *0-15* (default value is *0*)
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The **%** symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – do not run DDL create command if the table already exists
  - *recreate* – drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in

schema_name.table_name format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:

  - ➢ *skip* – do not run DDL drop command if the table doesn't exist

- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - ➢ *native* – to process DDL based on the user statement (default value)
  - ➢ *dictionary* – to generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".
- **ldtxsize** – Determines the number of records to be inserted in one transaction at the Initial Load stage. If the value is *0* (default) the table is loaded in a single transaction.
- **filter** – The set of the parameters to determine objects' filtering (see chapter 7.9.7 Objects Filtering for the details).
- **transformload** – Determines if the transformation rules must be applied to the Initial Load. The possible values are:
  - ➢ *0* – do not use transformation rules for the Initial Load (default)
  - ➢ *1* – use transformation rules for the Initial Load.

How to find "Snowflake Account" and "Snowflake Region" is described in chapter 7.1.8 Prepare Snowflake Database as Target Database.

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.11 Show Apply Process

The "**show**" Apply Process command is used to show an Apply configuration. This command will show the latest "Set" of configured Processes, regardless of whether or not the Apply Process has been "stopped and re-applied", this will not necessarily be the configuration of the currently running Apply Process.

Note – If the configuration has been changed when the Apply Process is running, the "**show**" command will still display the currently configured Apply Process, and not the "running" one.

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
   "command": "show",
   "parameters": [
      ["process","apply"],
      ["id","applyID"]
   ]
}
```

Server response:

- **HTTP Status** – Status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details",
  "parameters":[
      ["param1","value1"],
      ["param2","value2"],
       …
      ["paramN","valueN"],
   ]
}
```

The in the "parameters" section all enabled Process's parameters along with their values are provided.

CLI Syntax:

```
repcli show process=apply id=applyID
```

The input parameters are:

- **process** – The only possible value is *apply*

- **id** – The Apply Process Identifier

Note – Only in the case that the command completes successfully will the response contain all the "parameter" values and at the same time this information will be displayed in Plain Text in the "Message" display.

7.12 Validate Apply Process

The "**validate**" Apply Process command is used to perform a set of background checks to ensure that all the necessary parameters have been supplied to enable the functionality of an Apply configuration on the Target Database. This command will also provide the "last" successfully applied transaction's LSN/SCN but **only** if the Apply Process has previously been run or reset.

REST API:

- **Endpoint:** https://***repstance url***/configure/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command": "validate",
   "parameters": [
      ["process","apply"],
      ["id","applyID"]
   ]
}
```

Server response:

- **HTTP Status** – Status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body:**

```
{
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli validate process=apply id=applyID
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process Identifier

In case the **Status** is `Invalid` the **Message** will contain detailed information about the validation error.

7.13 Reset Apply Process

The "**reset**" command can **only** be used for an Apply Process – it is used where there is a need to force the Apply Process to start from specific LSN/SCN or after some LSN/SCN.

It is primarily used to "Set" or "Change" the "Startpoint" for an Apply Process.

REST API:

- **Endpoint:** https://***repstance url***/reset/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
   "command": "reset",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["LSN","LSN"],
      ["skip","{0|1}"],
   ]
}
```

Server response:

- **HTTP Status** – Status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details",
}
```

CLI Syntax:

```
repcli reset process=apply id=applyID LSN=LSN skip={0|1}
```

The input parameters are:

- **`process`** – The only possible value is *`apply`*
- **`id`** – The Apply Process Identifier
- **`LSN`** – The point at which we need to start reprocessing data
- **`skip`** – Determines if we need to jump over or not, a specific LSN:
  - ➢ *`0`* – start at the provided LSN
  - ➢ *`1`* –start after the provided LSN

The following example demonstrates how to reset an Apply Process to start from the first transaction found, after **LSN = 0x000000450000007B0004**:

```
repcli reset process=apply id=1 lsn=0x000000450000007B0004 skip=1
```

The following example demonstrates how to reset an Apply Process to start using the transactions from the **LSN = 0x000000450000007F0004:**

```
repcli reset process=apply id=1 lsn=0x000000450000007F0004 skip=0
```

7.14 Remove Apply Process

The "**remove**" Apply Process command is used to remove the Process configuration files and the checkpoint details, together with historical information. If the Target Database **is reachable** the "**remove**" command will cleanup all the checkpoint information on the Target Database.

The command can only be executed if the specified Apply Process is not running.

It is strongly recommended that you backup configuration files and last successfully processed LSN before running this command (see chapter 9.4 Backup Repstance Files).

This command cannot be applied to a group of Apply Processes at the same time, it must be used individually for each of the configured Apply Processes.

REST API:

- **`Endpoint:`** `https://`***`repstance url`***`/configure/process`
- **`Method:`** *`POST`*
- **`Header:`**
  - ➢ `Content-Type:` *`application/json`*
  - ➢ `X-Token:` *`token`*
- **`Body:`**

```
{
  "command": "remove",
  "parameters": [
    ["process","apply"],
    ["id","applyID"]
  ]
}
```

Server response:

- **HTTP Status** – Status of the command, possible codes are:
  - *200* – if no error occurs
  - *422* – if error occurs
- **Body**:

```
{
  "Status":"{Completed|Error|Warning}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli remove process=apply id=applyID
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The identifier of the Apply Process

The output parameters are:

- **Status** – Status of the command, possible values are:
  - *Error* – command has failed to run
  - *Warning* – process has failed to remove all the Apply Objects
  - *Completed* – command has completed successfully
- **Message** – Details of the command execution

Following example demonstrates how to remove Apply Process having id=2:

```
repcli remove process=apply id=2
```

7.15 Control Repstance Processes

There are three commands that are used to control the Capture and Apply Processes. They are :

- run
- stop
- status

Each of these commands can be used to control a single Process, a group of Processes or all Processes. **The default is to run "All" the Processes that are available to be run, stopped or get the status of, if no additional "Parameters" are specified**.

The syntax used for these commands is:

```
repcli run|stop|status [process=(capture|apply) [id=processID]]
```

Here are some examples of using **run** command.

This command will "run" everything (all existing Capture and Apply Processes):

```
repcli run
```

This command will "run" **all** Capture Processes:

```
repcli run process=capture
```

This command will "run" a single specified Capture Process:

```
repcli run process=capture id=1
```

The command format is identical in use for both the "stop" and "status" commands.

7.15.1 Run any Capture or Apply Processes

This command is used to "Run" either a single Capture or Apply Process or a group of either Capture, or Apply Processes or all configured Processes. **The default is to run all Processes that are available to be run, if no additional "Parameters" are specified**.

As part of the "**run**" command, Repstance carries out background checks to ensure that Source and/or Target Databases are properly configured with the necessary functionality to enable the Capture and/or Apply Processes to run. In the event that either of the Databases are not properly configured, it will display an error message listing the inconsistencies.

At the "first time" run of the Capture Process for MS SQL database it creates CDC tables needed for the replication and start extracting the changes from the first available LSN generated in the CDC tables.

For the Oracle database at the "first time" run the "current database SCN" is used as the start point of the replication.

Note – If there are any active transactions were running at the "current database SCN", and these transactions are committed after this SCN, the ALL changes generated by these transactions are included into the replication, so the tables will be in consistent state and no data is missed.

If the previously defined Objects (see chapter 7.4.4 Capture Objects Specification) are changed or the Transformation Rules (see chapter 7.4.7 Transformation Rules and Triggering Order) are altered, at this point all the DDL and DML changes will be re-implemented using the new criteria.

If any Objects are no longer configured for Capture Process the "**run**" command will "Turn Off" CDC on these Objects and they will no longer be written to the Trail Files – the reverse is true i.e. if new Objects are specified then the "run" command will enable CDC for them and they will now be written to the Trial Files.

If this is the "First time" an Apply Process has been run, it is **only** at this point that data from the specified Capture Process will be written to the Target Database using the first available LSN in the Trail File generated by this Capture Process.

The Apply Process can be configured to use a specific LSN from a Capture Process as a start point, i.e it can be configured to use any available LSN from a Trail File, but this is done by using the "**reset**" command (see chapter 7.13 Reset Apply Process).

If the specified Capture Process either is not configured or has never been run, the Apply Process will fail.

REST API:

- **Endpoint:** https://***repstance url***/control/process
- **Method:** *POST*
- **Header:**
  - Content-Type: *application/json*
  - X-Token: *token*
- **Body:**

```
{
  "command": "run",
  "parameters": [
    ["process","apply|capture"],
    ["id","processID"]
  ]
}
```

Server response:

- **HTTP Status** – Status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
{
  "Status":"{Completed|Error}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli run process=apply|capture id=processID
```

The input parameters are:

- **process** – Constant value *apply* for Apply Processes or constant value *capture* for Capture Processes
- **id** – Identifier of the Process

The output parameters are:

- **Status** – Status of the command, possible values are:
  - ➢ *Error* – the command has failed to run
  - ➢ *Completed* – the command has completed successfully
- **Message** – Details of command execution

7.15.2 Stop any Capture or Apply Processes

This command is used to stop either a single Capture or an Apply Process or a group of either Capture, or Apply Processes or all configured Processes. **The default is to stop everything that is running, if no additional parameters are specified**.

REST API:

- **Endpoint:** https://***repstance url***/control/process
- **Method:** *POST*
- **Header:**
  - ➢ Content-Type: *application/json*
  - ➢ X-Token: *token*
- **Body:**

```
{
  "command": "stop",
  "parameters": [
    ["process","apply|capture"],
    ["id","processID"]
```

```
  ]
}
```

Server response:

- **`HTTP Status`** – Status of the command, possible codes are:
  - ➢ *`200`* – if no error occurs
  - ➢ *`422`* – if error occurs
- **`Body`**:

```
{
  "Status":"{Completed|Error}",
  "Message":"The command execution details",
}
```

CLI Syntax:

```
repcli stop process=apply|capture id=processID
```

The input parameters are:

- **`process`** – Constant value *`apply`* for Apply Processes or constant value *`capture`* for Capture Processes
- **`id`** – Identifier of the Process

The output parameters are:

- **`Status`** – Status of the command, possible values are:
  - ➢ *`Error`* – the command has failed to run
  - ➢ *`Completed`* – the command has completed successfully
- **`Message`** – The details of command execution

7.15.3 Status of any Capture or Apply Processes

This command is used to show the status of either a single Capture or an Apply Process or a group of either Capture, or Apply Processes or all configured Processes. **The default is to show the status of every Process that is running, if no additional parameters are specified**.

REST API:

- **`Endpoint:`** `https://repstance url/control/process`
- **`Method:`** *`POST`*
- **`Header:`**
  - ➢ `Content-Type:` *`application/json`*
  - ➢ `X-Token:` *`token`*
- **`Body:`**

```
{
  "command": "status",
  "parameters": [
    ["process","apply|capture"],
    ["id","processID"]
  ]
}
```

Server response:

- **`HTTP Status`** – Status of the command, possible codes are:
  - ➢ *200* – if no error occurs
  - ➢ *422* – if error occurs
- **Body**:

```
[
   {
      "Process":"capture|apply",
      "ID":processID,
      "Parameters":{
         "Name|CaptureName":"captureName",
         "Server":"databaseHost",
         "Port":"databasePort",
         "DBname":"databaseName"},
      "Command":"INITIALISED|RUN|STOP|RESET",
      "CommandStatus":"Completed|In Process|Failed",
      "StatusDetails":"Waiting for the Next Command|
                       Processing the Transactions|
                       Waiting for Transactions",
      "CommandStartTime":"Time_in_UTC",
      "LastActivityTime":"Time_in_UTC",
      "StartLSN":"LSN",
      "StartLSNTime":"Time_in_UTC",
      "CheckpointLSN":"LSN",
      "CheckpointLSNTime":"Time_in_UTC",
      "LastDBLSN":"LSN",
      "LastDBLSNTime":"Time_in_UTC",
      "WAITING":"time_in_nanoseconds",
      "PROCESSING":"time_in_nanoseconds",
      "LOADING":"time_in_nanoseconds",
      "Message":"The command execution details",
      "CNTD":"Number_of_deletes",
      "CNTI":"Number_of_inserts",
      "CNTU":"Number_of_updates",
      "CNTDDL":"Number_of_ddls",
      "CNTLOBJ":"Number_of_loaded_objects",
      "CNTLOBJREC":"Number_of_loaded_records",
      "CNTTX":"Number_of_transactions"
   },
   {
    Next_Process_description
   }, ...
]
```

CLI Syntax:

```
repcli status process=apply|capture id=processID
```

The input parameters are:

- **process** – The *apply* value for Apply Processes or the *capture* value for Capture Processes
- **id** – The identifier of the Process

The output parameters are:

- **Process** – Name of the configured Process, possible values are:
  - *capture*
  - *apply*
- **ID** – The identifier of the Process
- **Parameters** – The list of the currently configured parameters:
  - *Name* – name of a Capture Process (only for Capture Processes)
  - *CaptureName* – name of Capture Process being used by the Apply Process (only for Apply Processes)
  - *Server* – Host name or IP address of the database server
  - *Port* – Database port
  - *DBname* – Database name
- **Command** – The last executed command on the listed Process, possible values are:
  - *INITIALISED* – means that the Process exists but currently has no command executed
  - *RUN* – means that the Process is running
  - *STOP* – means that the Process is stopping
  - *RESET* – means that the Apply Process is resetting to the specified LSN
- **CommandStatus** – Status of the command, possible values are:
  - *Waiting for the Next Command*
  - *Processing the Transactions*
  - *Waiting for Transactions*
  - *Loading the Data*
- **CommandStartTime** – Time in UTC format when the command was initialized (the parameter is available only in JSON response)
- **LastActivityTime** – Time in UTC format when the Process activity changed (the parameter is available only in JSON response)
- **StartLSN** – The first LSN (in hexadecimal format) when a transaction was processed (in **repcli** the parameter is displayed in the "Total since" section)
- **StartLSNTime** – Time of the transaction in UTC format the Process has started with (in **repcli** the parameter is displayed in the "Total since" section)

- **CheckpointLSN** – The LSN (in hexadecimal format) of the last transaction that has been successfully processed (in **repcli** the parameter is displayed in the "Last Captured/Applied DB Change" section)
- **CheckpointLSNTime** – Time in UTC format of the last transaction that has been successfully processed (in **repcli** the parameter is displayed in the "Last Captured/Applied DB Change" section)
- **LastDBLSN** – The last known transaction LSN (in hexadecimal format) in the Source Database. The parameter is provided by Capture Process and valid only in the case the Capture Process is running (in **repcli** the parameter is displayed in the "Last Sourced DB Change" section)
- **LastDBLSNTime** – Time in UTC format of the last known transaction LSN in the Source Database. The parameter is provided by Capture Process and valid only in the case the Capture Process is running (in **repcli** the parameter is displayed in the "Last Sourced DB Change" section)
- **WAITING** – Number of nanoseconds that the Process has been waiting for data to process (in **repcli** the parameter is displayed in the "Waiting for Transactions" section in `HH:MM:SS.FFF` format)
- **PROCESSING** – Number of nanoseconds that the Process has been processing the data (in **repcli** the parameter is displayed in the "Processing the Transactions" section in `HH:MM:SS.FFF` format)
- **LOADING** – Number of nanoseconds that the Process has been loading the data (in **repcli** the parameter is displayed in the "Loading the Data" section in `HH:MM:SS.FFF` format)
- **Message** – Details of the command execution
- **CNTD** – Number of Delete Operations currently performed by the Process (in **repcli** the parameter is displayed in the "Transactions" section on "Delete" position)
- **CNTI** – Number of Insert Operations currently performed by the Process (in **repcli** the parameter is displayed in the "Transactions" section on "Insert" position)
- **CNTU** – Number of Update Operations currently performed by the Process (in **repcli** the parameter is displayed in the "Transactions" section on "Update" position)
- **CNTDDL** – Number of DDL statements currently processed by the Process (in **repcli** the parameter is displayed in the "Transactions" section on "DDL" position)
- **CNTLOBJ** – Number of Objects currently processed by Initial Loading (in **repcli** the parameter is displayed in the "Loaded Objects" section)
- **CNTLOBJREC** – Number of records currently processed by Initial Loading (in **repcli** the parameter is displayed in the "Loaded Objects" section on "Rows" position)

- **CNTTX** – Number of transactions currently processed by the Process (in **repcli** the parameter is displayed in the "Transactions" section)

The **Average Speed** parameter is average number of records processed per second (calculated value).

The **Lag** parameter is the latency (delay) between the last record processed and the timestamp of the last transaction in Source Database (calculated value). The **Lag** is valid only in the case that the corresponding Capture Process is running. In **repcli** it is displayed in `HH:MM:SS.FFF` format.

The following example shows statuses of all existing Processes:

```
[http://localhost:8796/] repcli>status

Capture Process 1, Name: repv1, DB: mcdb, Server: source.repstance.com
  RUN (In Process), Waiting for Transactions
    Last Source DB Change   :  2019/03/10 21:47:00.610 UTC (0x000000E300001A750004)
    Last Captured DB Change :  2019/03/10 21:47:00.610 UTC (0x000000E300001A750004)
  Total since  2019/03/10 21:43:00.367 UTC (0x000000D400000A850004):
    Processing the Transactions :  00:00:02.214
    Waiting for Transactions    :  00:04:00.565
    Average Speed (ops)         :  45168.973
    Transactions   : 12 (DDL: 2,  Delete: 4,  Insert: 100016,  Update: 0)

Apply Process 1, Transactions Provider (Capture): repv1, DB: mcdb, Server:
target.repstance.com
  RUN (In Process), Waiting for Transactions
    Last Source DB Change   :  2019/03/10 21:47:00.610 UTC (0x000000E300001A750004)
    Last Applied DB Change  :  2019/03/10 21:47:00.610 UTC (0x000000E300001A750004)
    Lag                     :  00:00:00.000
  Total since  2019/03/10 21:44:00.696 UTC (0x00000000000000000001):
    Processing the Transactions :  00:00:09.230
    Waiting for Transactions    :  00:03:23.067
    Average Speed (ops)         :  10836.252
    Transactions   : 12 (DDL: 2,  Delete: 4,  Insert: 100016,  Update: 0)
```

The following example shows status of Capture Processes:

```
[http://localhost:8796/] repcli>status process=capture

Capture Process 1, Name: cap1, DB: prod1, Server: source.repstance.com
  RUN (In Process), Waiting for Transactions
    Last Source DB Change   :  2019/03/05 19:58:00.283 UTC (0x0000006F00004C900004)
    Last Captured DB Change :  2019/03/05 19:58:31.103 UTC (0x0000006F00005C340003)
  Total since  2019/03/05 19:58:00.283 UTC (0x0000006F00004C900004):
    Processing the Transactions :  00:00:00.588
    Waiting for Transactions    :  00:00:01.165
    Loading the Data            :  00:00:04.926
    Average Load Speed (ops)    :  190907.891
    Average Speed (ops)    :  0.000
    Transactions   : 0 (DDL: 0,  Delete: 0,  Insert: 0,  Update: 0)
    Loaded Objects : 2 (Rows: 940416)
```

8. WORK WITH REPSTANCE USING WEB UI

In both AWS and Azure environments the Repstance virtual machine is delivered with preconfigured Web UI Application, which is used to fully maintain and monitor Repstance Server using web browser.

This method can be used instead or along with the CLI or REST API commands. Any configuration changes that were performed via Web UI will be also available through the CLI or REST API commands and vice versa.

8.1 Connecting to Repstance Web UI

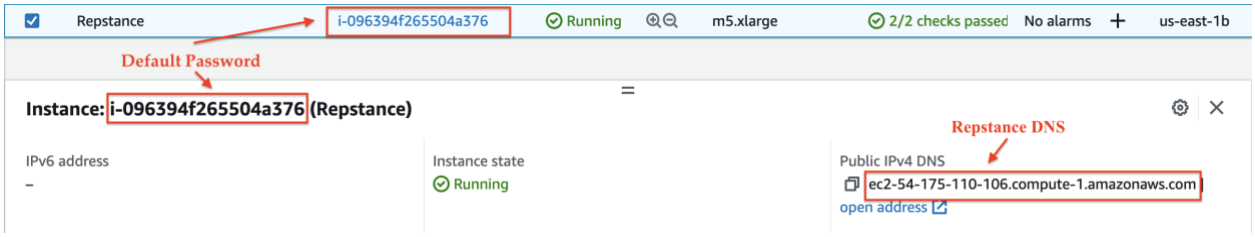
To connect to the Web UI using the HTTPS protocol open the following address in web browser:

```
https://<Repstance DNS>:3000/
```

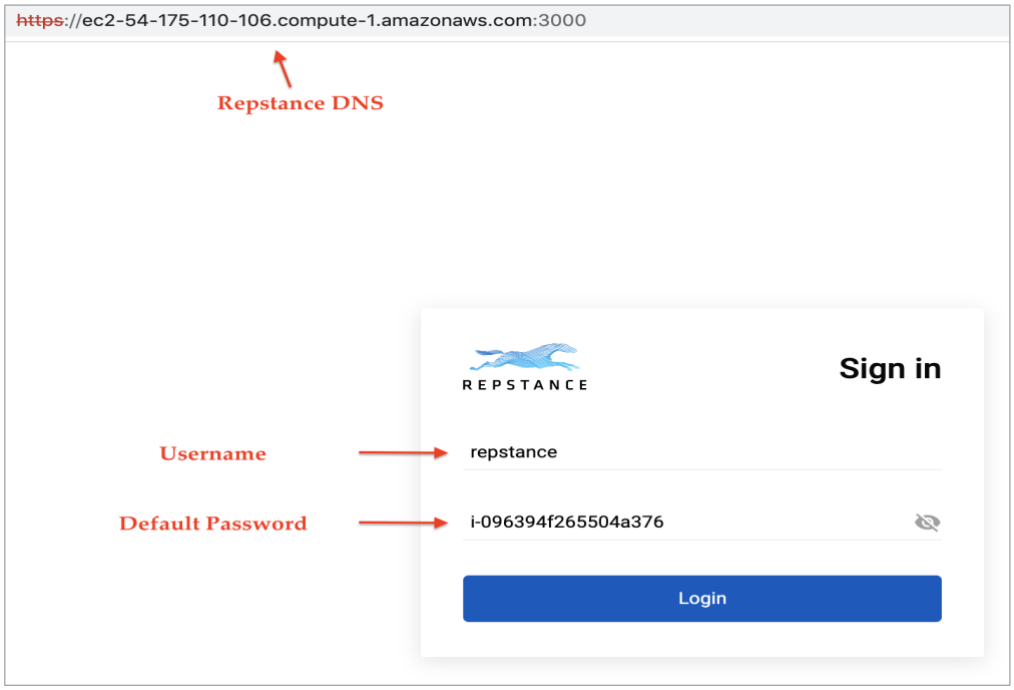
where <**Repstance DNS**> is the "Public DNS" of Repstance Instance. If "Public DNS" is not configured for the instance, the "Private IP" address should be used instead. The default port number is 3000.

The username is "**repstance**". The default password in the AWS environment is the **Instance ID** and the **VM Id** in the Azure environment.

The following example shows how to find Web UI connectivity parameters in the AWS environment.



So, the link to the Web UI and the default credentials for this example will be the following:



The "Repstance DNS" and the port number can be changed by editing "proxy_url" and "proxy_port" entries in the `/opt/repstance/ui/config.json` file.

The password is stored in the `/opt/repstance/conf/repstance.conf` file ("token" entry) and can be changed by editing the "token" entry.

8.2 How to work in Web UI

Repstance Web UI is an alternative way to configure, perform and maintain all the commands and processes described in chapter 7. Commands to be used.

All the functionality is allocated onto three tabs, which are:

- Database Configuration
- Process Configuration
- Wizard

The start page of Repstance Web UI is "Process Configuration". It displays general information about all existing Repstance Processes, created by any available way – using Web UI, **repcli** or REST API (see chapter 8.2.2 Process Configuration for more information).

If there are no any existing Repstance Processes yet, then Wizard mode will be suggested (see chapter 8.2.3 Wizard mode for the details).

8.2.1 Database Configuration

The Database Configuration form is to configure both Source and Target Databases, which are used by Repstance's Processes. As a part of database configuration, the following commands can be executed:

- **Prepare database**. This command is used to "prepare" either Source or Target database, by meaning that some replication functionality needs to be enabled and the Repstance's objects to support the replication need to be installed.

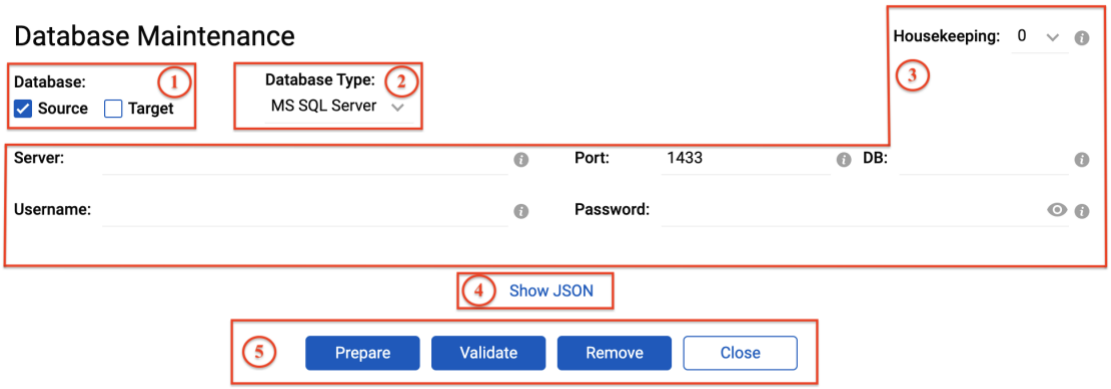
  Note – Both Source and Target Databases must be configured before running any replication process

- **Remove database**. This command is used to remove any replication functionality and Repstance's objects, which are installed by running the "prepare" command.
- **Validate database**. The command is used to check if the database has all necessary Repstance's objects and can be safety used by Repstance's Processes.

More details on these commands are described in chapter 5. How to use Repstance.

The "Database Configuration" is located on the top panel and opens the "Database Maintenance" form.

The form contains the following group of items:



1. Identify if the Database is "Source" or "Target", or both.
2. Select type of the database engine. The following database types can be selected for a Source Database:
   - Oracle

- MS SQL Server

The following database types can be selected for a Target Database:

- Oracle
- MS SQL Server
- MySQL
- PostgreSQL
- Redshift
- Snowflake

Note – MySQL, PostgreSQL, Redshift and Snowflake are supported in Repstance Advanced Edition only.

3. The database parameters. Depending on selected database type the different set of inputs to be displayed.
4. This link is used to generate JSON for "**prepare**", "**validate**" and "**remove**" commands, which is valid JSON for appropriate REST API command (see chapter 3.3 REST API for more details).
5. The buttons to execute "**prepare**", "**validate**" or "**remove**" commands. Once the command is executed the status is provided.

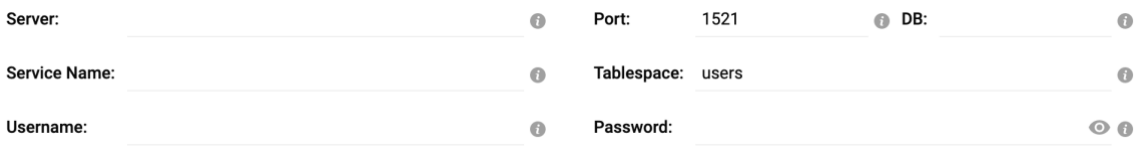
8.2.1.1 Oracle Database Configuration

In order to configure Oracle Database select "Oracle" under "Database Type" and enable "Source" or "Target" or both flags under "Database":

- **`Connection Type`** – There are two possible connection types – *`EZCONNECT`* and *`TNS`*. Depending on which one is chosen, there will be a different set of possible parameters.
- **`Createlogdirs`** – The parameter determines if the *`ONLINELOG_DIR`* and *`ARCHIVELOG_DIR`* Oracle directories should be created.

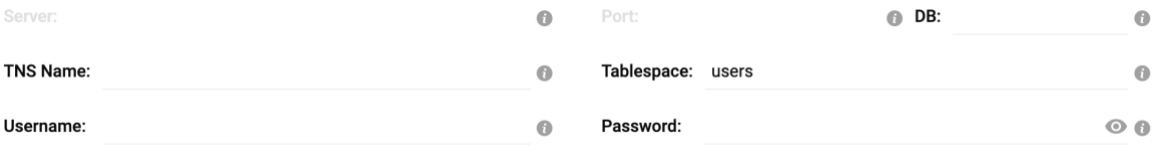
Note – The *`ONLINELOG_DIR`* and *`ARCHIVELOG_DIR`* directories are used if the Capture Process is configured to extract changes in the "DirectLog" mode **only** (see chapter 7.4.3 Overview of Data Capturing Methods for Oracle Database for the details).

If *`EZCONNECT`* connection type is selected the following connectivity parameters should be provided:



- **`Server`** – Host name or IP address of the database server
- **`Port`** – Database port number
- **`DB`** – Name of either container or pluggable database. Valid only for Source Database and for any Oracle versions 12c-19c but excluding RDS instances.
- **`Service Name`** – Database service name or SID
- **`Tablespace`** – Name of the tablespace that the Repstance's objects are to be installed in. The default is the *`USERS`* tablespace if no alternative has been specified.
- **`Username`** – Database User name
- **`Password`** – Database User password

If *`TNS`* connection type is selected the following connectivity parameters should be provided:

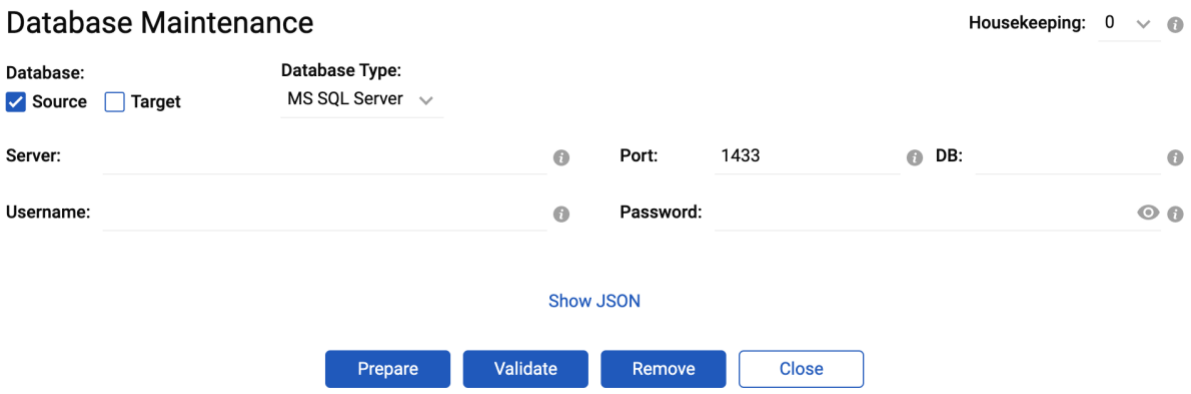


- **`DB`** – Name of either container or pluggable database. Valid only for Source Database and for any Oracle versions 12c-19c but excluding RDS instances.
- **`TNS Name`** – Name of the TNS alias
- **`Tablespace`** – Name of the tablespace that the Repstance's objects are to be installed in. The default is the *`USERS`* tablespace if no alternative has been specified.
- **`Username`** – Database User name
- **`Password`** – Database User password

Note – In order to use TNS connection type the "TNS Service Name" must be configured in the Repstance machine (see the details how the TNS method configuration https://docs.oracle.com/en/database/oracle/oracle-database/19/ntcli/specifying-connection-by-configuring-tnsnames.ora-file.html#GUID-D039649B-3A41-4BC8-BC29-EB1F9B0B6792).

8.2.1.2 MS SQL Server Database Configuration

In order to configure MS SQL Server Database select “MS SQL Server” under “Database Type” and enable “Source” or “Target” or both flags under “Database”:



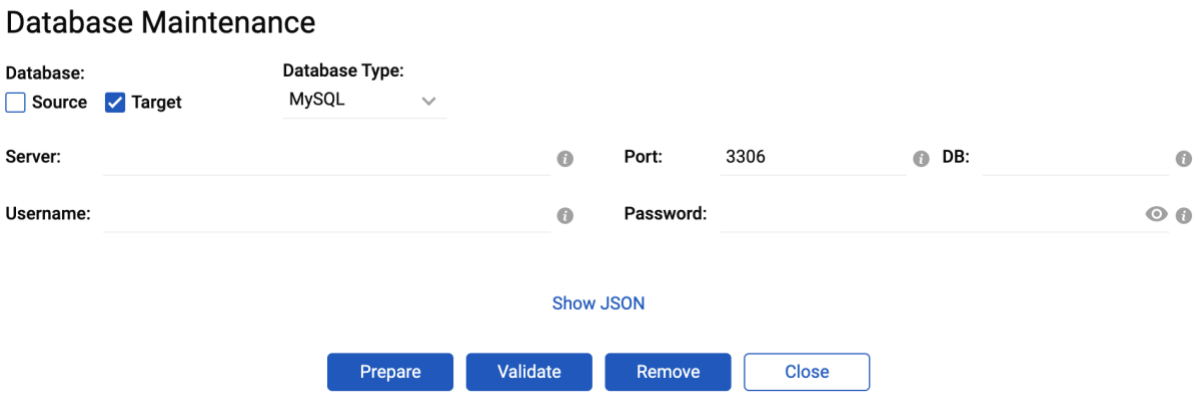
The following parameters should be provided:

- **`Server`** – Host name or IP address of the database server
- **`Port`** – Database port number
- **`DB`** – Database name to be connected to
- **`Username`** – Database User name
- **`Password`** – Database User password
- **`Housekeeping`** – This parameter determines how the CDC data will be cleaned up. The possible values are:
  - `0` – data will be cleaned up by an SQL Server job only,
  - `1` – install Repstance job to clean up data only after extraction and keep SQL Server cleanup job, which is a time based CDC setting,
  - `2` – install Repstance job and remove SQL Server cleanup job, data will be cleaned up only by the Repstance job.

8.2.1.3 MySQL Database Configuration

In order to configure MySQL Database select “MySQL” under “Database Type”. The database can be configured as Target Database only.

Note – MySQL database is supported in the Repstance Advanced Edition **only**.



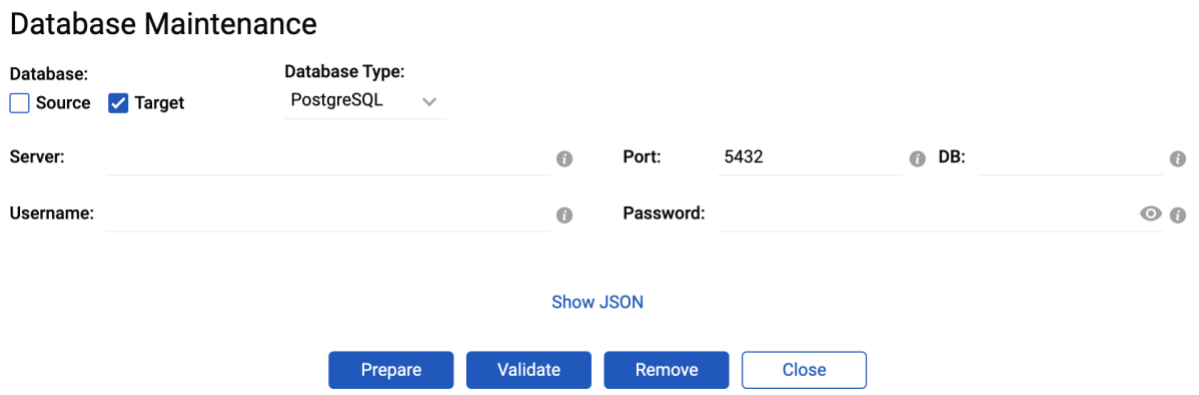
The following parameters should be provided:

- **`Server`** – Host name or IP address of the database server
- **`Port`** – Database port number
- **`DB`** – Database name to be connected to
- **`Username`** – Database User name
- **`Password`** – Database User password

8.2.1.4 PostgreSQL Database Configuration

In order to configure PostgreSQL Database select "PostgreSQL" under "Database Type". The database can be configured as Target Database only.

Note – PostgreSQL database is supported in the Repstance Advanced Edition **only**.



The following parameters should be provided:

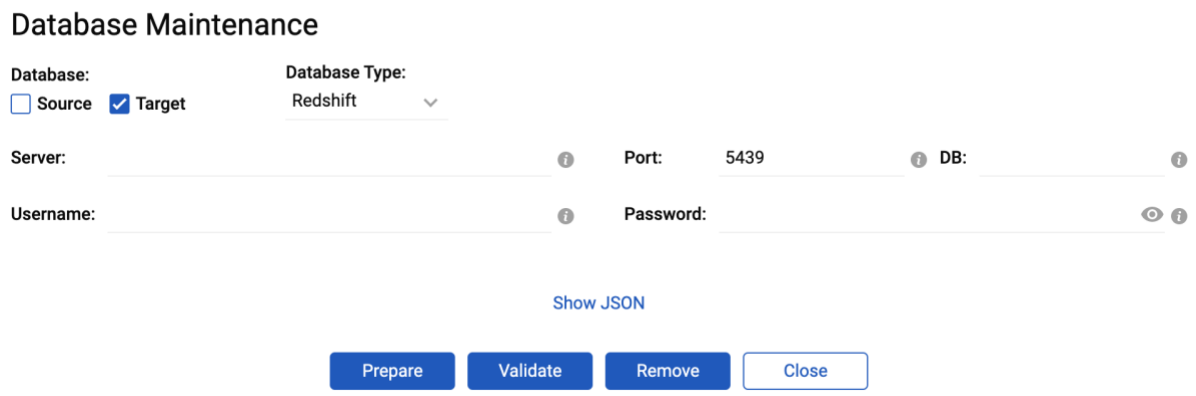
- **`Server`** – Host name or IP address of the database server
- **`Port`** – Database port number
- **`DB`** – Database name to be connected to
- **`Username`** – Database User name

- **`Password`** – Database User password

8.2.1.5 Redshift Database Configuration

In order to configure Redshift Database select "Redshift" under "Database Type". The database can be configured as Target Database only.

Note – Redshift Database is supported in the Repstance Advanced Edition **only**.



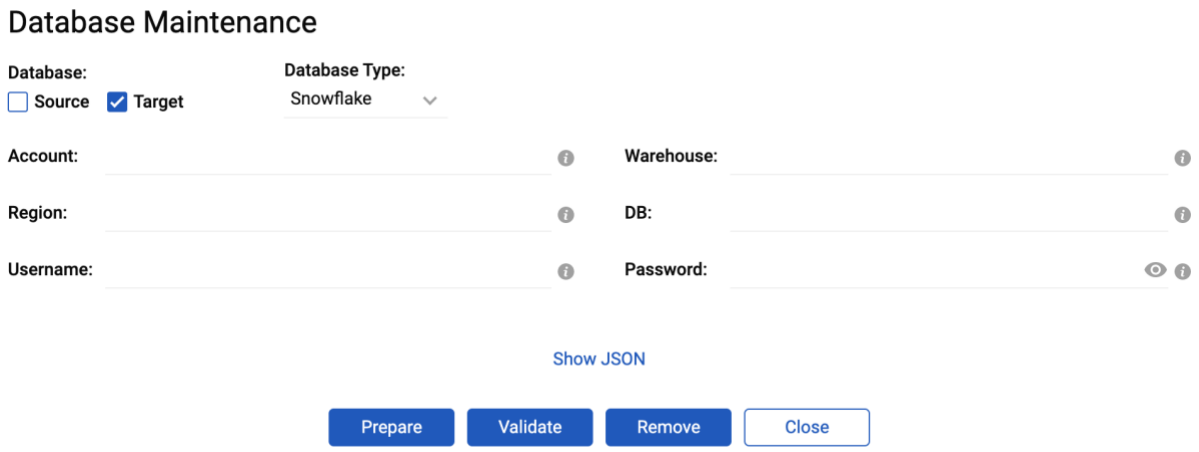
The following specific parameters should be provided:

- **`Server`** – Host name or IP address of the database server
- **`Port`** – Database port number
- **`DB`** – Database name to be connected to
- **`Username`** – Database User name
- **`Password`** – Database User password

8.2.1.6 Snowflake Database Configuration

In order to configure Snowflake Database select "Snowflake" under "Database Type". The database can be configured as Target Database only.

Note – Snowflake Database is supported in the Repstance Advanced Edition **only**.



The following specific parameters are to be provided:

- **`Account`** – Account name
- **`Warehouse`** – Warehouse name
- **`Region`** – Region name
- **`DB`** – Database name
- **`Username`** – Database User name
- **`Password`** – Database User password

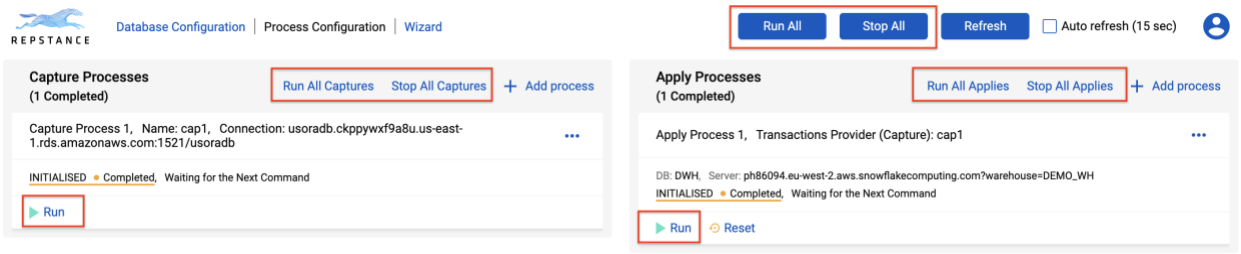
How to find "Snowflake Account" and "Snowflake Region" is described in chapter 7.1.8 Prepare Snowflake Database as Target Database.

8.2.2 Process Configuration

The "Process Configuration" tab is used to create, manage and control Repstance replication Processes. It displays details on the Processes created by any available method – using **repcli**, REST API or via Web UI and the items to maintain the Processes, which are:

- "Run All" – Start all Processes
- "Stop All" – Stop all Processes
- "Run All Captures" – Run All Capture Processes
- "Stop All Captures" – Stop All Capture Processes
- "+ Add Process" on the Capture Tab – Added Capture Process
- "Run All Applies" – Run All Apply Processes
- "Stop All Applies" – Stop All Apply Processes
- "+ Add Process" on the Apply Tab – Added Apply Process

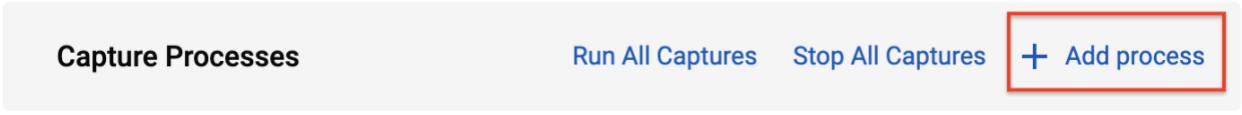
Each existing Process has individual control items which are to run, stop, modify, validate and remove the Process.



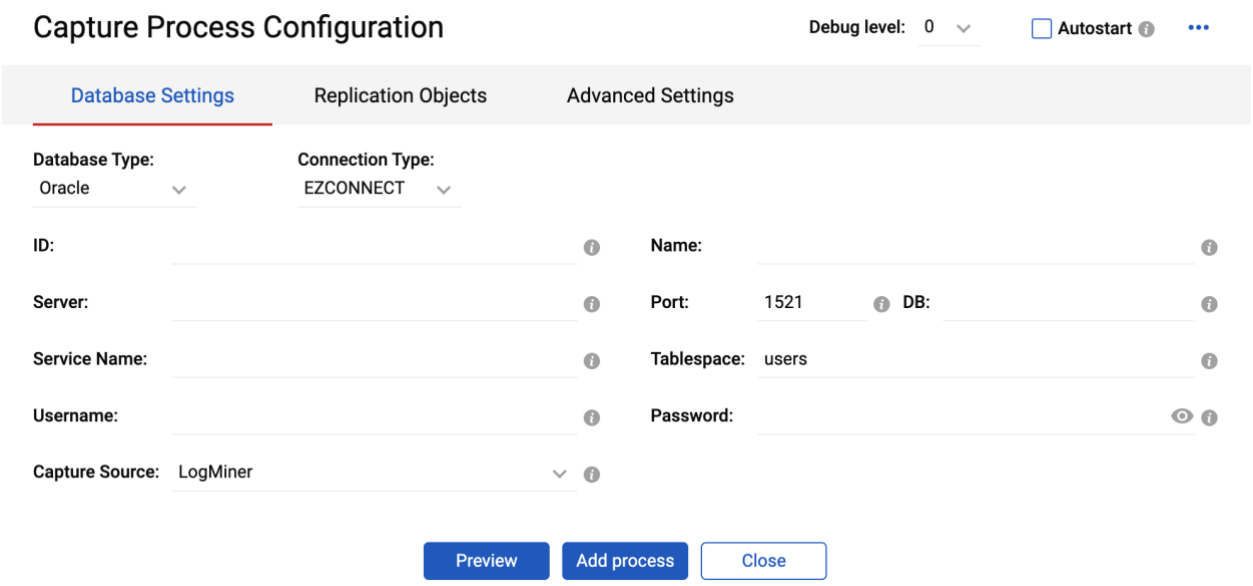
Execution details of these commands are described in chapter [7.15 Control Repstance Processes](#).

8.2.2.1 Create Capture Process

In order to create new Capture Process click "Add Process" button on "Capture Processes" tab:



The "Capture Process Configuration" form is opened:



On the top the following parameters should be provided:

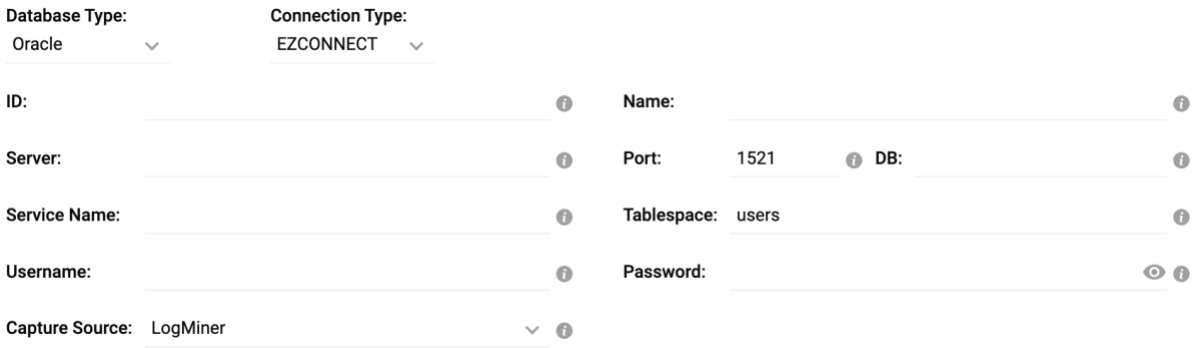
- **`Debug level`** – Level of debugging of the Capture Process.
- **`Autostart`** – Determines if the Process must be run automatically.

Database Settings Tab

Under "Database Settings" select the Database Type. Depending on which one is chosen, there will be a different set of possible parameters.

Database Settings for Oracle Database

For the Oracle database type the following parameters should be provided:



- **`Connection Type`** – There are two possible connection types: *`EZCONNECT`* and *`TNS`*. Depending on which one is chosen, there will be a different set of possible parameters.
- **`ID`** – The Capture Process id.
- **`Name`** – Name of the Capture Process.
- **`Server`** – Host name or IP address of the database server.
- **`Port`** – Database port.
- **`DB`** – Name of either container or pluggable database. Valid only for Source Database and for any Oracle versions 12c-19c but excluding RDS instances.
- **`Service Name`** – Database service name or SID.
- **`Tablespace`** – Name of the tablespace that the Repstance's objects are to be installed in. The default is the *`USERS`* tablespace if no alternative has been specified.
- **`Username`** – Database User name.
- **`Password`** – Database User password.
- **`Capture Source`** – Used to define Data Capturing Method (see chapter 7.4.3 Overview of Data Capturing Methods for Oracle Database). The possible values are:
  - *`LogMiner`* – use LogMiner method to capture the data (default value),
  - *`RedoMiner`* – use Direct Log Mining method to capture the data,
  - *`ASMMiner`* – use Direct Log Mining for Redo Logs located on the ASM device.

If the *RedoMiner* or *ASMMiner* is selected the addition parameter is to be provided:

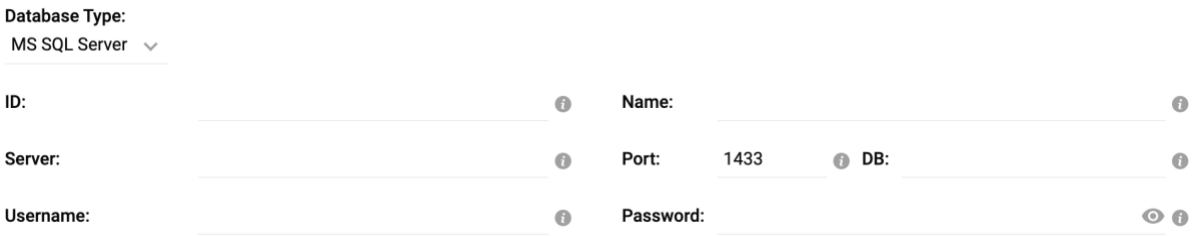
- **Mine Archivelogs Only** parameter determines if the Process must process Archived Redo Logs only.

If the *ASMMiner* is selected the ASM credentials parameters must be provided:

- **ASM User** – ASM User name
- **ASM Password** – ASM User password
- **ASM Service Name** – ASM service name or SID

Database Settings for MS SQL Server Database

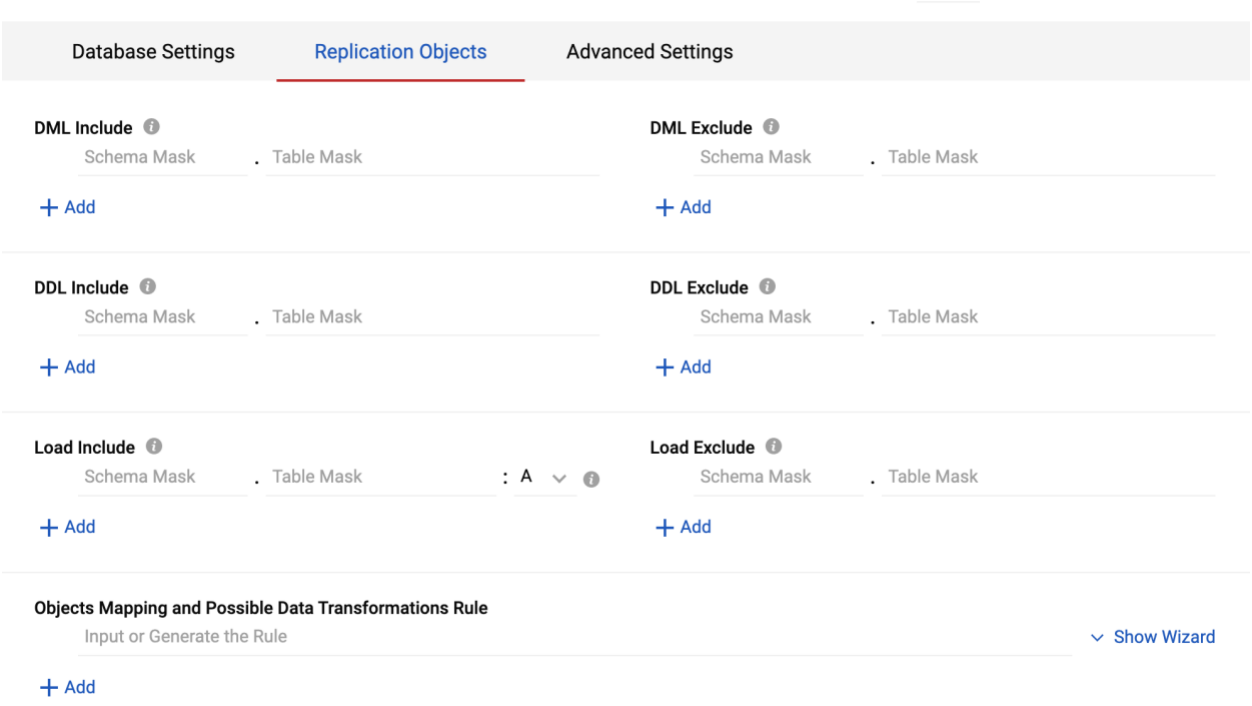
For MS SQL Server database the following connection parameters should be provided:



- **ID** – The Capture Process id
- **Name** – Name of the Capture Process
- **Server** – Host name or IP address of the database server
- **Port** – Database port
- **DB** – Name of the database
- **Username** – Database User name
- **Password** – Database User password

Replication Objects

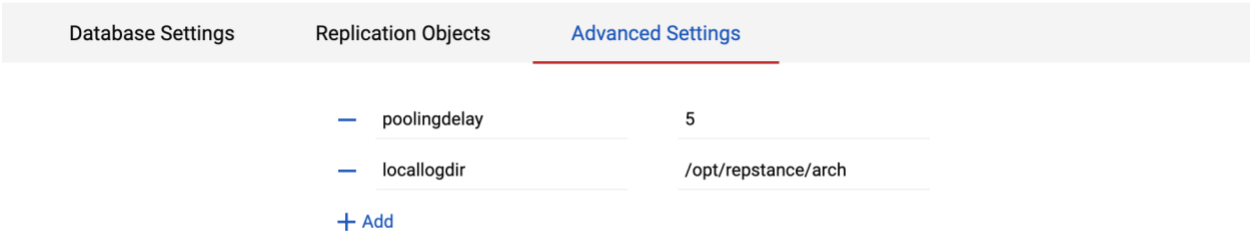
Under “Replication Objects” tab the details of the tables to be included in the replication should be provided:



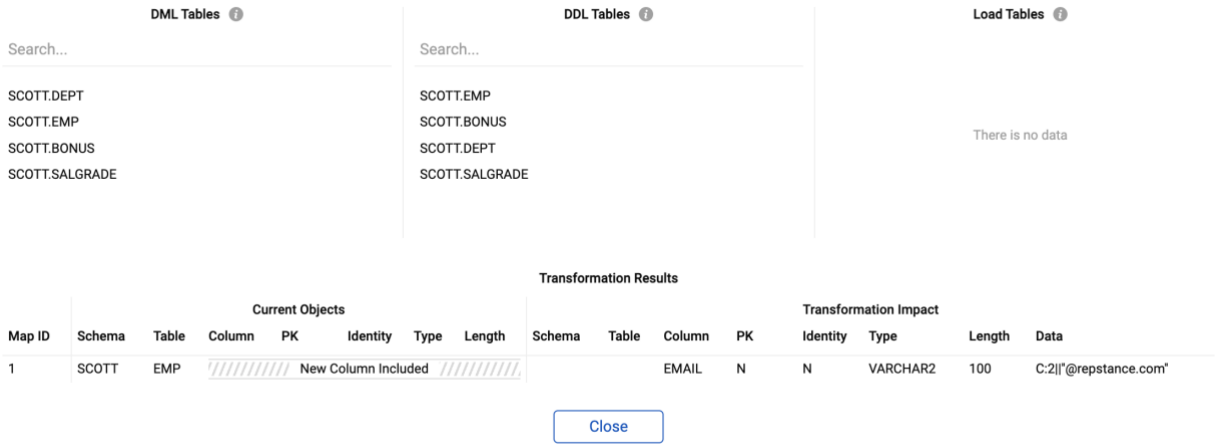
- **`DML Include`** – Mask of the DML objects to be captured.
- **`DML Exclude`** –Mask of the DML objects to be skipped by the Capture Process.
- **`DDL Include`** – Mask of the DDL objects to be captured.
- **`DDL Exclude`** – Mask of the DDL objects to be skipped by the Capture Process.
- **`Load Include`** – Mask of the objects to be included into the Initial Load along with the load option. The following load methods are currently supported:
  - *`A`* – preserve the data (default value),
  - *`T`* – use truncate statement to clean up the data,
  - *`D`* – use delete statement to clean up the data,
  - *`C`* – create table if it doesn’t exist,
  - *`R`* – recreate table if it already exists.
- **`DDL Exclude`** – Mask of the objects to be skipped during Initial Load.
- **`Object Mapping and Possible Data Transformation Rule`** – This section is to provide the transformation rules (see chapter 7.4.6 Objects Mapping and Possible Transforms for the details).

Advanced Settings

Under “Advanced Settings” tab the additional Capture parameters can be provided:



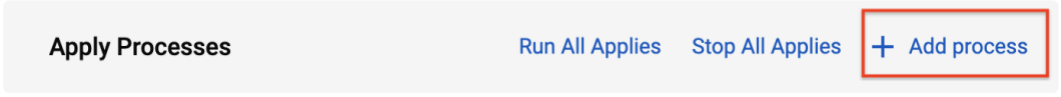
Clicking “Preview” button displays the report of the tables to be replicated and loaded along with the details on the tables’ mapping and data transformations, so the Capture Process configuration can be validated before to create it:



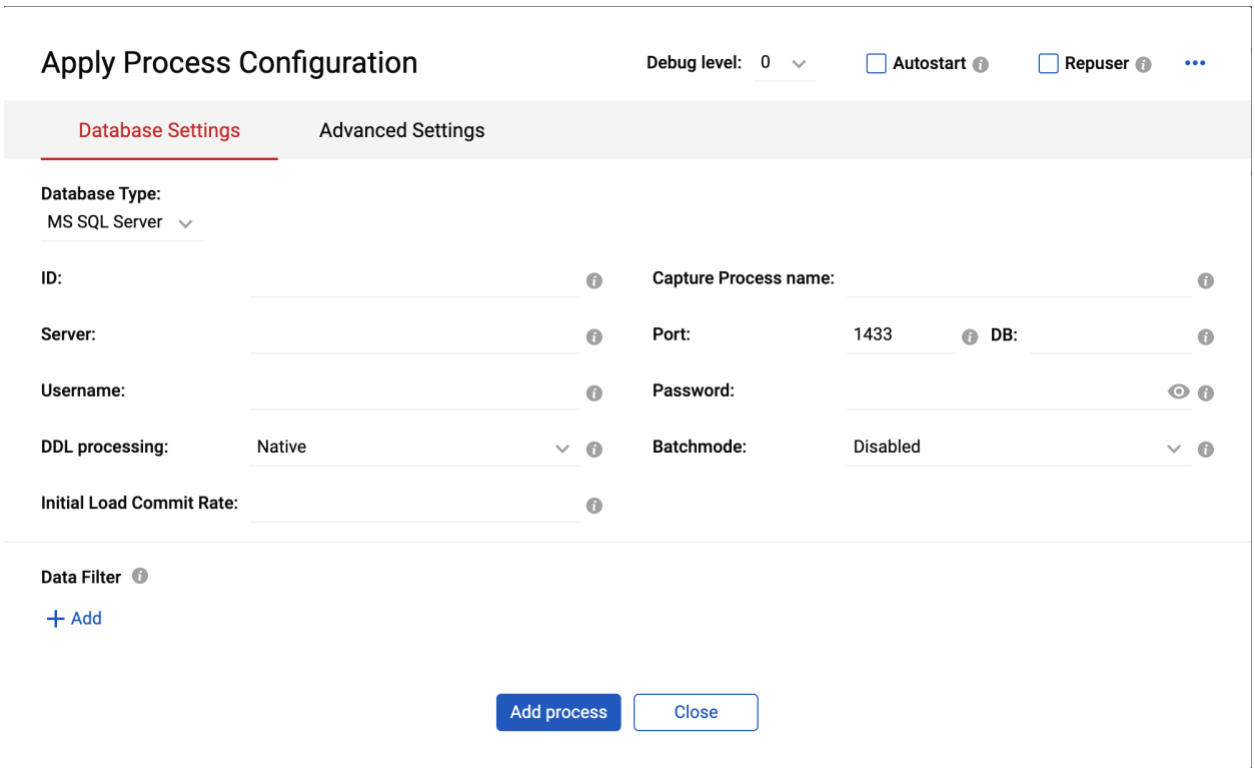
Clicking “Add Process” button creates the Capture Process and exits the form.

8.2.2.2 Create Apply Process

In order to create new “Apply Process” click “Add Process” button under “Apply Processes” tab:



The “Apply Process Configuration” form is opened:

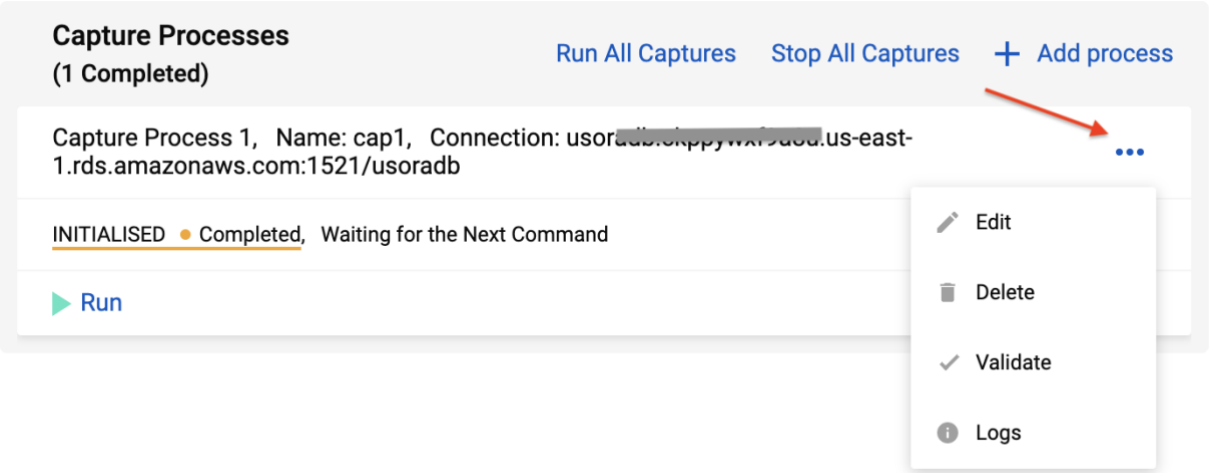


Under "Database Settings" the following parameters should be provided:

- **`Debug level`** – Level of the Apply Process debugging.
- **`Autostart`** – Determines if the Apply Process must be run automatically.
- **`Repuser`** – Determines if the Apply Process must represent itself as SQL Server replication process (valid for MS SQL Server database only).
- **`ID`** – The Apply Process id.
- **`Capture Process name`** – Name of the Capture Process which is providing the data to be used.
- **`Server`**, **`Port`**, **`DB`**, **`Username`** and **`Password`** are to provide DB connectivity details and might be different depending on selected database type.
- **`DDL Processing`** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - ➢ *`native`* – to process DDL based on the user statement (default value),
  - ➢ *`dictionary`* – to generate DDL based on the dictionary changes.
- **`Batchmode`** – This parameter is to specify transaction processing method. If the parameter is enabled the Apply Process combines and processes set of transactions in a single transaction.
- **`Initial Commit Rate`** – Determines the number of records to be inserted in one transaction at the Initial Load stage. If the value is 0 (default) the table is loaded in a single transaction.
- **`Data filter`** – The set of the parameters to determine objects' filtering (see chapter 7.9.7 Objects Filtering for the details).

8.2.2.3 Modify, Delete and Validate Processes

In order to modify, validate or delete existing Capture or Apply Process click on the "..." button and select the action:



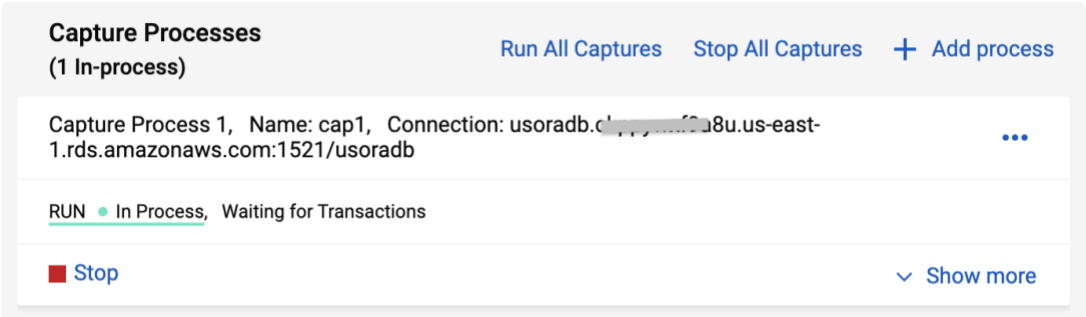
Clicking "Edit" button opens the same form, which is used to create the Process, so the Process parameters can be modified there. "Edit" is the analog of "**alter**" command (see chapters 7.5 Alter Capture Process and 7.10 Alter Apply Process for details).

"Delete" is used to remove the Process (see chapters 6. HOW TO REMOVE REPSTANCE, 7.8 Remove Capture Process and 7.14 Remove Apply Process for more details).

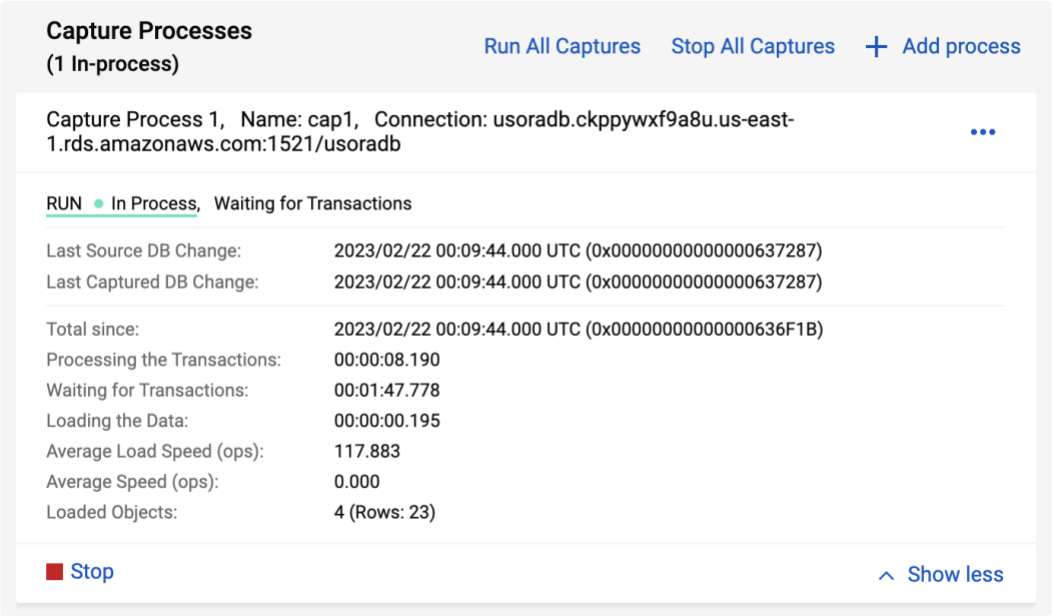
"Validate" is used to verify if the process has been configured properly (see chapters 7.7 Validate Capture Process and 7.12 Validate Apply Process for the details).

8.2.2.4 Processes Monitoring and Maintenance

The dashboard displays general information about the Processes such as *process state*, *name* of the Process and *connectivity details*:



To view the detailed information about the Process, click “Show more”:

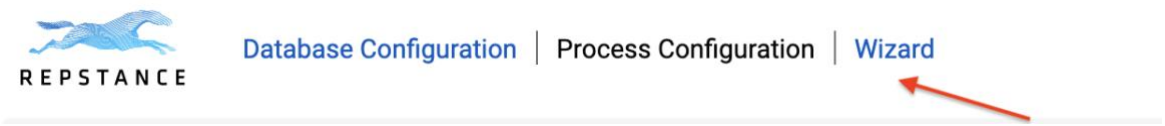


8.2.3 Wizard Mode

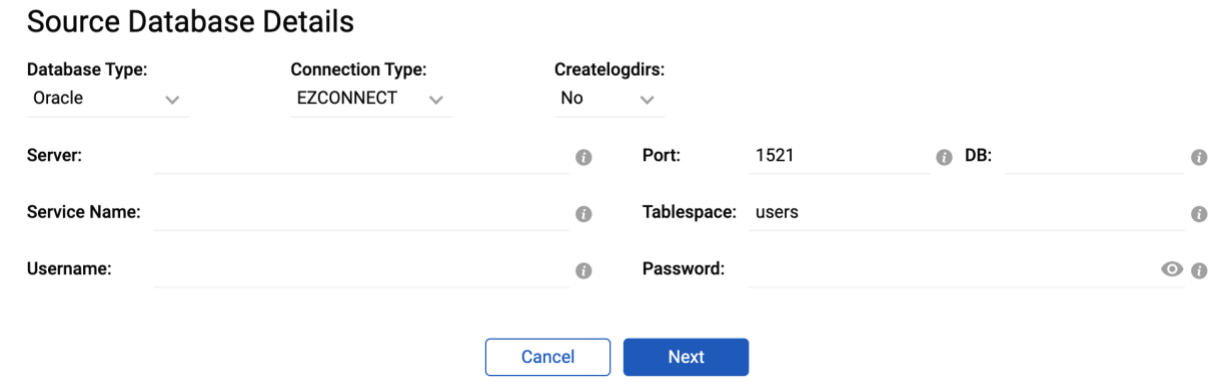
The Wizard Mode passes user through all the necessary steps to configure data replication between two databases. It makes the configuration process much easy and fast to implement. These steps are:

- **Prepare Source Database**. This step is to validate if the Source Database meets the requirements to run the Capture Process and if it doesn't to configure the Database.
- **Prepare Capture Process**. This step is to create the Capture Process and define list of the tables to be migrated and replicated along with the transformation rules.
- **Prepare Target Database**. This step is to validate if the Target Database is ready for the replication and if it is not to configure it.
- **Prepare Apply Process**. This step is to create the Apply Process and provide data processing settings.

Wizard Mode is automatically run at the first login or by clicking “Wizard” tab:

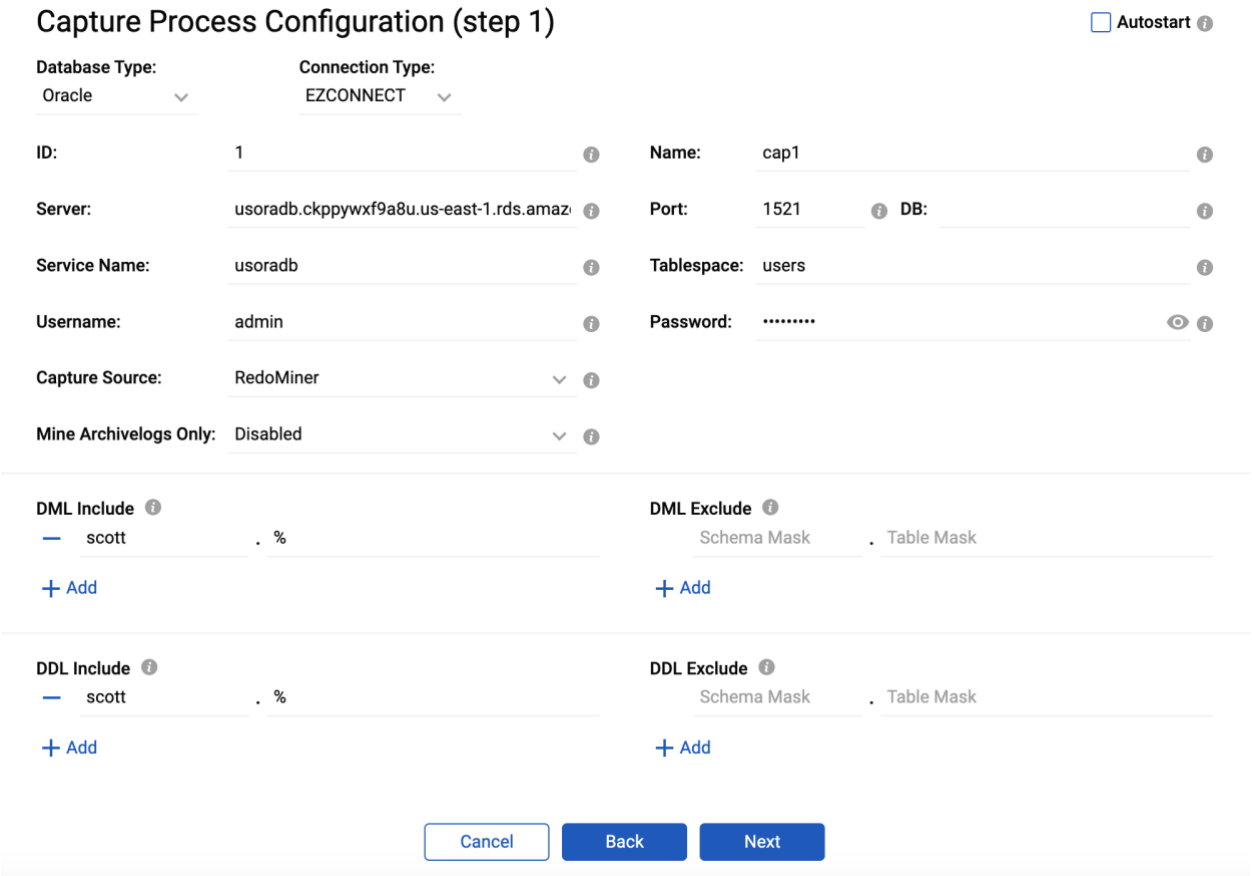


At the first step Wizard Process verifies if the Source Database is ready for the replication. It requires the Source Database details to be provided. The set of the details depends on the selected database type.



If the Source Database is not ready for replication, Wizard Process offers to prepare the database (see chapter 8.2.1 Database Configuration for the details).

Once the Source Database has been successfully prepared, Wizard Process moves on the next step, which is to prepare Capture Process.

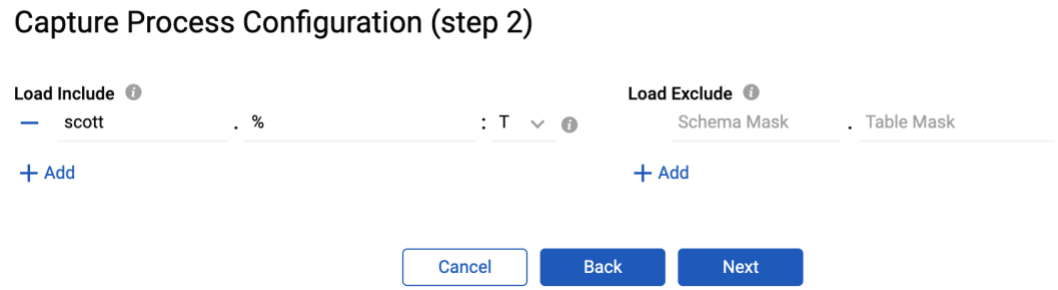


Wizard Process populates the database connection settings from the previous form. The settings can be changed if required.

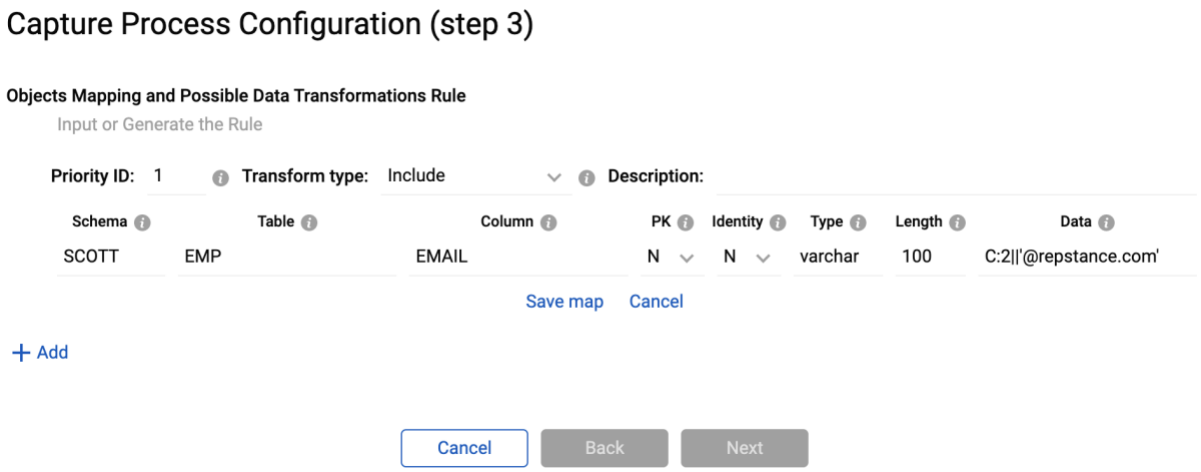
It is necessary to provide name of the Capture Process (**`Name`** parameter) and the tables to be included into the DML/DDL replication (**`DML Include`**, **`DML Exclude`** and **`DDL Include`**, **`DDL Exclude`** parameters).

If the Source Database is Oracle the Capturing method (**`Capture Source`** parameter) should be provided (see chapter 8.2.2.1 Create Capture Process for the details).

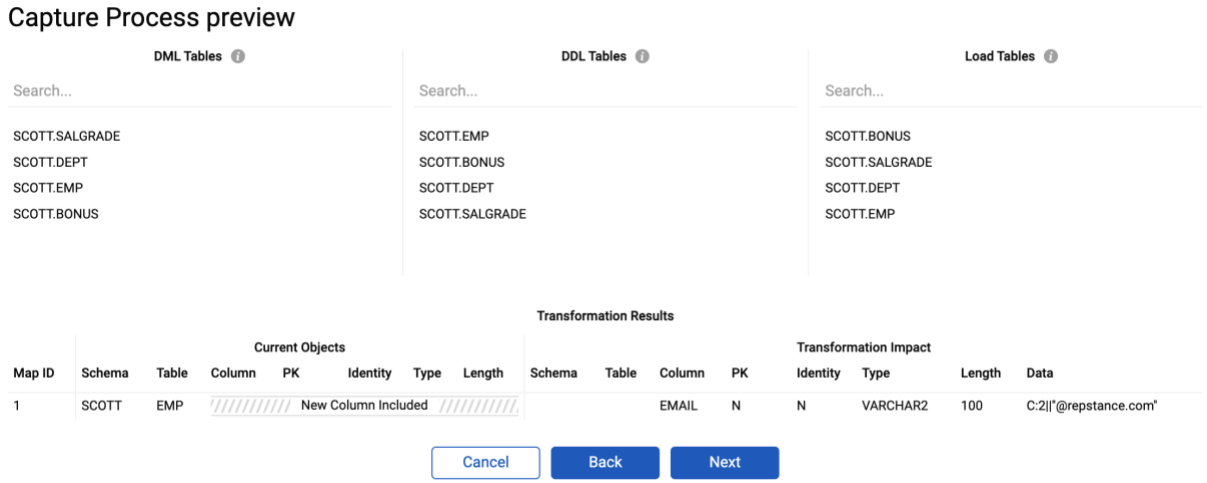
On the next form select the tables, which require being loaded before the replication along with the loading option (see chapter 7.4.5 Initial Load for the details):



The next form is used to configure Transformation Rules (see chapter 7.4.7 Transformation Rules and Triggering Order):



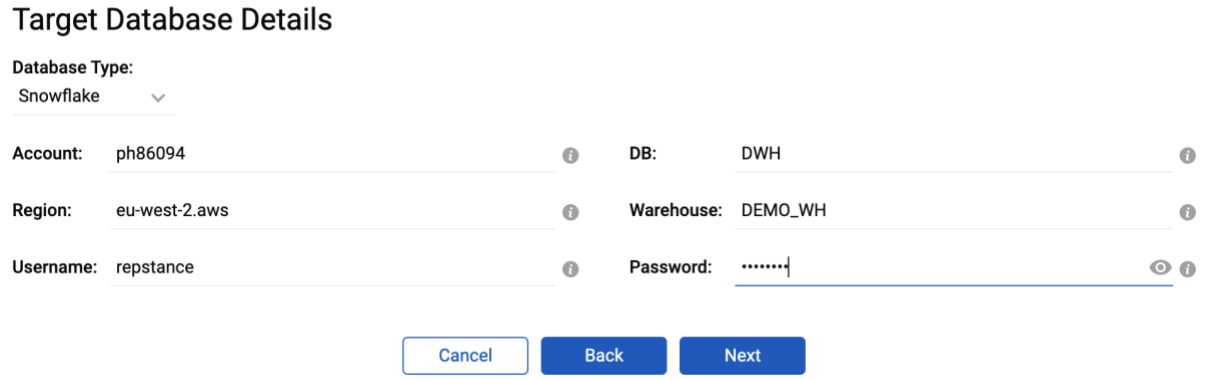
Press "Next" to view the report of the tables to be included into the replication along with the transformation details:



Press “Back” if anything needs to be adjusted or “Next” to complete the Capture Process configuration.

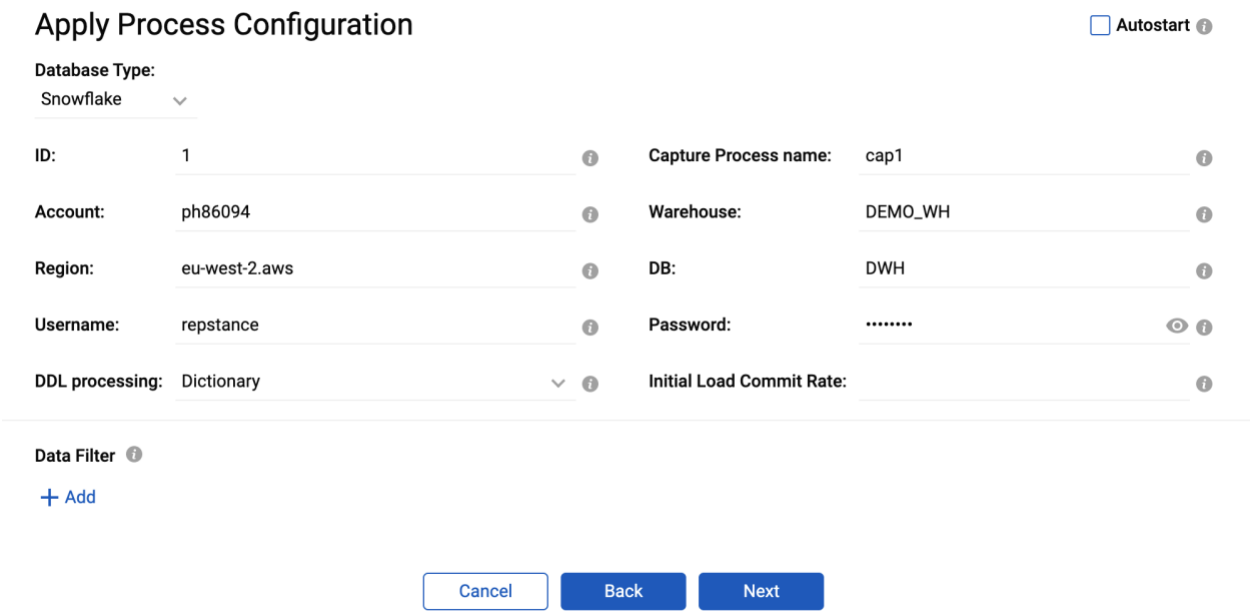
On the next form click “Yes”, if Capture Process needs to start immediately.

The next step is to configure the Target Database. At this step Wizard Process validates the Target Database and prepares it, if required (see chapter 8.2.1 Database Configuration for details).



Depending on the database type the different set of the parameters should be provided (see chapter 8.2.1 Database Configuration for the details).

Clicking “Next” validates Target Database and prepares it, if required, and moves to the Apply Process configuration step:



Repstance fills out database settings and the Capture Process name from the previous forms, which can be changed if desired.

It is recommended to use *`Dictionary`* value for the **`DDL processing`** parameter if the Source and Target Databases are of different types, otherwise the *`Native`* value to be preferable.

The **`Initial Load Commit rate`** and the **`Data Filter`** parameters are described in chapter 8.2.2.2 Create Apply Process.

On the next form click “Yes” to run the Process immediate.

The Wizard Mode is completed. As a result, the Capture and Apply Processes are created and once they are run the replication from Source Database to Target Database is started.

9. REPSTANCE SERVER MAINTENANCE

As Repstance is a pre-configured Linux service it starts/stops together with the Server and MUST be maintained using the Linux Service Command.

9.1 How to Stop Repstance Services

In order to stop the Repstance Server the "service repstance stop" must be used:

```
# service repstance stop
Stopping repstance:          [ OK ]
```

9.2 How to Start Repstance Services

In order to start Repstance Server the "service repstance start" must be used:

```
# service repstance start
Starting repstance:          [ OK ]
```

9.3 How to Check Repstance Service Status

In order to check Repstance Server status the "service repstance status" must be used:

```
# service repstance status
repstance (pid 3033) is running...
```

9.4 Backup Repstance Files

Regularly backup the Trail Files which are located in `/opt/repstance/trail` directory along with the Configuration Files, which are located in `/opt/repstance/conf` and Capture Checkpoint Files in `/opt/repstance/captureckpt.*` and lastly make sure that you record the last applied checkpoint in a file of you choosing.

9.5 Housekeeping

Schedule regular jobs which are responsible for cleaning up "old" Trail and Log Files which are found under these directories:

- `/opt/repstance/trail` – Trail Files
- `/opt/repstance/log` – Log Files

GLOSSARY

Apply Checkpoint – The last LSN identifier for any data change applied.

Apply Process – Is the "Process" that inserts the extracted Data into the specified Target Database.

Capture Checkpoint – The last LSN identifier for any data change captured.

Capture Process – Is the "Process" that extracts Data from the specified Source Database.

DDL – Type of the Database Statement, that is used to provide or modify a Dictionary Definition of a database object.

DML – Type of the Database Statement, that is used to modify Data in the tables.

LSN – Every record in Trail Files is identified by a Log Sequence Number generated by SQL Server CDC functionality.

Process – This can be either Capture or Apply Process.

SCN – every record in the Trail Files is identified by a System Change Number generated by Oracle.

Source Database – Is the Database from which you wish to extract/copy Data.

Target Database – Is the Database into which you wish to insert/copy Data.

Trail Files – These are the "Source Files" – containing Data extracted by a Capture Process and used by an Apply Process.

Transformation Rules – these are the rules provided by user to change/control transactions processing behavior.