

Redshift Artifact Extraction Prerequisites

Contents

1. Introduction	3
2. Artifact Extraction.....	3
2.1 DDL Scripts	3
2.1.1 Table DDLs.....	3
Table DDLs using Python	3
Table DDLs using Shell Script	4
2.1.2 View DDLs	5
View DDLs using Python	5
View DDLs using Shell Script.....	6
2.2 Query Execution Logs	7
2.2.1 Using UNLOAD Command.....	7
2.2.2 Using AWS Redshift Query Editor	8
2.3 Other Database Objects	10
2.4 Stored Procedure and Function DDL Extraction	13
2.4.1 Stored Procedure DDLs	14
Stored Procedure DDLs using Python.....	14
Stored Procedure DDLs using Shell Script	15
2.4.2 User Defined Function DDLs	16
2.5 Other Scripts and Artifacts	16
3. Getting Help.....	17

1. Introduction

LeapLogic's Assessment profiles existing inventory, identifies complexity, performs dependency analysis, and provides recommendations for migration to modern data platform.

2. Artifact Extraction

LeapLogic requires certain artifacts to perform an assessment. As a prerequisite, you need to have a super user privilege to fetch the required data. You can copy them from your GIT instance or the Redshift repository where all the artifacts such as DDL scripts, stored procedures, functions, query execution logs, DML scripts, and other database objects are stored. LeapLogic needs all these artifacts in the form of .sql files.

2.1 DDL Scripts

2.1.1 Table DDLs

There are two options available for extracting table DDL scripts. The first option is to use the Python script. The second option is to use the shell script with psql.

Note: To use the Python script, you need the redshift-connector module. For the Shell script, you need a psql client.

Table DDLs using Python

Install the “redshift_connector” module using Python pip command.

```
pip install redshift_connector
```

Use Python pip3 command if you have multiple Python versions installed.

```
pip3 install redshift_connector
```

Use the attached Python script and put it on your local machine or server. Run it from there to get the table DDLs.

```
#Connect to the cluster
import redshift_connector

conn = redshift_connector.connect(
    host='redshift_host',
    database='database',
    port=5439,
    user='username',
    password='password'
)

output_file = r'/path/table_ddl_output.sql'
```

```
# Create a Cursor object
cursor = conn.cursor()

# Query a table using the Cursor
res = cursor.execute("""          SELECT DISTINCT SCHEMAName ||'.'||tablename
FROM pg_tables
WHERE schemaname NOT IN ('pg_catalog','information_schema') limit
10""").fetchall()

for value in res:
    print(value[0])
    res2 = cursor.execute("""show table """ + value[0] + """
""").fetchone()
    # Write the result to the file
    with open(output_file, 'a') as f:
        f.write('-- Table: ' + value[0] + '\n')
        f.write(str(res2[0]) + '\n\n')
```

Modify the details highlighted in **Yellow**. This script exports all the table DDLs. Please run this script for every database in the Redshift cluster.

Additional Note: Please remember to remove the “**limit 10**” clause from the above script before executing it.

Also attached Python script in the form of text file.



Table DDLs using Shell Script

Use the attached Shell script and put it on your local machine or server. Run it from there to get the table DDLs.

Note: Run the script on a machine that has the psql client on it.

```
#!/bin/bash

# Database connection parameters
export PGHOST="redshift-host"
export PGDATABASE="database"
export PGPORT=5439
export PGUSER="username"
export PGPASSWORD="password"

# Output file
OUTPUT_FILE="/path/output/table_ddl_output.sql"

# Query to get view names
QUERY="SELECT DISTINCT SCHEMAName ||'.'||tablename FROM pg_tables WHERE
schemaname NOT IN ('pg_catalog','information_schema') limit 10;"

# Execute the query and save the result in a variable
TABLES=$(psql -AXqtc "$QUERY")
```

```
# Loop over the views
for TABLE in $TABLES
do
    # Query to get the view definition
    echo "Exporting Table DDL: $TABLE..."
    QUERY="SHOW TABLE $TABLE;"

    # Execute the query and save the result in a variable
    TABLE_DEFINITION=$(psql -AXqtc "$QUERY")

    # Write the view name and its definition to the output file
    echo "-- Table: $TABLE" >> $OUTPUT_FILE
    echo "$TABLE_DEFINITION" >> $OUTPUT_FILE
    echo "" >> $OUTPUT_FILE
done
```

Modify the details highlighted in **Yellow**. This script exports all the table DDLs. Please run this script for every database in the Redshift cluster.

Also attached shell script in the form of text file.



2.1.2 View DDLs

There are two options available for extracting the View DDL scripts. The first option is to use the Python script. The second option is to use the shell script with psql.

Note: To use the Python script, you need redshift-connector module. For the shell script, you need a psql client.

View DDLs using Python

Use the attached Python script and put it on your local machine or server. Run it from there to get the table DDLs.

```
#Connect to the cluster
import redshift_connector

conn = redshift_connector.connect(
    host='redshift_host',
    database='database',
    port=5439,
    user='username',
    password='password'
)

output_file = r'/path/view_ddl_output.sql'

# Create a Cursor object
cursor = conn.cursor()
```

```
# Query a table using the Cursor
res = cursor.execute(""" SELECT DISTINCT SCHEMAname || '.' || viewname FROM
pg_views
WHERE schemaname NOT IN ('pg_catalog','information_schema') limit
10""").fetchall()

for value in res:
    print(value[0])
    res2 = cursor.execute("""show view """ + value[0] + """ """).fetchone()
    # Write the result to the file
    with open(output_file, 'a') as f:
        f.write('-- Table: ' + value[0] + '\n')
        f.write(str(res2[0]) + '\n\n')
```

Modify the details highlighted in **Yellow**. This script exports all the view DDLs. Please run this script for every database in the Redshift cluster.

Also attached Python script in the form of text file.



view_ddl.txt

View DDLs using Shell Script

Take the shell script that's attached and put it on your local machine or server. Run it from there to get the table DDLs.

Note: Run the script on a machine that has the psql client on it.

```
#!/bin/bash

# Database connection parameters
export PGHOST="redshift-host"
export PGDATABASE="database"
export PGPORT=5439
export PGUSER="username"
export PGPASSWORD="password"

# Output file
OUTPUT_FILE="/path/output/view_ddl_output.sql"

# Query to get view names
QUERY="SELECT DISTINCT schemaname || '.' || viewname FROM pg_views WHERE
schemaname NOT IN ('pg_catalog','information_schema') LIMIT 10;"

# Execute the query and save the result in a variable
VIEWS=$(psql -AXqt " $QUERY")

# Loop over the views
for VIEW in $VIEWS
do
    # Query to get the view definition
    echo "Exporting view DDL: $VIEW..."
    QUERY="SHOW VIEW $VIEW;"

    # Execute the query and save the result in a variable
```

```

VIEW_DEFINITION=$(psql -AXqtc "$QUERY")

# Write the view name and its definition to the output file
echo "-- View: $VIEW" >> $OUTPUT_FILE
echo "$VIEW_DEFINITION" >> $OUTPUT_FILE
echo "" >> $OUTPUT_FILE
done

```

Modify the details highlighted in **Yellow**. This script exports all the View DDLs. Please run this script for every database in the Redshift cluster.

Also attached Shell script in the form of text file.



2.2 Query Execution Logs

Similarly, for extracting the query execution logs, there are two options available. The first option leverages the UNLOAD command while the second option uses the AWS Redshift Query Editor.

Note: We recommend using the UNLOAD command, especially when the file size is substantial.

2.2.1 Using UNLOAD Command

The UNLOAD command generates the required data file on an S3 bucket. Please see the prerequisites below.

- 's3://bucket_name/path' – The S3 path where files are expected to be generated.
- iam_role 'arn:aws:iam::<aws acct num>:role/<redshift role>' – IAM role of the Redshift cluster.

To export the required query execution logs, use the following UNLOAD command as a reference. Change the start date trunc(sqlog.starttime) between '2024-03-29' and '2024-04-01' as required. All the required inputs are marked in **Yellow** in the below script.

```

unload
(
$$
SELECT
    'IDWWM' || '~' ||
    coalesce(sui.username, '') || '~' ||
    0 || '~' ||
    'client_u' || '~' ||
    NULLIF(sqlog.starttime, '9999-12-31') ::varchar || '~' ||
    coalesce(sqm.query_cpu_time, 0) ::varchar || '~' ||
    0 || '~' ||
    coalesce(x.byts, 0) ::varchar || '~' ||
    coalesce(datediff(millisecond, sc.endtime, sc.starttime), 0) ::varchar || '~' ||
    NULLIF(sqlog.starttime, '9999-12-31') ::varchar || '~' ||
    NULLIF(sqlog.starttime, '9999-12-31') ::varchar || '~' ||

```

```

coalesce(sqlog.pid,999) ::varchar|| '~' ||
coalesce(sqlog.query,999) ::varchar|| '~' ||
'NA' ::VARCHAR|| '~' ||
'NA' ::VARCHAR|| '~' ||
coalesce(max(sqm.query_cpu_time) OVER(Partition by sqm.query ORDER BY NULL ROWS
BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW),0)::varchar || '~' ||
coalesce(x.max_bytes,0)::varchar || '~' ||
coalesce(sqm.query_execution_time,0) ::varchar || '~' ||
'Not Present' || '~' ||
ss.sequence::varchar || '~' ||
coalesce(ss.type,'') || '~' ||
'' || '~' ||
coalesce(sqlog.label,'') || '~' ||
ss.text ::VARCHAR(MAX)
from
SVL_STATEMENTTEXT ss
LEFT JOIN
SVL_QLOG sqlog
on ss.xid = sqlog.xid and
ss.pid = sqlog.pid and
ss.userid = sqlog.userid
LEFT JOIN
SVL_QUERY_METRICS sqm
on sqm.query = sqlog.query
LEFT JOIN
svl_compile sc
ON sqm.query = sc.query and
sqlog.pid = sc.pid and
sqlog.xid = sc.xid
Left JOIN
SVL_USER_INFO sui
ON sui.usesysid = sqm.userid
LEFT JOIN
( select query,sum(bytes) as byt,coalesce(max(bytes),0) as max_bytes from
SVL_QUERY_SUMMARY
WHERE userid > 1
group by query) X
on sqlog.query = x.query
WHERE
ss.userid > 1
and sqlog.userid > 1
and cast( ss.starttime AS date) between '2023-12-01' and '2024-04-30' $$)
to 's3://bucket_name/path'
iam role 'arn:aws:iam::<aws acct num>:role/<redshift role>'
DELIMITER '|'
GZIP
ALLOWOVERWRITE;

```

Note: The UNLOAD command runs in parallel which essentially means it generates multiple files in the S3 bucket.

2.2.2 Using AWS Redshift Query Editor

To export the required query execution logs, refer to the below script. Please remember to change the start date `trunc(sqlog.starttime) between '2024-03-29' and '2024-04-01'`; as required.

The total recommended timeframe is of three months for the end-to-end query logs. Next, execute the given script below using the AWS Redshift Query Editor UI and download the output after execution.

```
SELECT
    'IDWWM' || '~~' ||
    coalesce(sui.username, '') || '~~' ||
    0 || '~~' ||
    'client_u' || '~~' ||
    NULLIF(sqlog.starttime, '9999-12-31') ::varchar || '~~' ||
    coalesce(sqm.query_cpu_time, 0) ::varchar || '~~' ||
    0 || '~~' ||
    coalesce(x.bytt, 0) ::varchar || '~~' ||
    coalesce(datediff(millisecond, sc.endtime, sc.starttime), 0) ::varchar || '~~' ||
    NULLIF(sqlog.starttime, '9999-12-31') ::varchar || '~~' ||
    NULLIF(sqlog.starttime, '9999-12-31') ::varchar || '~~' ||
    coalesce(sqlog.pid, 999) ::varchar || '~~' ||
    coalesce(sqlog.query, 999) ::varchar || '~~' ||
    'NA' ::VARCHAR || '~~' ||
    'NA' ::VARCHAR || '~~' ||
    coalesce(max(sqm.query_cpu_time) OVER(Partition by sqm.query ORDER BY NULL ROWS
    BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW), 0) ::varchar || '~~' ||
    coalesce(x.max_bytes, 0) ::varchar || '~~' ||
    coalesce(sqm.query_execution_time, 0) ::varchar || '~~' ||
    'Not Present' || '~~' ||
    ss.sequence ::varchar || '~~' ||
    coalesce(ss.type, '') || '~~' ||
    '' || '~~' ||
    coalesce(sqlog.label, '') || '~~' ||
    ss.text ::VARCHAR(MAX)
from
    SVL_STATEMENTTEXT ss
LEFT JOIN
    SVL_QLOG sqlog
on ss.xid = sqlog.xid and
ss.pid = sqlog.pid and
ss.userid = sqlog.userid
LEFT JOIN
    SVL_QUERY_METRICS sqm
on sqm.query = sqlog.query
LEFT JOIN
    svl_compile sc
ON sqm.query = sc.query and
sqlog.pid = sc.pid and
sqlog.xid = sc.xid
Left JOIN
    SVL_USER_INFO sui
ON sui.usersysid = sqm.userid
LEFT JOIN
    ( select query, sum(bytes) as bytt, coalesce(max(bytes), 0) as max_bytes from
    SVL_QUERY_SUMMARY
WHERE userid > 1
group by query) X
on sqlog.query = x.query
WHERE
    ss.userid > 1
and sqlog.userid > 1
and cast( ss.starttime AS date) between '2023-12-01' and '2024-04-30';
```

2.3 Other Database Objects

For better assessment results of your environment and workloads, we recommend exporting additional database objects as well. Please refer to the below script to export the data as separate delimited files.

```
----Database objects: we need to run the Below query for every database using  
Super user credentials  
  
SELECT n.nspname AS schema_name  
  
  , CASE WHEN c.relkind = 'v' THEN 'view' when c.relkind = 'i' THEN 'index' ELSE  
'table' END  
    AS table_type  
  
  , count(c.relname)  
  
FROM pg_class As c  
LEFT JOIN pg_namespace n ON n.oid = c.relnamespace  
LEFT JOIN pg_tablespace t ON t.oid = c.reltablespace  
LEFT JOIN pg_description As d  
      ON (d.objoid = c.oid AND d.objsubid = 0)  
WHERE  
  n.nspname  not in ('information_schema', 'pg_catalog')  
  
group BY n.nspname, CASE WHEN c.relkind = 'v' THEN 'view' when c.relkind = 'i'  
THEN 'index' ELSE 'table' END  
  
UNION  
SELECT  
  n.nspname,  
  'Stored_procedure' as table_type  
  
  ,count(p.prosrc)  
FROM  
  pg_catalog.pg_namespace n  
JOIN pg_catalog.pg_proc p ON  
  pronamespace = n.oid  
join pg_catalog.pg_user b on  
  b.usesysid = p.proowner  
where  
  nspname not in ('information_schema',  
  'pg_catalog')  
group by      n.nspname, table_type  
  
---Databases:  
  select 'Databases',count(*) from pg_database  
  UNION  
  select 'Schemas',count(*) from pg_namespace where nspname  not in  
  ('information_schema',  
  'pg_catalog')  
  
----High Data Volume Tables  
  
WITH tbl_ids AS  
(
```

```

SELECT DISTINCT oid
FROM pg_class c
WHERE --relowner > 1
      relkind = 'r'
),
pcon AS
(
  SELECT conrelid,

         CASE
           WHEN SUM(
             CASE
               WHEN contype = 'p' THEN 1
               ELSE 0
             END
           ) > 0 THEN 'Y'
           ELSE 'N'
         END pk,
         CASE
           WHEN SUM(
             CASE
               WHEN contype = 'f' THEN 1
               ELSE 0
             END
           ) > 0 THEN 'Y'
           ELSE 'N'
         END fk,
         conname
  FROM pg_constraint
  WHERE conrelid > 0
  AND   conrelid IN (SELECT oid FROM tbl_ids)
  GROUP BY conrelid,conname
)
SELECT
database
,SCEMA as schemaname
,"table" AS tablename
,tbl_rows as num_rows
,size AS size_mb
,pcon.pk
,pcon.conname
FROM
svv_table_info ti
LEFT JOIN pcon ON pcon.conrelid = ti.table_id
WHERE ti.SCEMA !~ '^information_schema|catalog_history|pg_' and size_mb =
10000

---Data for Partitioning / Bucketing
SELECT
database
,SCEMA as schemaname
,"table" AS table_name
,size AS size_mb
,tbl_rows as num_rows

,pg.attname column_name
,' ' as num_unique_values
FROM
svv_table_info ti

```

```

inner JOIN pg_attribute pg ON pg.attrelid = ti.table_id
WHERE ti.SCHEMA !~ '^information_schema|catalog_history|pg_'

---Count of stored procedures and Functions
select database_name,schema_name,function_type,count(*) from
SVV_REDSHIFT_FUNCTIONS WHERE schema_name !~
'^information_schema|catalog_history|pg_'
group by database_name,schema_name,function_type

---List of stored procedure
select
database_name
,schema_name
,function_type
,function_name
from
SVV_REDSHIFT_FUNCTIONS
where
    schema_name not in ('information_schema',
        'pg_catalog')

---Count of external Tables/Views in Redshift

select
redshift_database_name
,schemaname
,tabletype
,count(tablename)
from
SVV_EXTERNAL_TABLES
group by
redshift_database_name
,schemaname
,tabletype

---List of External Tables
select
redshift_database_name
,schemaname
,tabletype
,tablename
from
SVV_EXTERNAL_TABLES

---Total I/O Usage by Days
SELECT trunc(start_time) as RUNDATE,sum(local_read_IO+remote_read_IO)as
TOTALIOREADS
FROM SYS_QUERY_DETAIL
where trunc(start_time) between '2024-03-28' and '2024-04-03'
group by trunc(start_time)

```

Note: '2024-03-28' and '2024-03-31' Please change the Date range To the period of High usage and of minimum 30 Days.

----Database Volume:



v_space_used_per_tbl.sql

We need to create the above view in any schema and then execute the Below query to pull the details.

```
SELECT
dbase_name
,schemaname
,SUM(megabytes) as total_mb
```

```
FROM
    public.v_space_used_per_tbl
GROUP BY
dbase_name
,schemaname
```

--Distinct application name

```
select distinct application_name
from pg_catalog.stl_connection_log where (recordtime between '2024-05-01' and
'2024-05-30' )
and application_name is not null
```

Note: '2024-05-01' and '2024-05-30' Please change the Date range To the period of High usage and of minimum 30 Days.

--Distinct client ID

```
select distinct client_id from
pg_catalog.stl_network_throttle where (log_time between '2024-05-01' and
'2024-05-30' )
```

Note: '2024-05-01' and '2024-05-30' Please change the Date range To the period of High usage and of minimum 30 Days.

-- The date range is subject to change - as needed for query log assessment.

-- (15day/1month/6months whatever is possible for extraction)

2.4 Stored Procedure and Function DDL Extraction

2.4.1 Stored Procedure DDLs

There are two options available for extracting Redshift stored procedure DDL scripts. The first option is to use the Python script. The second option is to use the Shell script with psql.

Note: To use the Python script, you need redshift-connector module. For the shell script, you need a psql client.

Stored Procedure DDLs using Python

Take the Python script that's attached and put it on your local machine or server. Run it from there to get the table DDLs.

```
#Connect to the cluster
import redshift_connector

conn = redshift_connector.connect(
    host='redshift_host',
    database='database',
    port=5439,
    user='username',
    password='password'
)

output_file = r'/path/sp_ddl_output.sql'

# Create a Cursor object
cursor = conn.cursor()

# Query a table using the Cursor
res = cursor.execute("""      select
database_name ||'.'||
schema_name ||'.'||
function_name ||'('||
argument_type ||')'
from
SVV_REDSHIFT_FUNCTIONS
where
    schema_name not in ('information_schema',
'pg_catalog')
AND function_type = 'STORED PROCEDURE'""").fetchall()

for value in res:
    print(value[0])
    res2 = cursor.execute("""show procedure """ + value[0] + """
""").fetchone()
    # Write the result to the file
    with open(output_file, 'a') as f:
        f.write('-- Procedure: ' + value[0] + '\n')
        f.write(str(res2[0]) + '\n\n')
```

Modify the details highlighted in **Yellow**. This script exports all the stored procedure DDLs. Please run this script for every database in the Redshift cluster.

Also attached Python script in the form of text file.



proc_ddl_export.txt

Stored Procedure DDLs using Shell Script

Take the shell script that's attached and put it on your local machine or server. From there, you can run it to get the table DDLs.

Note: Run the script on a machine that has the psql client on it.

```
#!/bin/bash

# Database connection parameters
export PGHOST="redshift-host"
export PGDATABASE="database"
export PGPORT=5439
export PGUSER="username"
export PGPASSWORD="password"

# Output file
OUTPUT_FILE="/path/output/sp_ddl_output.sql"

# Query to get view names
QUERY="select database_name || '.' || schema_name || '.' || function_name
|| '(' || replace(replace(replace(replace(argument_type, '-', ''),
'charactervarying', 'varchar'), 'timestampwithouttimezone', 'timestamp'
), 'binaryvarying', 'varbyte' ), 'timestampwithtimezone', 'timestampz' )
|| ')' from SVV_REDSHIFT_FUNCTIONS where schema_name not in
('information_schema', 'pg_catalog') AND function_type = 'STORED
PROCEDURE'"

# Execute the query and save the result in a variable
SPS=$(psql -AXqt "$QUERY")

# Loop over the views
for SP in $SPS
do
    # Query to get the view definition
    echo "Exporting Procedure DDL: $SP..."
    QUERY="SHOW PROCEDURE \"$SP\";"

    # Execute the query and save the result in a variable
    SP_DEFINITION=$(psql -AXqt "$QUERY")

    # Write the view name and its definition to the output file
    echo "-- Stored Procedure: $SP" >> $OUTPUT_FILE
    echo "$SP_DEFINITION" >> $OUTPUT_FILE
    echo "" >> $OUTPUT_FILE
done
```

Modify the details highlighted in Yellow. This script exports all the stored procedure DDLs. Please run this script for every database in the Redshift cluster.

Also attached the Shell script in the form of text file.



sp_ddl.sh

2.4.2 User Defined Function DDLs

There are two options available for extracting user-defined function DDL scripts. The first option is to use the AWS Redshift console query editor. The second option is to use any Redshift client to execute and export the results of the query.

Please follow the below steps to export the DDL scripts using the query editor from the AWS Query Editor.

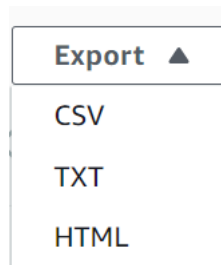
- Click **Amazon Redshift** from the AWS Console.
- Select the Redshift cluster from the **Cluster Overview** tab.
- Click **Query Data** highlighted in Orange on the right side.
- Next, click **Query in query Editor**.

Query in query editor

- The Query Editor opens where the below attached procedure can be compiled along with the other steps as mentioned below.

- Execute the SELECT statement. When the query execution is complete, click

Export ▼



- a. To export the required DDLs, refer to the below script.

Note: Please execute the below query using super user for any schema.



generate_udf_ddls.
sql

2.5 Other Scripts and Artifacts

Copy any other scripts such as DML scripts etc. from your GIT and share them with the LeapLogic team to produce more extensive insights.

3. Getting Help

Contact LeapLogic technical support at info@leaplogic.io