

Broadcom CA Test Data Manager and Amazon Redshift Database

Continuous Testing Solution Engineering Team

DRAFT version 0.5

August, 2021



Table of Contents

Introduction	3
TDM Architecture Diagram	3
Broadcom CA Test Data Manager and Amazon Redshift Demo Overview	4
Synthetic Data Generation	4
Masking	5
Demo Setup	6
Synthetic Data Generation Detail	7
Masking Detail	13

DRAFT

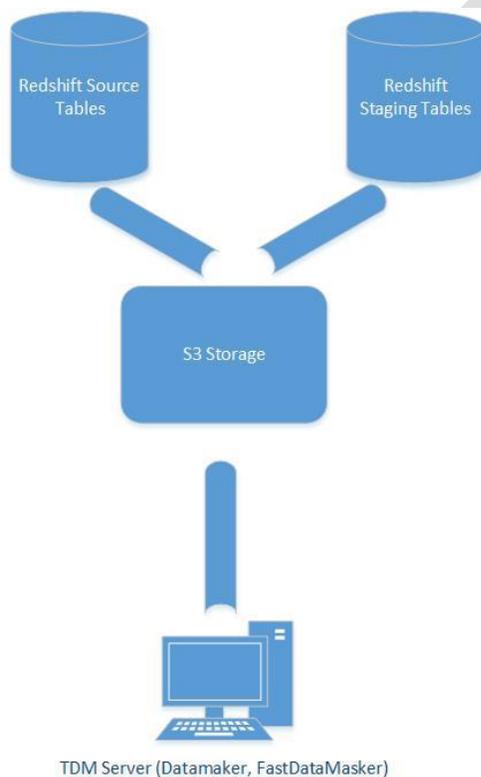
Introduction

Amazon Redshift comes from a PostgreSQL foundation, but the interface has significantly changed in recent years, so that the standard PostgreSQL drivers no longer work with it. Amazon has created their own Redshift ODBC/JDBC drivers, that are not supported with TDM. Therefore, Redshift capabilities in TDM is limited to whatever use cases that can be implemented using a bulk load/unload methodology (Synthetic Data Generation, Masking).

The steps described below are for a PROOF OF CONCEPT implementation.

TDM Architecture Diagram

The below diagram shows a basic TDM – Amazon Redshift deployment architecture.



Broadcom CA Test Data Manager and Amazon Redshift Demo Overview

There are two TDM Use Cases that are suitable for Amazon Redshift – Synthetic Data Generation and Masking. Details about the steps required to enable these use cases follow:

Synthetic Data Generation

Amazon's documentation itself recommends [using bulk loading](#) for large quantities of information:

"We strongly recommend using the COPY command to load large amounts of data. Using individual INSERT statements to populate a table might be prohibitively slow."

This document describes how to setup & execute the use cases with Test Data Manager.

The options for the COPY do not include a simple upload of a delimited file. Therefore, we'll use the [load from S3](#) option to demonstrate this capability.

S3 Buckets can be [created via the Amazon CLI](#)

Followed by a [AWS S3 CLI mv command](#) to load the files from local storage to the S3 Bucket:

Followed by a [RedShift COPY command](#)

Followed by a [Delete Bucket](#) (once the COPY is complete into Redshift) - unless you want this to always be available instead of transient

Masking

To mask Redshift data, one will need to Unload, Mask, and Upsert the data tables. The tables/fields to mask will need to be manually identified (no PII audit scan is available due to driver incompatibility). The Test Data Engineer will then be very selective determining how much data to extract to minimize the data flow.

NOTE: You will need to export key row identifier(s) along with the columns to mask to ensure the Upsert/Merge can take place after masking.

Redshift table data would need to be exported to a delimited file format that Fast Data Masker can process.

Unload the data you wish to mask

Redshift provides a [number of options](#) for export/unload. We will use the unload to S3 option for this exercise. Any of the [delimited file options](#) should work with FDM. Once the delimited files are in the S3 bucket, you can use the [AWS S3 CLI mv command](#) to move them locally for processing.

Mask the data

Now that the files are in a local directory, follow standard Fast Data Masker methods to mask delimited files. The output will be .scramble files.

Upsert of the data will require a multi-step process

- (1) Use the [AWS S3 CLI mv command](#) to upload the .scramble files into an S3 Bucket (as you did for the Synthetic Data Generation above)
- (2) Use the [Redshift COPY](#) command to copy the data into Staging Table(s)
- (3) Use the [documented best practices to Upsert](#) the data

Demo Setup

Pre-requisites:

Amazon AWS ID, Tutorial database available (for this proof, we'll use the [Tutorial: Loading data from Amazon S3](#) to setup our testbed), and the [AWS CLI v2 client](#) downloaded, installed, and configured on the TDM Server.

Install & Configure:

In order for TDM to work with the RedShift data structures, we need to identify the most efficient way to catalog those structures. You'll want to use the export to Delimited Files function to get .csv files to register into TDM.

Extract Table Structures for Masking Setup

If you still have the S3 bucket from the Tutorial installation, we'll reuse it. For each table that has columns that require masking of PII, create & execute commands to export the data structure (with a single header line, a single data line, and including the primary key/unique identifier of the row) to the S3 Bucket. Example for customer table, where we'll just mask the customer name (in the c_name column):

```
unload ('select c_custkey,c_name from customer where c_custkey = 1')
to 's3://broadcom-tdm-bucket/unload/customer_'
ACCESS_KEY_ID 'YOURKEYID'
SECRET_ACCESS_KEY 'YOURACCESSKEY'
header
CSV
parallel off;
```

Repeat for each table & field that will need to be masked. Once you've completed this, use the AWS CLIv2 to move the files from the bucket to a local directory:

```
aws s3 mv s3://broadcom-tdm-bucket/unload . --recursive
```

(If you have not already setup the [AWS CLI config files](#), do so before executing this command)

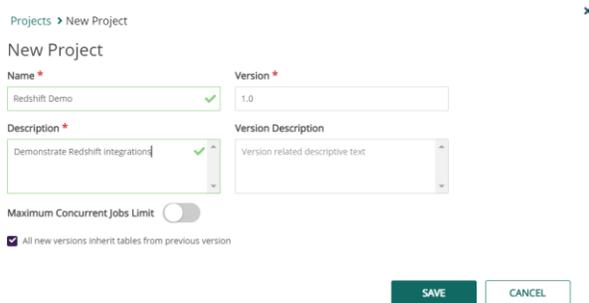
Extract Table Structures for Data Generation

Use "select *" to unload all the columns as the generator will need to generate all. Unload into a separate prep directory for registration.

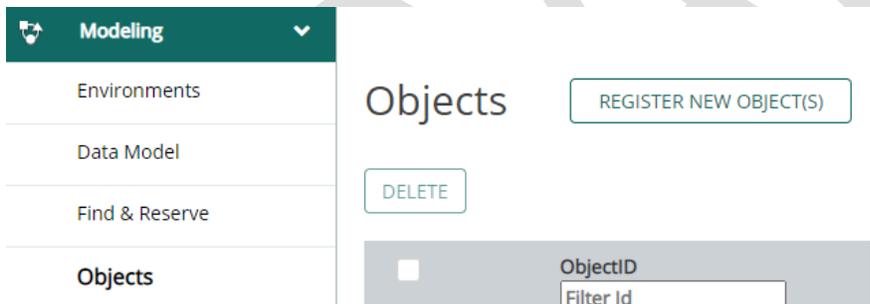
Synthetic Data Generation Detail

Project Configuration

Use TDM Portal to create a new Project using the New Project Wizard on the homepage



Select Modeling, then Objects, then click the Register New Objects button



Change to Delimited(CSV) and select the files you've downloaded (you may need to add a .csv extension to allow for registration).

Register New Object(s)

Object Type *

File(s) to Upload *

Drag and Drop file (or click)



File Name	Table Name	Status
customer_000.csv	customer_000	New Table
dwwdate_000.csv	dwwdate_000	New Table
lineorder_000.csv	lineorder_000	New Table
part_000.csv	part_000	New Table
supplier_000.csv	supplier_000	New Table

Click the Register button.

Create a new generator:

[Generators](#) > [Create New Generator](#)

Create New Generator

Name *

Description *

Choose Data Generation functions that align with the data in the existing tables. For the unique ids, choose a starting number that is unique/much larger than the existing number to indicate the data is generated. Sample functions follow:

```

customer_000  c_custkey      9@leftpad(~NEXT~,0,4)@
customer_000  c_name
               @left(@randlov(0,@seedlist(Companies))@@randlov(0,@seedlist(Flowers))@,25)@
customer_000  c_address
               @left(@randrange(1,9999)@
               @percval(10%N,5%North,10%E,5%East,10%S,5%South,10%W,5%West,40%)@
               @percval(10%Second St.,10%Main St.,10%Park Ave.,10%Oak St.,10%Pine St.,10%Maple
               Ln.,10%Washington St.,10%Lake Dr.,10%Hill Ave.,10%Ninth St.)@,25)@
customer_000  c_city          @randlov(0,@seedlist(US Zip-Codes))@,3)@
customer_000  c_nation        US
customer_000  c_region        @randlov(0,@seedlist(US Zip-Codes))@,2)@
customer_000  c_phone         @randlov(0,@seedlist(US Phone no))@)@
customer_000  c_mktsegment
               @percval(20%BUILDING,20%FURNITURE,20%AUTOMOBILE,20%HOUSEHOLD,20%MACHINERY
               )@

dwdate_000   d_datekey       @date(~CDATE-4000~,YYYYMMDD)@
dwdate_000   d_date          @string(^d_datekey^,mmmmmmm dd yyyy)@
dwdate_000   d_dayofweek
               @case(@dow(^d_datekey^)=1,Sunday,@dow(^d_datekey^)=2,Monday,@dow(^d_dateke
               y^)=3,Tuesday,@dow(^d_datekey^)=4,Wednesday,@dow(^d_datekey^)=5,Thursday,@dow(^d
               _datekey^)=6,Friday,@dow(^d_datekey^)=7,Saturday)@
dwdate_000   d_month         @string(^d_datekey^,mmmmmmm)@
dwdate_000   d_year          @string(^d_datekey^,yyyy)@
dwdate_000   d_yearmonthnum @string(^d_datekey^,YYYYMM)@
dwdate_000   d_yearmonth     @string(^d_datekey^,mmmyyyy)@
dwdate_000   d_daynuminweek @dow(^d_datekey^)@
dwdate_000   d_daynuminmonth @string(^d_datekey^,dd)@
dwdate_000   d_daynuminyear
dwdate_000   d_monthnuminyear
dwdate_000   d_weeknuminyear

```

```

dwdate_000    d_sellingseason
dwdate_000    d_lastdayinweekfl
dwdate_000    d_lastdayinmonthfl
dwdate_000    d_holidayfl
dwdate_000    d_weekdayfl

lineorder_000 lo_orderkey  ~NEXT~
lineorder_000 lo_linenum    1
lineorder_000 lo_custkey   ^customer_000.c_custkey(1)^
lineorder_000 lo_partkey   ^part_000.p_partkey(1)^
lineorder_000 lo_suppkey   ^supplier_000.s_suppkey(1)^
lineorder_000 lo_orderdate ^dwdate_000.d_datekey(1)^
lineorder_000 lo_orderpriority @percval(50%2-HIGH,50%5-LOW)@
lineorder_000 lo_shippriority 0
lineorder_000 lo_quantity  @randrange(1,100)@
lineorder_000 lo_extendedprice @randrange(5,5000)@
lineorder_000 lo_ordertotalprice @multiply(^lo_quantity^,^lo_extendedprice^)@
lineorder_000 lo_discount  @randrange(2,20)@
lineorder_000 lo_revenue    @subtract(^lo_extendedprice^,@multiply(^lo_extendedprice^,@multiply(@subtract(100,^lo_discount^),.01)@)@)@
lineorder_000 lo_supplycost @multiply(^lo_extendedprice^,.20)@
lineorder_000 lo_tax        @randrange(2,12)@
lineorder_000 lo_commitdate @randdate(@adddays(^lo_orderdate^,2)@,@adddays(^lo_orderdate^,90)@)@
lineorder_000 lo_shipmode  @percval(20%MAIL,20%REG AIR,20%FOB,20%AIR,20%TRUCK)@

part_000      p_partkey   9@leftpad(~NEXT~,0,4)@
part_000      p_name      @left(@randlov(0,@seedlist(Fruit)@)@@randlov(0,@seedlist(Name)@)@,22)@

```

```
part_000    p_mfgr      MFR@randtext(3,3,UPPER)@
part_000    p_category  @left(@randlov(0,@seedlist(Flowers)@),7)@
part_000    p_brand1    ABC
part_000    p_color     @left(@randlov(0,@seedlist(Flowers)@),11)@
part_000    p_type      @percval(34%MODULE,33%COMPONENT,33%PART)@
part_000    p_size      @percval(34%SMALL,33%MEDIUM,33%LARGE)@
part_000    p_container @percval(10%JUMBO PKG,10%MED BAG,10%JUMBO CAN,10%MED
DRUM,10%WRAP PKG,10%WRAP DRUM,10%LG PACK,10%SM JAR,10%MED PKG,10%LG JAR)@

supplier_000 s_suppkey  9@leftpad(~NEXT~,0,5)@
supplier_000 s_name     @left(@randlov(0,@seedlist(Stocks)@),2),25)@
supplier_000 s_address  @left(@randrange(1,9999)@
@percval(10%N.,5%North,10%E.,5%East,10%S.,5%South,10%W.,5%West,40%)@
@percval(10%Second St.,10%Main St.,10%Park Ave.,10%Oak St.,10%Pine St.,10%Maple
Ln.,10%Washington St.,10%Lake Dr.,10%Hill Ave.,10%Ninth St.)@,25)@
supplier_000 s_city     @randlov(0,@seedlist(US Zip-Codes)@,3)@
supplier_000 s_nation    US
supplier_000 s_region    @randlov(0,@seedlist(US Zip-Codes)@,2)@
supplier_000 s_phone    @randlov(0,@seedlist(US Phone no)@)@
```

Publish to CSV.

Once you've completed this, use the AWS CLIv2 to move the files from the local directory to a bucket:

```
aws s3 cp . s3://broadcom-tdm-bucket/load --recursive
```

where the . signifies the directory where the published .csv files have been published or downloaded to.

Then use the COPY command to load the tables

```
copy part from 's3://broadcom-tdm-bucket/load/part_000.csv'
```

```
ACCESS_KEY_ID 'YOURID'
```

```
SECRET_ACCESS_KEY 'YOURKEY'
```

```
CSV
```

```
IGNOREHEADER 1;
```

Query to confirm the new part (in the 9xxxx range exists).

90001	PearClassmates	MFRGCO	Califor	ABC	California	PART	4	MED DRUM
-------	----------------	--------	---------	-----	------------	------	---	----------

Repeat for each of the tables.

Consider creating an AWS [Pipeline](#) to configure the imports.

Masking Detail

For each table that has columns that require masking of PII, create & execute commands to export the data structure (with a single header line, a single data line, and including the primary key/unique identifier of the row) to the S3 Bucket. Example for customer table, where we'll just mask the customer name (in the c_name column):

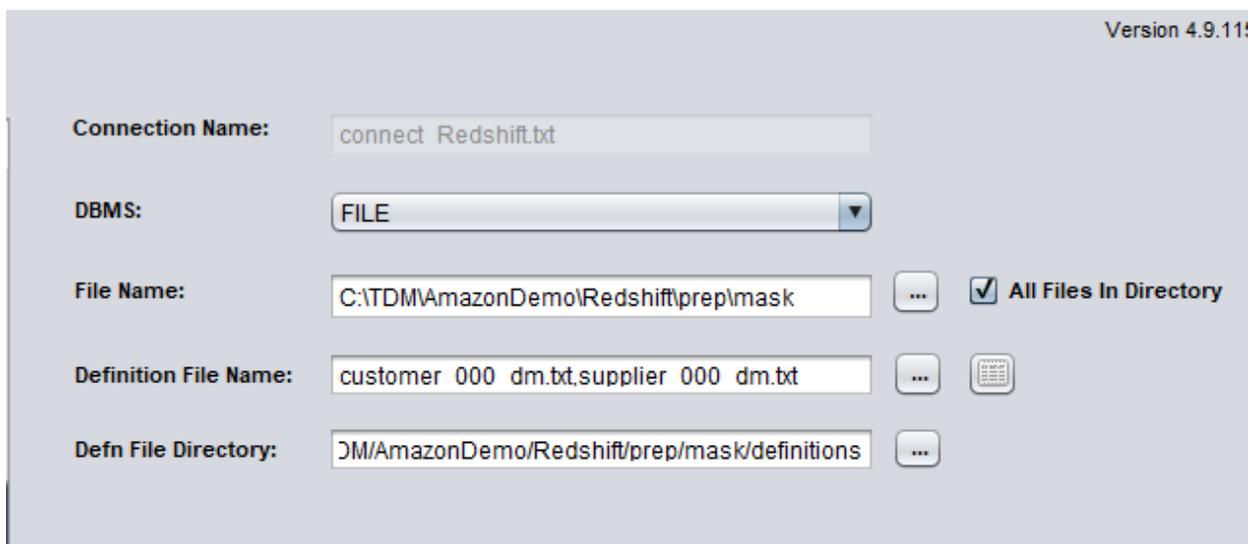
```
unload ('select c_custkey,c_name from customer where c_custkey = 1')
to 's3://broadcom-tdm-bucket/unload/customer_'
ACCESS_KEY_ID 'YOURKEYID'
SECRET_ACCESS_KEY 'YOURACCESSKEY'
header
CSV
parallel off;
```

Repeat for each table & field that will need to be masked (skip those that do not). Once you've completed this, use the AWS CLIv2 to move the files from the bucket to a local directory:

```
aws s3 mv s3://broadcom-tdm-bucket/unload . --recursive
```

Launch Fast Data Masker and create a connection to the directory where the examples have been downloaded. Add a .csv extension on the files.

Create a connection, and individually create definition files for each of the table extracts. Use the definitions folder capability on the Connection string to reference all the definition files (comma separated list).



Version 4.9.11!

Connection Name: connect Redshift.txt

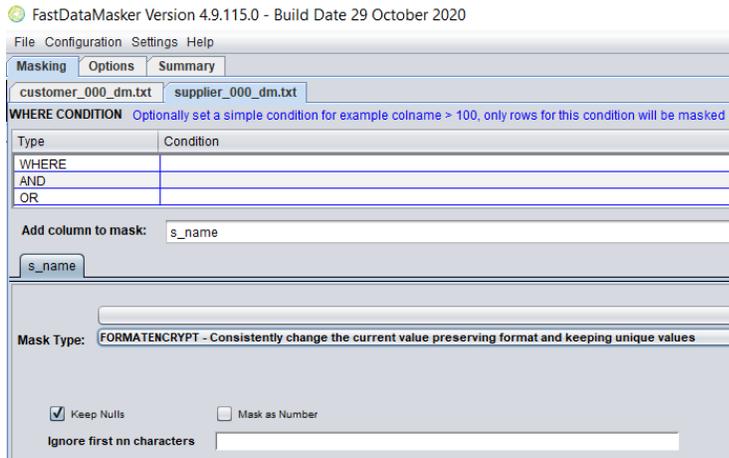
DBMS: FILE

File Name: C:\TDM\AmazonDemo\Redshift\prep\mask All Files In Directory

Definition File Name: customer 000 dm.txt,supplier 000 dm.txt

Defn File Directory: \DM\AmazonDemo\Redshift\prep\mask\definitions

After connection, specify the masking techniques desired on the specific fields in each table.



Execute the mask job. The resulting .scramble files will be produced.

Upload the .scramble files to the S3 bucket:

```
aws s3 cp . s3://broadcom-tdm-bucket/load --recursive --exclude "*" --include "*.scramble"
```

Create Staging Tables for the modified records

```
create table customer_temp(c_custkey int, c_name varchar(25), primary key(c_custkey) );
```

COPY the records into the Staging Tables

```
copy customer_temp from 's3://broadcom-tdm-bucket/load/customer_000.csv.scramble'
```

```
ACCESS_KEY_ID 'YOURID'
```

```
SECRET_ACCESS_KEY 'YOURKEY'
```

```
CSV
```

```
IGNOREHEADER 1;
```

Execute the SQL updates to merge/overwrite the Database tables.

```
begin transaction;
```

```
update customer
```

```
set c_name = customer_temp.c_name
```

```
from customer_temp
```

```
where customer.c_custkey = customer_temp.c_custkey;
```

```
end transaction;
```

Query the table to validate:

Rows returned (1)

c_custkey	c_name
1	Xuykwpxf#789123457