# Amazon Redshift Data Loading

Amazon Redshift offers industrial grade business intelligence data store at a fraction of the cost of traditional BI databases. But it poses challenges when it comes to upload data such as INSERT, UPDATE, and DELETE queries. There are some Redshift specific techniques that one should know to perform the queries quickly and efficiently.

In this article, I’d like to reiterate some of such techniques that I use frequently. Redshift is not write optimized, rather it is optimized columnar data store for high performance read operations. If one wants to insert many rows into a Redshift table, the INSERT query is not a practical option. This is because of slow performance due to large number of connections it internally creates per insert. Instead, Redshift offers an alternative using COPY command. It lets upload rows stored in S3, EMR, Dynamo-DB, or a remote host via SSH to a table. The COPY command is much more efficient compared to INSERT queries when run on a huge number of rows.

### UPSERT

What if one want to UPDATE and/or DELETE a large number of records? UPSERT (UPdate or inSERT) is a common technique to insert or update a large number of rows to a table. It works this way:

1. Create a staging table using the same table definition as the target table.

CREATE TABLE product\_stage (id integer, name varchar(255), description varchar(255), make varchar(20), serial varchar(15), PRIMARY KEY (id) );

1. Upload rows to the staging table using the COPY command. Rows you want to insert and rows you want to update may be mixed together in the staging table.

COPY product\_stage (id, name, description, make, serial) FROM 's3://.......' CREDENTIALS 'aws\_access\_key\_id=xxxxxxx;aws\_secret\_access\_key=xxxxxxx';

1. Run an UPDATE query to update rows in the target table, whose corresponding rows exist in the staging table.

UPDATE products SET name = ps.name, description = ps.description, make=ps.make, serial=ps.serial FROM products\_stage ps WHERE products.id = ps.id;

1. Run an INSERT query to insert rows which do not exist in the target table.

INSERT INTO products SELECT ps.\* FROM products\_stage ps LEFT JOIN products ON ps.id = products.id WHERE products.id is NULL;

In short one can perform bulk inserts and updates with 3 commands, COPY, UPDATE and INSERT.

### UPSERT + Bulk DELETEs

Since UPSERT doesn’t handle deletes, one need to issue another set of commands to delete rows from the target table. One can use a staging table to delete rows all at once.

1. Create a staging table. This staging table, unlike the staging table for UPSERT, may omit columns other than the primary key columns because only the primary key columns will be used.

CREATE TABLE products\_stage ( id integer, PRIMARY KEY (id) );

1. Upload the rows to be deleted to a staging table using a COPY command.

COPY products\_stage (id) FROM 's3://.......' CREDENTIALS 'aws\_access\_key\_id=xxxxxxx;aws\_secret\_access\_key=xxxxxxx';

1. Run a DELETE query to delete rows from the target table whose primary key exist in the staging table. The query might look like this:

DELETE FROM products USING products\_stage ps WHERE products.id = ps.id;

With the two additional commands (COPY and DELETE) you can bulk insert, update and delete rows. But it’s a total of 2 COPY commands and 3 data manipulation commands INSERT, UPDATE and DELETE.

It’s a lot of queries especially if you have many tables or if you want to update data frequently. Also, if you looks at these INSERT, UPDATE and DELETE queries, all 3 involves a JOIN. This is a very expensive operation we’d like to avoid if possible. Now, what can we do?

### DELSERT

A technique DELSERT (DELete and inSERT) will to improve the bulk upload performance with Redshift. It’ll cut down the number of commands from 5 to 3 and the number of JOIN queries from 3 to 1.

Create a staging table. In addition to columns from the target table, add an extra column which tells that the row is for i.e. i-> insert, u->update or d->delete.

CREATE TABLE products\_stage ( id integer, name varchar(255), description varchar(255), make varchar(20), serial varchar(15) row\_type varchar(1), PRIMARY KEY (id) );

1. Upload all rows (insert, delete, update) to a staging table using a COPY command. Each row has a value indicating what it’s for, insert/update/delete, in the extra column.

COPY products\_stage (id, name, description, make, serial, row\_type) FROM 's3://.......' CREDENTIALS 'aws\_access\_key\_id=xxxxxxx;aws\_secret\_access\_key=xxxxxxx';

1. Run a DELETE query to delete rows from the target table whose primary key exists in the stage table for delete or update.

DELETE FROM products USING products\_stage ps WHERE products.id = ps.id AND (row\_type = ‘u’ OR row\_type = ‘d’);

1. Run an INSERT query to insert rows marked for insert or update.

INSERT INTO products (id, name, description, make, serial) SELECT id, name, description, make, serial FROM products\_stage s WHERE row\_type = ‘i’ OR row\_type = ‘u’;

As one can see, a set of updates are done using only 3 SQL queries ( COPY, DELETE and INSERT) instead of the previous 5. Also, unlike our original UPSERT, this INSERT does not involve a JOIN, so it is much faster than the INSERT query used in an UPSERT.

### Conclusion

While UPSERT is a fairly common and useful practice, it has some room for performance improvement, especially if one need to delete rows in addition to just INSERTs and UPDATEs. DELSERT is a more streamlined alternative, which minimizes the number of queries and also improves the performance of some of the queries.

### References

Amazon Redshift Database Developer Guide: COPY [http://docs.aws.amazon.com/redshift/latest/dg/r\_COPY.html](http://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html" \t "_blank)