CASSANDRA SQL BATCHES AND MAPS

There are too many SQL commands and data types to look at all of them in one blog post, so we will look at a few here.

In order the understand the examples below you should have some understanding of regular, i.e., Oracle or other RDBMS, SQL, as Cassandra SQL is almost the same. And to get started with Cassandra, you can read what wrote here. Look at that even if you already know and have installed Cassandra so that you can the SQL from there to set up a table and index that we will start with here.

(This article is part of our Cassandra Guide. Use the right-hand menu to navigate.)

Cassandra SQL Shell

First step, to open the Cassandra SQL command line enter: **cqlsh**.

Paste in the SQL below. If you looked at that other blog post you will have some data here.

```
select * from Library.book;
```

Here is another query.

(Note that Cassandra requires the use of single and not double quotes, or you will get the error **no viable alternative at input.**)

```
select ssn from Library.patron where checkedOut contains '1234';
```

Collections, Maps, and Sets

Cassandra supports complex data types including user-defined objects, collections, maps, and sets.

For example. if we type:

```
describe library.patron;
```

We can see that commands that were used to create the table. Below is a truncated view of that as the complete view shows lots of system administration type options, which we will look at in another post.

Below you can see that the column **checkedout** is a set of type **text**. A **set** is a collection of unique values.

```
CREATE TABLE library.patron (
ssn int PRIMARY KEY,
checkedout set<text>
)
```

Maps

Maps store values in {key->value} format.

First create this keyspace.

```
CREATE KEYSPACE shopping
WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 3 };
```

Then create a shopping cart for an ecommerce app where the items purchased are a map. The first field is the customer order number. The second is the combination of what they bought and how much, e.g, {"shoes", 1}.

```
CREATE TABLE shopping.cart (
customerOrder int PRIMARY KEY,
items map<text,int>
);
```

Add some data, i.e., make a sale:

```
INSERT INTO shopping.cart (customerOrder, items)
VALUES (
  1,
 {'toys' : 12, 'socks': 5});
```

And list it:

```
select * from shopping.cart;
customerorder | items
---------------+-------------------------
1 | {'socks': 5, 'toys': 12}
```

Batches

With an ecommerce application you would want to use batches.

Batches ensure **atomicity**, which RDBMS programmers would say ensure **referential integrity**. This ensures that if any of the SQL statements in a batch have an error then none of the statements are applied.

That's import for, for example, a shopping cart app. We would only want to update the inventory if the sales transaction worked.

To illustrate that first create an inventory table and put an item into it:

```
CREATE TABLE shopping.inventory (
      item text PRIMARY KEY,
      quantity int
);

INSERT INTO shopping.inventory (item, quantity) VALUES ('eggs', 48);
```

Now make a sale of 4 eggs in a BATCH:

```
BEGIN BATCH

INSERT INTO shopping.cart (customerOrder, items)
VALUES (
2,
{'eggs' : 4}
);

UPDATE shopping.inventory
SET quantity = 44
where item = 'eggs';

APPLY BATCH;
```