INTRODUCTION TO MONGODB TRANSACTIONS

In computer science there are the concepts **ACID** (Atomicity, Consistency, Isolation, Durability), **atomic**, and **referential integrity**. Which term you know, depends largely on your age. I know the last one, which gives you a clue to hold old I am.

Basically all of these terms mean the same thing: it means the computer should kick out any transaction that would result in a logically-inconsistent set of transactions.

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

Why is this important?

Imagine if you have a sales order and inventory control system. When you make a sale it should reduce on-hand inventory. But what happens if the sales transaction works then the inventory update fails? Then the database would no longer be consistent: the inventory would not match the sales.

The way to avoid that is to group the two transactions into one larger transaction. Oracle database programmers know this as the **begin** and **commit** statements. MongoDB supports the same concept, which we illustrate below.

Performance Hit

While MongoDB supports transactions, it stresses that due to performance issues it's better to embed documents as JSON arrays. In other words, put all the data for one item, and its associated data, into one document.

Oracle programmers would say that results in a database which is not **normalized**. But that does not matter, as big data databases have gotten rid of that concept, saying it is OK to store the same data in two records, as their focus is on speed and using extra disk space to store something twice does not matter, since storage become cheaper than in the mainframe and Solaris server days.

In other words, its OK to have zip code **29607** and the word **Greenville, SC** in two records. You do not need to keep a zip code table as a separate entity, as you would with the rdbms approach.

Transactions Illustrated

You need a clustered MongoDB installation to do this, but you can pretend you have a cluster with only server on your laptop. Just pretend that the replicas are offline, by not installing but one node. In other words, follow these instructions we wrote, but remove the section in /etc/mongodConfig.conf as shown below, and only set up one server.

```
sharding:
  clusterRole: configsvr
```

To use transactions you must have replication turn on and sharding turned off. Support for sharded databases comes with MongoDB 4.2. Below we use 4.0.5.

Create Some Data

Open the mongo shell and create the products and sales collections then add one product to the inventory. We add 100 items of product 123.

```
use products
db.createCollection("products")
db.createCollection("sales")
db.products.insert({product: 123, count: 100})
```

Now, sell 4 items of product 123. The result would be 96 items remaining in inventory. So add a sales transaction and update the inventory count to 96 shown below.

Either paste that code into the MongoDB shell or save it in a JavaScript text file and then run it with **load('<file name>')**.

```
session = db.getMongo().startSession( { readPreference: { mode: "primary" } }
);

productsCollection = session.getDatabase("products").products;
salesCollection = session.getDatabase("products").sales;

session.startTransaction( { readConcern: { level: "snapshot" }, writeConcern:
{ w: "majority" } } );

try {
   salesCollection.insertOne( {
        product: 123,
        count: 4
```

```
        }
      );
   productsCollection.update(
   { product: 123 },
   { $set:
      {
        count: 96
       }
   }
);

} catch (error) {
   session.abortTransaction();
   throw error;
}

session.commitTransaction();

session.endSession();
```

Explaining the Code

First, test that is worked:

```
db.products.findOne();
{
        "_id" : ObjectId("5cb8ccb664ae78fe855e9431"),
        "product" : 123,
        "count" : 96
}
```

The key parts of the code are:

session.startTransaction({ readConcern: { level: "snapshot" }, writeConcern: { w: "majority" } });	Start transaction.
session.commitTransaction(); session.endSession();	Commit transaction. It will only reach here if the transactions between the two statements worked because of the **try-catch** block.
session.abortTransaction()	Means do not commit either transaction.

```
productsCollection.update(
{ product: 123 },
{ $set:
{
count: 96
}
}
)
```

The format of this is db.collection.update(query to find record, what fields to update).
We update product 123 and set the count to 96.

Troubleshooting

If you are trying to do this with your existing installation and did not turn on replication or turn off sharding or made some illogical statements in your code, you will get any of these errors:

- **Transaction numbers are only allowed on a replica set member or mongos.**
- **Multi-document transactions cannot be run in a sharded cluster unless using 4.2.**
- **Transactions are only supported in featureCompatibilityVersion 4.0. See http://dochub.mongodb.org/core/4.0-feature-compatibility for more information.**