The traditional E-commerce shopping cart can be modeled in MongoDB by using double bookkeeping. Given that we have the following initial documents.

First let's create a product

```
var col = db.getSisterDB("shop").products;
col.insert({
  , _id: "111445GB3"
  , name: "Simsong Mobile"
  , description: "Awesome new 70G Phone"
  , quantity: 99
  , price: 1000
});
```

Adding the product to the Shopping Cart

When the user indicates they want to add the product to their shopping cart we need to perform the following 3 steps.

1. Add the item to the shopping cart, creating the cart if it does not exist
2. Update the inventory only if we have enough quantity

If we don't have enough inventory to fulfill the request we need to rollback the shopping cart.

Let's add the selected product to the cart, creating the cart if it does not exist. We are assuming that the users unique session **id** in this case is **1**.

```
var quantity = 1;
var userId = 1;
var productId = "111445GB3";

var col = db.getSisterDB("shop").carts;
col.update(
    { _id: userId, status: 'active' }
  , {
      $set: { modified_on: new Date() }
    , $push: { products: {
          _id: productId
        , quantity: quantity
        , name: "Simsong Mobile"
```

```
      , price: 1000
    }}
  }, true);
```

The update above is an **upsert** meaning the cart is created if it does not exist. The next step is to reserve the quantity from the product ensuring there is inventory to cover the customers request.

```
var quantity = 1;
var col = db.getSisterDB("shop").products;
col.update({
    _id: productId
  , quantity: { $gte: quantity }
}, {
    $inc: { quantity: -quantity }
  , $push: {
    reserved: {
      quantity: quantity, _id: userId, created_on: new Date()
    }
  }
});
```

This reserves the quantity of the customer request only if there is product inventory to cover it. If there is inventory we decrement the available inventory and push a reservation into the **reserved** array.

Not Enough Inventory

If we don't have enough inventory to cover the customer request we need to rollback the addition to the shopping cart.

```
var quantity = 1;
var col = db.getSisterDB("shop").carts;
col.update({
  _id: userId
}, {
    $set: { modified_on: new Date() }
  , $pull: { products: { _id: productId }}
})
```

This removes the shopping cart reservation.

Adjusting The Number Of Items In The Cart

If the customer changes their mind about the number of items they want to shop we need to perform an update of the shopping cart. We need to perform a couple of steps to ensure proper recording of the right value.

First let's update the quantity in the shopping cart. First we need to fetch the existing quantity, then we need to calculate the delta between the old and new quantity and finally update the cart.

```
var col = db.getSisterDB("shop").carts;
var cart = db.findOne({
    _id: userId
  , "products._id": productId
  , status: "active"});
var oldQuantity = 0;

for(var i = 0; i < cart.products.length; i++) {
  if(cart.products[i]._id == productId) {
    oldQuantity = cart.products[i].quantity;
  }
}

var newQuantity = 2;
var delta = newQuantity - oldQuantity;

col.update({
    _id: userId
  , "products._id": productId
  , status: "active"
}, {
  $set: {
      modified_on: new Date()
    , "products.$.quantity": newQuantity
  }
});
```

Having updated the quantity in the cart we now need to ensure there is enough inventory to of the product to cover the change in quantity. The needed amount is the difference between (newQuantity and oldQuantity)

```
var col = db.getSisterDB("shop").products;
col.update({
    _id: productId
```

```
  , "reserved._id": userId
  , quantity: {
    $gte: delta
  }
}, {
  , $inc: { quantity: -delta }
    $set: {
      "reserved.$.quantity": newQuantity, modified_on: new Date()
    }
})
```

This correctly reserves more or returns any non-needed product to the inventory.

1. If delta is a **negative** number the $gte will always hold and the product **quantity** get increased by the delta, returning product to the inventory.
2. If delta is a **positive** number the $gte will only hold if inventory is equal to the delta and is then decreased by delta, reserving more product.

Rolling back Attempted Increase of Reservation for A Product

If there is not enough inventory to fulfill the new reservation we need to rollback the change we made in the cart. We do that by re-applying the old quantity.

```
var col = db.getSisterDB("shop").carts;
col.update({
    _id: userId
  , "products._id": productId
  , status: "active"
}, {
  $set: {
      modified_on: new Date()
    , "products.$.quantity": oldQuantity
  }
});
```

Expiring Carts

It's common for customers to have put items in a cart and then abandon it. This means there is a need for a process to expire carts that have been abandoned. For each expired cart we need to.

1. Return the reserved items to the product inventory
2. Expire the cart

Below is a script that will look for any cart that has been sitting inactive for more than 30 minutes and automatically expire them returning stock to the inventory for each product reserved in the carts.

```
var cutOffDate = new Date();
cutOffDate.setMinutes(cutOffDate.getMinutes() - 30);

var cartsCol = db.getSisterDB("shop").carts;
var productCol = db.getSisterDB("shop").products;

var carts = cartsCol.find({ modified_on: { $lte: cutOffDate }});
while(carts.hasNext()) {
  var cart = carts.next();

  for(var i = 0; i < cart.products.length; i++) {
    var product = cart.products[i];

    productCol.update({
          _id: product._id
        , "reserved._id": cart._id
        , "reserved.quantity": product.quantity
    }, {
          $inc: { quantity: product.quantity }
        , $pull: { reserved: { _id: cart._id }}
    });
  }

  cartsCol.update({
    _id: cart._id
  }, {
    $set: { status: 'expired' }
  });
}
```

For each cart we iterate over all the products in it and for each product we return the quantity to the product inventory and at the same time remove that cart from the **reserved** array of the product. After returning the inventory we set the status of the cart to **expired**. Notice that we don't clean up the cart. We are keeping the expired cart as history. Any new customer will create a new cart.

Checkout

The customer clicked the checkout button on the website and entered their payment details. It's time to issue an purchase order and clean up the cart and product reservations.

```
var cartsCol = db.getSisterDB("shop").carts;
var productCol = db.getSisterDB("shop").products;
var orderCol = db.getSisterDB("shop").orders;

var cart = cartsCol.findOne({ _id: userId })

orderCol.insert({
    created_on: new Date()
  , shipping: {
      name: "Joe Dow"
    , address: "Some street 1, NY 11223"
  }
  , payment: { method: "visa", transaction_id: "2312213312XXXTD" }
  , products: cart.products
});

cartsCol.update({
  { _id: userId }
}, {
  $set: { status: 'complete' }
});

productCol.update({
  "reserved._id": userId
}, {
  $pull: { reserved: {_id: userId }}
}, false, true);
```

We perform the following actions during checkout.

1. Add an a finished order document to the **orders** collection
2. Set the cart to **done** status
3. Removing the cart from the **reserved** arrays of all products where it's present using a multi update

Indexes

Some Possible Changes

It's possible to split out product information from the inventory by creating an **inventory** collection that references the production metadata collection and contains the amount of the product.