Cleansing Time – SQL Free Applications

Cleansing Time - SQL Free Applications

Release 3.4

MongoDB, Inc.

Jun 05, 2017

Contents

1 RDBMS to MongoDB Workshop

1.1 RDBMS to MongoDB Introduction

Welcome

This is a full day workshop where we are going to be covering the following topics:

- Migration strategies
- Architecture and application implications
- Relation to Document modeling
- MongoDB CRUD operations

Where do we start?

You will be given a fully functional application called **Mongomart** backed by a relational database.

You will be tasked with defining the different tasks required to move this application to a MongoDB supported back-end.

After each task the instructor will provide a solution for each of the labs/tasks.

How are we going to do that?

Given a migration strategy, defined by the instructor, we will migrating our application.

We need to understand:

- What's the relation schema looks like. *(ERD)*
- How do we want to store the same information in a MongoDB document model.

What to expect by the end of the workshop?

By the end of this workshop you will be more suited to:

- Understand the practical tasks required to move a relational system to MongoDB
- The benefits and tradeoffs of the different migration approaches
- Considerations about schema design and relational to document mapping

1.2 System Requirements

Before you get started

Before we jump into coding and making a migration plan let's review the list of software components required to run this workshop.

In this workshop we will be using a set of software requirements apart from the actual code and workbooks.

Mongomart Java Version

- Java 8[1]
- Apache Maven[2]

Code has been tested using Java 8[3] and built using Apache Maven[4] 3.5.0 .

Other versions may function correctly but we cannot provide efficient support.

[1] http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
[2] https://maven.apache.org/install.html
[3] http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
[4] https://maven.apache.org/install.html

Mongomart Databases

- MongoDB 3.4[5]
- MySQL[6]

We will be using MySQL[7] Ver 5.7.18 and MongoDB 3.4[8].

System Editor or IDE

During the course of the workshop we will be requiring some code changes and recompilations.

Make sure to have your prefered editor or java IDE.

Otherwise take some time to download and install Eclipse[9] or Intellij[10].

MongoDB Compass

To review and analyze the MongoDB documents schema we will be using MongoDB Compass[11].

Not mandatory for this workshop but highly recommended to be installed.

/modules/compass

1.3 Environment Setup

Setup Workshop Environment

After we completed the download of our workshop material, it is then time to bring our **MongoMart** up.

To do so, we will need the following:

- Unzip `rdbms2mongodb.zip` file
- Launch local MySQL server
- Import dataset
- Run `mongomart` process

[5] https://docs.mongodb.com/manual/installation/
[6] https://dev.mysql.com/downloads/installer/
[7] https://dev.mysql.com/downloads/installer/
[8] https://docs.mongodb.com/manual/installation/
[9] http://www.eclipse.org/downloads/eclipse-packages/?show_instructions=TRUE
[10] https://www.jetbrains.com/idea/download
[11] https://www.mongodb.com/products/compass

Unzip `rdbms2mongodb.zip`

After inflating the `rdbms2mongodb.zip` file, we will find this directory structure:

```
unzip rdbms2mongodb.zip
ls rdbms2mongodb
> README  dataset java
```

Launch local MySQL server

Time to launch our relational database server:

- In your *NIX system

```
mysql.server start
```

- Or Windows

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqld"
```

Import dataset

Within your `rdbms2mongodb` folder, there is a `dataset` folder.

This folder contains the dataset that we will be working with.

To import this dataset:

```
# creates the relational system schema
mysql -uroot < dataset/create_schema.sql
# imports previously generated dump
mysql -uroot < dataset/dump/mongomart.sql
# run a few checks - mandatory step!
mysql -uroot < dataset/check.sql
```

And for our Windows friends:

```
cmd.exe /c "mysql -u root < dataset\create_schema.sql"
cmd.exe /c "mysql -u root < dataset\dump\mongomart.sql"
cmd.exe /c "mysql -u root < dataset\check.sql"
```

Run the `mongomart` app

Once we have our dataset fully imported it is time for us to give `mongomart` a spin:

- First we generate the java package

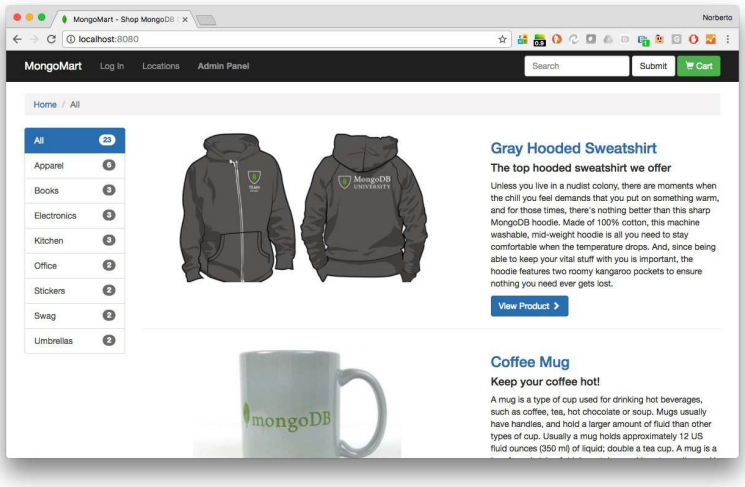
```
mvn package -f java/pom.xml
```

- Then we run the application process

```
java -jar java/target/MongoMart-1.0-SNAPSHOT.jar
```

Final Step

Once the process is correctly up and running, the final step is to connect to http://localhost:8080 using your system browser.



1.4 MongoMart RDBMS

What is MongoMart

MongoMart is an on-line store for buying MongoDB merchandise. In this workshop we will start with a RDBMS version and end up with a MongoDB one.

MongoMart Demo of Fully Implemented Version

- View Items
- View Items by Category
- Text Search
- View Item Details
- Shopping Cart

View Items

- http://localhost:8080
- Pagination and page numbers
- Click on a category

View Items by Category

- http://localhost:8080/?category=Apparel
- Pagination and page numbers
- “All” is listed as a category, to return to all items listing

Text Search

- http://localhost:8080/search?query=shirt
- This functionality is not yet implemented.
- Will be part of this workshop to add Text Search functionality

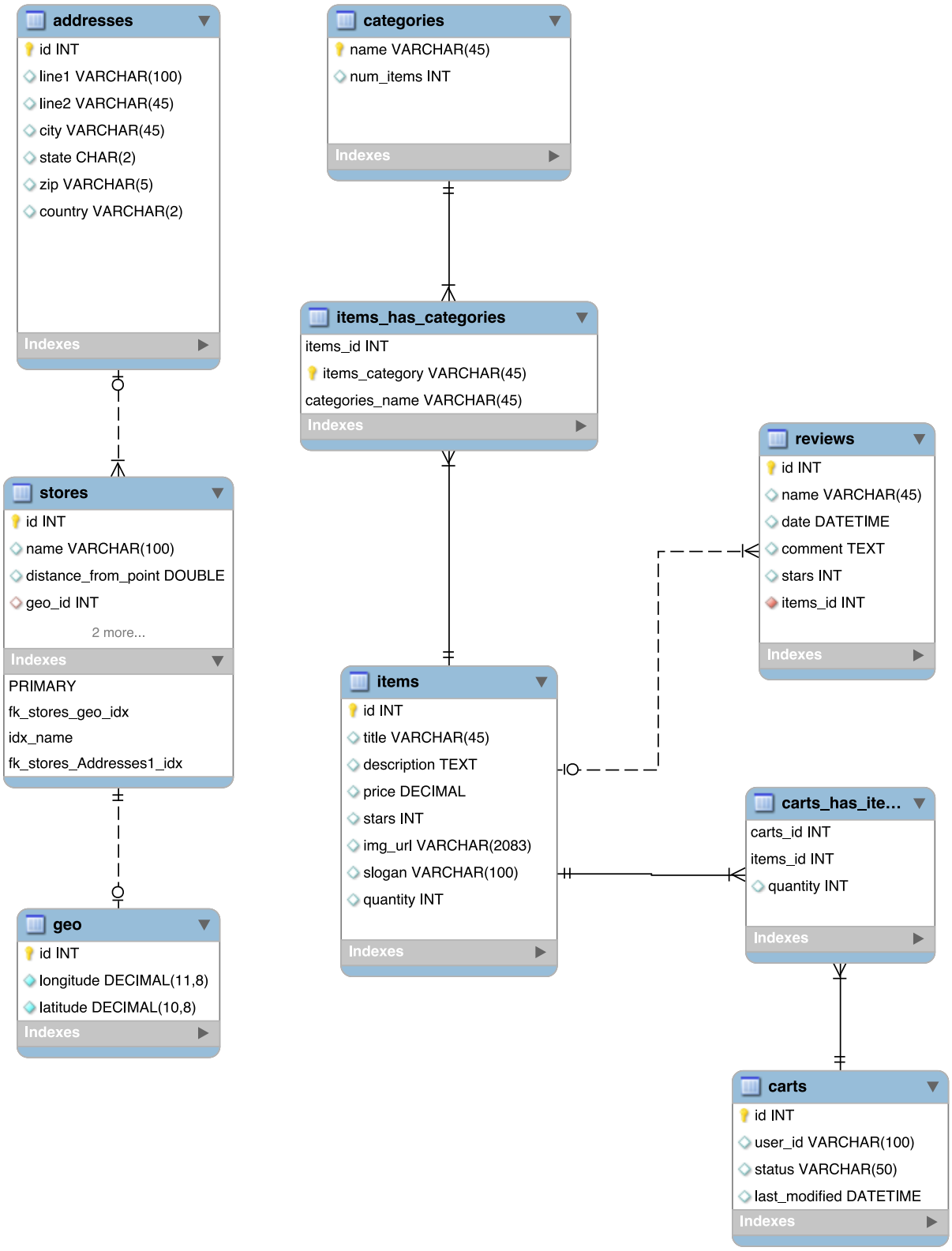
View Item Details

- http://localhost:8080/item?id=1
- Star rating based on reviews
- Add a review
- Related items
- Add item to cart

Shopping Cart

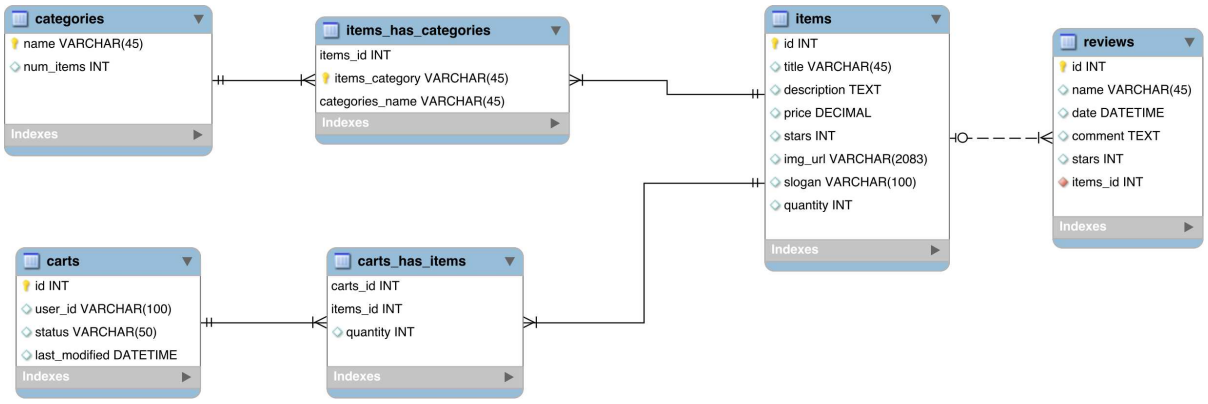
- http://localhost:8080/cart
- Adding an item multiple times increments quantity by 1
- Change quantity of any item
- Changing quantity to 0 removes item

Entity Relationship Diagram (ERD)



Stores ERD

Items ERD



1.5 Migration Strategies

Learning Objectives

In this module we will be covering the following topics:

- Different strategies to deal with database migration
- Different development implications of such strategies
- Pros and cons of different strategies and approaches

To extend your understanding about this topic, we recommend you to read our Migration Guide White Paper[12] that explores this topic in detail.

[12] https://www.mongodb.com/collateral/rdbms-mongodb-migration-guide

Migration Strategies

When migrating between databases, there are two typical strategies that we can follow:

- Full one stop shop migration
- Step-by-step hybrid migration

Database Migration Phases

But regardless of the strategy/approach taken, there are a few steps or phases that we need to attend:

- Schema Design: Plan how our schema is going to look like in the new system
- App Integration: What different libraries and CRUD operations will need to be modified.
- Data Migration: The actual migration of our data between databases
- Operations: All the backup/monitoring/management procedures will require a review.

Schema Design

The way that data is organized in a RDBMS system is considerably different from the one used by MongoDB.

Having that in mind, we need to understand how to restructure our data structures to make the most out of the new system that we are migrating to.

App Integration

New database means new libraries and new instructions to operate with:

- Review the list of components and classes that will be affected
- If you have clear architecture in place this tends to be easier
- New tests to be re-written (unit, integration, regression)
- Data types will have to considered
  - How data is stored in one database might not have a direct translation with the previous system and needs to be handled in the application

Operations

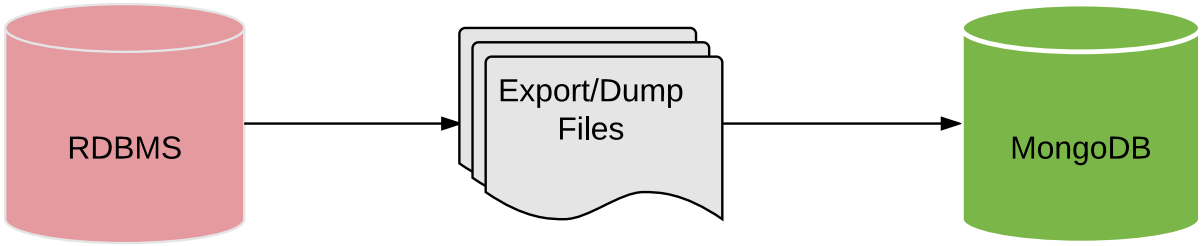
With new infrastructure like a database, the operations teams needs to bring into place new monitoring an management tools:

- New monitoring tools
- New alerting and stats diagrams
- Backup procedures will be different
- Management and operational procedures will need to be updated
- New APM tooling might be required

Full one stop shop migration

In a one stop shop migration, data is moved all at once:

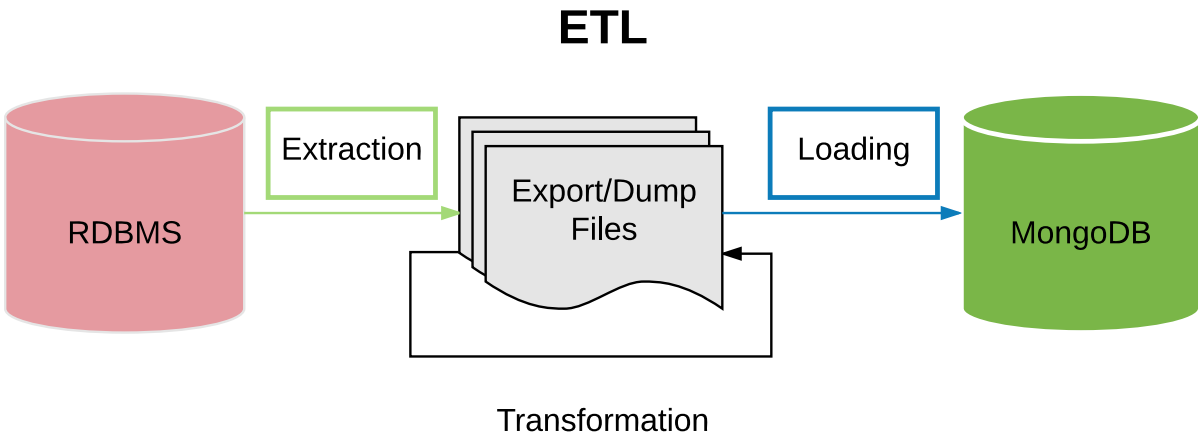
- We export a set of data that we want to see migrated



Full one stop shop migration (cont')

Generally we use ETL tooling to support this type of migration strategy:

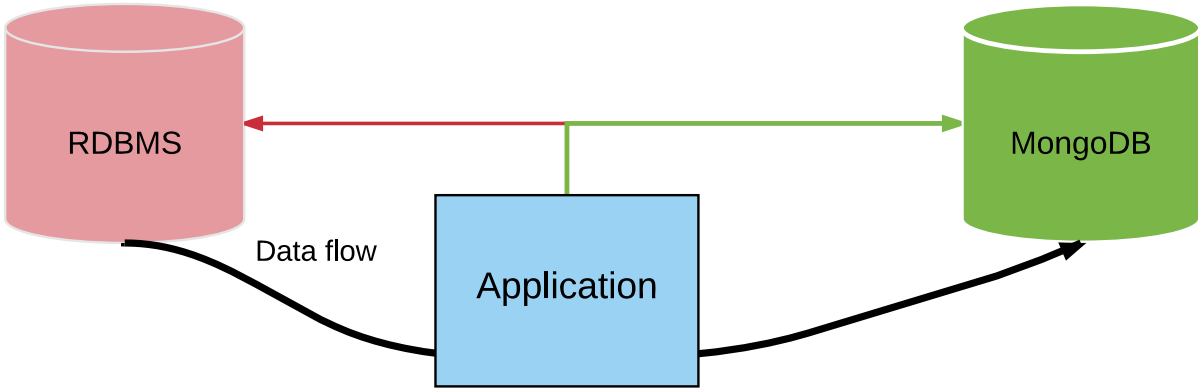
- Different databases will export data into different formats
- To handle database type impedance and schema issues we need to perform some transformation
- Loading data to the end destination will required a well defined process.



Step-by-step hybrid migration

In a hybrid solution we will have the application itself driving the migration process.

- The application will be talking to both backends (RDBMS and MongoDB)
- Data will flow from one system to another based on the application operational workload
- This strategy will increase your code complexity, however moving smaller portions of code at a time



1.6 Lab1: `Reviews` Migration

Learning Objectives

In this lab, we'll be setting our application into an hybrid mode where both databases will be providing service.

We will be covering:

- Benefits of using an hybrid solution for migration purposes.
- Implementation strategies to perform such migration.
- Schema design review.

Introduction

Currently we have our `ReviewsController` using only one RDBMS system.

For this lab we purpose that `Reviews` should be migrated to MongoDB.

To do that we will change the `ReviewsController` to use both RDBMS and MognoDB.

Strategy

To do this, the data migration we will follow this approach:

- For each `item`, we will requests all `reviews` stored in MongoDB **mongomart.review** collection.
- Then, we will request all `reviews` from the RDBMS system that are not present in MongoDB
- For each `review` retrieved from RDBMS we will store a copy `review` in MongoDB
- All new reviews added to the system should be stored in MongoDB

`review` Collection Schema Design

In this collection we will be storing data in a format that reflects the `Reviews` class.

```
{
  "_id": <ObjectId>,
  "id": <integer>,
  "name": <string>,
  "date" <ISODate>,
  "comment": <string>,
  "stars": <integer>,
  "itemid": <integer>
}
```

Step 1: Connection to MongoDB

In order for us to be able to store data into MongoDB we will have to establish a connection from our application to our server:

- Bring up a MongoDB server instance

```
# default dbpath for MongoDB
mkdir -p /data/db
# launch MongoDB
mongod
```

Once the instance is up and running lets establish a connection from our application.

- Change the `MongoMart` class to include a connection to MongoDB.

Step 2: Create `mongodb.ReviewDao`

To interact with MongoDB collection we will need to create a `ReviewDao` class.

- Create a new package `/mongomart/dao/mongodb`
- Create the `ReviewDao` class within this new package - `mongomart.dao.mongodb.ReviewDao`
- This dao should reimplement all rdbms package public methods:
  - `getItemReviews(...)`
  - `avgStars(...)`
  - `numReviews(...)`
- And add the a new method
  - `documentToReview(...)`

Step 3: Add `addReview` method

As part of our strategy, we will need to add reviews to the `review` collection.

Our `dao.mongodb.ReviewDao` class should have a method that adds reviews to the collection

- Implement an `addReview()` method

Step 4: Integrate new Dao into `StoreController`

After we've added our new `dao.mongodb.ReviewDao` we will need to start using it.

The `reviews` table is access by the `StoreController`. The corresponding MongoDB collection, will also be accessed from the same class.

- `StoreController` constructor should receive a MongoDB database object
- Use the new `dao.mongodb.ReviewDao` to find reviews in MongoDB
- After collecting all reviews, insert those reviews into MongoDB.

Add Reviews to Items

At this point you should have a fully functional **MongoMart** that stores all new reviews on `Item` into MongoDB.

If that's not the case, don't worry, you can use the emergency eject script:

```
./solvethis.sh lab1
```

1.7 Lab2: `Items` Migration

Learning Objectives

In this lab, we will be exploring the following operations:

- Implementing a full migration using a hybrid solution
- Enabling Text Search

Step 1: Add `dao.mongodb.ItemDao` to MongoMart

After receiving this new `ItemDao` class, you will have to:

- Integrate this class into the **MongoMart** application
  - `StoreController` is a good candidate to hold it!
- Once the new Dao is integrated:
  - Verify that `Items` are getting stored into MongoDB

Step 2: Iterate over `Items`

First, migrate the items into MongoDB:

```
# iterate over all 23 items
for i in {1..23}
do
  curl -I http://localhost:8080/item?id=$i
done
```

... or for Windows:

```
FOR /L %%A IN (1,1,23) DO( curl -I http://localhost:8080/item?id=%%A )
```

Step 3: Enable Text Search

The Text Search[13] functionality has not been activated yet.

Let's get it up and running:

- Enable Text Search[14] on the `items` collection.
- This functionality is provided by MongoDB
  - Enable Text Search[15] on following fields:
    * `title`
    * `slogan`
    * `description`

[13] https://docs.mongodb.com/manual/reference/operator/query/text/
[14] https://docs.mongodb.com/manual/reference/operator/query/text/
[15] https://docs.mongodb.com/manual/reference/operator/query/text/

1.8 Lab3: Full Migration

Learning Objectives

In this lab, we will use MongoDB tools for the data migration. We will *not* use the standard ETL process.

Do the following:

- Import an RDBMS data dump into MongoDB using `mongoimport`
- Transform that data using **aggregation** and **views**
- Use the `$out` stage of the aggregation framework to *materialize* the schema design

Step 1: Apply the Solution to lab3

Apply the solution to **lab3** to be sure we're all on the same page:

```
./solvethis.sh lab3
```

We will give you the commands for importing the data, and you will focus on the data migration using data loading tools.

Step 2: Import RDBMS CSV Files

Go to the `dataset/csv` folder containing RDBMS data export files.

- Use `mongoimport` to import the data contained in these files.

```
mongoimport -d mongomart --type CSV --headerline -c geo_sql < dataset/csv/geo_sql.csv
mongoimport -d mongomart --type CSV --headerline -c address_sql < dataset/csv/address_
↪sql.csv
mongoimport -d mongomart --type CSV --headerline -c store_sql < dataset/csv/store_sql.
↪csv
mongoimport -d mongomart --type CSV --headerline -c zip_sql < dataset/csv/zip_sql.csv
```

Step 3: Transform the Schema using `Views`

The relational schema we're importing is very normalized, and isn't a good schema design for `mongodb.StoreDao`, as it doesn't take advantage of what MongoDB does well.

Do the following:

- Create a view that transforms the `store` collection into the schema expected by `mongodb.StoreDao`: (see next slide)

Transform the Schema using `Views` (continued)

```
{
  "_id": <ObjectID>,
  "address": {
    "address1": <string>,
    "address2": <string>,
    "city": <string>,
    "zip": <string>,
    "country": <string>,
    "state": <string>,
  }
  "coords": [lon, lat]],
  "name": <string>,
  "storeId": <string>,

}
```

Step 4: Persist the `view` using `$out`

Views are a good option to store an aggregation pipeline so that a transformation of the original data can be treated as a queryable object.

They can be great tool for figuring out which data model to use.

But views do not allow us to **change** the content of the data directly. We want to *transform* our data from the normalized collections, and maintain our transformation separately.

- "Materialize" the view into a collection by using aggregation pipeline's $out stage[16].

1.9 Lab4: Final Migration Cleanup

Learning Objectives

This is the *final* lab of the workshop. Here, we will:

- Clean up our environment (code, data)
- Create a backup of the **legacy** database
- Safely shut down the RDBMS system that we started with.
- Build any indexes that could improve performance on the new database.

[16] https://docs.mongodb.com/manual/reference/operator/aggregation/out/

Step 1: Remove All `dao.rdbms` Calls

Let's review our controllers and pinpoint which calls to `dao.rdbms` libraries and other RDBMS-specific objects are still floating around:

- `StoreController`
- `LocationController`
- `CartController`
- `ItemController`

Your will remove all calls to *any* legacy RDBMS data access objects.

Step 2: Change Controller Constructor Signatures

Some of our controller objects are getting a `Connection` object in their constructors.

They are no longer required.

- Change the controllers constructors to expunge the reference to `java.sql.Connection`

Step 3: Remove the RDBMS Connection Creation Code

Remove the code that creates the RBMS connection.

- Specifically, remove the JDBC driver library and any associated connection management code

Step 4: Stop the RDBMS Server

Now that our code is clean, we can safely stop the legacy RDBMS server.

```
mysql.server stop
```

Validate that the application is fully functional... and celebrate!

Step 5: Create an RDBMS Backup

Bring up the RBMS system for one last operation:

```
mysql.server start
```

- Create a backup

```
mysqldump --databases mongomart > mongomart_backup.sql
```

Step 6: Remove Temporary Collections

In Lab3, we imported the relational data into a set of collections we no longer use.

- Remove any collection of our `mongomart` database that is no longer used.

```
db.address_sql.drop()
db.geo_sql.drop()
db.store_sql.drop()
```

- Create additional indexes to optimize for the **MongoMart** application

Step 7: Have a Great Day!

Congratulations! Your work is done!

Find out more
mongodb.com | mongodb.org
university.mongodb.com

Having trouble?
File a JIRA ticket:
jira.mongodb.org

Follow us on twitter
@MongoDBInc
@MongoDB