
# Cleansing Time – SQL Free Applications

# Cleansing Time - SQL Free Applications

***Release 3.4***

**MongoDB, Inc.**

**May 30, 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.

---

**Note:** Note to students that they will be given enough time to complete the tasks on their own, however, to avoid leaving people behind we will be sharing incremental solutions to the different purposed labs.

---

### 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 looking like. *(ERD)*
- How do we want to store the same information in a MongoDB document model.

---

**Note:** Bring students attention to:

- different strategies might end up with different schemas
- the followed strategy might not apply to their real life scenarios
- this is an exercise that does not take into consideration actuall production level load. Something that should not be neglected in production environments .

---

### What to expect by the end of the workshop?

By the 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 reqirements 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].

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

### Unzip `mongomart.zip`

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

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

---

**Note:** Take some time to get the students acquainted with the directory structure.

- In case students are using an IDE let them known that they can import the `java` folder as a Maven project.

---

[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

### 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"
```

---

**Note:** Although students should have already checked that they do have MySQL installed, make sure to ask around to get everyone up to speed.

---

### Import dataset

Within your `mongomart` 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
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\mongomart.sql"
```

---

**Note:** Take some time to let the students perform this operation.

- They might not be used to running shell commands, specially Windows people
- Files have been generated in a Unix system. There might be some incompatibilities between file formats.
- If so, have the students download the executable file available in this link:

http://www.efgh.com/software/unix2dos.htm

---

### Run the `mongomart` app

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

- First we generate the java package

```
cd mongomart/java
mvn package -Dmaven.test.skip=true
```

- Then we run the application process

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

---

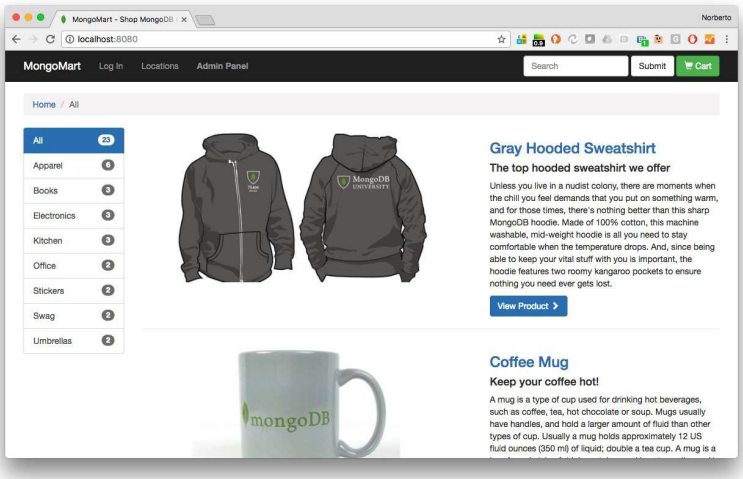
**Note:** This might be a critical point for many students. Specially if they have not properly installed maven.

- Make sure that everyone has correctly setup their $MAVEN_HOME and $JAVA_HOME in their environment $PATH
- Students using the IDE and familiar with Maven might be running the process from the IDE itself. Nevertheless it's have students bring up the process, at least once, from the command line.

---

### 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

---

**Note:**

- Mongomart is in the training repo in the training portal, packaged has mongomart.zip .
- Ask the students to login and download it.
- In case they are not able to login, have a spare pen drive that you can pass along the classroom.
- You should also have an instructor only mongomart-instructor.zip file in the training portal.
- This will include the solutions for the different exercises.
- Ensure you've loaded the dataset from /mongomart/dataset as described in the /mongomart/rdbms/README file)
- DO NOT WALK THROUGH THIS VERSION OF THE SOURCE CODE WITH THE STUDENTS, ONLY THE STUDENT VERSION

---

### View Items

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

---

**Note:**

- Click around the site to show them how it works, click through various pages and categories (into next slide)

---

### 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

---

**Note:**

- Once we move the application to be supported by MongoDB we will enable text search
- Search for a few strings, such as “shirt”, “mug”, “mongodb”, etc.

---

### View Item Details

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

---

**Note:**

- Add reviews, show how stars change, etc.

---

### 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

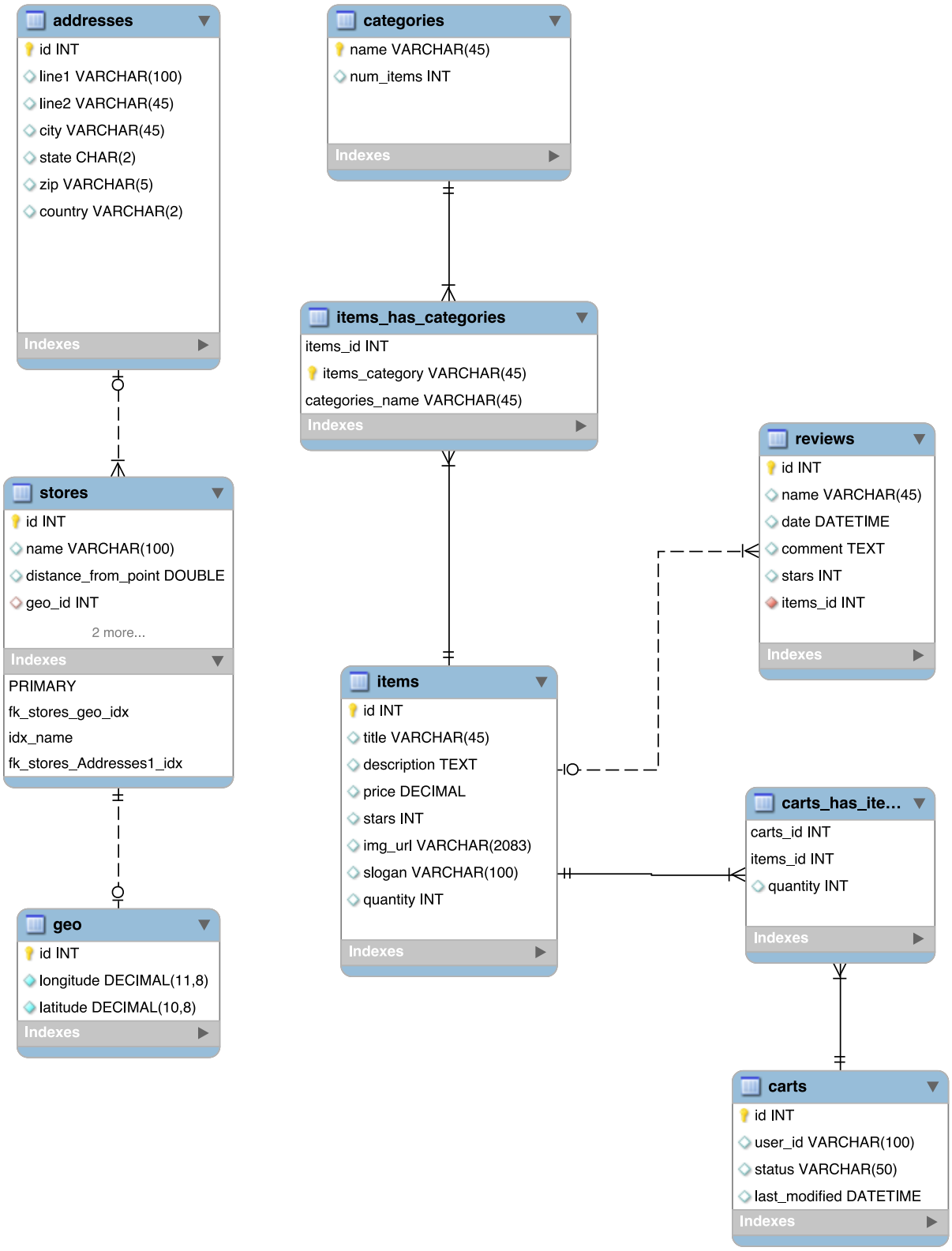
---

**Note:**

- Add an item to your cart
- Update the quantity in the cart
- Checkout doesn’t work, we are only interested in the cart functionality

---

## Entity Relationship Diagram (ERD)



**Note:** This is the mongomart rdbms relational model diagram. Take a few moments to walk them through the model. The next sections have a more detailed view.

Important to bring up that we have 2 clear model components in this app, that can be treated in parallel.

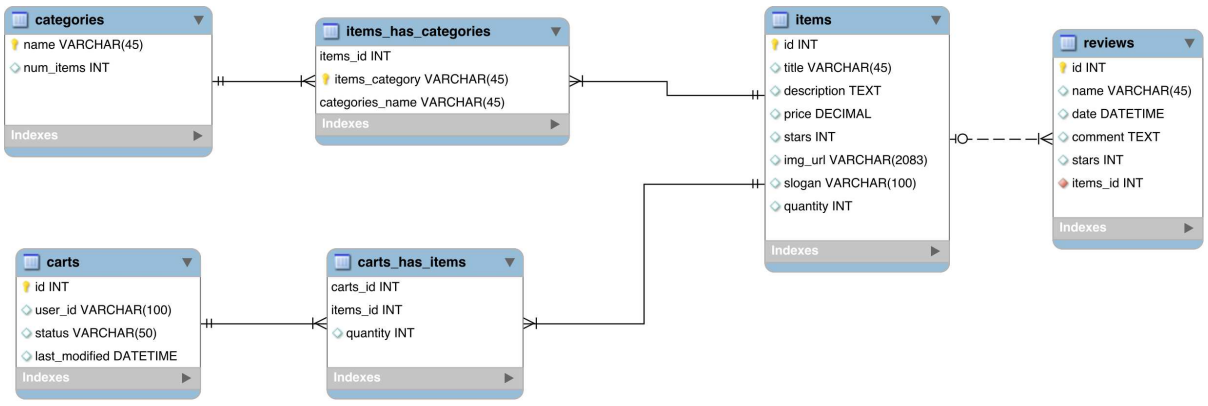
This is an important aspect of the migration process, understand the model that we are working with, and make decisions on how to approach the migration process.

### Stores ERD


**Note:** In this diagram, bring students attention to the following:

- This a pretty straight forward approach
- A store can have multiple addresses, and multiple geo locations (which is wrong on purpose!)
- Stores can be found either by their address or by their geo location.

### Items ERD




**Note:** In this diagram, bring students attention to the following:

- An item has categories and a category can be referenced by sevaral different items
- A cart will have several items and an item can be included in several different carts
  - Apart from that, the join/junction table will also account for the cart item quantity.
- Then we have items with reviews. Each review corresponds only to one item , but an item can have multiple reviews.

Our migration will start with this component of the application.

---

## 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[11] that explores this topic in detail.

### 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

---

**Note:** There might be different variations of these approaches, but in essence these are the typical approaches teams may opt to while doing a migration.

- **Full one stop shop migration** means that there's one event where data is migrated to another system. This might be phased but it assumes that all data will eventually by migrated.
- **Step-by-step hybrid migration** means that the two database systems co-exist and provide service to the application for a period of time, while the transition takes place.

---

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

### 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.

---

**Note:** Highlight that this phases can occur in parallel by different teams.

---

### 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 restructure our data structures to make the most out-of the new system that we are migrating to..

### App Integration

// TODO

### Operations

// TODO

### Full one stop shop migration

// TODO

### Step-by-step hybrid migration

// TODO

## 1.6 Lab1: `Reviews` Migration

---

**Note:**

- We'll start by migrating the `reviews` table since the data is totally user generated data, therefore we can easily integrate a hybrid mechanism
- Students will be asked to maintain the databases in place and define the access pattern.
- The exercise proposed solution opts for:
  - Find all reviews of an `item` in MongoDB.
  - Find all reviews in the rdbms that are not present in MongoDB
  - For each review in rdbms insert them into MongoDB
  - Merge both list
- After a few iterations all reviews should have been moved to MongoDB
- Once all data is present in MongoDB we can remove the connection to the RDBMS
- Given that this is the first lab where students will have to do some coding it would be usefull to do a bit of hand-holding and complete the exercise together

---

### 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 `review` 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` retrived 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>,
  "items_id": <integer>
}
```

---

**Note:** You can bring the attention to the students that we are using `_id` along side the `id` field.

- This will be the correspondant primary key from the rdbms table
- since MongoDB does not have an autoincrement field, we need to keep the compatibility between databases.

---

### 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.

---

**Note:** Show the students how to do this:

- Edit java/mongomart/MongoMart.java file

```
import com.mongodb.MongoClient;
...
public MongoMart(String connectionString) throws IOException {
  ...
```

```
  // Create a Database connection
  try{
    ...
    // MongoClient connection
    MongoClient mongoClient = new MongoClient();
    ...
  }
}
```

### Step 2: Create `mongodb.ReviewDao`

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

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

**Note:** A few things worth noticing to students:

- Students need to make sure we have methods that translate Bson Document to POJO classes

```
public static Review documentToReview(Document doc){
  Review review = new Review();

  review.setId(doc.getInteger("id", "-120"));
  review.setComment(doc.getString("comment"));
  review.setName(doc.getString("name"));
  review.setStars(doc.getInteger("stars"));
  review.setDate(doc.getDate("date"));
  review.setItemid(doc.getInteger("itemid"));
  review.set_Id(doc.getObjectId("_id"));


  return review;
}
```

- `ReviewsDao` class needs to receive a `Collection` or `Database` object

### Step 3: Add `addReview` method

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

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

- Implement an `addReview()` method

---

**Note:** Students will have to:

- create the `addReview()` method that takes a `Review` object and inserts the corresponding document into MongoDB

```
public void addReview(Review review) {
  reviewCollection.insertOne(reviewToDoc(review));
}
```

- A method to unmarshal `Review` objects into `bson.Document` will also be required

```
public void addReview(Review review) {
  reviewCollection.insertOne(reviewToDoc(review));
}
```

---

### Step 4: Integrate new Dao into `StoreController`

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

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

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

---

**Note:** Several different actions will need to be accomplished here:

- Change `StoreController` class to accept a `MongoDatabase` object

```
public StoreController(Configuration cfg, Connection connection, MongoDatabase␣
↪mongoMartDatabase) {
  ReviewDao reviewDao = new ReviewDao(connection);
  mongomart.dao.mongodb.ReviewDao revDao = new mongomart.dao.mongodb.
↪ReviewDao(mongoMartDatabase);
  ...
}
```

- For any call on `dao.rbms.ReviewsDao` also call `dao.mongodb.Reviews`

```
private HashMap<String, Object> buildItemResponse(){...}

get("/add-review", (request, response) -> { ... });
```

- After collecting all item reviews, insert reviews into MognoDB collection

```
buildItemResponse(int itemid, ItemDao itemDao, ReviewDao reviewDao, mongomart.dao.
↪mongodb.ReviewDao mongoRevDao){
  // Get reviews from MongoDB
  List<Review> mongoReviews = mongoRevDao.getItemReviews(itemid);

  // Get reviews from rdbms
  List<Review> reviews = reviewDao.getItemReviews(itemid);
  reviews.addAll(mongoReviews);
  // Remove duplicates by using a set
  Set<Review> nodups = new HashSet<Review>(reviews);
  item.setReviews(new ArrayList<Review>(nodups));

  // Set num reviews for item
  int num_reviews = reviewDao.numReviews(itemid) + mongoRevDao.numReviews(itemid);
  item.setNum_reviews(num_reviews);

  // insert into MongoDB all reviews not present in MongoDB
  reviews.removeAll(mongoReviews);
  for(Review review :  reviews ){
    mongoRevDao.addReview(review);
  }
  ...
}
```

---

### Add Reviews to Items

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

Test this by selecting a few random items and add reviews into them.

To fully migrate, we could run script that iterates over all items.

This will use the internal application code to perform the migration!

---

**Note:** After given the students a chance to try on their own, navigate over the exercise solution, and share with them the purpose code implementation.

- `mongomart/rdbms/solutions/reviews_migration`

---

## 1.7 Lab2: `Items` Migration

---

**Note:** In this lab we will conducting 2 main learning objectives:

- After providing a new `mongodb.ItemDao` attendees should figure out the controller changes on their own.
- Restructure the schema design to integrate the last *10* comments within the `Items` collection using the aggregation pipeline

---

### Learning Objectives

In this lab we will be exploring the following operations:

- Enforce a full migration using hybrid solution
- Enable Text Search

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

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

- Integrate this class within the **MongoMart** application
  - `StoreController` is a good candidate for us to into!
- Once this new Dao is integrated
  - Validate that `Items` are getting stored into MongoDB

---

**Note:** Students will get the `dao.mongodb.ItemDao` file with solution for lab1:

- Students will have to integrate this new class into the `StoreController`

```
// StoreController.java
...
ItemDao itemDao = new ItemDao(connection);
            mongomart.dao.mongodb.ItemDao mongoItemDao = new mongomart.dao.mongodb.
↪ItemDao(mongoMartDatabase);
...
get("/add-review", (request, response) -> {
...
  HashMap<String, Object> attributes = buildItemResponse(itemid, itemDao,␣
↪mongoItemDao, reviewDao, mongoReviewDao);
...
});

get("/add-review", (request, response) -> {
...
  HashMap<String, Object> attributes = buildItemResponse(itemid, itemDao,␣
↪mongoItemDao, reviewDao, mongoReviewDao);
...
});
```

---

### Step 2: Iterate over all `Item`

To migrate all items into MongoDB, we can simply run the following script:

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

... and for our Windows friends:

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

---

**Note:** This step is important to move all `item` data back to mongod. Import notes to touch base with students:

- We are using the internal code application to do the migration
- Once we know that the data is fully migrate we should start disconnecting the *old* database, in this case MySQL.
- This makes the migration process easy to test
- Reduces the data impedance that can occur during migration process
- Reduces the need for external tools like ETL tools.

---

### Step 3: Enable Text Search

At this point, the Text Search[12] functionality is not yet enabled. Time to get it up and running:

- Go ahead and enable Text Search[13] on `items` collection.
- This functionality is provided by MongoDB
  - Text Search[14] should be enabled in the following fields
    * `title`
    * `slogan`
    * `description`

---

**Note:**

- Students will have to connect to the MongoDB shell and create the required Text Index:

```
mongo mongomart --eval 'db.item.createIndex( { "title" : "text", "slogan" : "text",
↪"description" : "text" } )'
```

- And enable the commented `/search` route, present on `StoreController`
- The Dao object to be used must be `mongoItemDao`

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

```
// Text search for an item, requires a text index
get("/search", (request, response) -> {
  String query = request.queryParams("query");
  String page = request.queryParams("page");

  List<Item> items = mongoItemDao.textSearch(query, page);
  long itemCount = mongoItemDao.textSearchCount(query);

  // Determine the number of pages to display in the UI (pagination)
  int num_pages = 0;
  if (itemCount > mongoItemDao.getItemsPerPage()) {
    num_pages = (int)Math.ceil(itemCount / mongoItemDao.getItemsPerPage());
  }

  HashMap<String, Object> attributes = new HashMap<>();
  attributes.put("items", items);
  attributes.put("item_count", itemCount);
  attributes.put("query_string", query);
  attributes.put("page", Utils.getIntFromString(page));
  attributes.put("num_pages", num_pages);

  return new ModelAndView(attributes, "search.ftl");
}, new FreeMarkerEngine(cfg));
```

- Show students that this works by opening the following url:

http://localhost:8080/search?query=mug

---

// TODO need to finish this one

// TODO need to finish this one

// TODO need to finish this one


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