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 backend.

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 looks 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 actual production level load. Something that should not be neglected in production environments .

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
```

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.

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 `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"
```

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

- Beware that *check.sql* is not optional. Make sure students do follow all all the instructions.

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

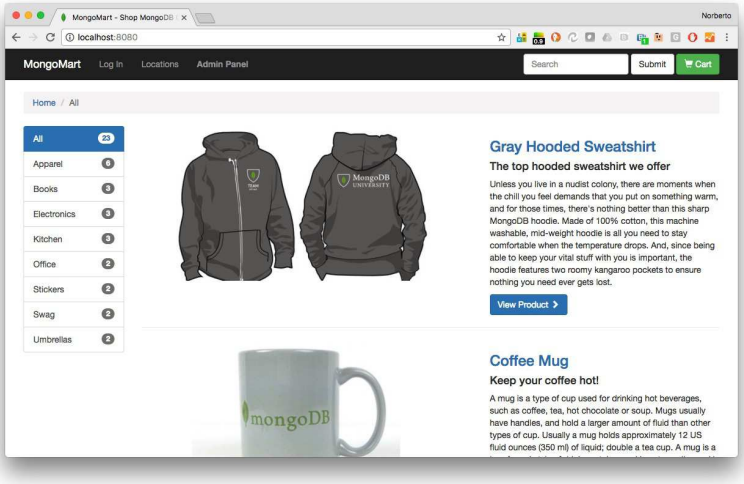
```
java -jar java/target/MongoMart-1.0-SNAPSHOT.jar
```

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 nice to 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

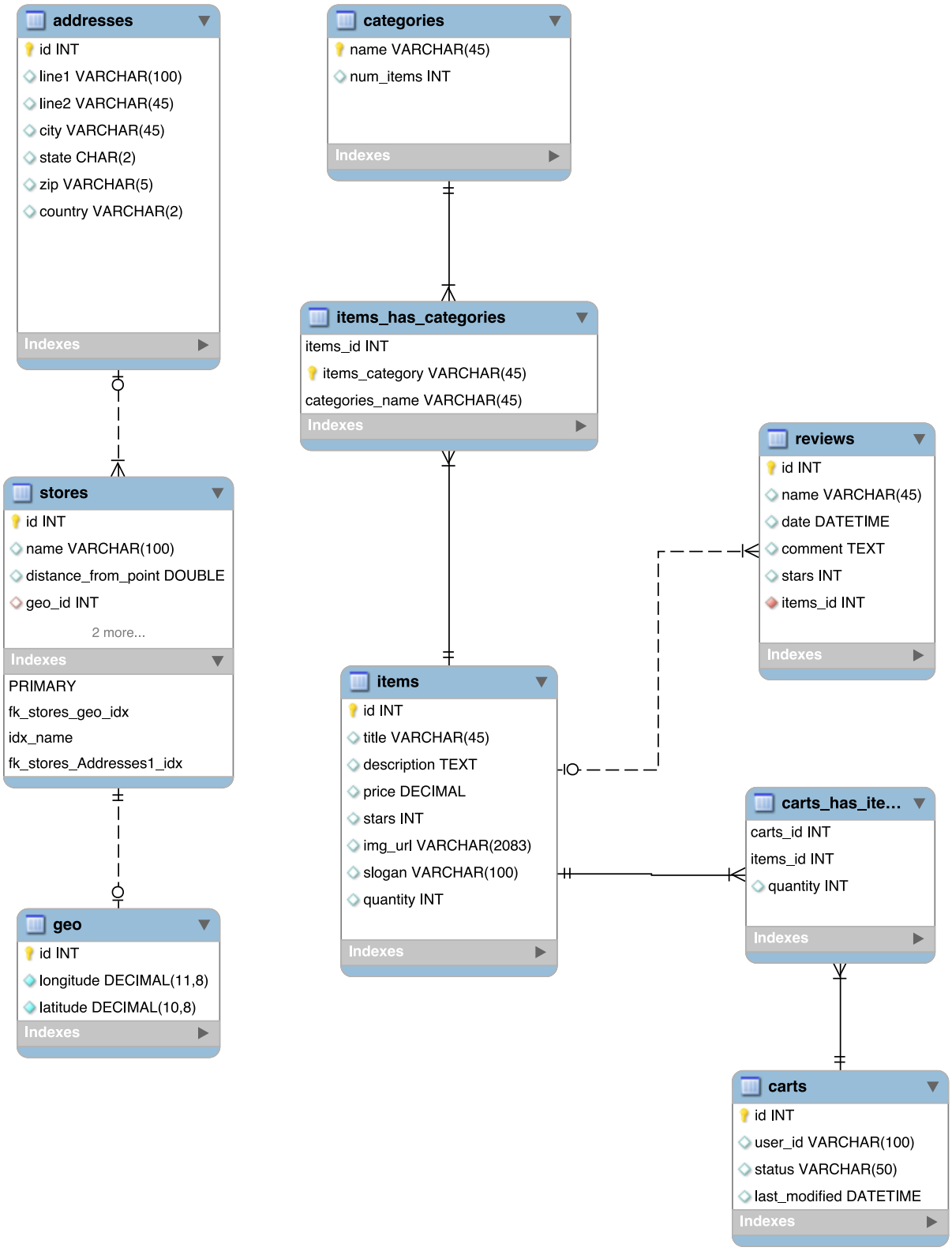
- http://localhost:8080/cart
- Adding an item multiple times increments quantity by 1
- Change quantity of any item
- Changing quantity to 0 removes item

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.

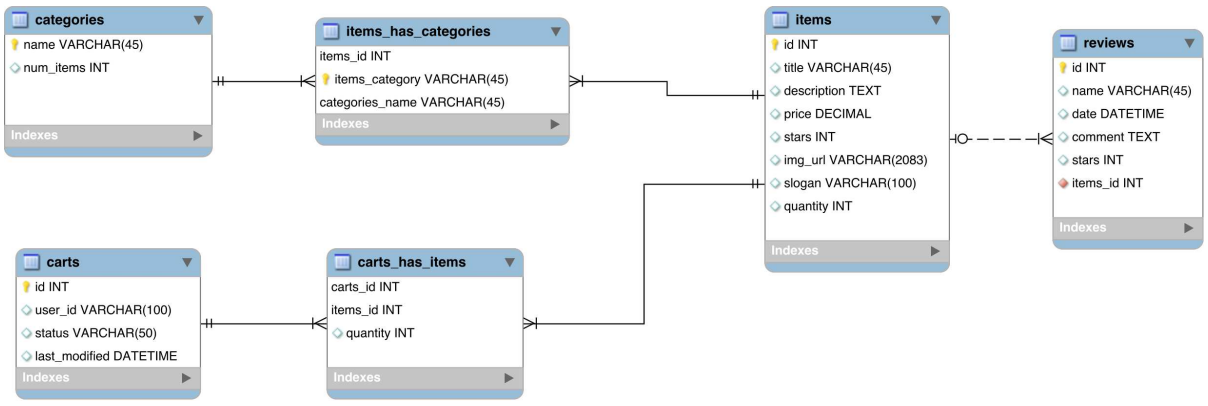
This is an important aspect of the migration process, understand the model that we are working with, and make decisions on how to approach the migration process.

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 several 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[12] 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.

---

[12] 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 these 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 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

---

Note: Raise awareness to:

- This happens with any database migration, not particular of MongoDB
- Intensive testing will need to carried away to ensure the viability of the application
- Sometimes customers do this to clean and force a code refactoring of their products
  - added benefit of any infrastructure change
  - code ends up being reviewed and refactored

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

Note: Briefly introduce Ops/Cloud Manager

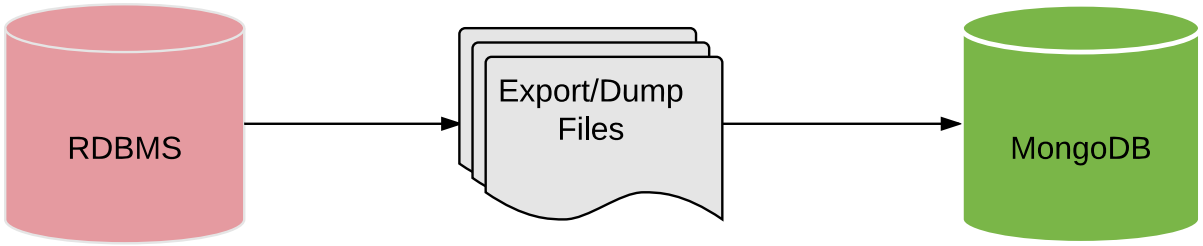
- https://cloud.mongodb.com/v2/57640e5ae4b07cb327ad53aa#metrics/host/a2d96b6d5f2fee00644f49fab7d19bc0/status

Use this link with extreme care. Don't expose any sensitive information.

Full one stop shop migration

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

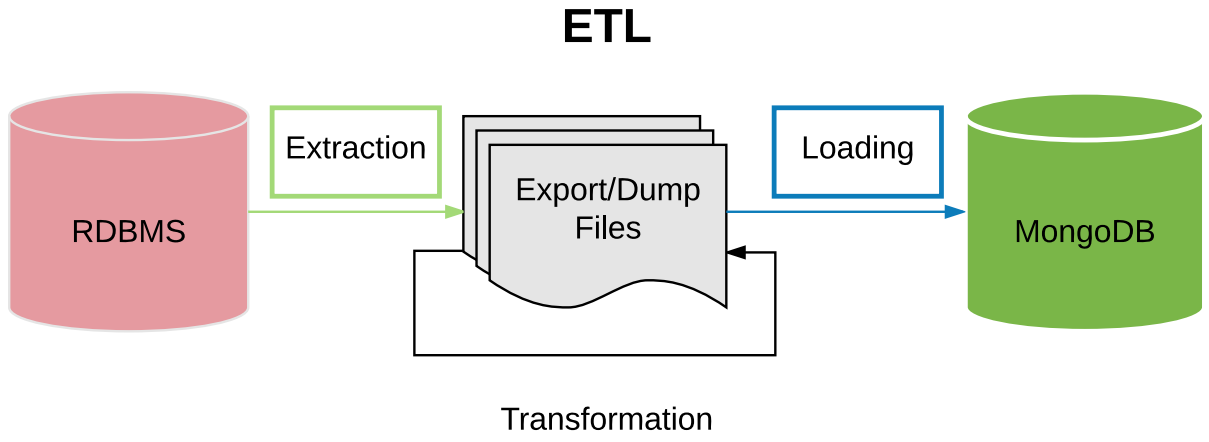
- We export a set of data that we want to see migrated



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.



Note: In this scenario we can highlight that:

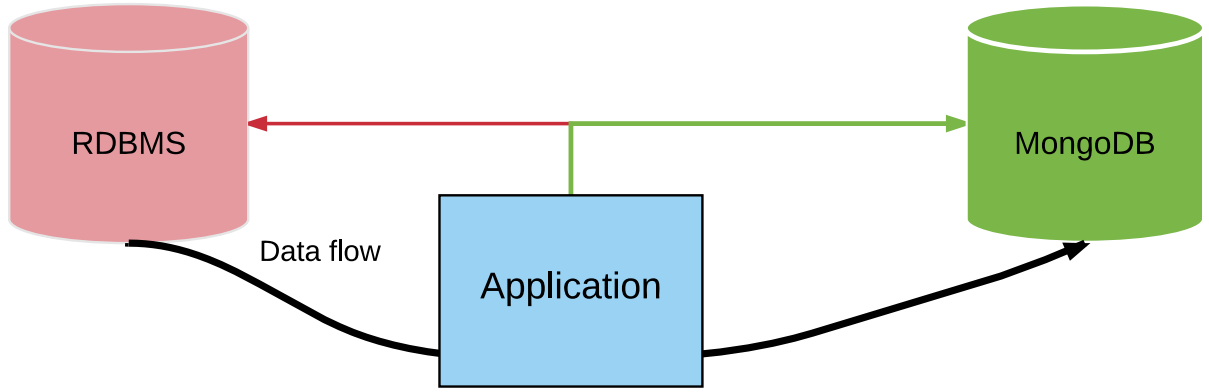
- This tends to be a process that requires lots of preparation and pre-testing
- May cause service interruption while the migration is taking place

---

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



Note: We should mention that:

- This strategy is driven by the application code
- Can make our migration process longer
- The components that are being migrated might suffer some extended latency given that we might be accessing two different databases for the same data
- The process is included within the release cycle and development practices
- Has code get's added, eventually some other code will be required to be removed since it's function will be temporary

---

Also mention that we will be using both strategies, for different components, throughout the workshop.

1.6 Lab1: `Reviews` Migration

Note:

- Given that this is the first lab where students will have to do some coding it would be useful to do a bit of hand-holding and complete the exercise together
- 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

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>
}
```

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

- This will be the correspondent primary key from the rdbms table
- since MongoDB does not have an auto-increment 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 `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(...)`

Note: A few things worth mentioning 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;
}
```

- This method is analogous to `resultSetToReview`, but for MongoDB. Parses a `Document` instead of a `ResultSet` into a `Review` object.
- `ReviewDao` class needs to receive a `Collection` or `Database` object

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

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 static Document reviewToDoc(Review review) {
  Document document = new Document();
  if(review.getId() >= 0){
    document.append("id", review.getId());
  }
  document.append("name", review.getName());
  document.append("date", review.getDate());
  document.append("comment", review.getComment());
  document.append("stars", review.getStars());
  document.append("itemid", review.getItemid());
  if (review.get_Id() != null){
    document.append("_id", review.get_Id());
  }
  return document;
}
```

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.

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.ReviewDao` also call `dao.mongodb.ReviewDao`

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

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

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

```
private HashMap<String, Object> buildItemResponse(int itemid, ItemDao itemDao,␣
↪ReviewDao reviewDao, mongomart.dao.mongodb.ReviewDao mongoRevDao) {
  List<Item> related_items = itemDao.getItems("0");


  Item item = itemDao.getItem(itemid);

  // Get reviews from MongoDB
  List<Review> reviews = mongoRevDao.getItemReviews(itemid);
  // Get reviews from rdbms
  for( Review rev : reviewDao.getItemReviews(itemid)){
    // Remove duplicates
    if (!reviews.contains(rev) ){
      reviews.add(rev);
      mongoRevDao.addReview( rev );
    }
  }

  item.setReviews(reviews);

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

  HashMap<String, Object> attributes = new HashMap<>();
  attributes.put("item", item);
  attributes.put("itemid", itemid);
  attributes.put("related_items", related_items.subList(0, 4));
  return attributes;
}
```

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
```

Note: After giving the students a chance to try on their own, navigate over the exercise solution code.

- `mongomart/rdbms/solutions/lab1`

1.7 Lab2: `Items` Migration

Note: In this lab, we will focus on two primary learning objectives:

- After providing a new `mongodb.ItemDao`, attendees will figure out the controller changes on their own.
- Restructureing 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:

- 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

Note: Point the students to the `dao.mongodb.ItemDao` file with the solution for lab1.

- Students will have to integrate this new class into `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 `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 )
```

Note: This step is important to move all `items` data back to `mongod`. Important things to mention:

- We are using the *internal* code application to do the migration
- Once we know that the data is fully migrated, we should start disconnecting the *old* database (MySQL in this case).
  - This makes the migration process easy to test
  - Minimizes the chance of data impedance mismatch during migration process
  - Eliminates the need for external tools (e.g., ETL tools).

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`

Note:

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

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

- Next, they'll need to enable the commented `/search` route, present on `StoreController`
- The `Dao` object should be `mongoItemDao`

[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/

```
// 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

1.8 Lab3: Full Migration

Note: In this lab, we will import & transform our data.

- Students will first apply the code solution for **lab3**.
- Next, they will import the `dataset/csv` folder containing a RDBMS data dump.
  - Point them to the `README` file with import instructions.

Once the data is loaded, they will transform it using the aggregation framework and views as follows:

- First, create the views.
- Next, use the `$out` aggregation stage to persist the data in that view.

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

---

Note: Make sure to clarify that the term *materialize* is unrelated to *materialized views*.

- Instead, it means that after we create a view that reflects our schema, we can store that data using the `$out` command.

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.

Note: Students still have to compile their application after applying the code solution.

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
```

Note: Ask the question:

- Is exporting/importing `CSV` the best way to migrate data between databases?

The idea is to bring the students' attention to the fact that this operation is only possible here because the data in this table is in easy transferable data types, i.e., integers and strings. If higher precision data types were present, such as dates, binary data and others, the migration would not be possible with `mongoimport` and would require for ETL tools.

With this operation, data will be placed in MongoDB in a very denormalized format, i.e., one MongoDB collection per table in MySQL.

We will transform this data into something more "application friendly," with hierarchy and nesting manifest in the data, which you can do with MongoDB but not with relational databases.

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>,

}
```

Note: This is the primary challenge of this lab. We want students to use the aggregation framework to execute the required transformation using only MongoDB.

Do mention to students:

- This is a sub-optimal strategy that works *here* because we set it up to work in this lab. In a real-world scenario, data would need to be imported directly into the expected format.
- That said, this **is** a nice tool to have in their tool kit, **if** they need to perform a data transformation on already-imported data.

The students may come up with their own pipeline, but it should be functionally identical to (and probably similar to) the following:

```
var view_pipeline = [
  {"$lookup": { "from": "address_sql", "foreignField": "id", "localField": "address_id
↪", "as": "addresses"  }} ,
  {"$lookup": {"from": "geo_sql", "foreignField": "geo_id", "localField": "geo_id",
↪"as": "coords" }},
  {"$project": {"coords": {"$arrayElemAt": ["$coords", 0]}, "address": {"$arrayElemAt
↪": ["$addresses", 0]}, "name": 1}},
  {"$project":  {"address.id":0, "address._id": 0, "coords.geo_id": 0, "coords._id":␣
↪0} },
  {"$project":  {"coords":["$coords.lon", "$coords.lat"], "name": 1, "address": 1} },
```

```
];
db.createView("store", "store_sql", view_pipeline)
```

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].

Note:

- Persisting this data into is own collection makes more sense than to use a view, since it will simplify our schema.
- The advantages of doing it this way are fewer collections to maintain, and more efficiency in handling the data.
  - Using a view with several `$lookup` stages would use system resources.
- Views should only be used as an intermediary stage, while we are figuring out the schema that we want to use to support the application

To solve this step, students will have to run something like the following script:

```
db.store.drop()
var out_pipeline = [
  {"$lookup": { "from": "address_sql", "foreignField": "id", "localField": "address_id
↪", "as": "addresses"  }} ,
  {"$lookup": {"from": "geo_sql", "foreignField": "geo_id", "localField": "geo_id",
↪"as": "coords" }},
  {"$project": {"coords": {"$arrayElemAt": ["$coords", 0]}, "address":
  {"$arrayElemAt": ["$addresses", 0]}, "name": 1}}, {"$project":
  {"address.id":0, "address._id": 0, "coords.geo_id": 0, "coords._id": 0}
  }, {"$project":  {"coords":["$coords.lon", "$coords.lat"], "name": 1,
  "address": 1} }, {"$out": "store"} ];

db.store_sql.aggregate(out_pipeline); db.getCollectionNames()
```

Another issue with using views to simulate a collection is that our `$geoNear` stage would not work. `$geoNear` must be the first stage of an aggregation pipeline, so persisting the view with `$out` is mandatory here.

- When the view is done, have the students create the corresponding `2dSphere` indexes.

```
db.store.createIndex( { "coords" : "2dsphere" } )
```

---

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

1.9 Lab4: Final Migration Cleanup

Note: This is the final lab of the workshop. After completing this lab, it will be safe to turn off the RDBMS system that we started with.

We will be looking into:

- Cleaning up the code post-migration
- Data handling operations, like making sure we create a backup of the RDBMS system.
- Cleaning up any intermediate collections created during the loading and data modeling phases.

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.

Note: Tell the students that:

- They should fire up their text editors / IDE's to refactor and purge some code.
- We will review our collections and check which can go, and what indexes are missing from the ones we keep.
- Finally, we will shut down our **legacy** database.

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.

Note: Given that this is the final lab, if you are running out of time, jump directly to the solution. Otherwise, let the students dive into the code on their own for about 10 mins or so.

Students should:

- Start by replacing all `import``s that include ``dao.rdbms` libraries with `dao.mongodb` ones.

```
//StoreController.java
...
import mongomart.config.Utils;
import mongomart.dao.mongodb.ItemDao;
import mongomart.dao.mongodb.ReviewDao;
import mongomart.model.Category;
...
```

- Expunge references to these libraries and remove the extended path call for `dao.mongodb` objects

```
//StoreController.java
...
// ItemDao itemDao = new ItemDao(connection);
ItemDao mongoItemDao = new ItemDao(mongoMartDatabase);
// ReviewDao reviewDao = new ReviewDao(connection);
ReviewDao mongoReviewDao = new ReviewDao(mongoMartDatabase);
...
// replace itemDao with mongoItemDao
```

- Change the `buildItemResponse` method signature

```
private HashMap<String, Object> buildItemResponse(int itemid, ItemDao mongoItemDao,␣
↪ReviewDao mongoRevDao) { ... }
```

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`

Note: Without the need to instantiate the `dao.rdbms` objects, we no longer need to pass the `Connection` object to controllers.

- Update `StoreController` constructor
- Update the `MongoMart` store controller call.

```
//MongoMart.java
...
StoreController storeController = new StoreController(cfg, mongoClient.getDatabase(
↪"mongomart"));
...
```

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

---

Note: Remove all jdbc related code from `MongoMart` class and switch the `connectionString` to a mongodb default URI

```
// MongoMart.java
...
import java.io.IOException;
//import java.sql.Connection;
//import java.sql.DriverManager;
//import java.sql.SQLException;
...

try {
    // The newInstance() call is a work around for some
    // broken Java implementations
    // Class.forName("com.mysql.jdbc.Driver").newInstance();

    String connectionString = "mongodb://localhost";
    if (args.length != 0) {
      connectionString = args[0];
    }

    new MongoMart(connectionString);

} catch (Exception ex) {
    System.out.println("Bad things happen: " + ex.getMessage());
}

...

// Create a Database connection
try{
    // Connection connection = DriverManager.getConnection(connectionString);

    ...

    // MongoClient connection
    mongoClient = new MongoClient();

...
```

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

---

Note: Ask students which indexes they should be creating:

- item: name, category
- address: city, country, zip

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