MongoDB Relation Migrator

Aug 21, 2023

What is Relation Migrator

MongoDB Relational Migrator is a tool to help you migrate relational workloads to MongoDB.

MongoDB Relational Migrator

By the end of this tutorial you will be able to:

- Design an effective MongoDB schema, derived from an existing relational schema.
- Migrate data from Oracle, SQL Server, MySQL, PostgreSQL to MongoDB, while transforming to the target schema.
- Generate code artifacts to reduce the time required to update application code.

App Explanation

This tutorial will create a MySQL database for an event company, in which we will convert it to MongoDB schemas.

Agenda

In this forum we are going to discuss the basics of Relation Migrator and how we can configure the Relation Migrator to migrate data from MySQL database hosted on AWS RDS to MongoDB Atlas.

1. RDMS database setup
2. Understanding how relational migrator works
3. Create RDBMS schema for MongoDB from scratch
4. Migrate data with 2 different modes (snapshot and continuous)
5. Troubleshooting

Prerequisites

RDBMS database

Create an instance of any of the following databases: Oracle, SQL Server, MySQL or PostgreSQL, in this tutorial we will use MySQL through AWS RDS.

To help us in this tutorial we will use a CloudFormation template to create a MySQL instance demo

Method 1 (Quick create stack)

1. Click here
2. Click on "Create Stack"

Method 2 (Manual create stack)

1. Go to CloudFormation page on your AWS console
2. Select the option "Template is ready"
3. Select "Amazon S3 URL", paste the url and click on next
    a. Template url:
    https://delbridge-tutorials.s3.amazonaws.com/relationalmigrator/mysql.yml
4. Type any name in the stack name and click on next
5. In the step 3 (Configure stack options) keep the default values and click on next
6. Click on Submit

this process can take between 5~10 minutes to complete, you can continue to the next step until this progress is complete

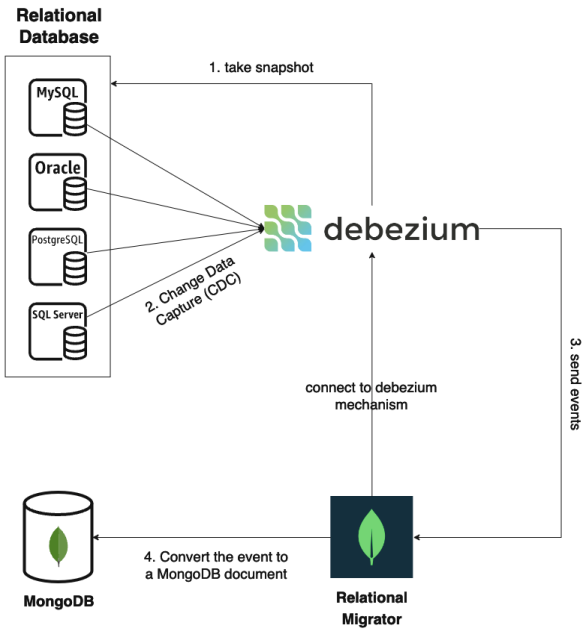
How works

MongoDB Relational Migrator is a web application running on localhost written in React (WebUI) and Java, you can download the application on the download page in the mongodb website.

This tool uses some other software to help in the migration process, the principals are SchemaCrawler and Debezium.

SchemaCrawler will act as schema discovery, reading the schema from the relational database to use inside the relational migrator tool

Debezium is an open-source distributed platform designed for change data capture (CDC). When utilized in conjunction with Relational Migrator, Debezium assumes the role of a listener, capturing every instance of inserts, updates, and deletes made within the database. Subsequently, this data is then replicated over to MongoDB, facilitating a smooth and comprehensive migration process.



The Migrator can analyze database schemas from live databases and create ERDs for relational and MongoDB schemas. It allows denormalisation and consolidation of tables into documents. It is a job-based system for managing data synchronization, enabling you to map relational schemas to MongoDB Schemas. You can currently only migrate a snapshot of the data, but continuous sync is possible as well.

Steps

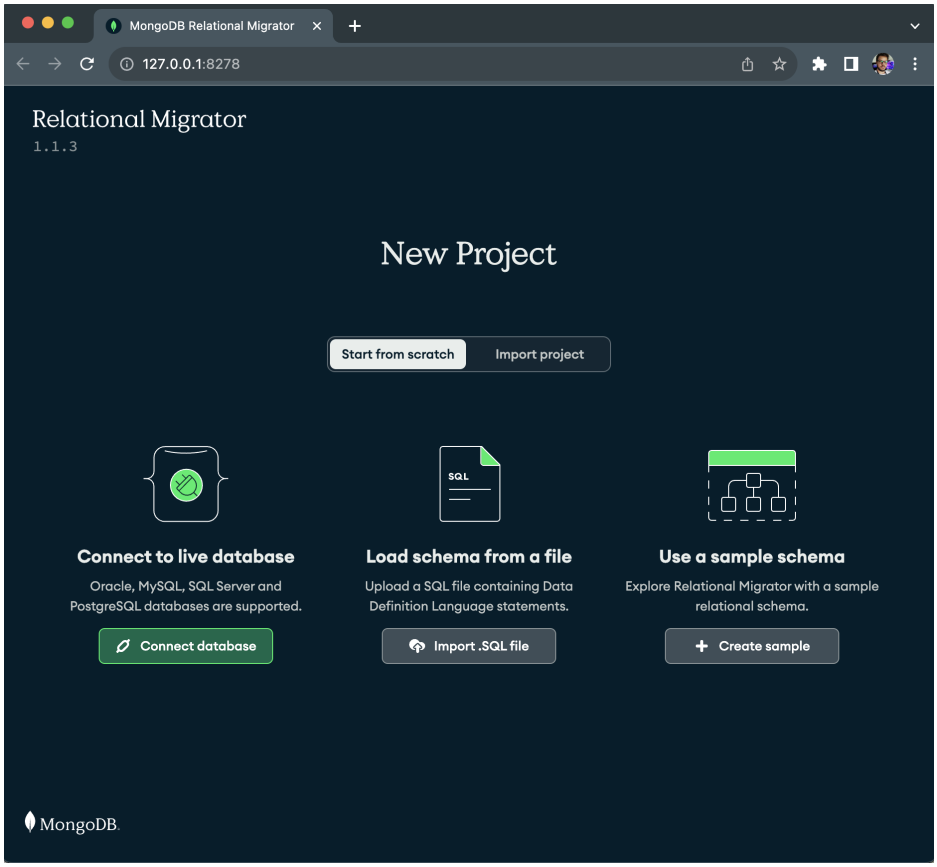
1. Download and install MongoDB Relational Migrator tool

Since this tool operates as a standalone application on your local machine, you need to download the software and complete the installation on either your Windows or Mac system. Alternatively, you can choose to install it on a server, be it Debian or RedHat. The future production-scale deployment will run on a server and migrations will use Kafka

- Click here to download
- Install
  - Install on Windows
  - Install on Mac
  - Install on Ubuntu
  - Install on RHEL

2. Execute the application

Once installed, open the Relation Migrator. Next, click on the "Launch-UI" button to initiate the process. This action will prompt a web page to open at the following URL: http://127.0.0.1:8278/



3. Create new project

Upon launching the application for the first time, users will be directed to the new project page. Here, a choice between two database migration options awaits:

- Live Database Connection: This option involves directly connecting to the active database.
- Load Schema from a file: Alternatively, users can opt to utilize a dump file (.sql) containing all the necessary data for migration.

For the purpose of this tutorial, we will be focusing on the "Connect to Live Database" method.

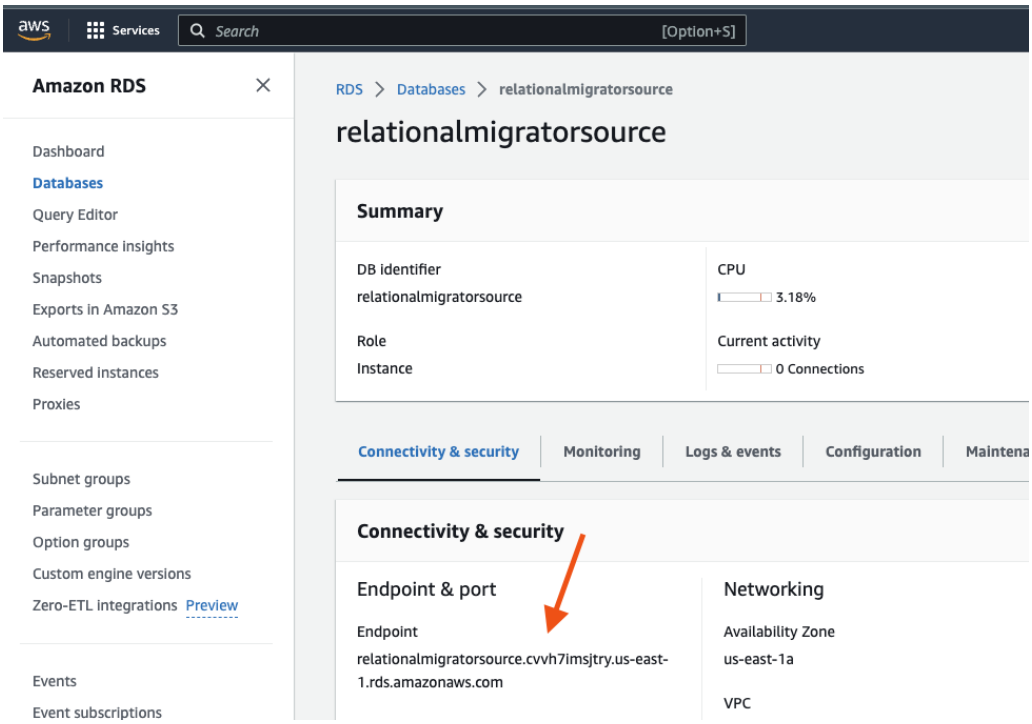
3.1 Connect to the database

On this page, we need to provide the credentials of the relational database established in the preceding prerequisite phase. Alternatively, you can also input the credentials of any other RDMS you intend to utilize.

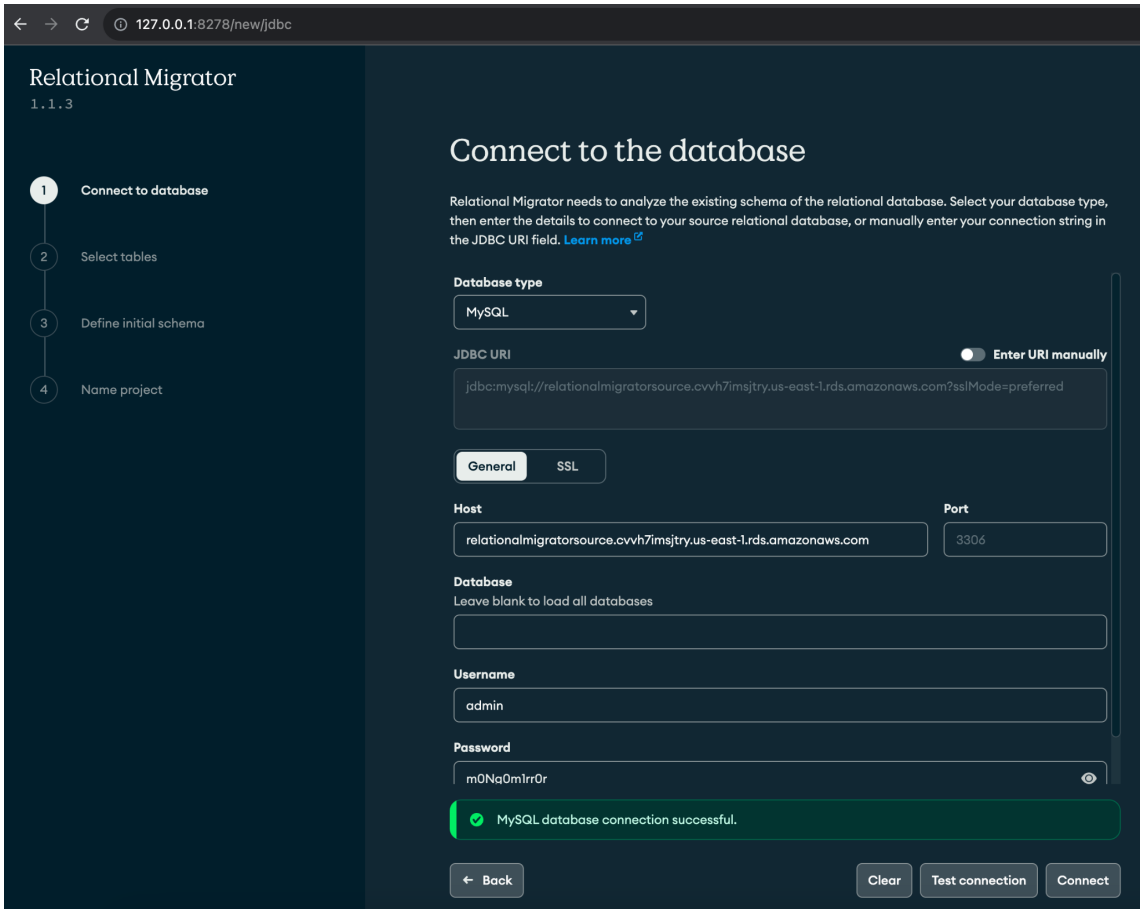
To get the endpoint for MySQL on AWS RDS, go to the RDS page in your account and click on the instance name “relationalmigratorsource”

In the "Connectivity & security" tab copy the endpoint and the credentials is

- Username: admin
- Password: m0Ng0m1rr0r



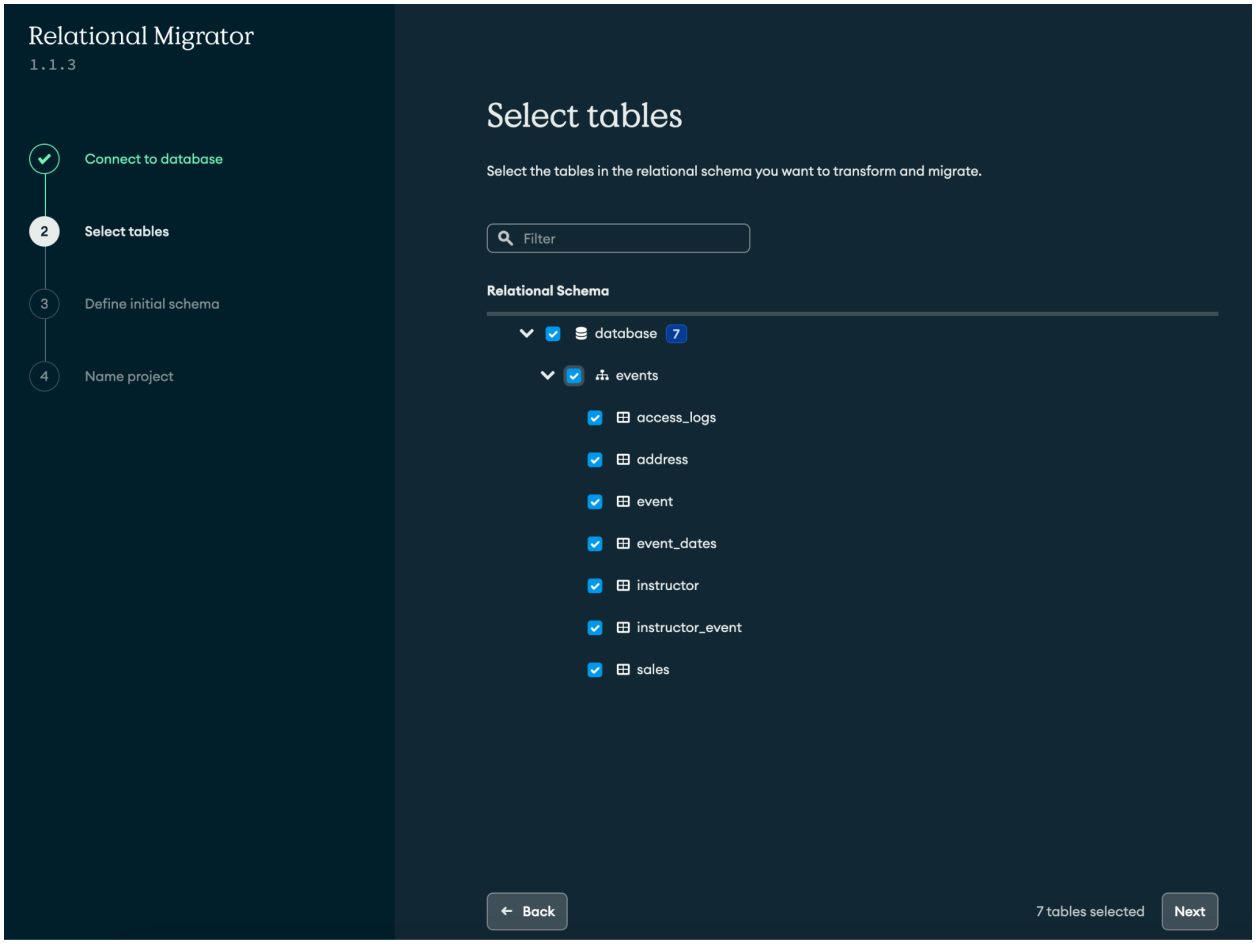
Prior to progressing to the Relation Migrator tool, ensure that you establish a connection to your MySQL database using any preferred client. Once connected, proceed to restore the sample data. You can conveniently download the sample data employed in this tutorial by clicking on the provided link.



After configuring the connection, click on the “Test connection” button, if everything is ok you will see this message “MySQL database connection successful” and then click in the connect button

3.2 Select the database and tables

At this stage, you should select all the desired databases and tables that will be targeted for migration to MongoDB. After making your selections, click the "Next" button.



4. Initial Schema

During this step, the Relation Migrator employs SchemaCrawler to extract the relational schema from your RDMS. This process results in the generation of suggestions for your consideration. You will have three options to choose from.

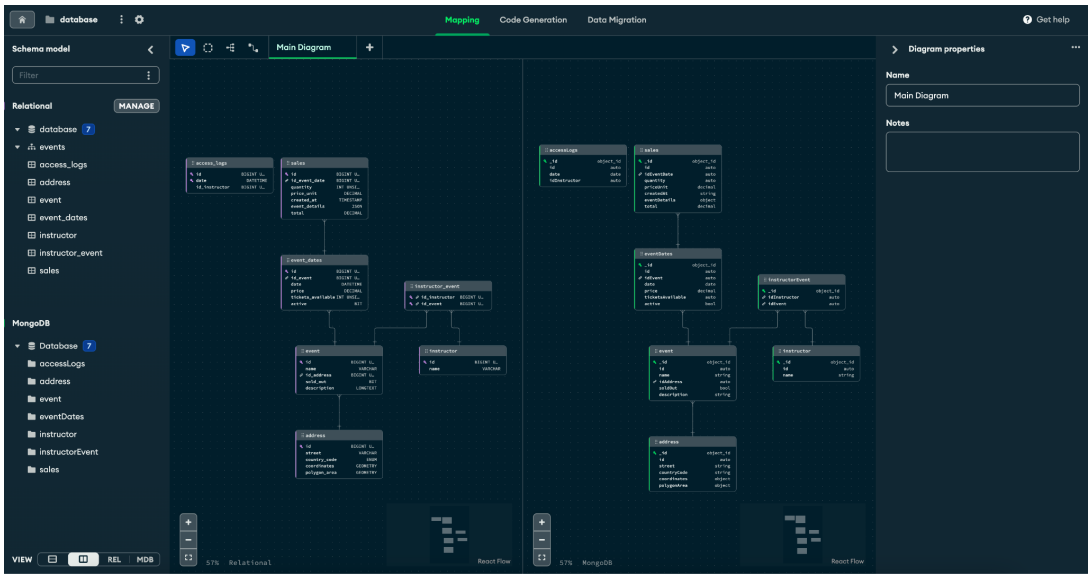
- Start with a MongoDB schema that matches your relational schema
    - Basically will create one collection for each table
- Start with a recommended MongoDB schema
    - Will be suggested a possible schema
- Start with an empty MongoDB schema
    - Will not create any collection, you will need each desired collection on the next stage manually

This selection doesn't lock you into your choice; you can still modify the schema in the upcoming diagram design stage. It serves as an initial step to prevent starting from scratch.

For the purposes of this tutorial, we will select the first option to create all collections for each table and redraw the schema in the diagram design phase.

5. Schema Model

This page shows the relational schema extracted from your database on the left, along with the initial schema created for MongoDB on the right.



As we selected the first option in the Initial Schema "Start with a MongoDB schema that matches your relational schema" we have all collections for each table, basically it's a copy from or relational database but in collection.

At this stage you should analyze your MongoDB schema and check if the recommended schema (in case you choose the 2nd option in the initial schema) is a good choice based on the principles of MongoDB Schema Design Best Practices, or if you chose the 1st or 3rd option you need create manually your MongoDB Schema, with this tool is very straightforward to design just click in your collection and add or remove new mappings

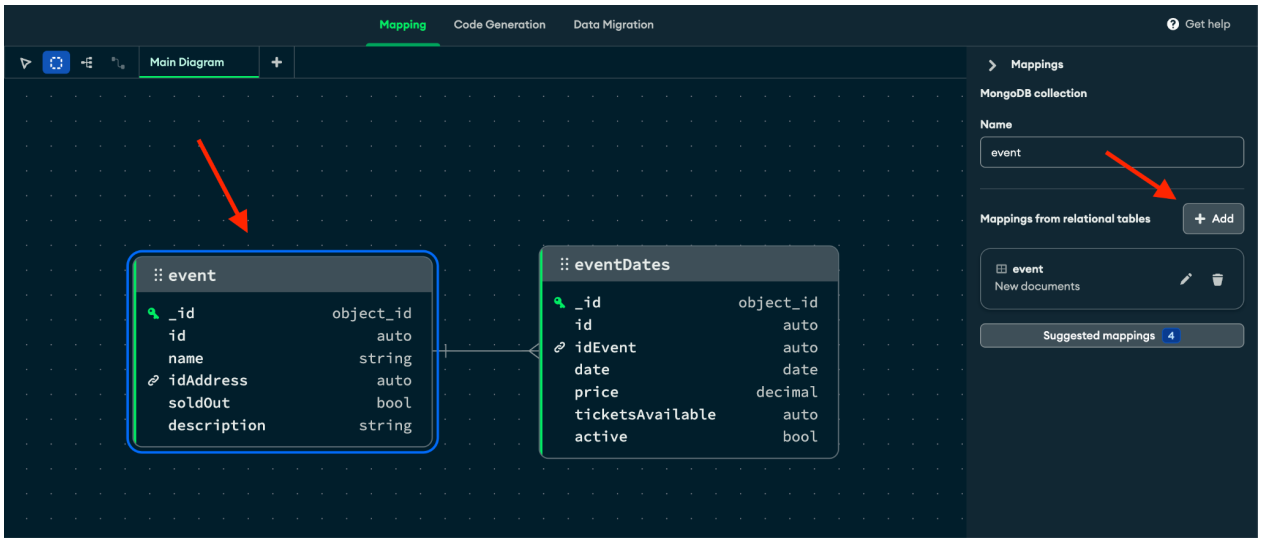
As we start with no-schema recommend we need to transform our schema manually

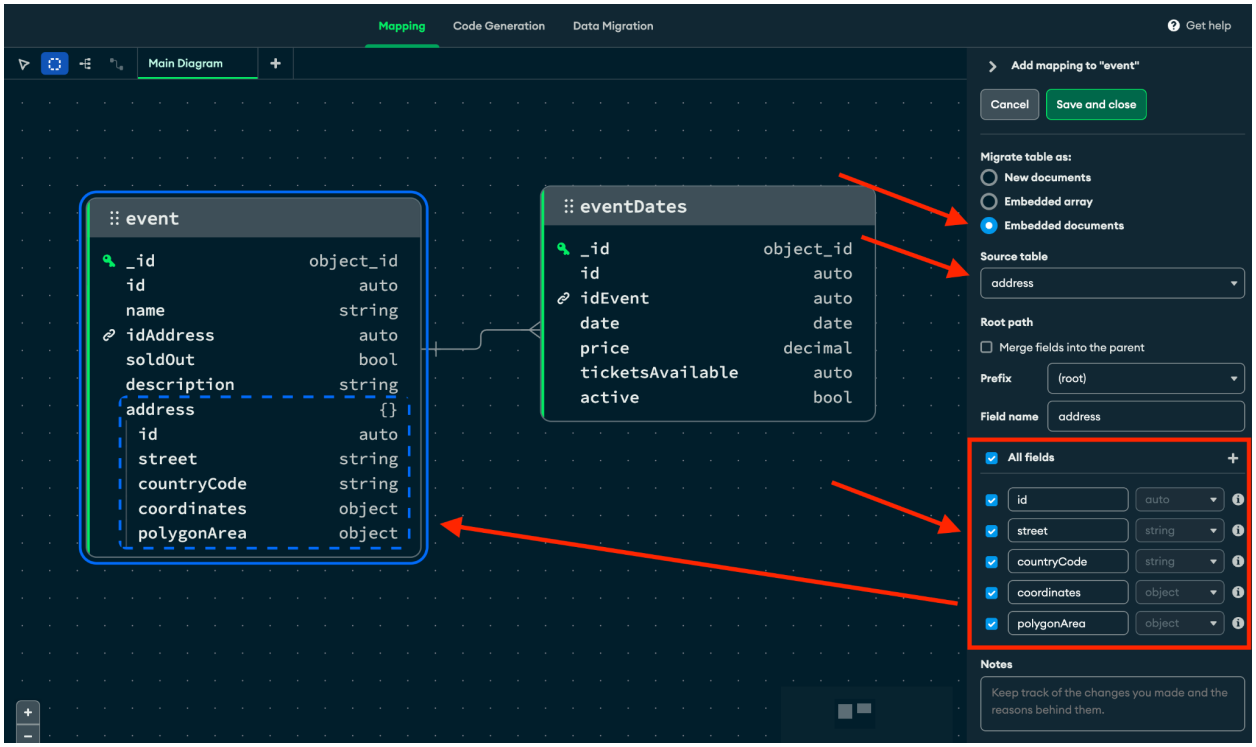
5.1 Convert 1-1 relationship with subdocuments in MongoDB schema

In your design, there exists a one-to-one relationship between "event" and "eventAddress," signifying that each event is associated with just one address. To translate this relationship into a MongoDB document, we will employ the technique of embedded documents. Therefore, within the RelationMigrator, we are required to establish this mapping rule. Fortunately, the process for doing so is relatively straightforward.

Steps

1. In the MDB Diagram (the right panel)
2. Click on the main collection (event)
3. In the right panel click in the “+ add” button
4. Select the “Embedded document” option
5. Select the target collection (event_address)
6. Unselect the unnecessary fields or keep all fields selected to be part of the sub document “address” that will be created in the “event” collection
7. Optionally we can mark the option “Merge fields into the parent” to add the fields in the root document and not as a subdocument


Mapping
Code Generation
Data Migration
Get help
Main Diagram
event
_id object_id
id auto
name string
idAddress auto
soldOut bool
description string
eventDates
_id object_id
id auto
idEvent auto
date date
price decimal
ticketsAvailable auto
active bool
Mappings
MongoDB collection
Name
event
Mappings from relational tables
Add
event
New documents
Suggested mappings 4

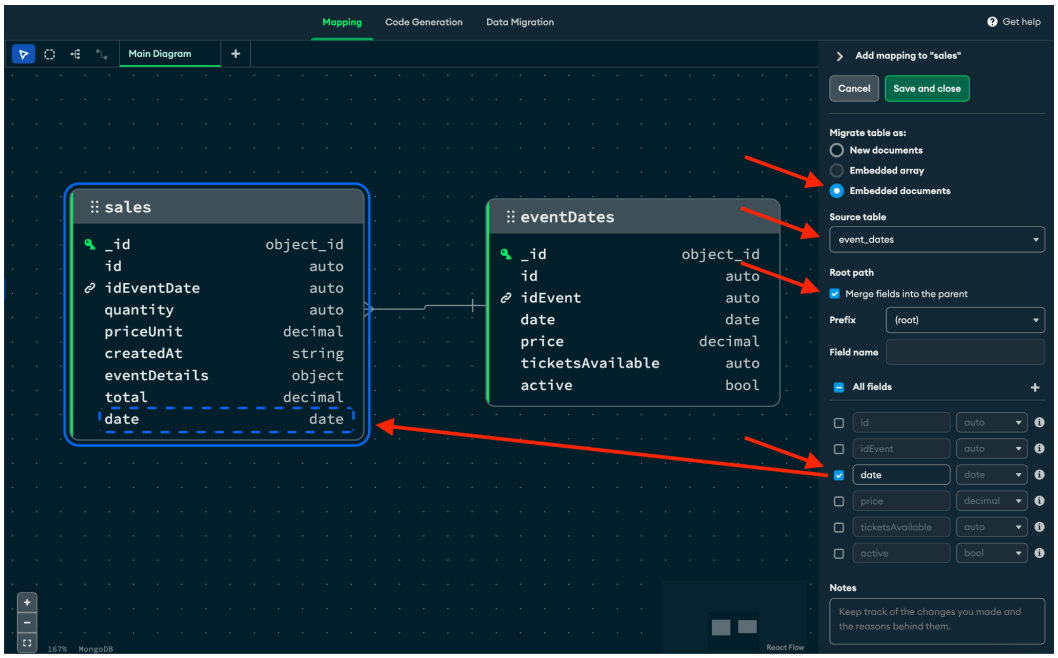

Mapping
Code Generation
Data Migration
Get help
Main Diagram
event
_id object_id
id auto
name string
idAddress auto
soldOut bool
description string
address {}
id auto
street string
countryCode string
coordinates object
polygonArea object
eventDates
_id object_id
id auto
idEvent auto
date date
price decimal
ticketsAvailable auto
active bool
Add mapping to "event"
Cancel
Save and close
Migrate table as:
New documents
Embedded array
Embedded documents
Source table
address
Root path
Merge fields into the parent
Prefix
(root)
Field name
address
All fields
id auto
street string
countryCode string
coordinates object
polygonArea object
Notes
Keep track of the changes you made and the reasons behind them.

5.2 Convert 1-1 relationship with field in the root MongoDB document

For the “sales” collection we can use the same 1-1 mapping but in this case not create one subdocument, we can store the field directly in the root collection for this scenario

Steps

1. Click on the main collection (sales)
2. In the right panel click in the “+ add” button
3. Select the “Embedded documents” option
4. Select the target collection (event_dates)
5. Select “Merge fields into the parent”
6. Unselect all fields except the “date” field
7. Click and “Save and Close”



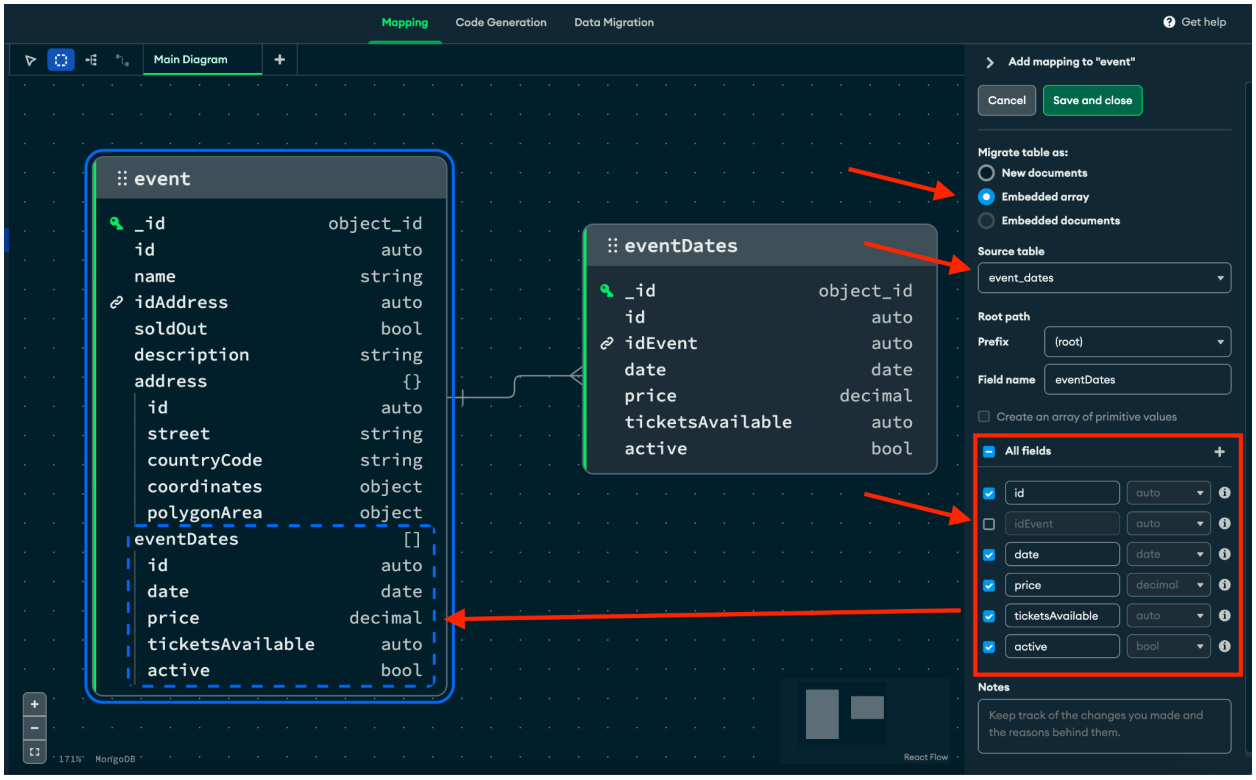
Repeat the same process but with the relation between sales and events to get the event name

5.3 Convert 1-N relationship to MongoDB schema

For a one-to-many relationship we will use the collections "event" and "eventDates". To translate this relationship into a MongoDB document, we will employ the technique of Embedded array.

Steps

1. Click on the main collection (event)
2. In the right panel click in the “+ add” button
3. Select the “Embedded array” option
4. Select the target collection (event_dates)
5. Unselect the foreignKey in the eventDates collection (idEvent)
6. Click and “Save and Close”

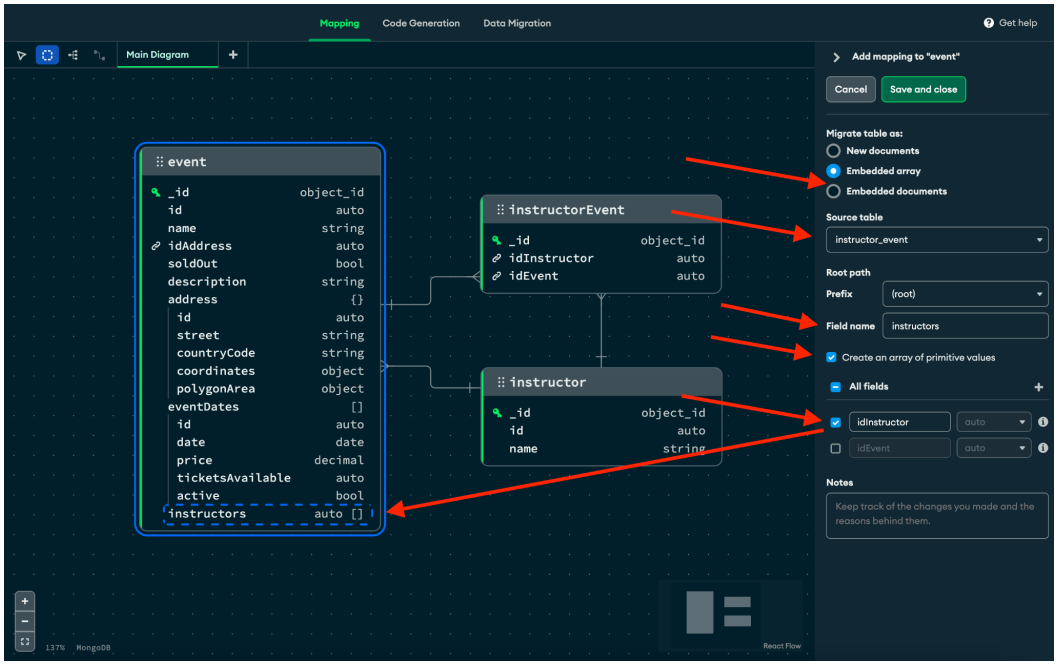


5.3 Convert N-N relationship to MongoDB schema

For a many-to-many relationship we will use the collections "event" and "instructor" through the “instructorEvent” collection. To translate this relationship into a MongoDB document, we will employ the technique of Embedded array with Referencing.

Steps

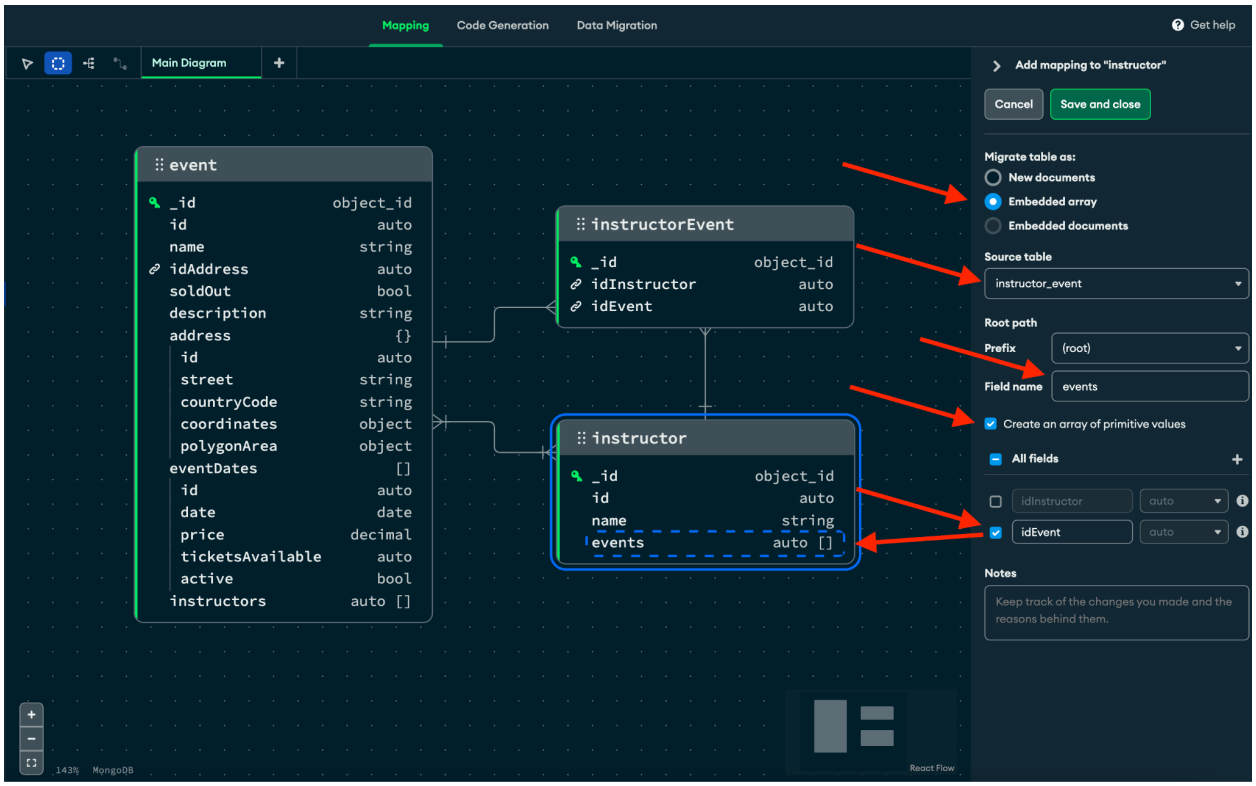
1. Click on the main collection (event)
2. In the right panel click in the “+ add” button
3. Select the “Embedded array” option
4. Select the target collection (instructor_event)
5. Rename the field name to “instructors”
6. Unselect the “idEvent” field
7. Select the “Create an array of primitive values”
8. Click and “Save and Close”



Do the same in the instructor collection to keep the relationship in the both side

Steps:

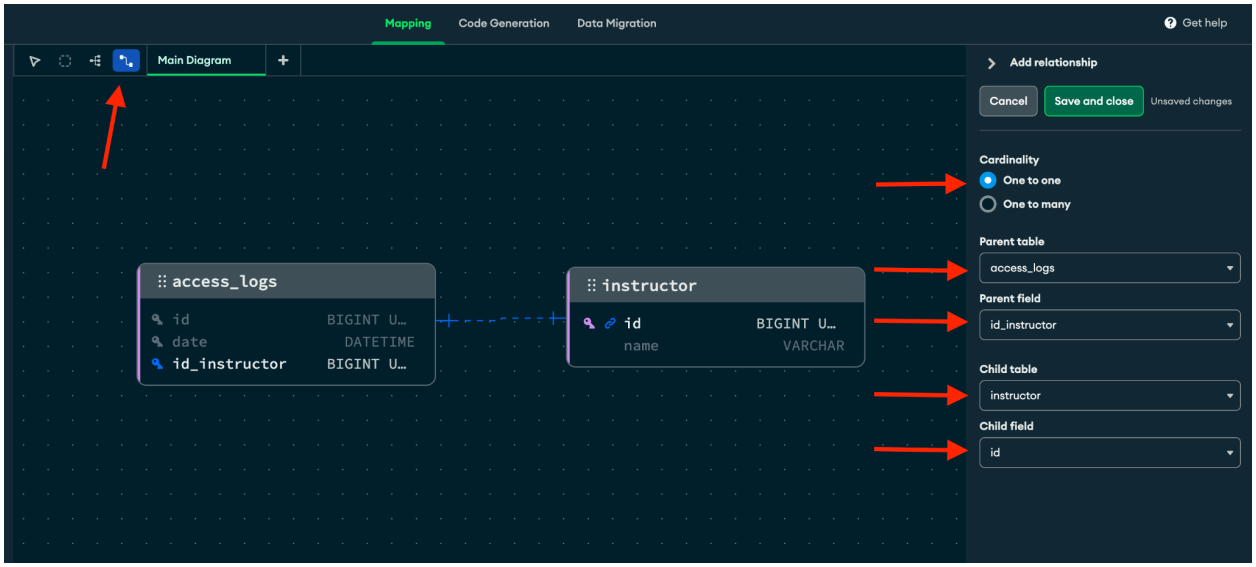
1. Click on the main collection (instructors)
2. In the right panel click in the “+ add” button
3. Select the “Embedded array” option
4. Select the target collection (instructor_event)
5. Rename the field name to “events”
6. Unselect the “idInstructor” field
7. Select the “Create an array of primitive values”
8. Click and “Save and Close”



6. Synthetic Foreign Key

In our accessLog collection on MongoDB we have the idInstructor, but in our RDMS we do not have this relationship created, to bypass that, we can create a “virtual” relationship to give access to instructor collection through accesLog collection. To do that follow the steps below

1. On the left panel (RDMS schema) click in the “add synthetic foreign key”
2. On the right panel click in the “parent table” and select the “access_log”
3. Select One to one in the cardinality
4. Select the id_instructor on the “Parent field” option
5. Select instructor on the “Child table” option
6. Select id on the “Child field” option
7. Click on “Save and close”



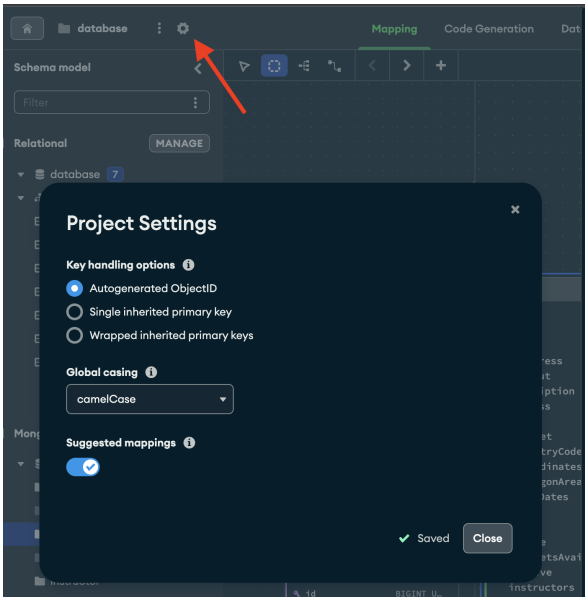
After that we can put the instructor as embedded document in the accessLog collection

7. Project settings

We can define some project settings like how will be the key handling or the collections name format

Key handling options

1. **Autogenerated ObjectId**: the Relational Migrator will generate a new ObjectId for each document and will keep the primary key from RDMS
2. **Single inherited primary key**: will use the primary key from RDMS as _id
3. **Wrapped inherit primary keys**: will use the _id as object where contains the primary key from RDMS inside this object. e.g: {_id: {id: 123}}



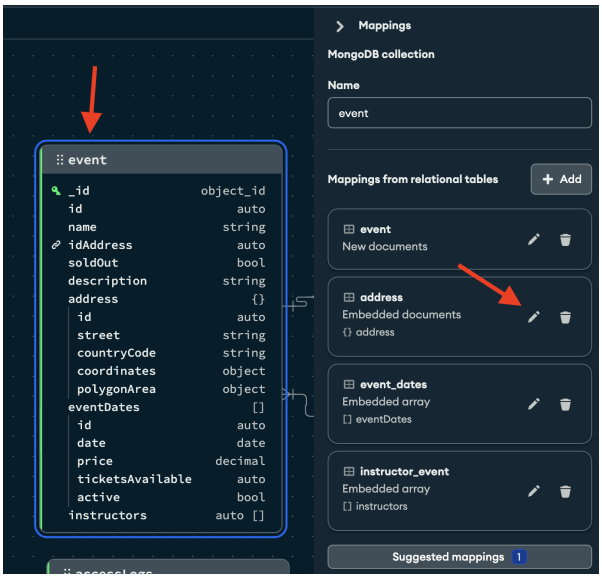
8. Calculated fields

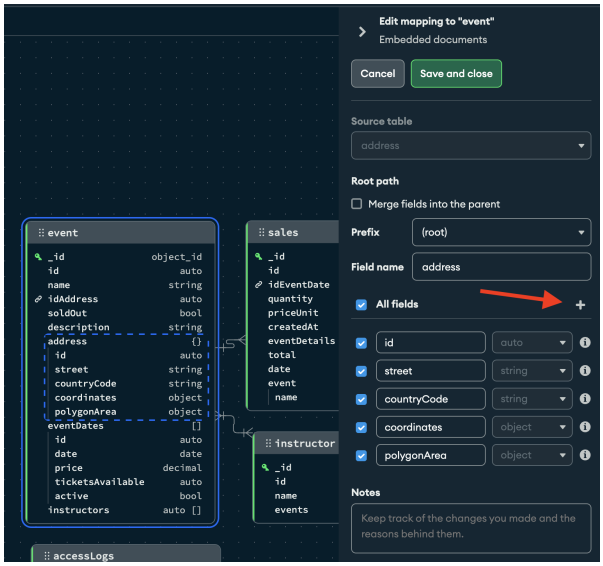
We can create a javascript expression to create a [calculated field] in our MongoDB document. In our case let's convert the MySQL coordinate object to a string split by comma, to do that follow the steps below

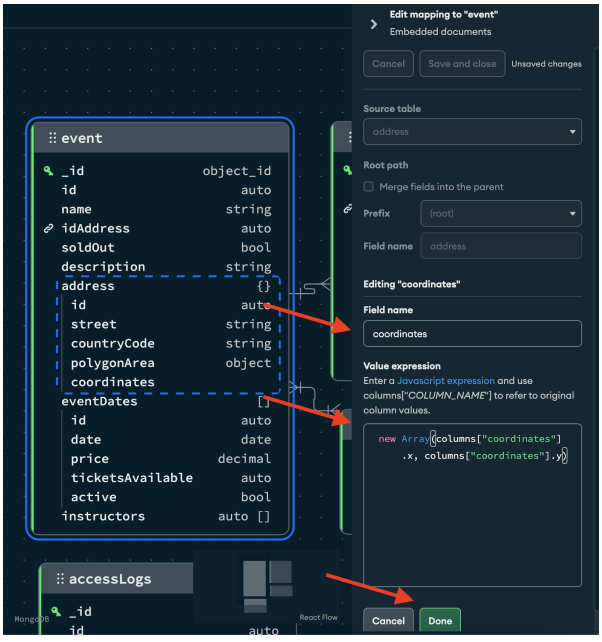
1. Click in the event collection
2. Click on “pencil” icon in the address mappings
3. Click in the “plus” icon in the fields sections
4. Add the name in the new field
5. And type the follow code

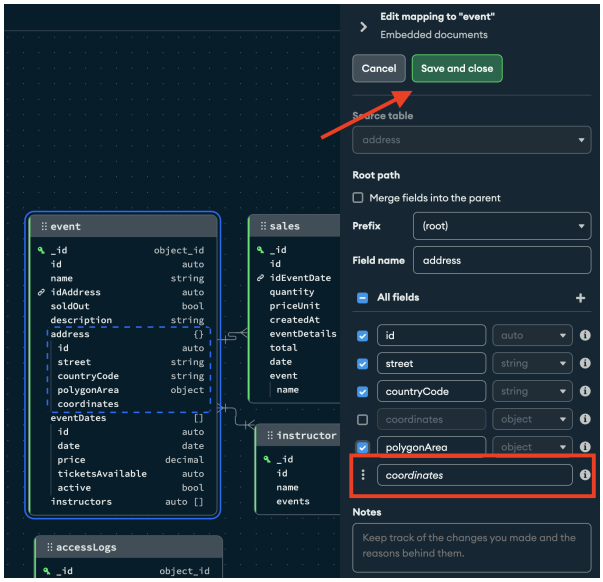
JavaScript

```
columns["coordinates"] !== null ? columns["coordinates"].x + ", " +
columns["coordinates"].y) : null
```


Mappings
MongoDB collection
Name
event
Mappings from relational tables
+ Add
event
New documents
address
Embedded documents
{} address
event_dates
Embedded array
[] eventDates
instructor_event
Embedded array
[] instructors
Suggested mappings 1
event
_id object_id
id auto
name string
idAddress auto
soldOut bool
description string
address {}
id auto
street string
countryCode string
coordinates object
polygonArea object
eventDates []
id auto
date date
price decimal
ticketsAvailable auto
active bool
instructors auto []

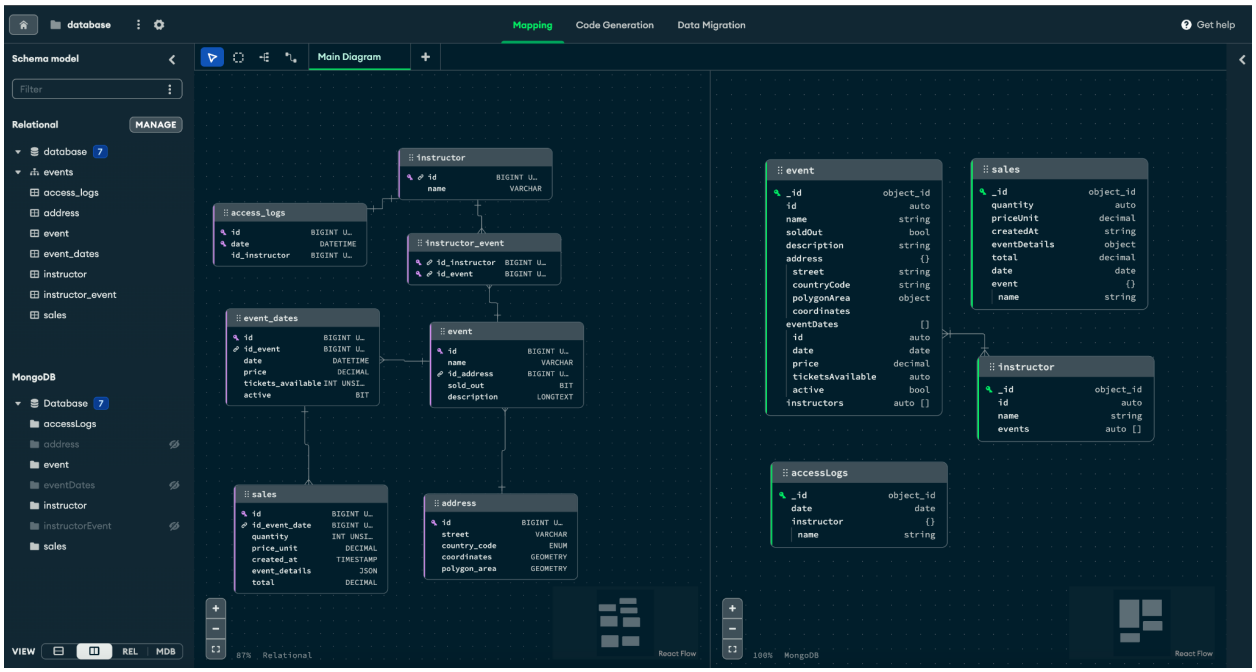

Edit mapping to "event"
Embedded documents
Cancel
Save and close
Source table
address
Root path
Merge fields into the parent
Prefix
(root)
Field name
address
All fields
id auto
street string
countryCode string
coordinates object
polygonArea object
Notes
Keep track of the changes you made and the reasons behind them.
event
sales
instructor
accessLogs


Edit mapping to "event"
Embedded documents
Cancel
Save and close
Unsaved changes
Source table
address
Root path
Merge fields into the parent
Prefix
(root)
Field name
address
Editing "coordinates"
Field name
coordinates
Value expression
Enter a Javascript expression and use columns["COLUMN_NAME"] to refer to original column values.
new Array(columns["coordinates"].x, columns["coordinates"].y)
Cancel
Done
event
accessLogs


Edit mapping to "event"
Embedded documents
Cancel
Save and close
Source table
address
Root path
Merge fields into the parent
Prefix
(root)
Field name
address
All fields
id auto
street string
countryCode string
coordinates object
polygonArea object
coordinates
Notes
Keep track of the changes you made and the reasons behind them.
event
sales
instructor
accessLogs

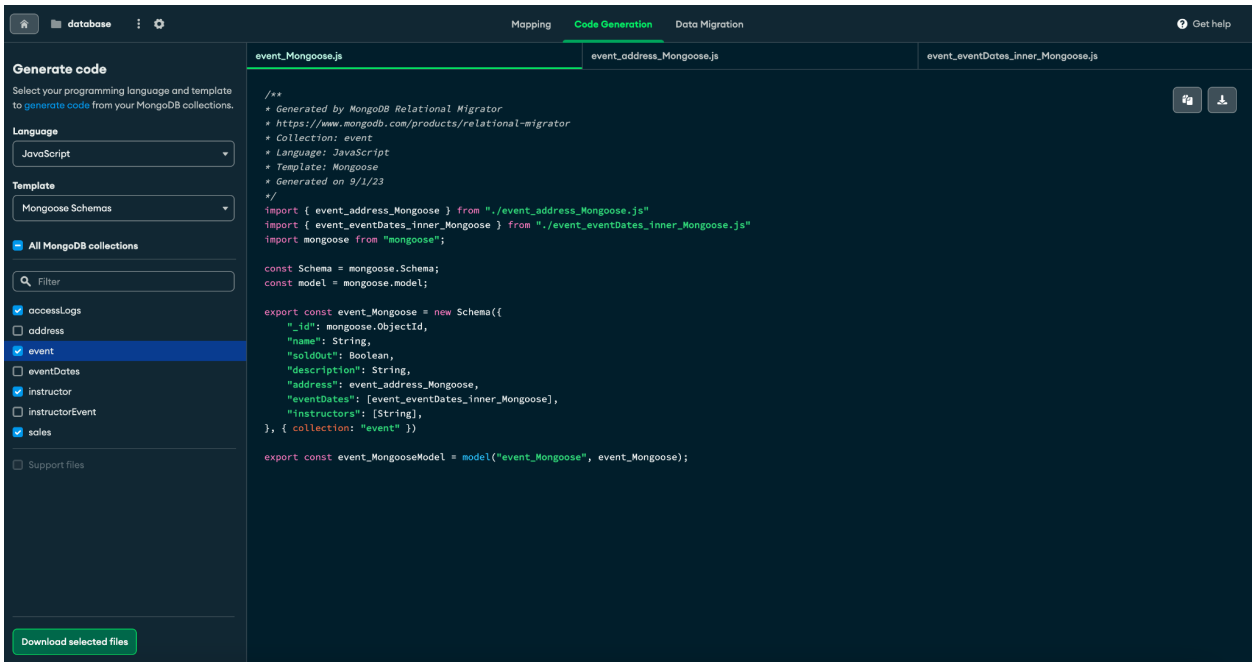
9. Delete / Hidden Collections

On the left-hand sidebar menu, you can access both the Relational Schema and MongoDB collections. To enhance clarity within the database collections, there's an option to selectively hide certain entities. It's important to note that this feature only affects visibility and doesn't exclude hidden collections from the migration process. If you intend to exclude an entity from being migrated entirely, you'll need to click on the respective collection and then select the "delete mapping" option.



10. Code Generation

Located within the top horizontal menu's second tab, you'll find the code generation page. This particular page serves the purpose of assisting you in generating the foundational segments of code required to establish MongoDB object mappings in languages such as C#, Java, and JavaScript. Additionally, it offers the capability to create a JSON schema if needed.



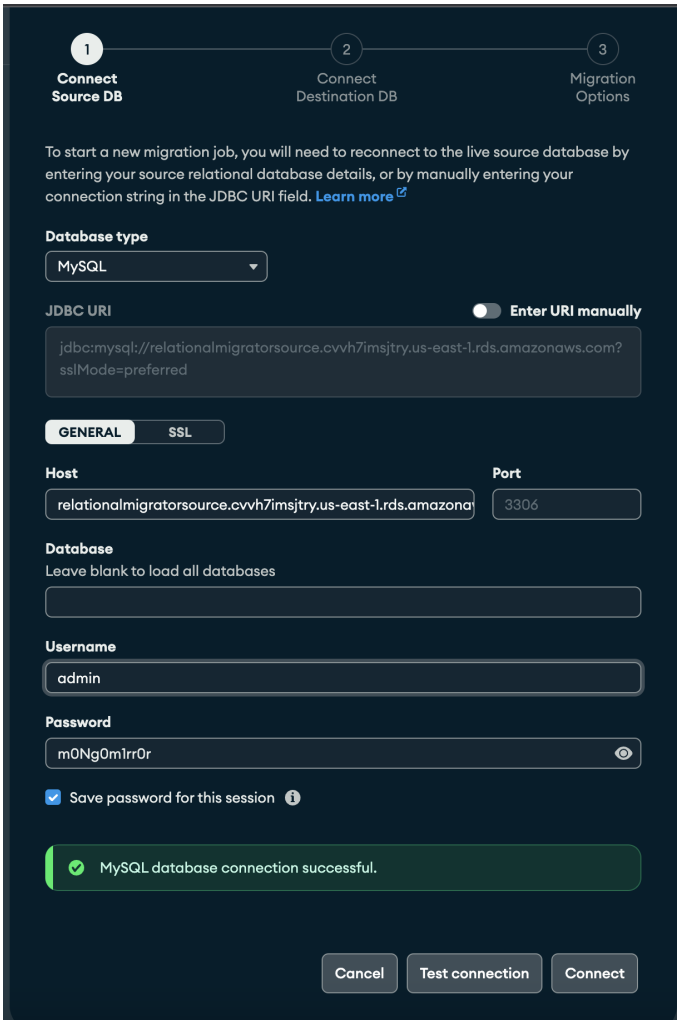
11. Data Migration

The functionality for creating data migration jobs can be found on the third tab of the top horizontal menu. This page allows you to view a list of previous migration jobs in the History section on the left side, as well as create new synchronization jobs.

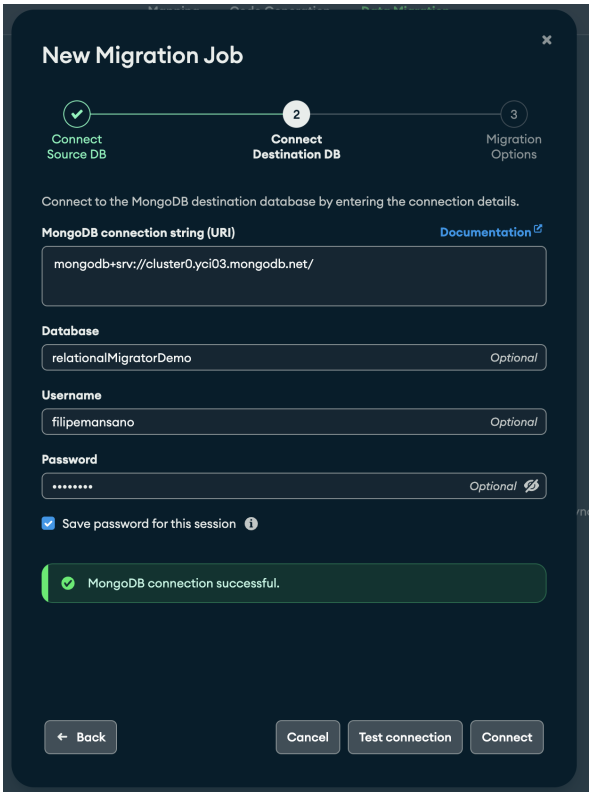
11.1 Create sync job

After clicking on the Create sync job, we need complete 3 steps to initialize the sync

1. Connect Source DB
    a. In this step we need put the same credentials used in the step 3.1



2. Connect Destination DB
    a. In this step we need provide the credential of our MongoDB Cluster

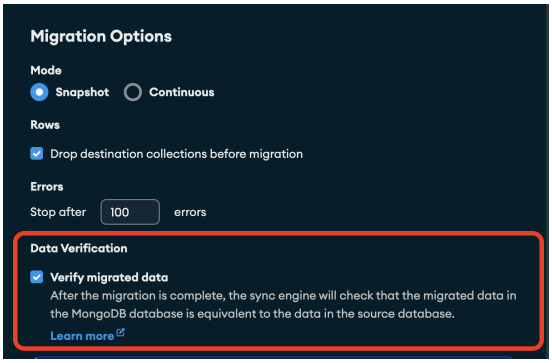


3. Migration Options: on this step we have 2 different ways to migrate our data
    a. Snapshot: Migrate all existing data to MongoDB
    b. Continuous: Migrate all existing data to MongoDB and keep the Relational Migration active to watch any changes from the RDMS to replicate to MongoDB

11.2 Snapshot Stage

During the snapshot stage, the Relational Migrator's primary function is to meticulously replicate all available data. Following this step, the migration process is automatically finalized. For enhanced precision, there's also an option to activate the Data Verification stage. This critical

phase involves a comprehensive validation of the migrated data, ensuring its integrity before proceeding to the final step of transferring all data to MongoDB.

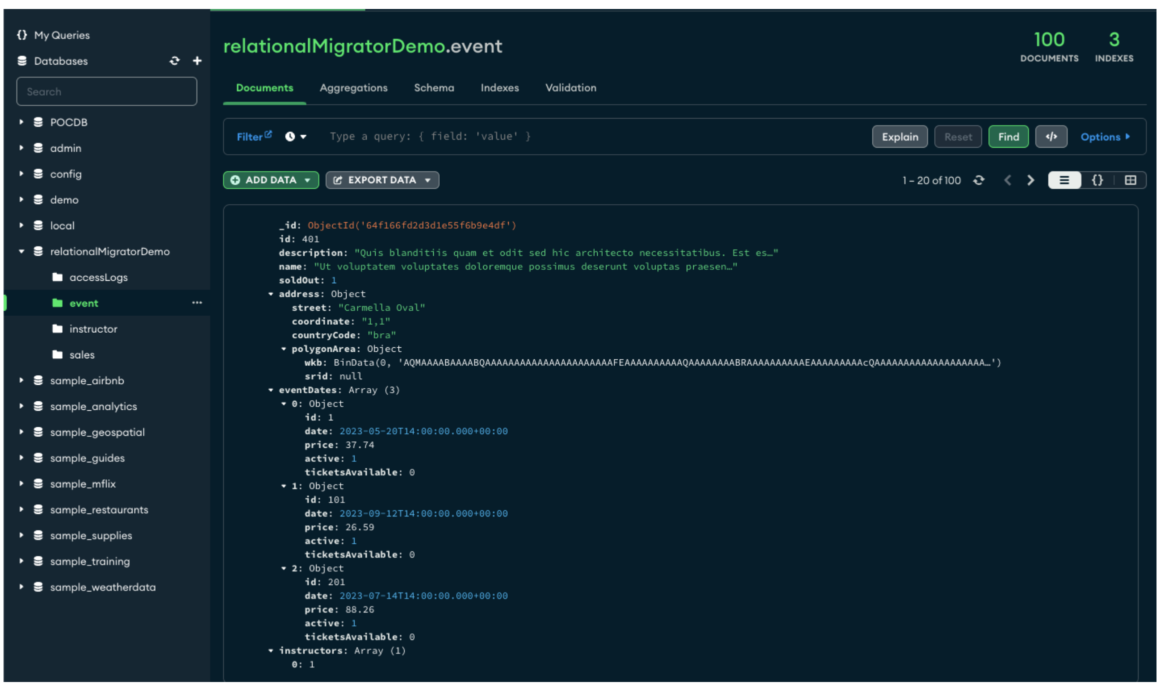


11.3 CDC Stage (optionally)

Once the snapshot stage is completed, the CDC (Change Data Capture) stage will commence. This stage involves continuous monitoring of binary log files from the RDMS (Relational Database Management System), ensuring that any data modifications are replicated to MongoDB. This process will persist until you choose to cease by clicking the "Complete CDC" button located at the upper right corner.


Job overview
Create sync job
Source URI: jdbc:mysql://relationalmigratorsource.cvvh7imsjtry.us-east-1.rds.amazonaws.com?sslMode=preferred
Destination URI: mongodb+srv://filipemansano:*****@cluster0.yci03.mongodb.net/relationalMigratorDemo
Job Mode: Continuous
CDC in progress
Complete CDC
Snapshot stage COMPLETED
Started: Today at 1:10 AM
Duration: 12 sec
Tables migrated: 7 of 7
Rows migrated: 711
CDC stage RUNNING
Started: Today at 1:11 AM
Last event time: Today at 1:13 AM
Total events seen: 0
Events in the last hour: 0
Issues 0

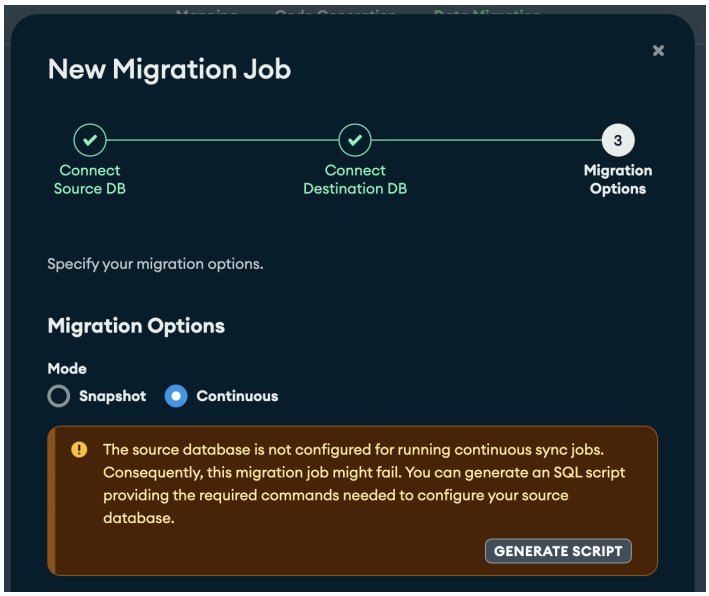
11.4 Result


My Queries
Databases
Search
POCDB
admin
config
demo
local
relationalMigratorDemo
accessLogs
event
instructor
sales
sample_airbnb
sample_analytics
sample_geospatial
sample_guides
sample_mflix
sample_restaurants
sample_supplies
sample_training
sample_weatherdata
relationalMigratorDemo.event
100 DOCUMENTS
3 INDEXES
Documents
Aggregations
Schema
Indexes
Validation
Filter
Type a query: { field: 'value' }
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
1 – 20 of 100
_id: ObjectId('64f166fd2d3d1e55f6b9e4df')
id: 401
description: "Quis blanditiis quam et odit sed hic architecto necessitatibus. Est es…"
name: "Ut voluptatem voluptates doloremque possimus deserunt voluptas praesen…"
soldOut: 1
address: Object
street: "Carmella Oval"
coordinate: "1,1"
countryCode: "bra"
polygonArea: Object
wkb: BinData(0, 'AQMAAAABAAAABQAAAAAAAAAAAAAAAAAAAAAAFEAAAAAAAAAAQAAAAAAAABRAAAAAAAAAAEAAAAAAAAAcQAAAAAAAAAAAAAAAAAA…')
srid: null
eventDates: Array (3)
0: Object
id: 1
date: 2023-05-20T14:00:00.000+00:00
price: 37.74
active: 1
ticketsAvailable: 0
1: Object
id: 101
date: 2023-09-12T14:00:00.000+00:00
price: 26.59
active: 1
ticketsAvailable: 0
2: Object
id: 201
date: 2023-07-14T14:00:00.000+00:00
price: 88.26
active: 1
ticketsAvailable: 0
instructors: Array (1)
0: 1

11.5 Troubleshooting

Once you have established the database connection and specified the job type on the Migration Options form, Relational Migrator conducts various checks to ensure that the database is configured correctly. If any configuration is missing, a banner appears indicating that the database is not properly configured, with a GENERATE SCRIPT button to download a SQL script.

This script includes the required configuration statements and additional instructions in the form of comments.



After running the script generated by RelationalMigrator, navigate back and return to Migration Options to run the verification process again. If the error persists, it is advisable to troubleshoot the problem by looking at some common issues. For example, make sure that replication is enabled on the Relational Database Management System (RDMS) for continuous mode (this action is not needed for Snapshot mode).

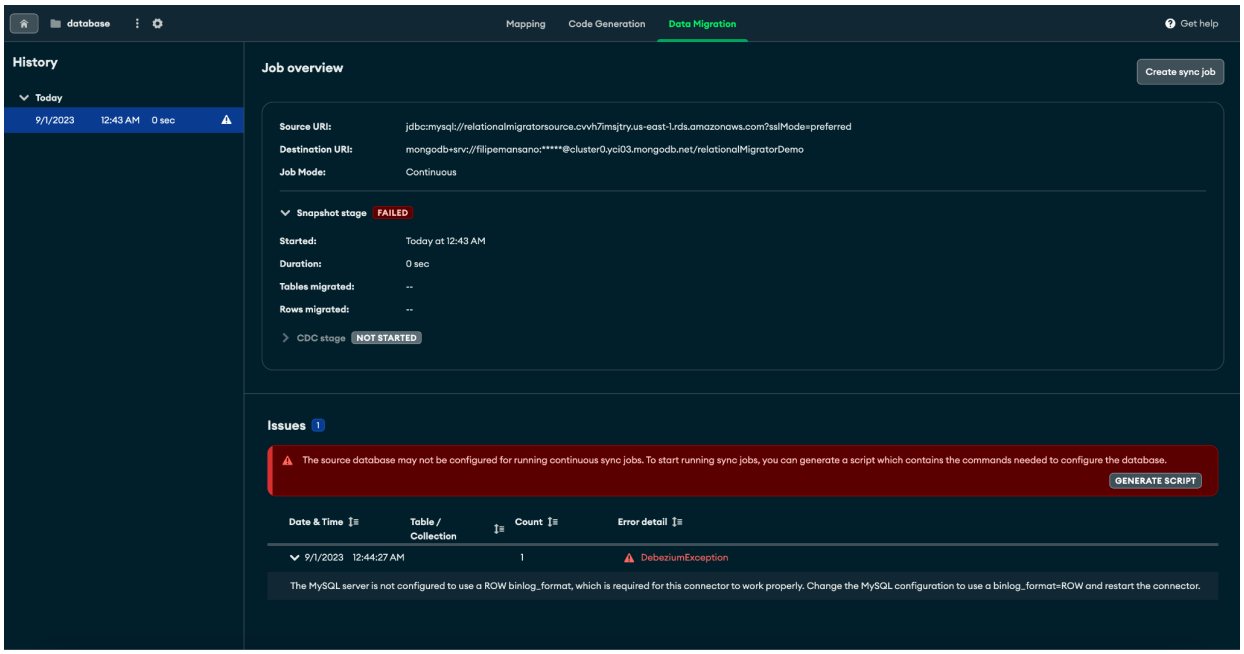
Unset

```
// MySQL
SELECT variable_value as "BINARY LOGGING STATUS (log-bin) ::"
FROM performance_schema.global_variables
WHERE variable_name='log_bin';
```

For detailed information regarding the configuration requirements for each database, see the follow Debezium reference links, which are internally utilized by Relational Migrator:

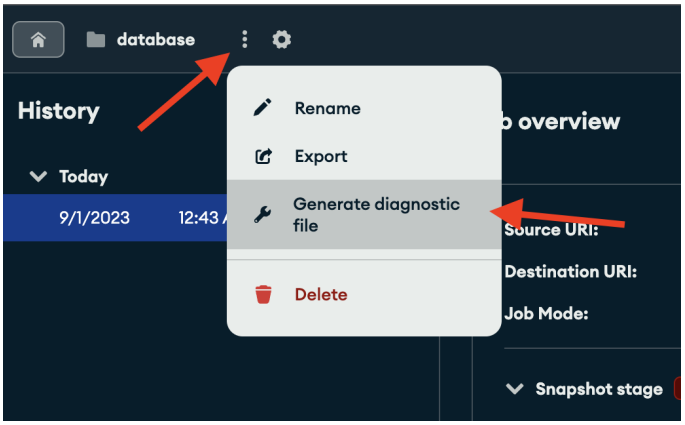
- MySQL
- Oracle
- PostgreSQL
- SQL Server

If any error occurred after start the sync process the RelationalMigrator will show you the reason in the issues section



You can download the diagnostic file as well to see with details all errors occurred in any stage, to download the file follow the steps below

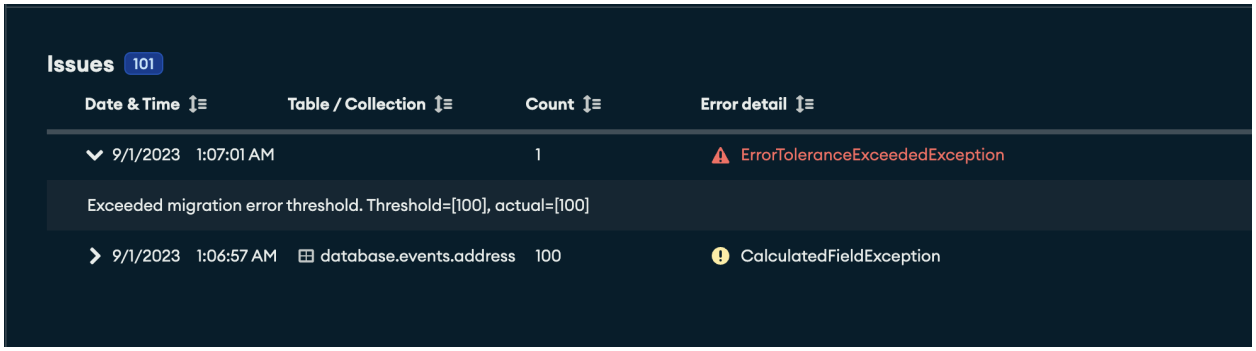
1. Click on the 3 dots icon aside from the project name in the top menu
2. Click on “Generate diagnostic file”



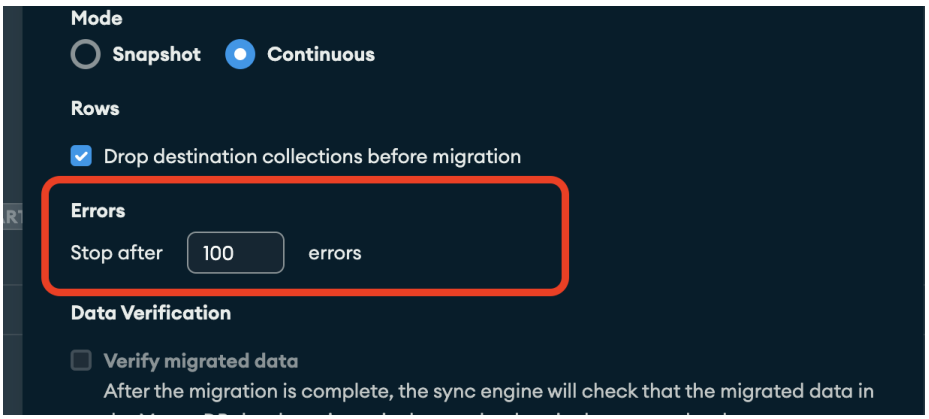
In this file we can see the stack trace of the error that happen above

```
2023-09-01 00:44:27,646 WARN
com.mongodb.migrator.application.migration.job.MigrationJobService
[pool-304-thread-1] Migration job finished with exception
io.debezium.DebeziumException: The MySQL server is not configured to
use a ROW binlog_format, which is required for this connector to work
properly. Change the MySQL configuration to use a binlog_format=ROW
and restart the connector.
  . . .
at java.base/java.lang.Thread.run(Unknown Source)
```

In addition, if any data errors arise during the migration process and these errors surpass the predefined error rate threshold, the relational migration will not succeed, as depicted in the image below.



You can configure the error rate limit on the Migration Options stage



12. Limitations

12.1 Only data is migrated

Only the raw data undergoes migration. Auxiliary elements such as triggers, views, indexes, stored procedures, and events are not transferred automatically. It's essential to undertake a manual conversion process to adapt and migrate these aspects to MongoDB.

12.2 Nested relationship

When we have some nested relationships like sales with events we need to do something like that in the relational database

```
SELECT events.name FROM sales INNER JOIN event_dates ON event_dates.id = sales.id_event_date INNER JOIN events ON events.id = event_dates.id_event
```

Isn't possible access this nested relationship in relation migrator, but in some cases we can do some workaround to figure out this problem, as mentioned in the section 5.2

12.3 Calculated field is very limited

We can't write complex expressions, or create an object to represent a subdocument in the field, for example we can't convert correctly the MySQL coordinates to a simple array because we can't create an array in the calculated field