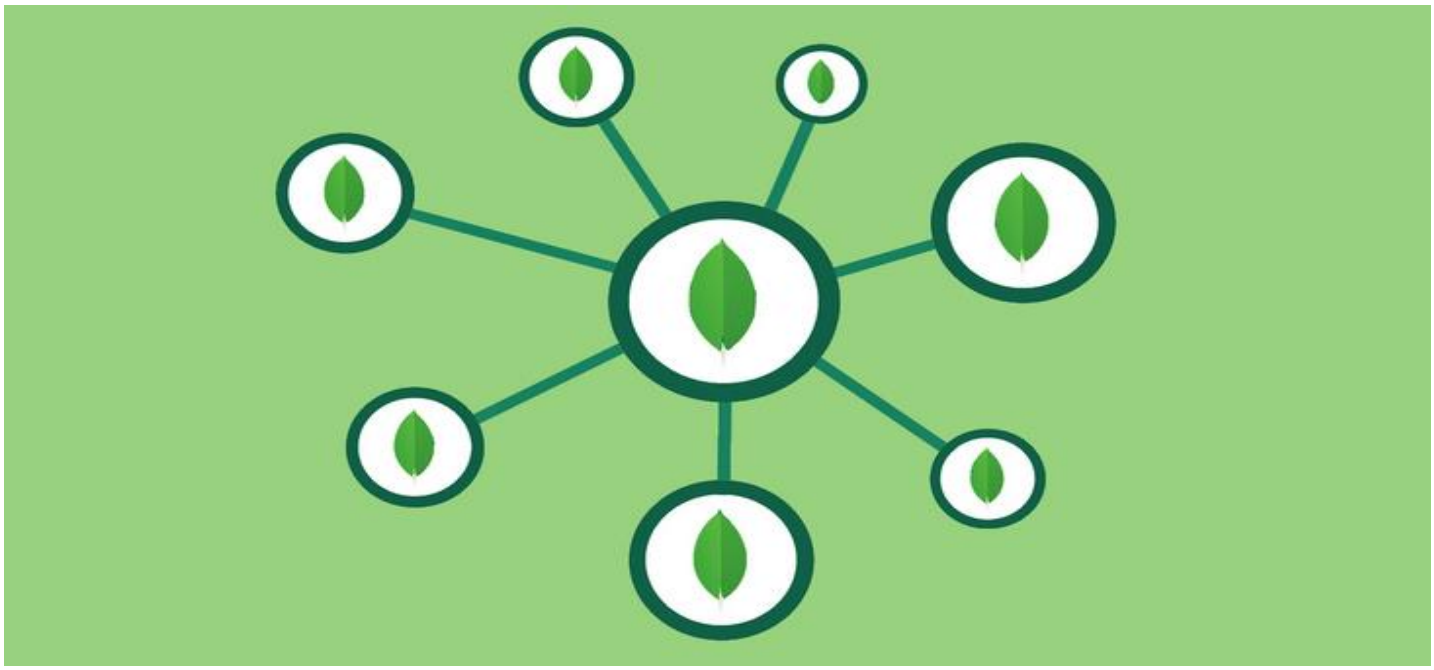
How to Create an ER Diagram for MongoDB

 Subrahmanyeswarao Karri

13th March 2025



I love flexibility and freedom that JSON and document databases such as MongoDB bring. However, when I'm trying to figure out schema of the data by looking at JSON documents it can be really painful.

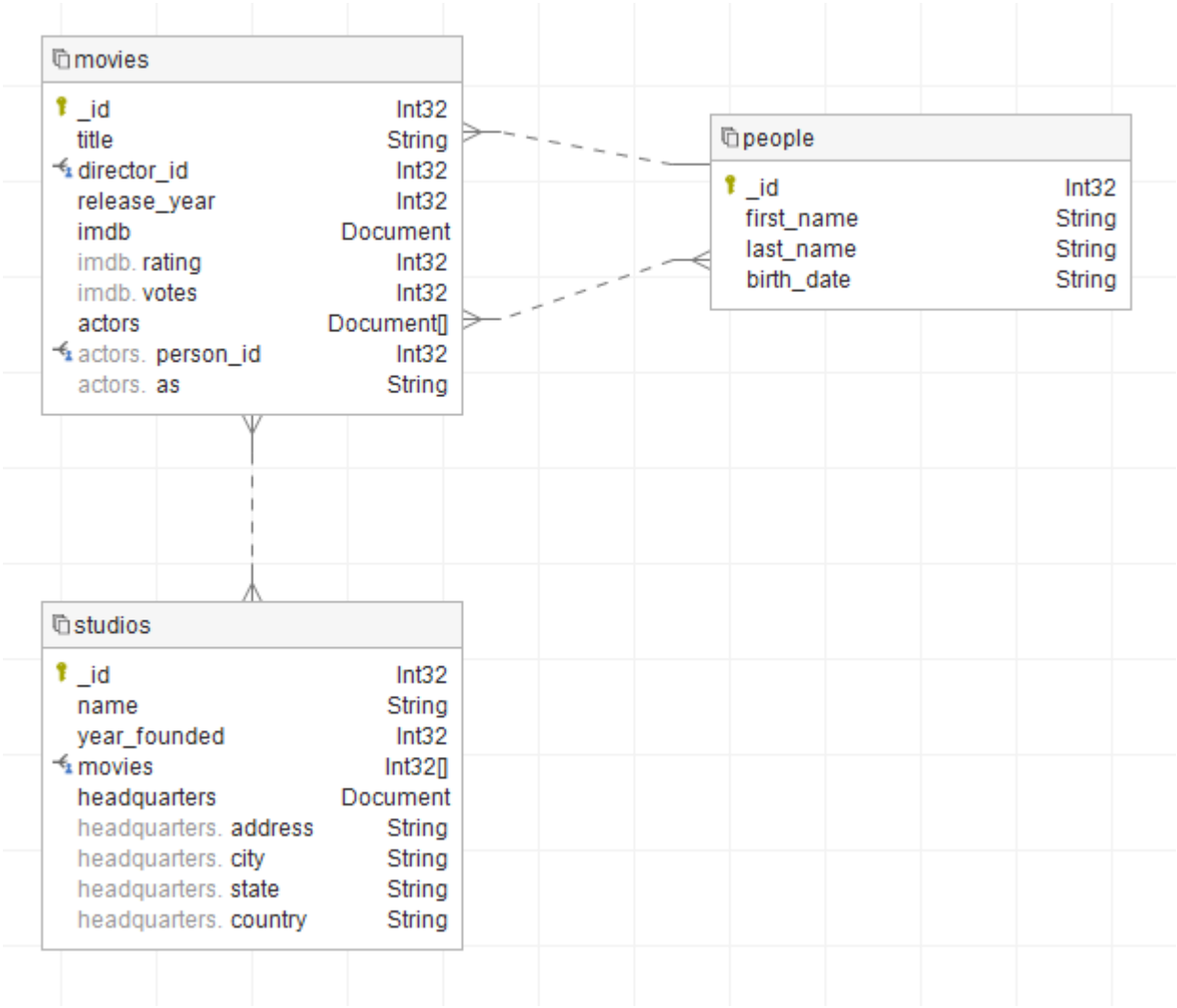
```
// movies
{
    "_id": 156,
    "title": "The Terminator",
    "plot": "A cyborg killing machine is sent from 2029 to 1984 to execute
    "director_id": 217,
    "release_year": 1984,
    "imdb": {
        "rating": 8.0,
        "votes": 782237,
        "url": "https://www.imdb.com/title/tt0088247/",
        "last_updated": null
    },
    "actors" [
        {
            "person_id": 180,
            "as": "Terminator",
            "salary": 15000000
        },
        {
            "person_id": 13122,
            "as": "Sarah Connor"
        },
        {
            "person_id": 13122,
            "as": "Kyle Reese"
        }
    ]
}

// studios
{
    "_id": 1,
    "name": "Warner Bros.",
    "year_founded": 1923,
    "movies": [156, 2234, 334, 2109],
    "headquarters": {
        "address": "4000 Warner Blvd.",
        "city": "Burbank",
        "state": "California",
        "country": "United States"
    }
}

// people

{
    "_id": 1,
    "first_name": "Arnold",
    "last_name": "Schwarzenegger",
    "birth_date": "1947-07-30"
}
```

Wouldn’t a diagram like below be much better? Or at least a great addition?



Well, what if I told you there is a tool that you can build this results in a few minutes? Well, there is – **Dataedo**.

What you get

This tutorial will teach you how you can:

1. Discover schema of MongoDB documents,
2. Document logical references,
3. Build a diagram,
4. Share a diagram as image

5. Share entire database documentation in interactive HTML or PDF documents

You will also learn about:

1. Different relationship types in MongoDB - embedded documents and references

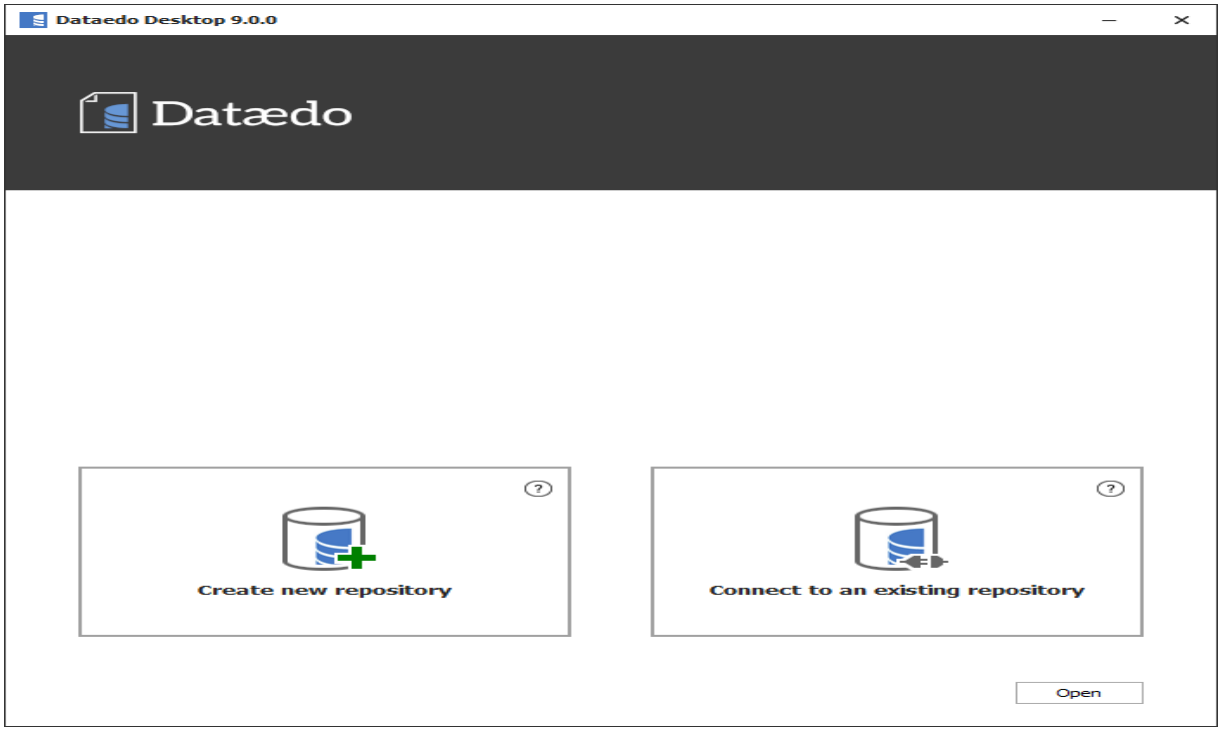
Prepare the tool

1. Install

First, you need to download and install Dataedo Desktop on your computer.

2. Create repository/file

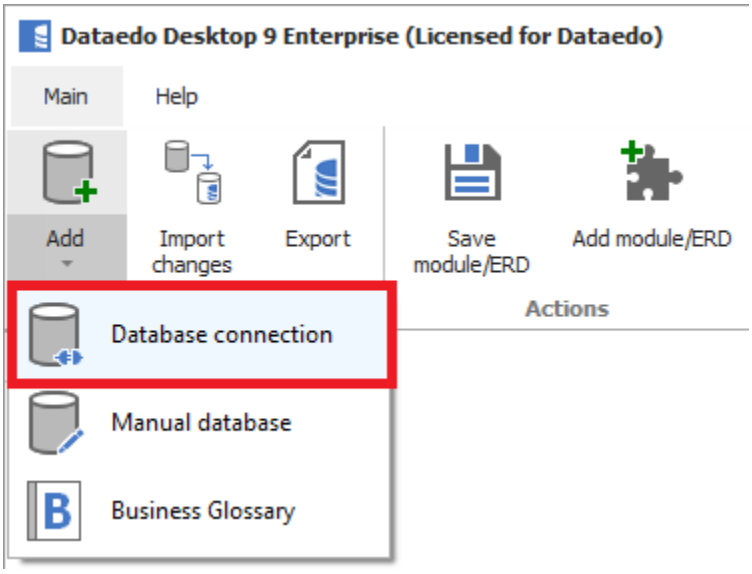
Next step is to create a repository. Repository is a file or database that will hold all the metadata. Database is regular SQL Server or Azure SQL database. It is more advanced option for multiuser environments so if you can't get your hands on an instance go with the file option. File is just a document you can save anywhere.



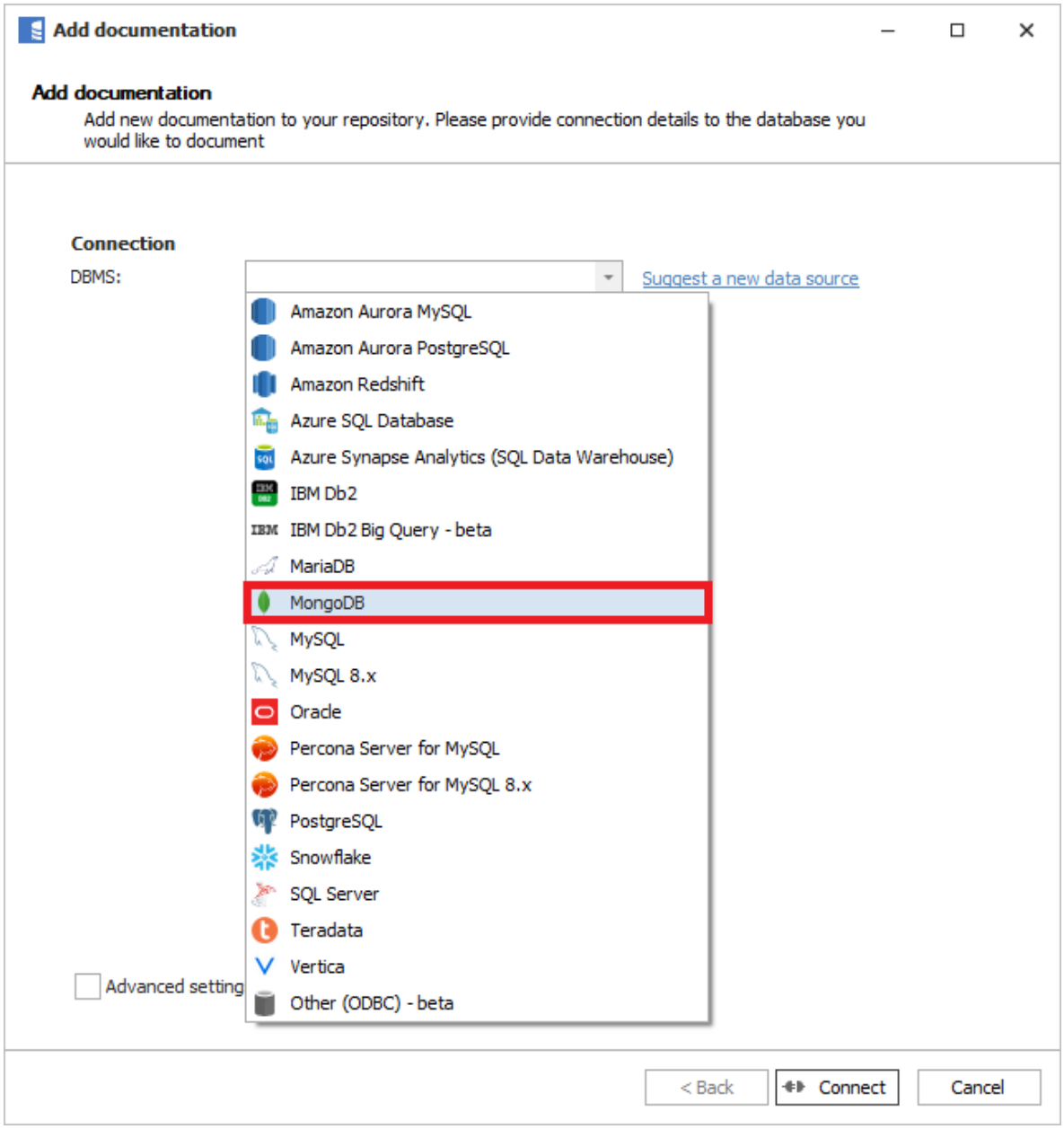
Now your set up is complete.

Connect to MongoDB and import Collections

Now that you have installed and configured Dataedo you can connect to your instance of MongoDB. To connect to database click Add in the ribbon and choose Database connection option.

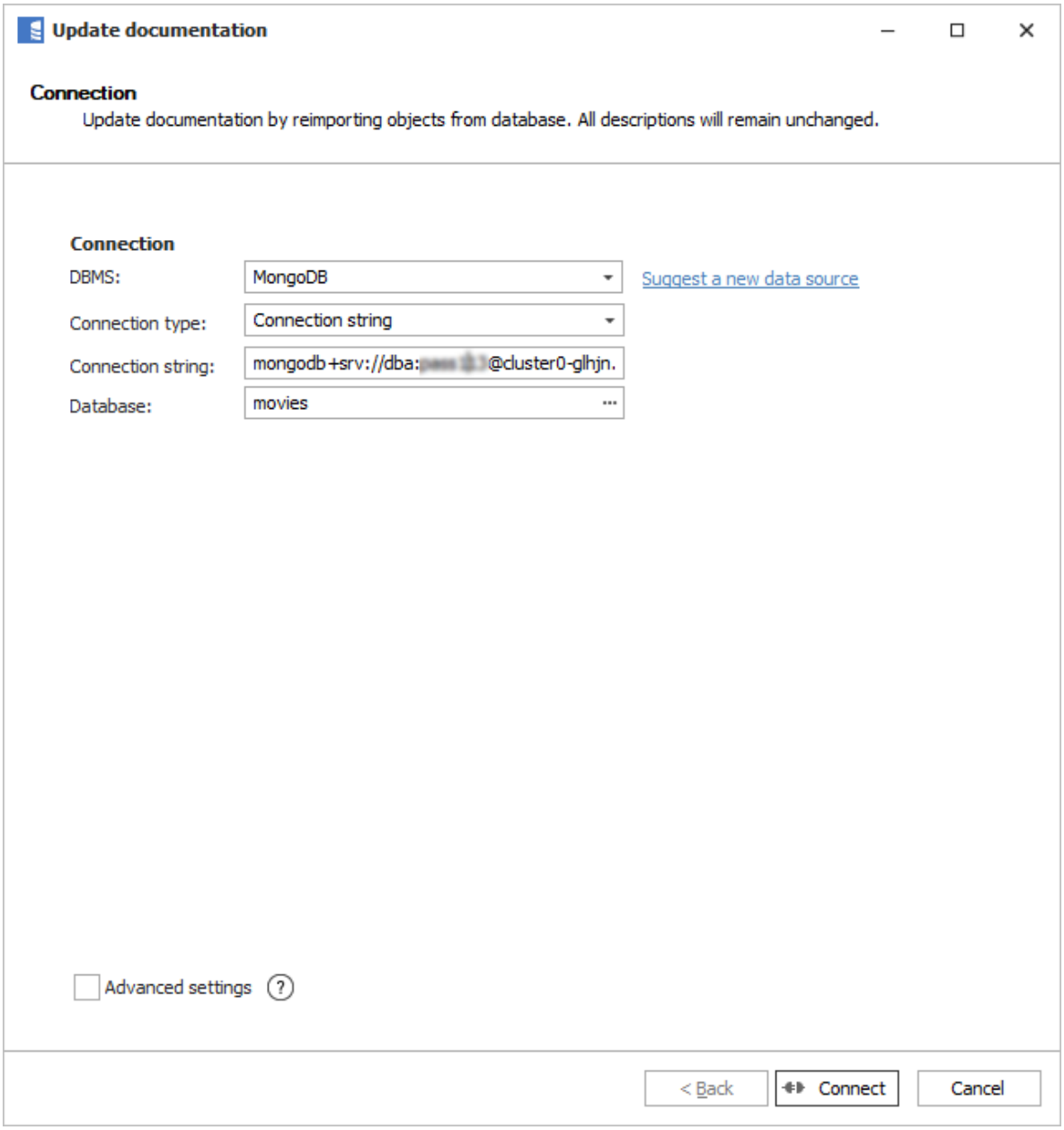


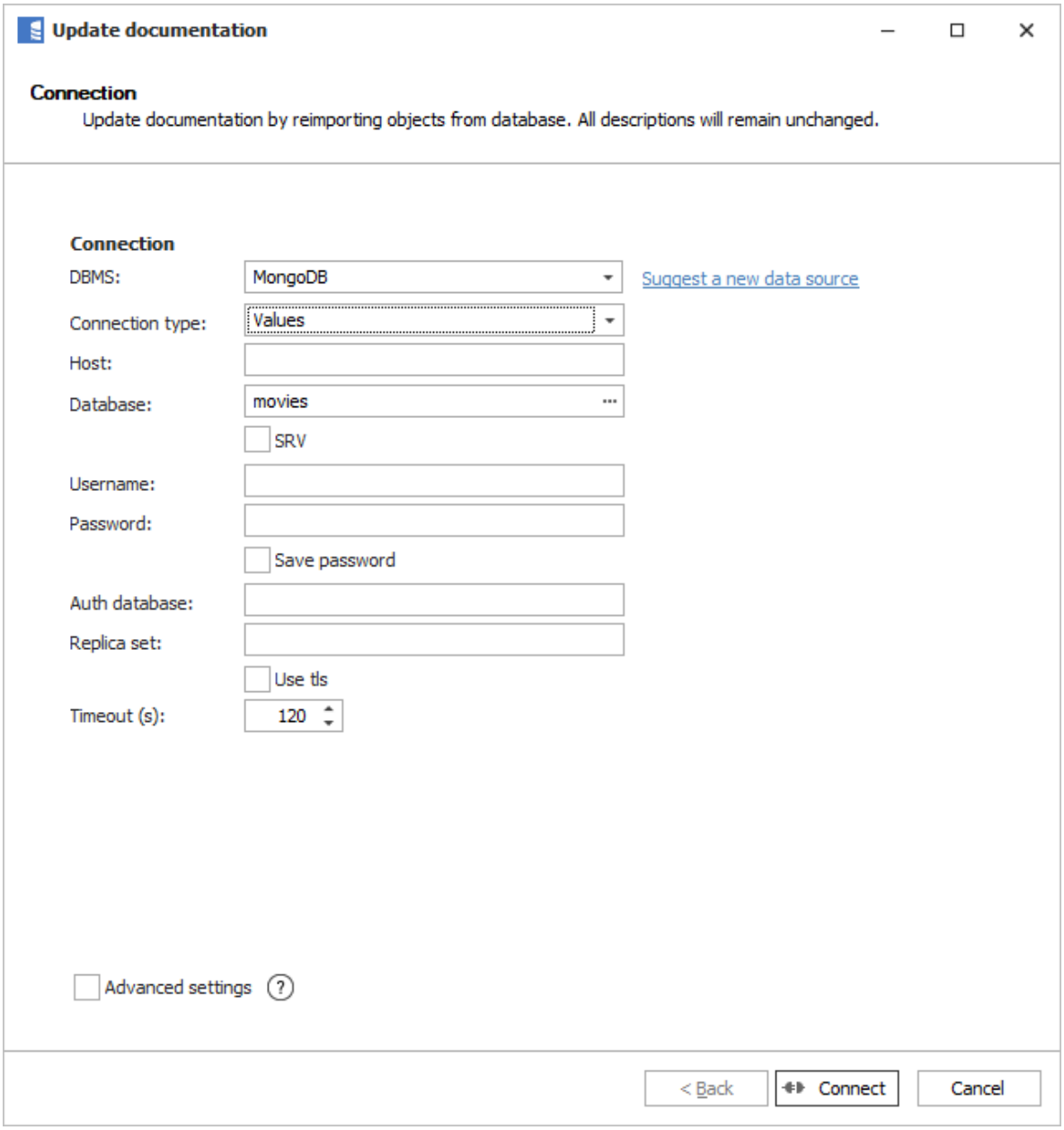
Now choose MongoDB in the DBMS field:



And connection type - Values or Connection string.

Subrahmanyeswarao Karri

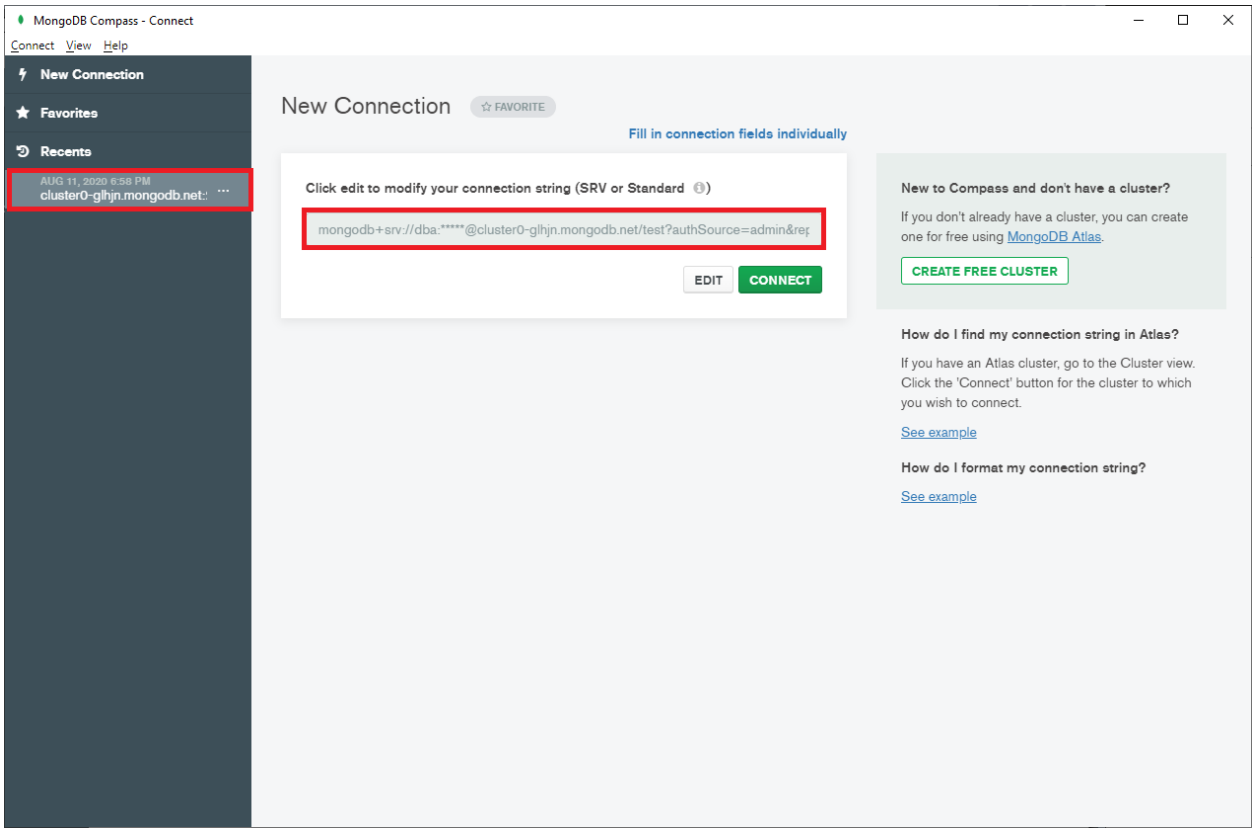




How to find connection string?

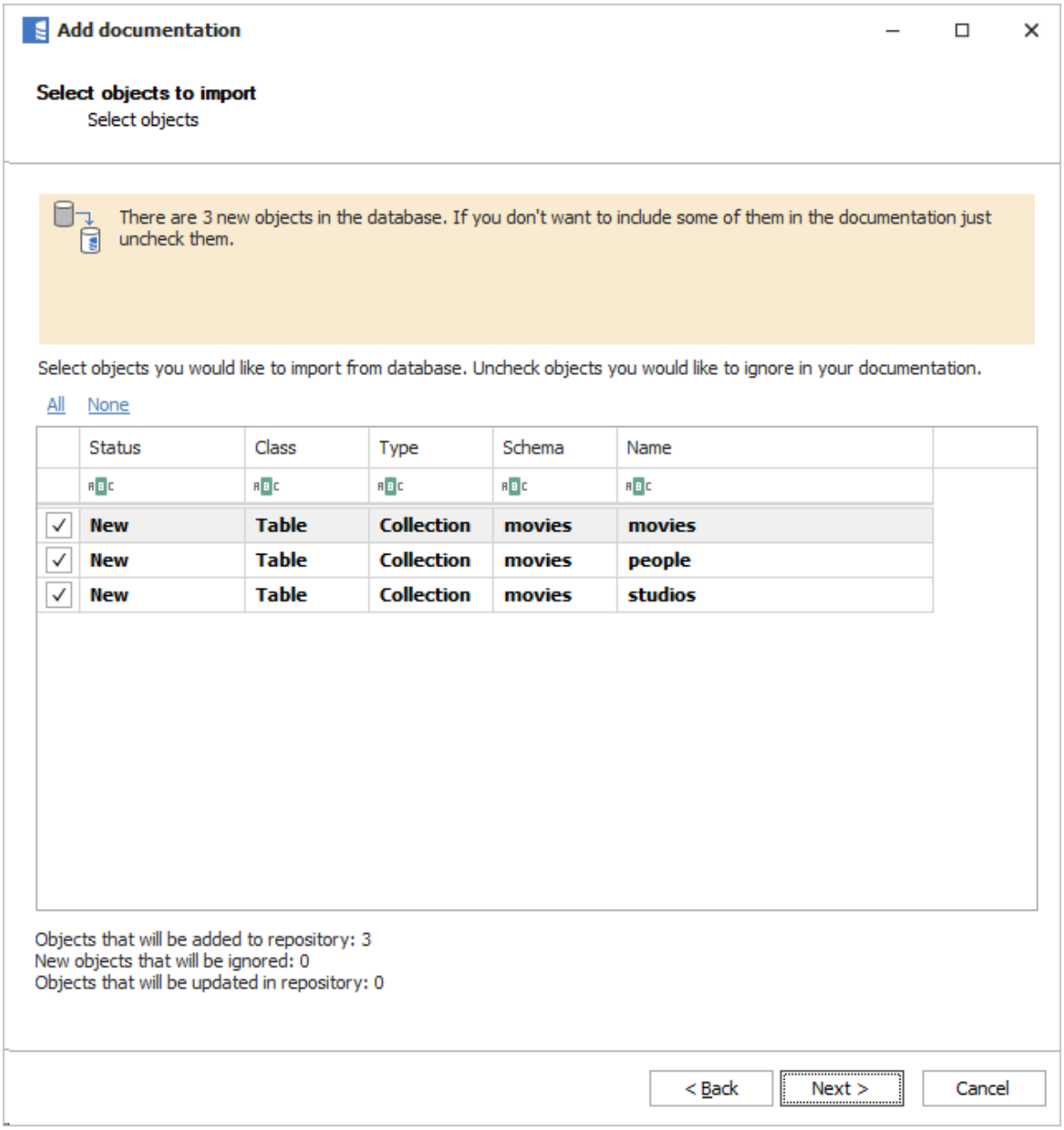
If you don't have it you can ask your admin, developers or anyone who might know.

If you used **MongoDB Compass** to connect to your MongoDB instance you will find it in **Recents** section. Click it, it will get copied into connection field. Now you can click edit to enable field. Copy it and paste into Dataedo.



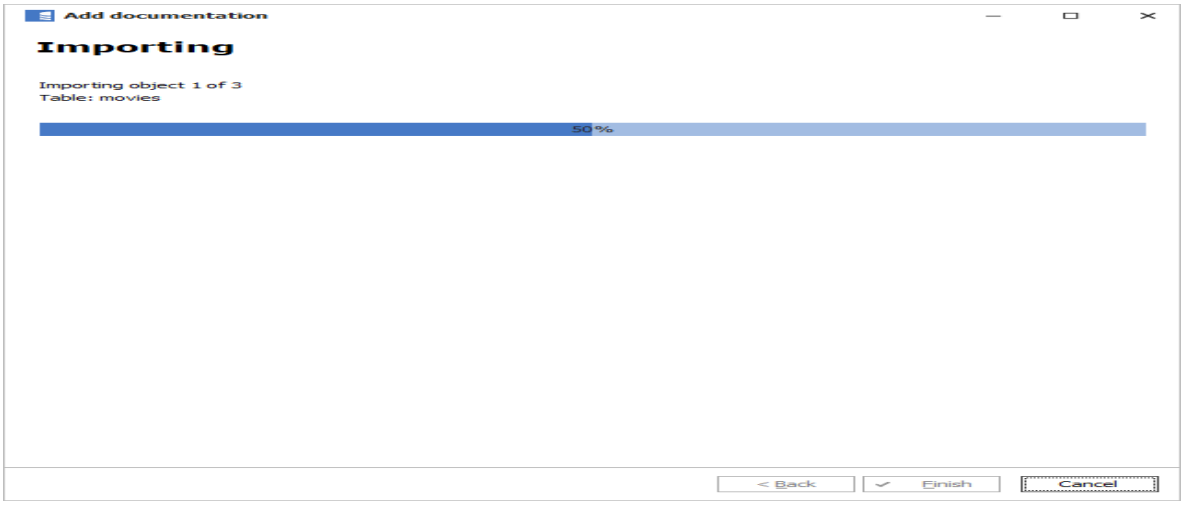
Connect

When you provide connection details and click **Connect**. Dataedo will connect to your MongoDB database and list collections. You can choose collections to import from this list, but you just want to skip this step with Next to import entire schema.



Schema Discovery

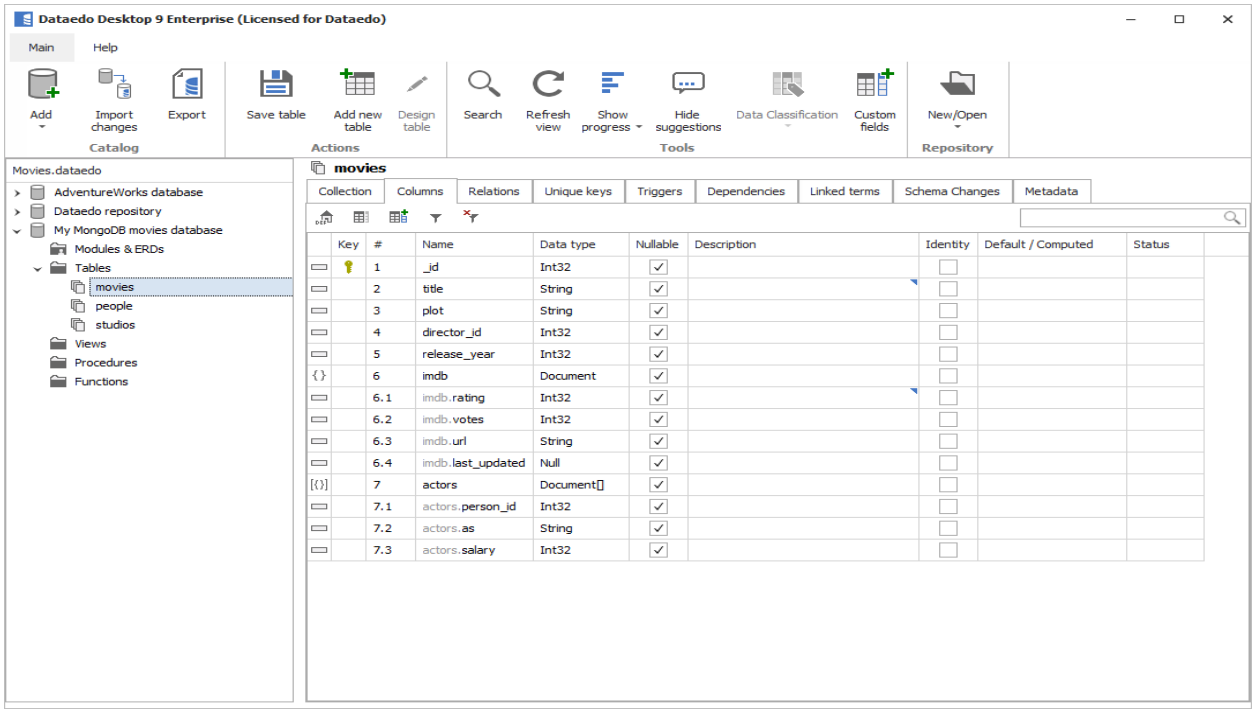
When you confirm, Dataedo will perform **automatic schema discovery**.



What happens here is that Dataedo:

1. **samples documents** in collections, parses the JSON documents, and
2. reads **Schema Validation** rules

and builds data dictionary from that information. When it finishes it creates complete data dictionary for your MongoDB database – list of collections and their attributes organized into hierarchy (documents, fields, arrays, etc.).



Discovering and Documenting Relationships

To create an ER diagram, you need entities (collections) and relationships. Dataedo discovered entities and their fields. It is a bit more complicated (as always) with the relationships. MongoDB is not a relational database, it is a document store, so traditional ER modeling does not apply. However, we can stretch the concept to fit JSON documents.

Let's have an overview of relationships in this kind of databases.

Relationships in MongoDB

MongoDB, or any document store, has in general two categories of relationships – **embedded documents** and **references**.

1. **Embedded documents** are nested in the data and can be discovered and visualized automatically.

2. **References** are logical, and as such, cannot be derived from data and needs to be documented manually in Dataedo. And this is where Dataedo shows its value – you end up with **additional information about data** (metadata) that cannot be easily obtained by people working with data.

Discovering Embedded Documents Relationships

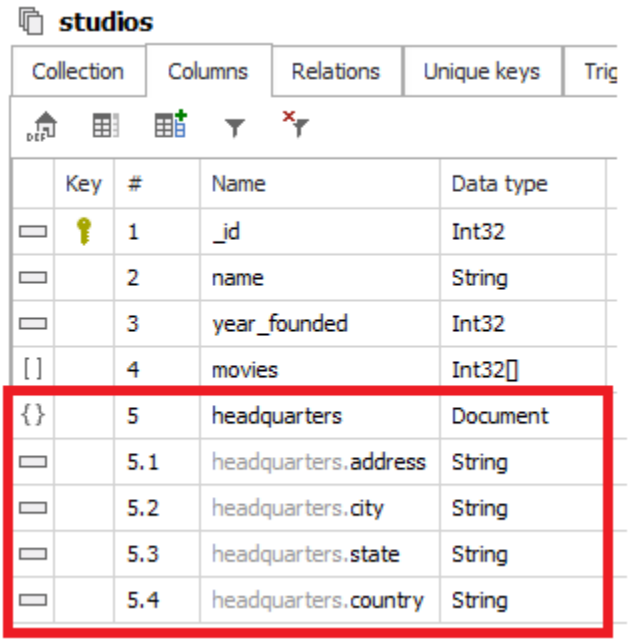
Embedded documents are specific to semi-structured data – ability to embed another record (document), or array of rows, into another record. It is defined directly in data and you can view those relationships in Dataedo right after schema import.

Embedded Document (One-to-One)

Basic embedded document is a one-to-one relationship. One parent record is related to one child record. In case below, (Hollywood) Studio has embedded one headquarters record.

```
// studios
{
    "_id": 1,
    "name": "Warner Bros.",
    "year_founded": 1923,
    "movies": [156, 2234, 334, 2109],
    "headquarters": {
        "address": "4000 Warner Blvd.",
        "city": "Burbank",
        "state": "California",
        "country": "United States"
    }
}
```

Dataedo shows this relationship as a hierarchy of fields in collection entity. Parent object has a **Document** type.

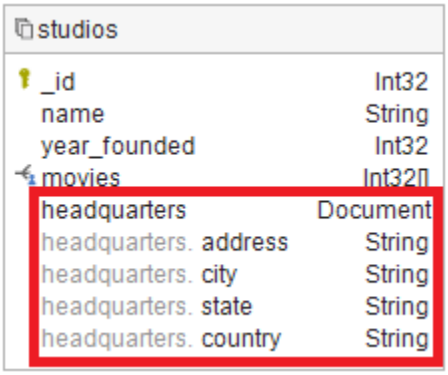


studios

Collection | Columns | Relations | Unique keys | Trig

	Key	#	Name	Data type
		1	_id	Int32
		2	name	String
		3	year_founded	Int32
[ ]		4	movies	Int32[]
{ }		5	headquarters	Document
		5.1	headquarters.address	String
		5.2	headquarters.city	String
		5.3	headquarters.state	String
		5.4	headquarters.country	String

On the diagram it is represented as a hierarchy in the entity.



studios

_id	Int32
name	String
year_founded	Int32
movies	Int32[]
headquarters	Document
headquarters. address	String
headquarters. city	String
headquarters. state	String
headquarters. country	String

Embedded Array of Documents (One-to-Many)

More complex design is the implementation of one-to-many relationship as embedded array of documents. In this case one parent record is related to multiple child record. Example below shows one Movie record having a list of actor records.

```
// movies
{
    "_id": 156,
    "title": "The Terminator",
    "plot": "A cyborg killing machine is sent from 2029 to 1984 to execute
    "director_id": 217,
    "release_year": 1984,
    "imdb": {
        "rating": 8.0,
        "votes": 782237,
        "url": "https://www.imdb.com/title/tt0088247/",
        "last_updated": null
    },
    "actors": [
        {
            "person_id": 180,
            "as": "Terminator",
            "salary": 1000000
        },
        {
            "person_id": 13122,
            "as": "Sarah Connor"
        },
        {
            "person_id": 31542,
            "as": "Kyle Reese"
        }
    ]
}
```

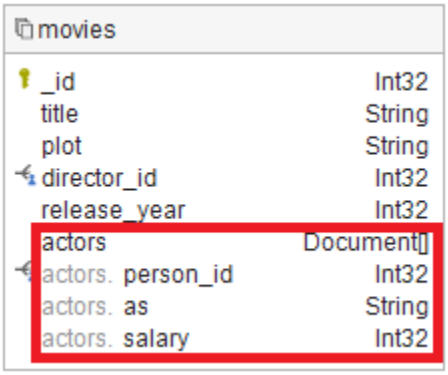
As in the case of embedded document Dataedo shows such relationship as hierarchy of fields, except in this case type of parent field is **Document[]** (array of documents) instead of Document.

movies

Collection | Columns | Relations | Unique keys

	Key	#	Name	Data type
	🔑	1	_id	Int32
		2	title	String
		3	plot	String
		4	director_id	Int32
		5	release_year	Int32
{}		6	imdb	Document
		6.1	imdb.rating	Int32
		6.2	imdb.votes	Int32
		6.3	imdb.url	String
		6.4	imdb.last_updated	Null
[{}]		7	actors	Document[]
		7.1	actors.person_id	Int32
		7.2	actors.as	String
		7.3	actors.salary	Int32

On the diagram, just as in the case of embedded document, it is represented as a hierarchy within an entity.



Documenting Reference Relationships

References are the same concept as in the case of relational databases – a data normalization technique where row in one set references row in another (or the same).

Document References (Many-to-One) - AKA Foreign Keys

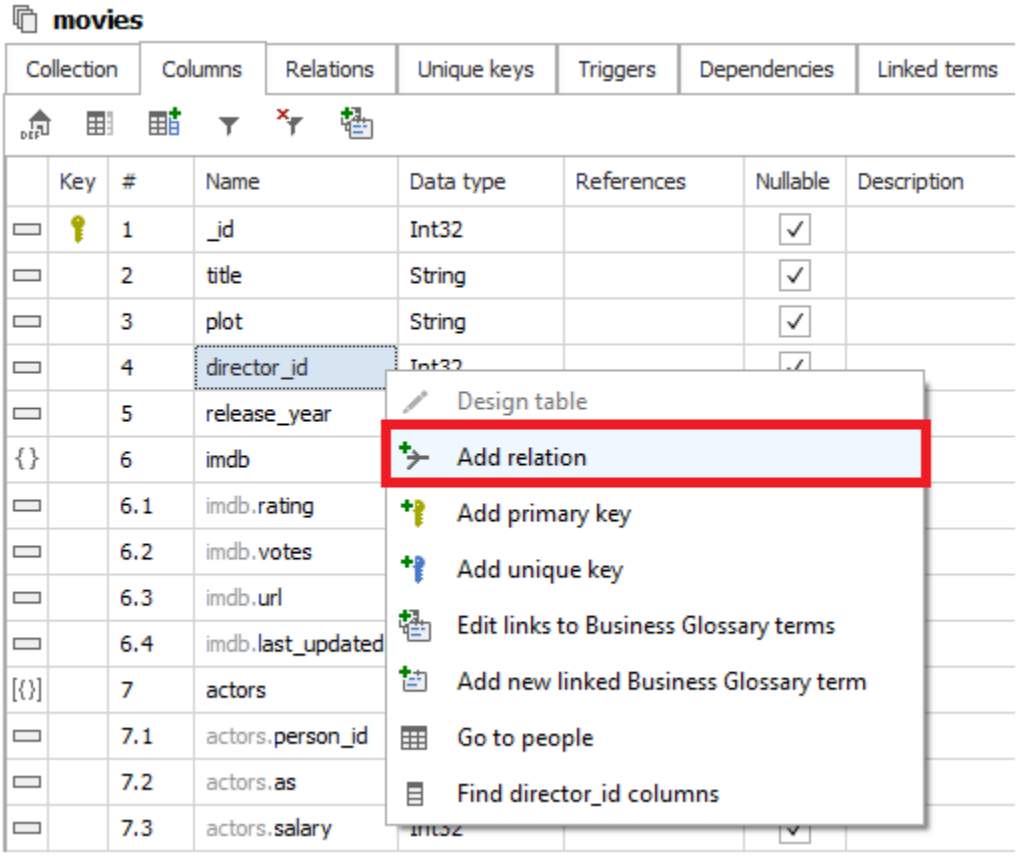
Most typical reference in MongoDB document works exactly like foreign keys in relational databases – field in one record (1) references record in another record (2). This called many-to-one relationship because field in record 1 (director in movies collection) can reference exactly one record (person), while record 2 (person) can be referenced by unlimited number of records (movies).

```
// movies
{
    "_id": 156,
    "title": "The Terminator",
    "plot": "A cyborg killing machine is sent from 2029 to
    "director_id": 217,
    "release_year": 1984,
    "imdb": {
        "rating": 8.0,
        "votes": 782237,
        "url": "https://www.imdb.com/title/tt0088247/",
        "last_updated": null
    }
}
```

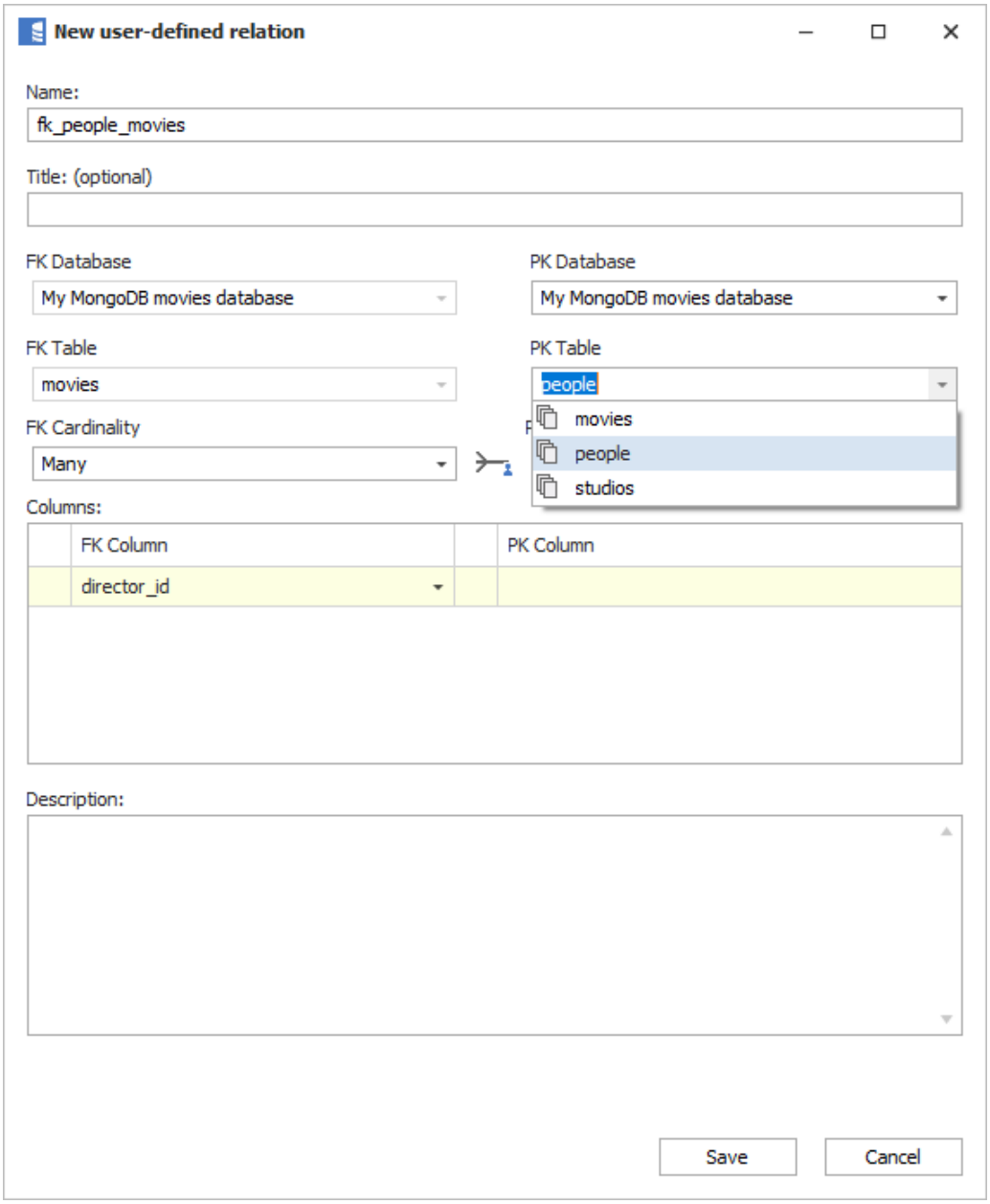
```
// people
{
    "_id": 217,
    "first_name": "James",
    "last_name": "Cameron",
    "birth_date": "1954-08-16"
}
```

This reference is logical only, i.e. it is not defined with the data or collections structure. You need to know how data elements are related to each other, and then add this information into Dataedo metadata repository.

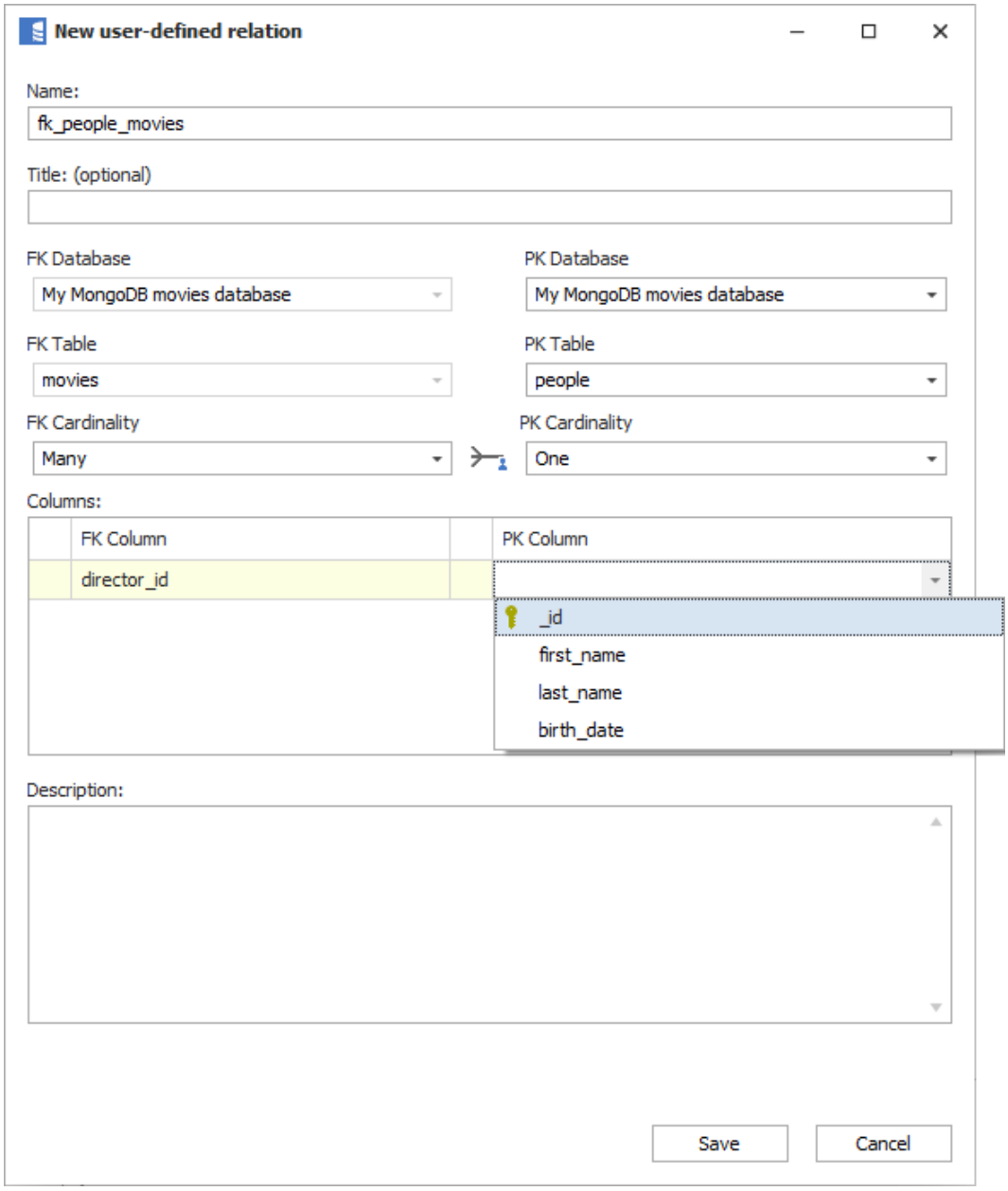
To define relationship (foreign key) with Dataedo, select the field that references other records, right click, and choose **Add relation**.



Now, in **PK Table** field choose a collection this field is referencing.



And in **PK Column** select primary key. Most likely that would be **_id** column



Confirm with **Save**. Now a relationship has been saved in Dataedo metadata repository linking the two collections. You can see this relationship in **References** column next to the field.

movies

Collection | Columns | Relations | Unique keys | Triggers | Deper

	Key	#	Name	Data type	References
		1	_id	Int32	
		2	title	String	
		3	plot	String	
		4	director_id	Int32	people
		5	release_year	Int32	
{}		6	imdb	Document	
		6.1	imdb.rating	Int32	
		6.2	imdb.votes	Int32	
		6.3	imdb.url	String	
		6.4	imdb.last_updated	Null	
[{}]		7	actors	Document[]	
		7.1	actors.person_id	Int32	
		7.2	actors.as	String	
		7.3	actors.salary	Int32	

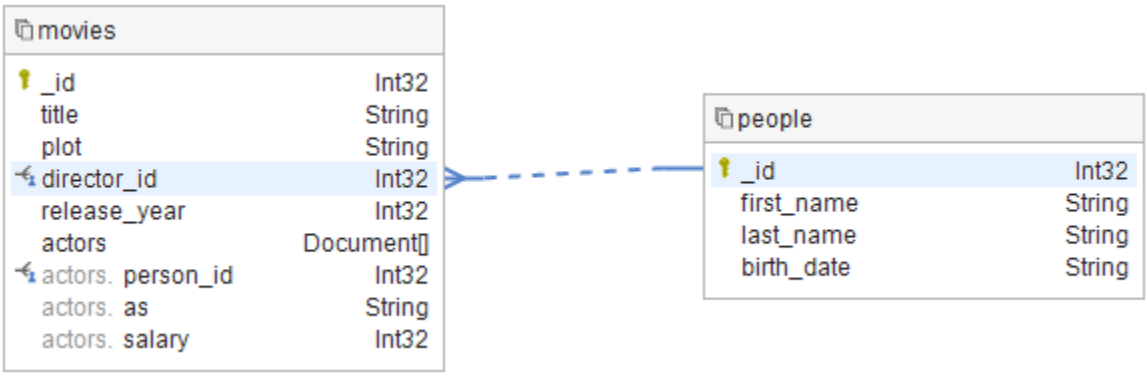
And in **Relations** tab as a separate row.

movies

Collection | Columns | Relations | Unique keys | Triggers | Dependencies | Linked terms

FK Table		PK Table	Title	Relation name	Join	Description
movies		people		fk_people_movies	director_id = _id	

On the diagram, it is represented as a regular relationship between entities.



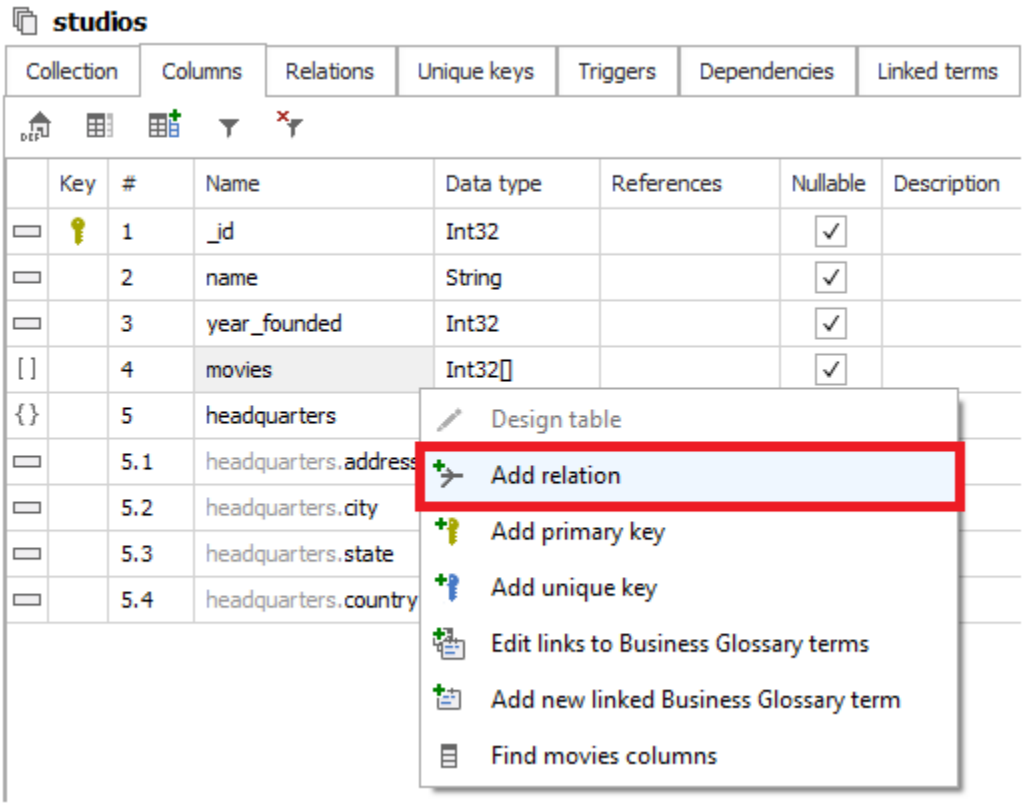
Document References (Many-to-Many)

More advanced reference modeling technique in MongoDB is keeping references in an array, rather than simple field. In case below Studio document stores references to all its Movies in an array of integers.

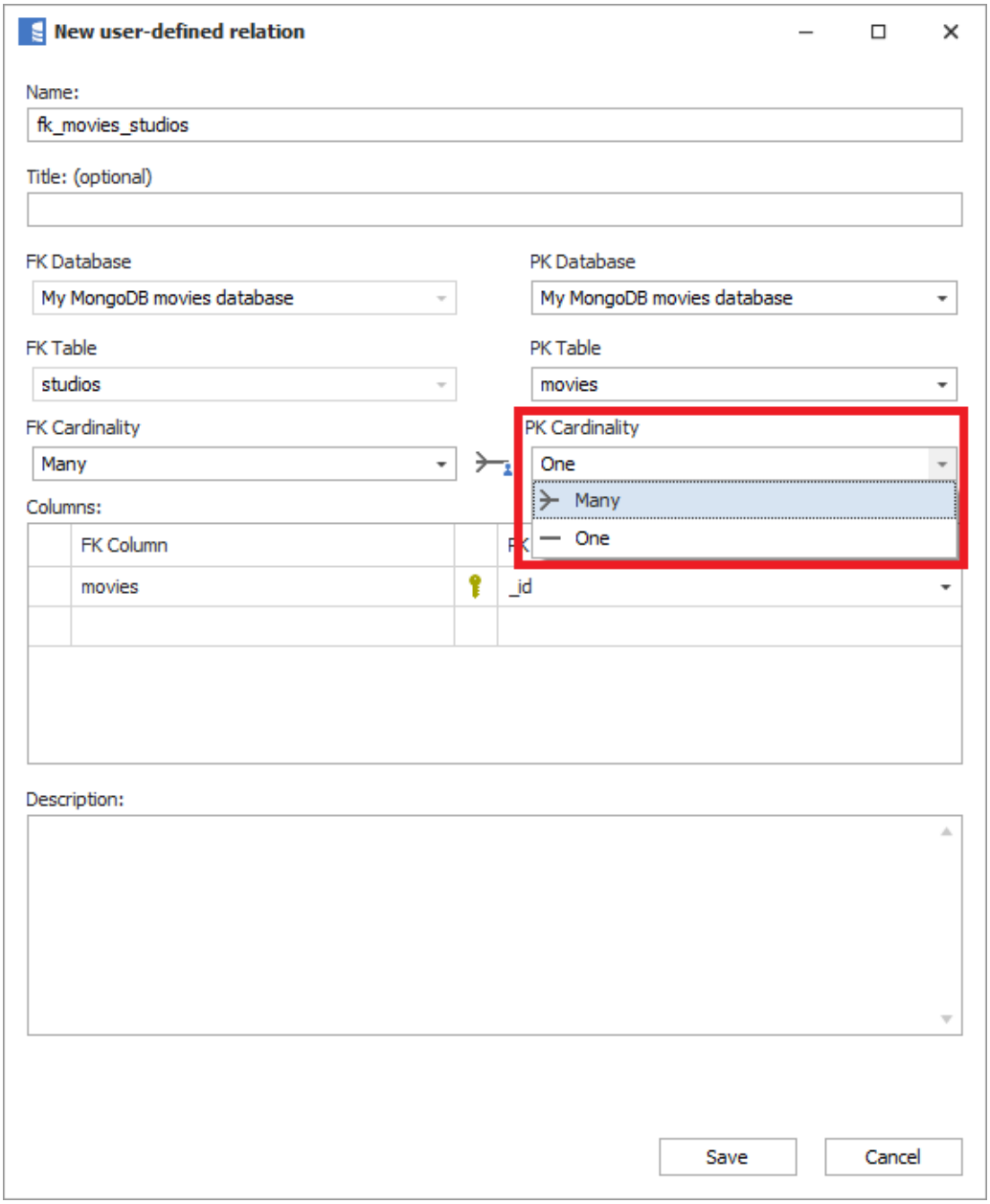
```
// studios
{
    "_id": 1,
    "name": "Warner Bros.",
    "year_founded": 1923,
    "movies": [156, 2234, 334, 2109],
    "headquarters": {
        "address": "4000 Warner Blvd.",
        "city": "Burbank",
        "state": "California",
        "country": "United States"
    }
}
```

```
// movies
{
    "_id": 156,
    "title": "The Terminator",
    "plot": "A cyborg killing machine is sent from 2029 to 1984
    "director_id": 217,
    "release_year": 1984,
    "imdb": {
        "rating": 8.0,
        "votes": 782237,
        "url": "https://www.imdb.com/title/tt0088247/",
        "last_updated": null
    }
}
```

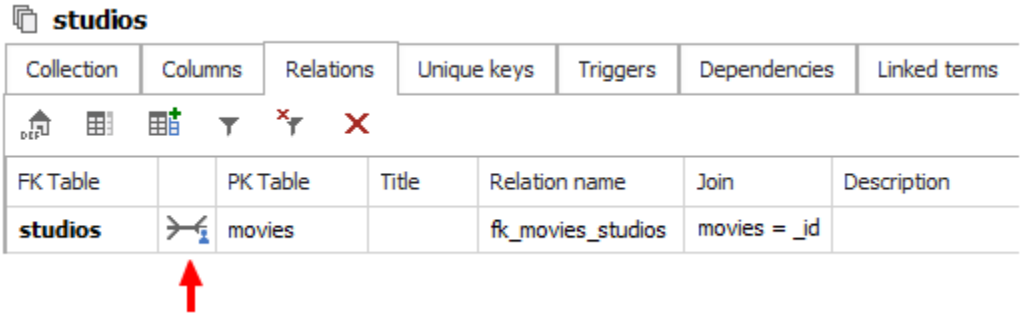
You document this relationship almost identically as the in the case of simple foreign key.



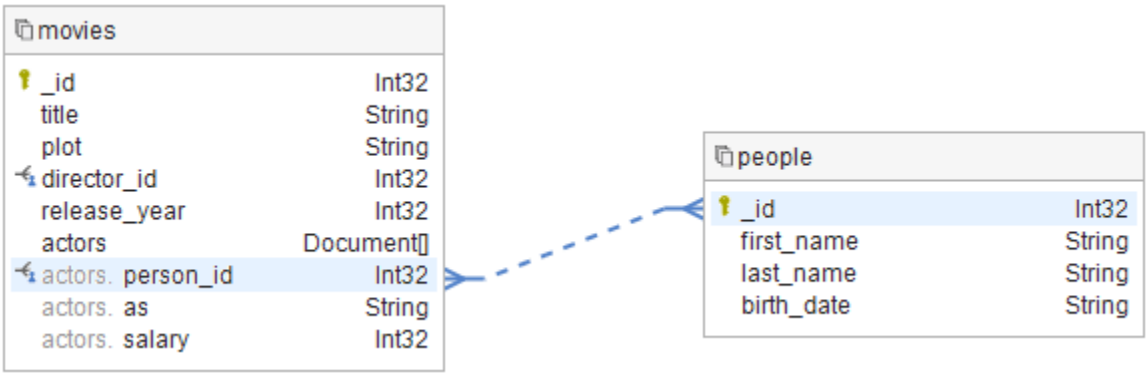
But you need to indicate Many-to-Many cardinality by setting to Many in **PK Cardinality** field.



It will be represented with a different icon.



On the diagram, it is represented as a many-to-many relationship between entities.

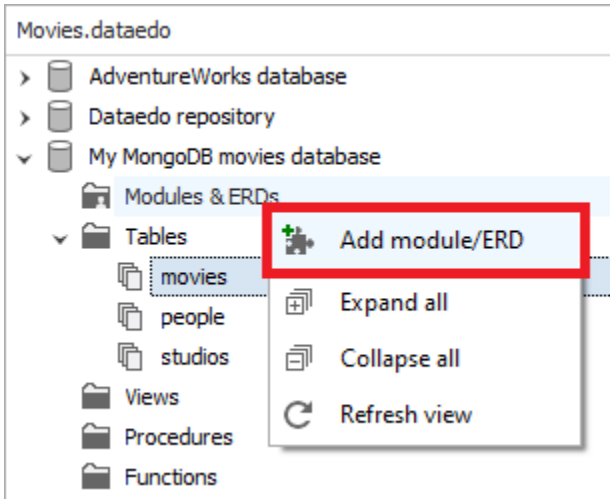


Creating a Diagram

So far, you have built a data dictionary and defined relationships. Now, it is time to build a diagram.

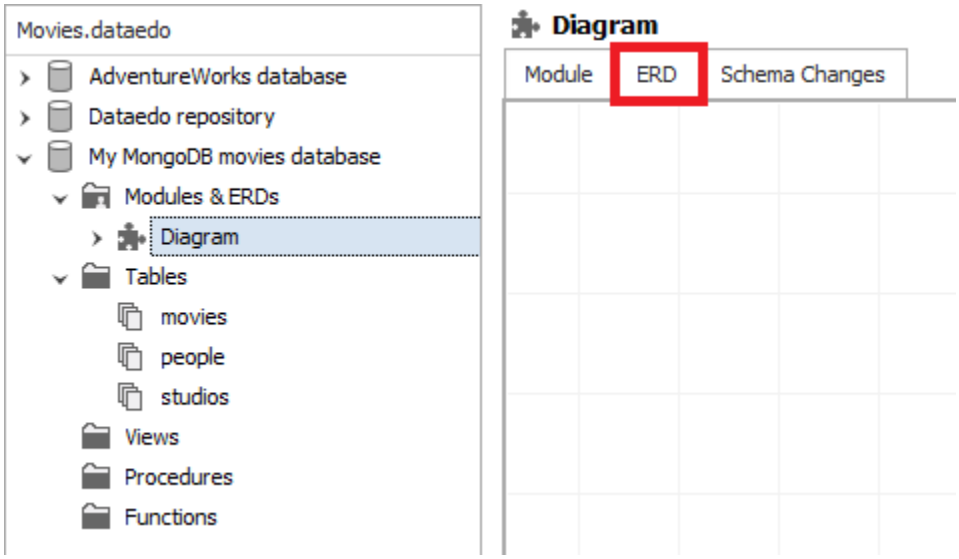
Create a module

To build a diagram you need to create a “module” that will be the container for the diagram. To create a module right click on **Modules & ERDs** and choose **Add module/ERD**. Provide a name for the module.



Create a diagram

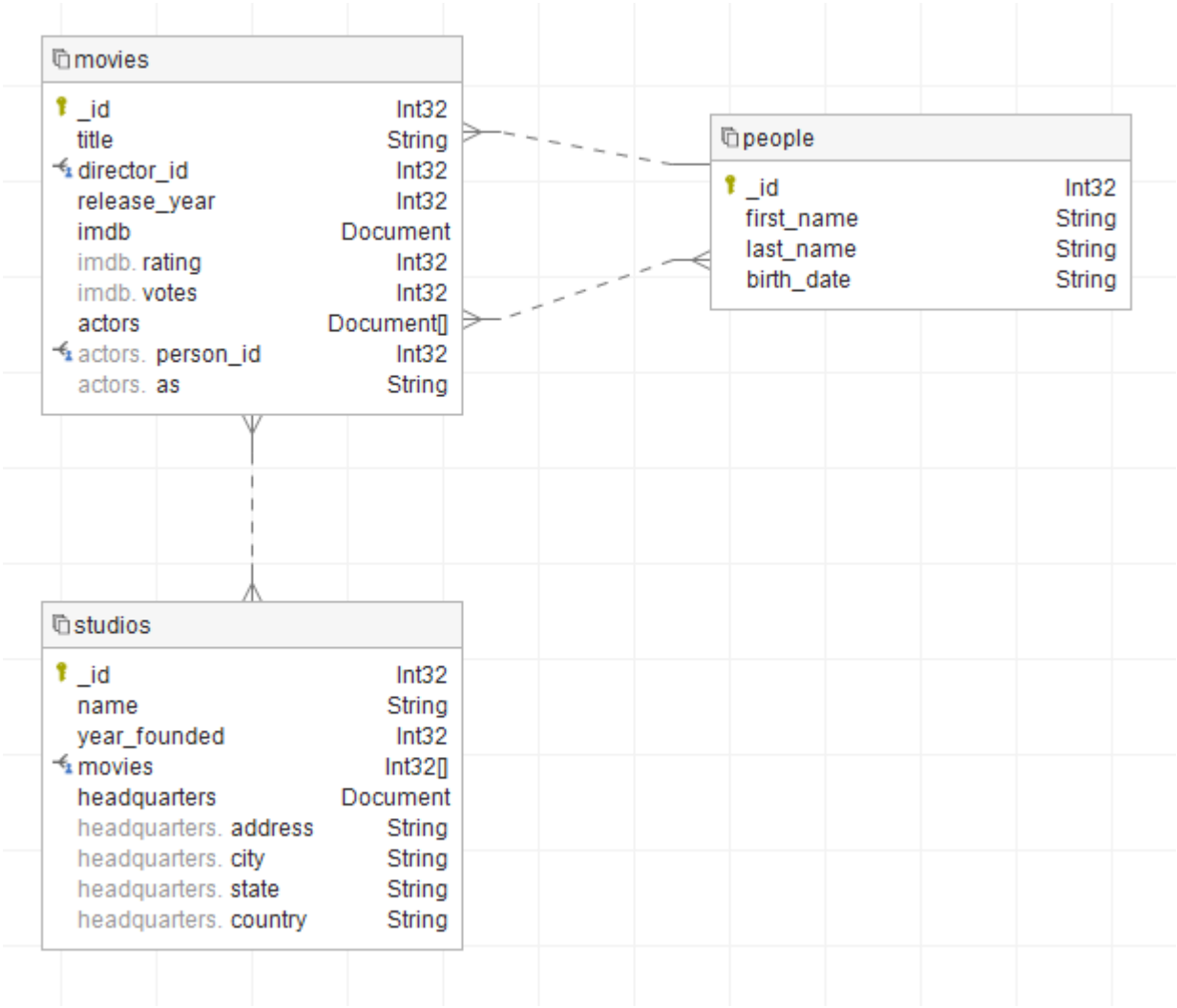
Now you are ready to create a diagram. You can do it on the ERD tab of the module.



On the ERD tab there is a diagram pane and a toolbar with list of available collections/entities. Let's drag & drop to the pane entities you'd like to include in the diagram. Relationships should appear automatically. You can choose which document fields you would like to show by double clicking on entity and selecting columns you would like to be visible.

You can include data types in the diagram with **Show column types** in context menu.

This is the result – an Entity-Relationship Diagram of documents in MongoDB:



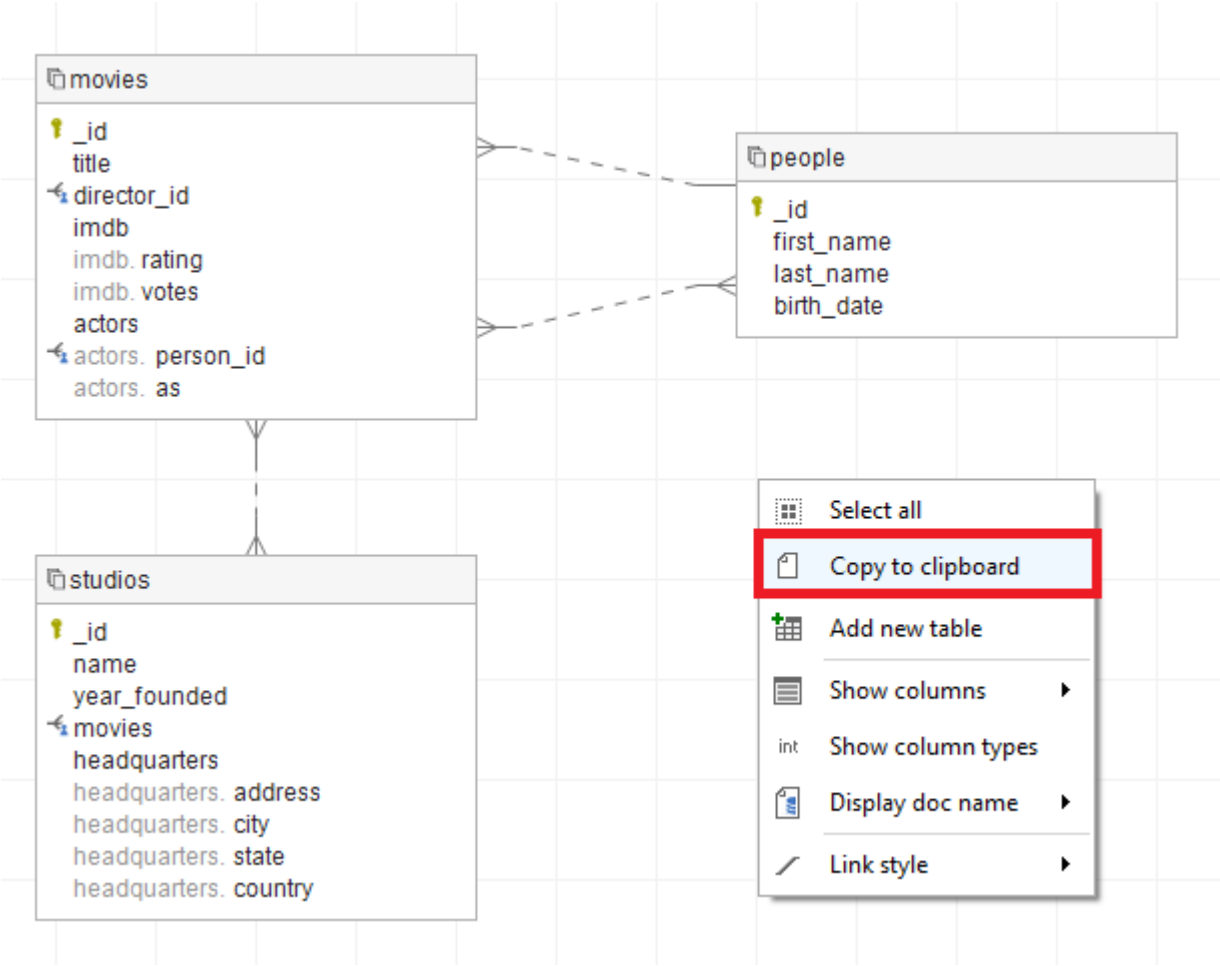
You can repeat that process multiple times creating multiple diagrams presenting different scope of the database.

Share diagram and documentation

Now that you have built the diagram, you should share it. After all, value of the diagram comes from looking at it, so you need to share it with broader community.

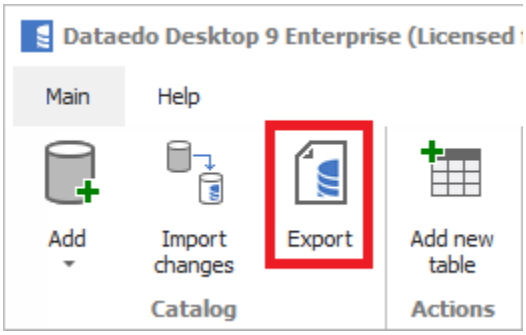
Share diagram as image

You can export diagram as image to clipboard. To do it simply right click pane and choose **Copy to clipboard**. Now you can paste it in a document or save with MS Paint or any other graphical software.

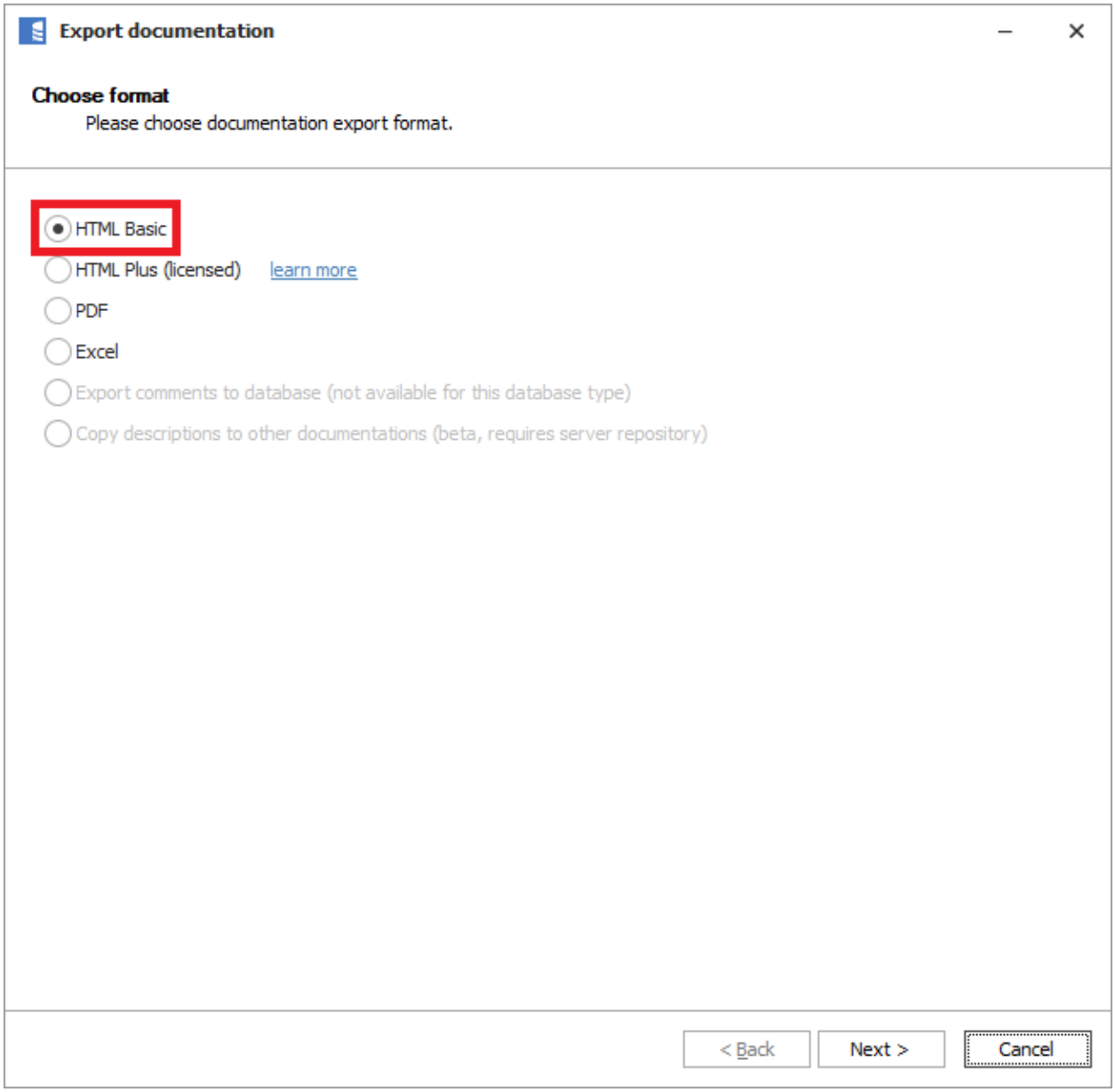


Share entire documentation as HTML

Much better option than just an image is to share the entire data dictionary and all diagrams in interactive HTML documentation. To export documentation, choose Export from the ribbon.

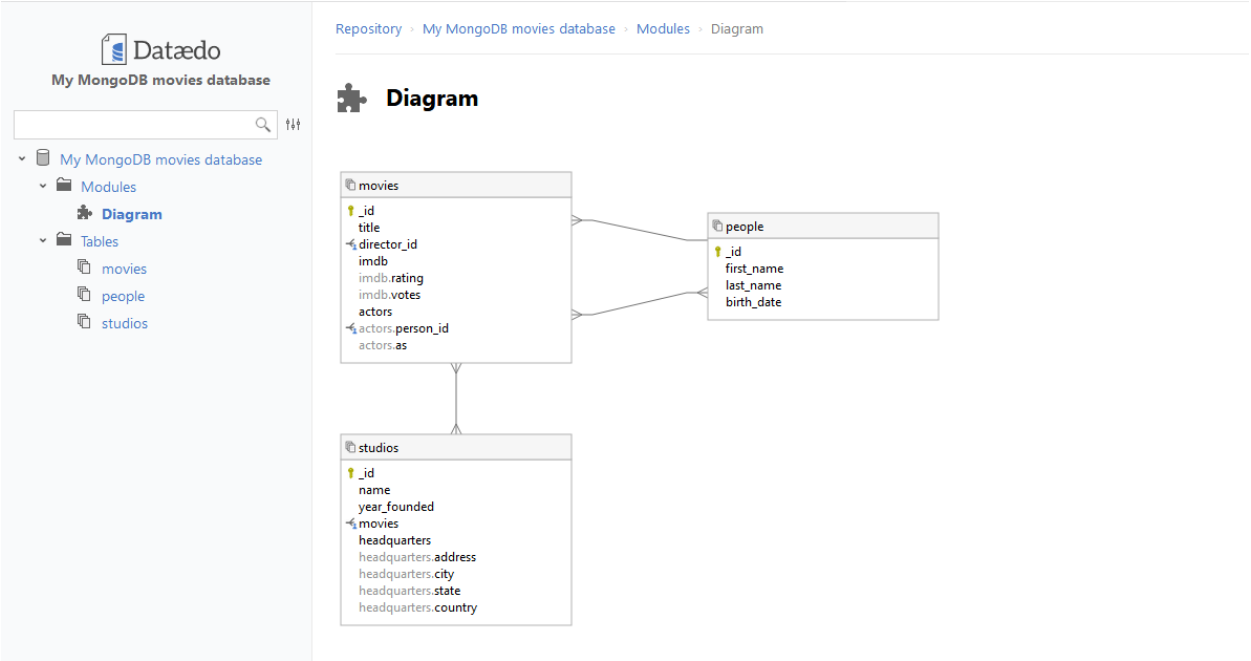


Then choose **HTML Basic**, then click **Next** on the following pages and select the path and a name on the **Choose folder** page. Confirm with **Export** and your documentation will be generated.



The result is this – interactive, searchable, lightweight HTML documentation.

Subrahmanyeswarao Karri



Subrahmanyeswarao Karri

Datædo

My MongoDB movies database

- My MongoDB movies database
  - Modules
    - Diagram
  - Tables
    - **movies**
    - people
    - studios

Repository › My MongoDB movies database › Tables › movies

movies

Documentation	My MongoDB movies database
Schema	movies
Name	movies
Type	Collection

Columns

	Name	Key	Data type	Null	References	Description
1	_id		Int32	✓		Intgernal uniquie movie identifier. Collection primary key.
2	title		String	✓		Movie title.
3	plot		String	✓		Short description of a plot from the cover.
4	director_id		Int32	✓	people	Identifier of the director of the movie.
5	release_year		Int32	✓		Year of the release of the movie.
6	imdb		Document	✓		Aggregated imdb review statistics.
1	imdb.rating		Int32	✓		Average rating on imdb.
2	imdb.votes		Int32	✓		Number of votes on imdb.
3	imdb.url		String	✓		Movie URL on imdb.com.
4	imdb.last_updated		Null	✓		Date the statistics were last updates from the website.
7	actors		Document[]	✓		Array of actors.
1	actors.person_id		Int32	✓	people	Identifier of the actor in the people collection.
2	actors.as		String	✓		Name of the character.
3	actors.salary		Int32	✓		Salary for the role if disclosed.

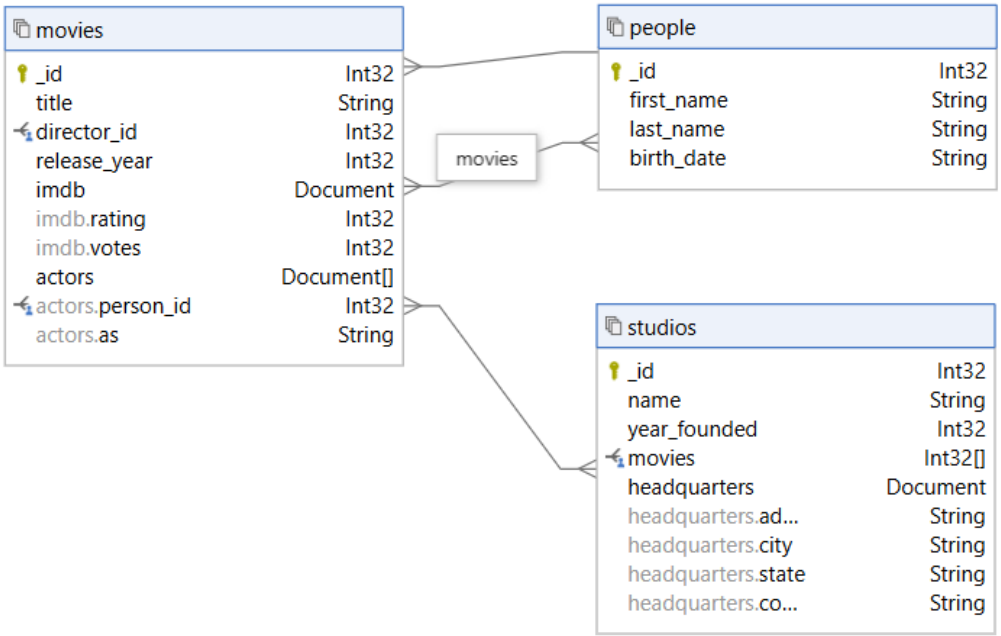
Relations

Foreign table		Primary table	Join	Title / Name / Description
movies		people	**movies**.director_id = people._id	fk_people_movies
movies		people	**movies**.actors.person_id = people._id	fk_people_movies
studios		movies	studios.movies = **movies**._id	fk_movies_studios

Unique keys

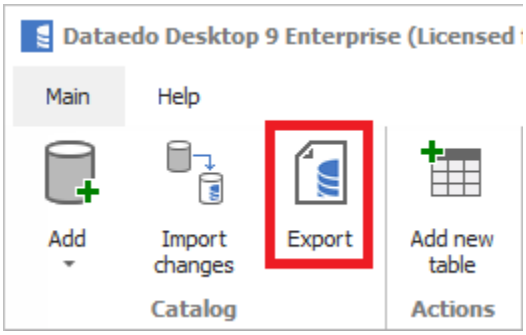
	Key name	Columns	Description
	_id	_id	

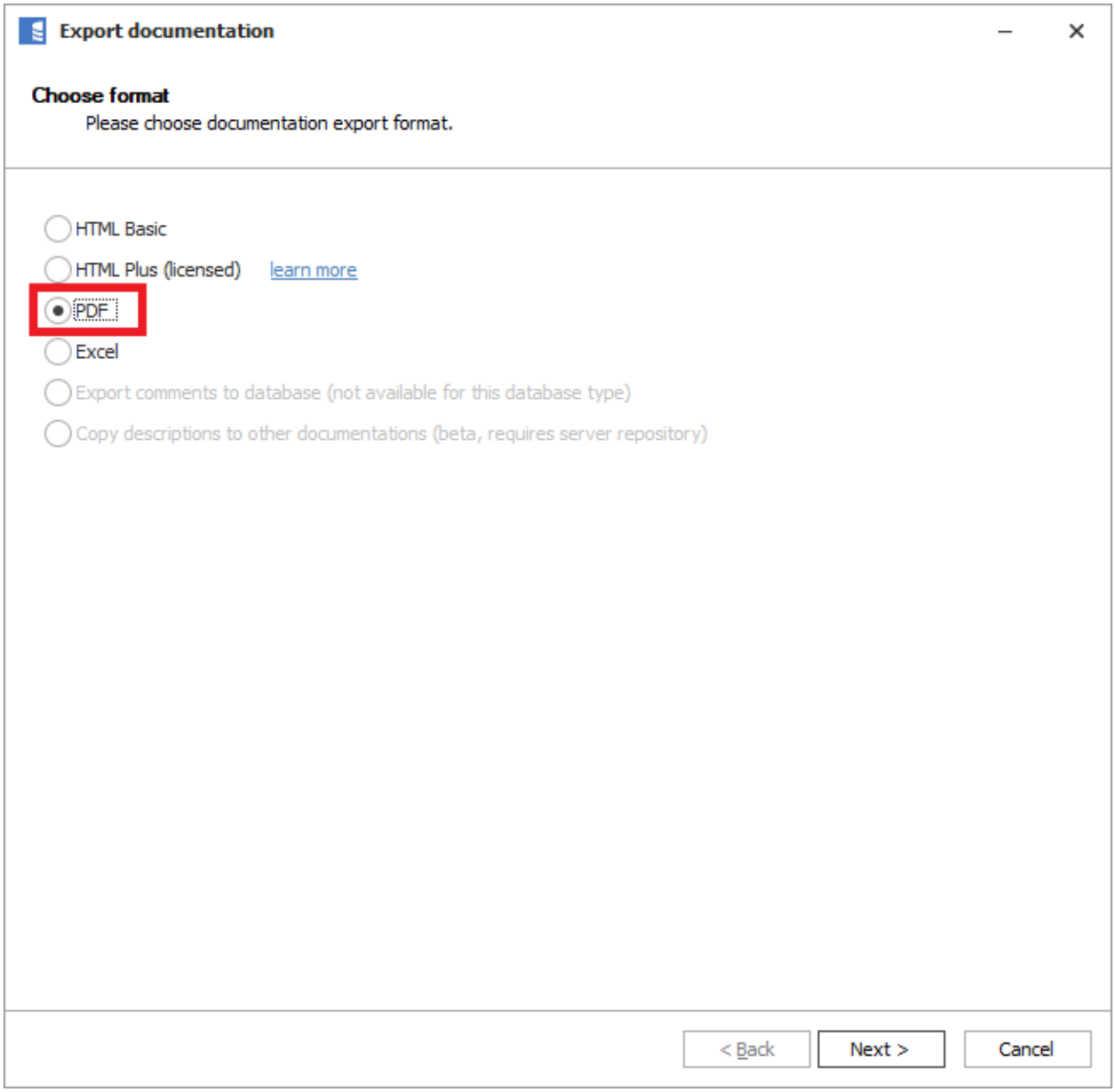
Diagram



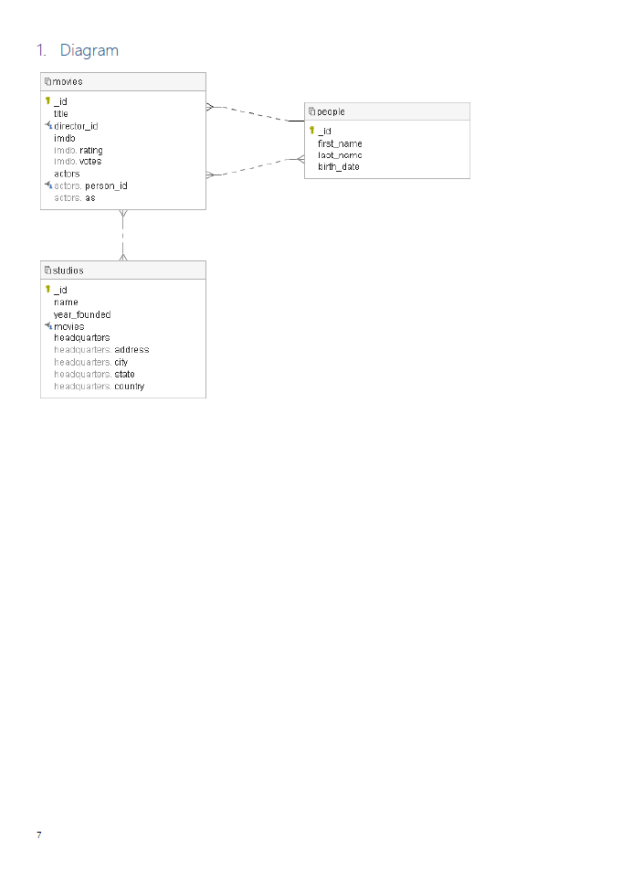
Share entire documentation in PDF

You can also export documentation to PDF document. Process is like exporting HTML, except in this case you choose **PDF** option.





PDF includes ER Diagrams and data dictionary.



1. Diagram

7

2. Other

2.1. Tables

2.1.1. Collection: movies

Columns

	Name	Data type	Description / Attributes
	_id	Int32	Nullable
	title	String	Nullable
	plot	String	Nullable
	director_id	Int32	Nullable References: people
	release_year	Int32	Nullable
	imdb	Document	Nullable
	imdb.rating	Int32	Nullable
	imdb.votes	Int32	Nullable
	imdb.url	String	Nullable
	imdb.last_updated	Null	Nullable
	actors	Document[]	Nullable
	actors.person_id	Int32	Nullable References: people
	actors.as	String	Nullable
	actors.salary	Int32	Nullable

Links to

	Table	Join	Title / Name / Description
	people	moviesdirector_id = people_id	fk_people_movies
	people	movies.actors.person_id = people_id	fk_people_movies

Linked from

	Table	Join	Title / Name / Description
	studios	movies_id = studiosmovies	fk_movies_studios

Unique keys

	Columns	Name / Description
	_id	_id

8

There you have it – a complete guide on how to build a diagram.