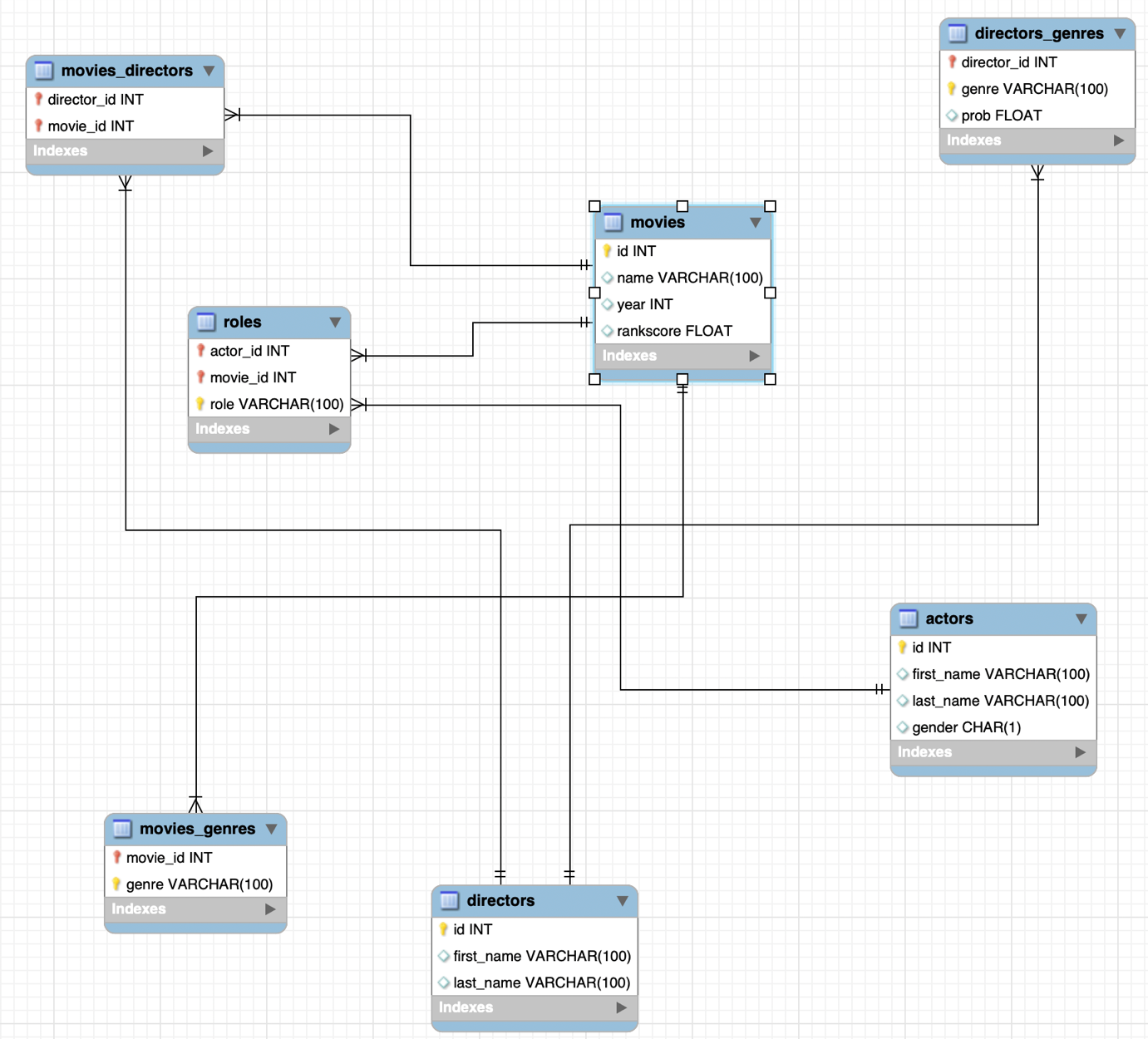
**Problem Statement –**

* Goal is to migrate data from a SQL database to a NoSQL database.
* You have been given SQL data and SQL schema along with NoSQL schema.
* Assuming the column names in SQL data is same as key used in NoSQL schema.
* Using algorithms and schema, migrate data from SQL to NoSQL.

**Data Used –**

IMDB dataset used.

ER diagram –



Since there are more than 300K data in each table. For simplicity, using only one relation.

movies -> roles -> actors

movies to roles is one to many relation.

Actors to roles is one to many relation.

So, “actor” acted in a “movie” for the “role”.

**Schema used for MongoDB –**

{

"movie\_id": "string",

"name": "string",

"year": "int",

"rankscore": "float",

"actors": [

{

"actor\_id": "int",

"role": "string"

}

]

}

**Serialization. –**

class Movies:

def \_\_init\_\_(self, data):

self.movie\_id = data[0]

self.name = data[1]

self.year = data[2]

self.rankscore = data[3]

class Roles:

def \_\_init\_\_(self, data):

self.actor\_id = data[0]

self.role = data[1]

**Possible Solutions –**

Pre-requisite of all solutions would be to map columns of SQL data to NoSQL schema. Since NoSQL schema is in JSON, it can be easily serialized to dictionary or an object.

1. **Brute force –**
   1. The easiest approach in terms of code complexity would be to read each row from SQL table (in this case “movies”). (Extract)
   2. Once read, we convert the data into serialized dictionary based on MongoDB schema. (Transform)
   3. Upon converting, we can push the transformed data to MongoDB. (Load)
2. **A better approach using batch processing –**
   1. Rather than reading Mysql table row by row, it can be easily read batch by batch. Lets say we read data by 1000 row each. (Extract)
   2. Once read, we transform all data into serialized dictionary based on MongoDB schema at once. (Transform)
   3. Upon transformation, we can push all transformed data at Once using insert\_many function. (Load)
3. **Incorporating thread pooling with batch processing (implemented approach) –**
   1. With batch processing in mind, we first divide the whole data into 6 or more threads. There are 10^5 data in movie table. We first divide this data into 60k each into multiple threads. (Extract)
   2. Now for each thread, we read independently for movies dataset along with roles table. (Extract)
   3. Each thread independently transforms the dataset to MongoDB schema. (Transform)
   4. Each thread independently write data to MongoDB with batch processing. (Load)
4. **More efficient approach with thread pooling (future improvement) –**
   1. With batch processing and thread pooling, in-memory database like redis or python dictionary could also be used to pre-load data which are independent of ETL pipeline.
   2. Like loading complete dataset of Roles tables to Redis. This way, the SQL table would be converted directly to a key-value store which can be directly accessed without iterating and searching for it.
   3. This way, the thread will not have to wait for the next process’s result to combine data from different tables. It can run parallely and combine once they are ready to be pushed to mongoDB.