MongoDB

database

Offered by NEILIT, Imphal in collaboration with Lamzing Technologies Pvt. Ltd.

About this course

Duration	~4-5 weeks
Timing	Mon to Fri 9:30am to 12:00pm
Instructor	Bijen Hemam, Lamzing Technologies Pvt Ltd

Brief Introduction

How MongoDB differs from MySQL

MySQL	MongoDB
Relational database	Document-based NoSQL DB
Fixed schema	Dynamic schema
Tables with rows and columns	Collections with JSON-like documents
Structured Query Language (SQL)	Query by MongoDB's JSON-based query language
Supports indexes	Supports indexes

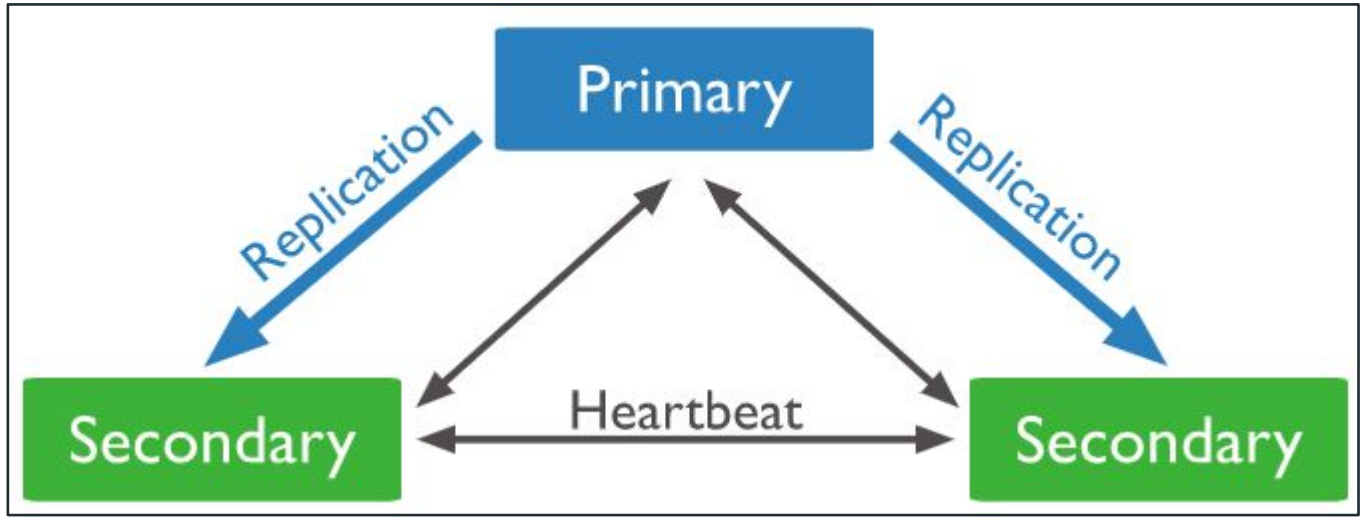
MySQL vs MongoDB

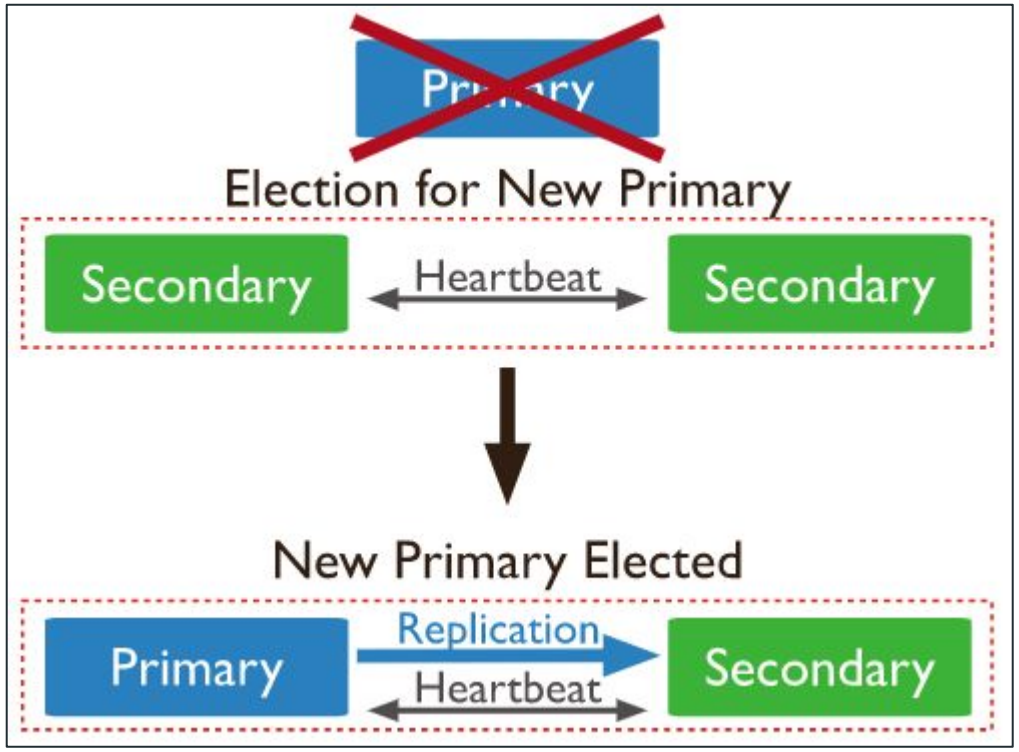
MySQL	MongoDB
Supports complex joins	No support for joins
ACID-compliant transactions	ACID Supported
Vertical scaling	Horizontal scaling through sharding
Limited JSON support	Native support for JSON data

How MongoDB differs from MySQL

MySQL	MongoDB
Supports atomic operations	Supports atomic operations
Strong consistency	Eventual consistency
Data relationships through foreign keys	Data relationships via embedded documents and manual references
Traditional applications	High-volume, unstructured data applications

MongoDB Automatic Replication



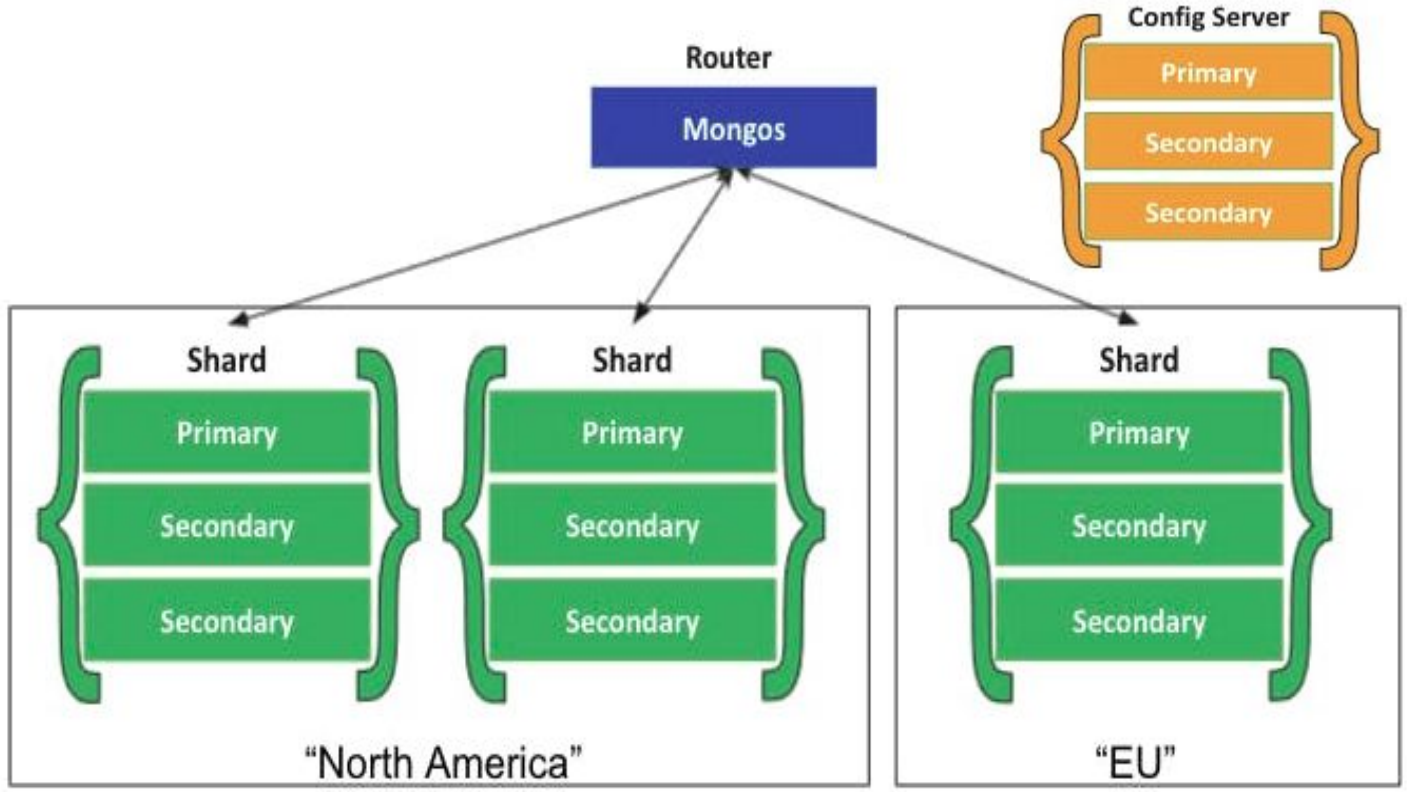


- Typical setup has 3 node, 1-primary and 2-secondary. Also called **ReplicaSet**
- When Primay is unavailable, a new primary is selected

https://www.mongodb.com/docs/manual/replication/

Sharding in MongoDB

- method for distributing or partitioning **data across multiple machines**
- Allows for horizontal scaling



https://www.mongodb.com/basics/sharding

MongoDB SETUP

Resources

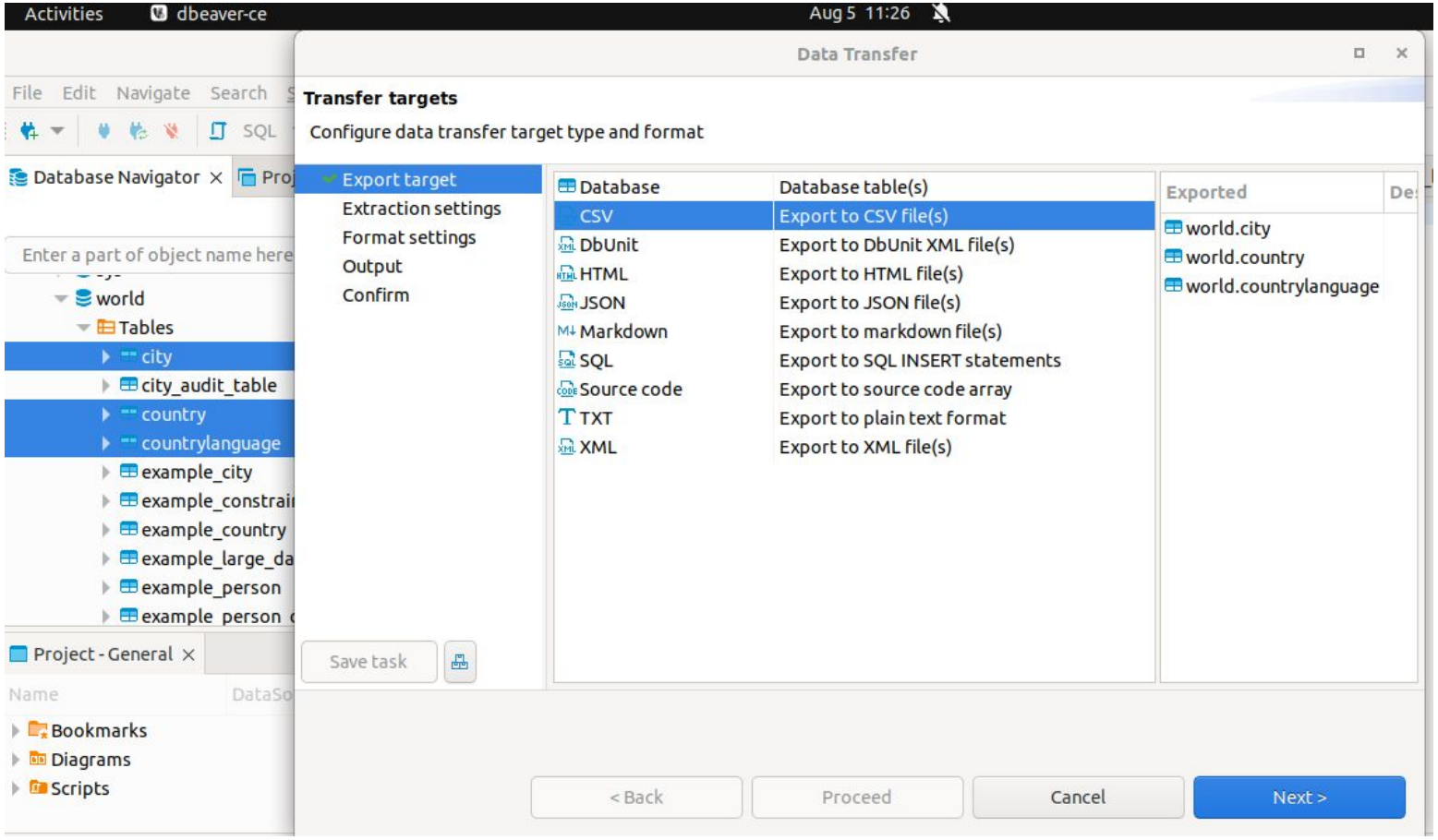
- MongoDB database installation
  - https://www.mongodb.com/try/download/community
- Clients to connect to the MongoDB database
  - https://www.mongodb.com/try/download/compass
  - https://www.mongodb.com/try/download/shell
- Tutorial
  - https://www.w3schools.com/mongodb/index.php
- Online Playground
  - https://www.humongous.io/app/playground/mongodb/new

JSON Format


```
{
    "Code": "ABW",
    "Name": "Aruba",
    "Continent": "North America",
    "Region": "Caribbean",
    "numberical_data": {
        "SurfaceArea": 193,
        "Population": 103000,
        "LifeExpectancy": 78.4,
    }
}
```

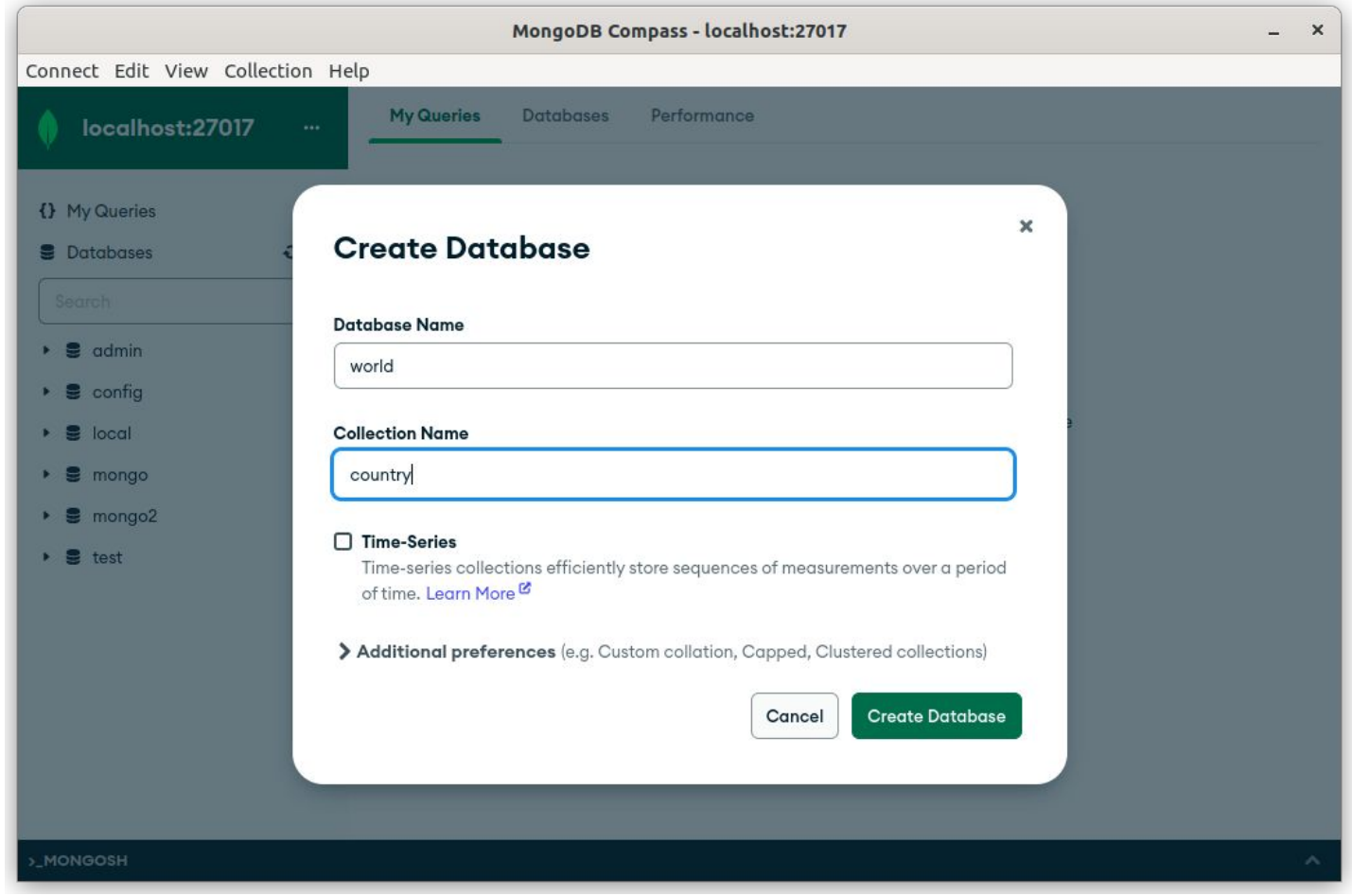
Load Data into MongoDB

- Export country, city and countrylanguage table from mysql in csv format



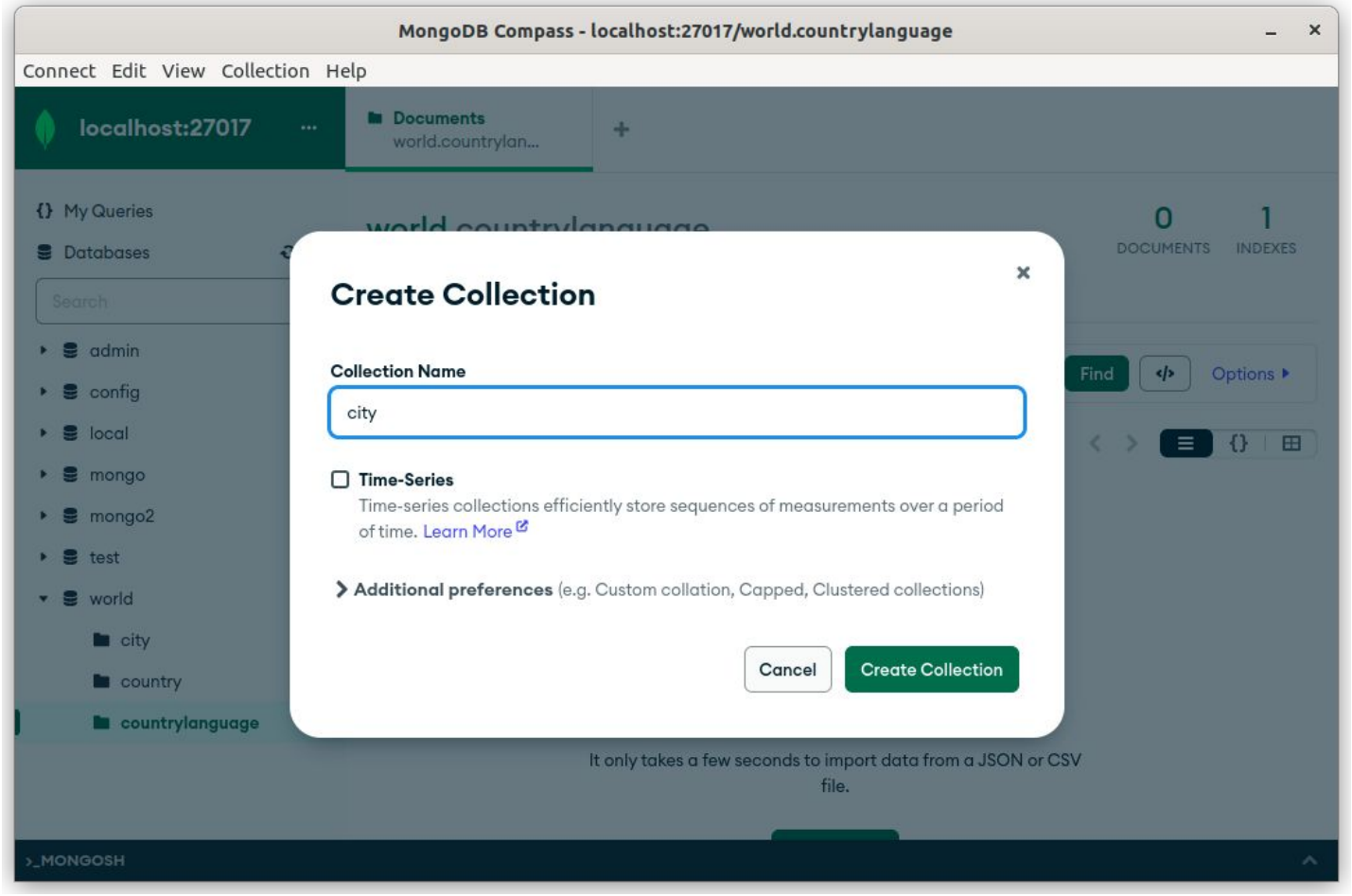
Load Data into MongoDB

- Create a database ‘world’
- Add a default collection



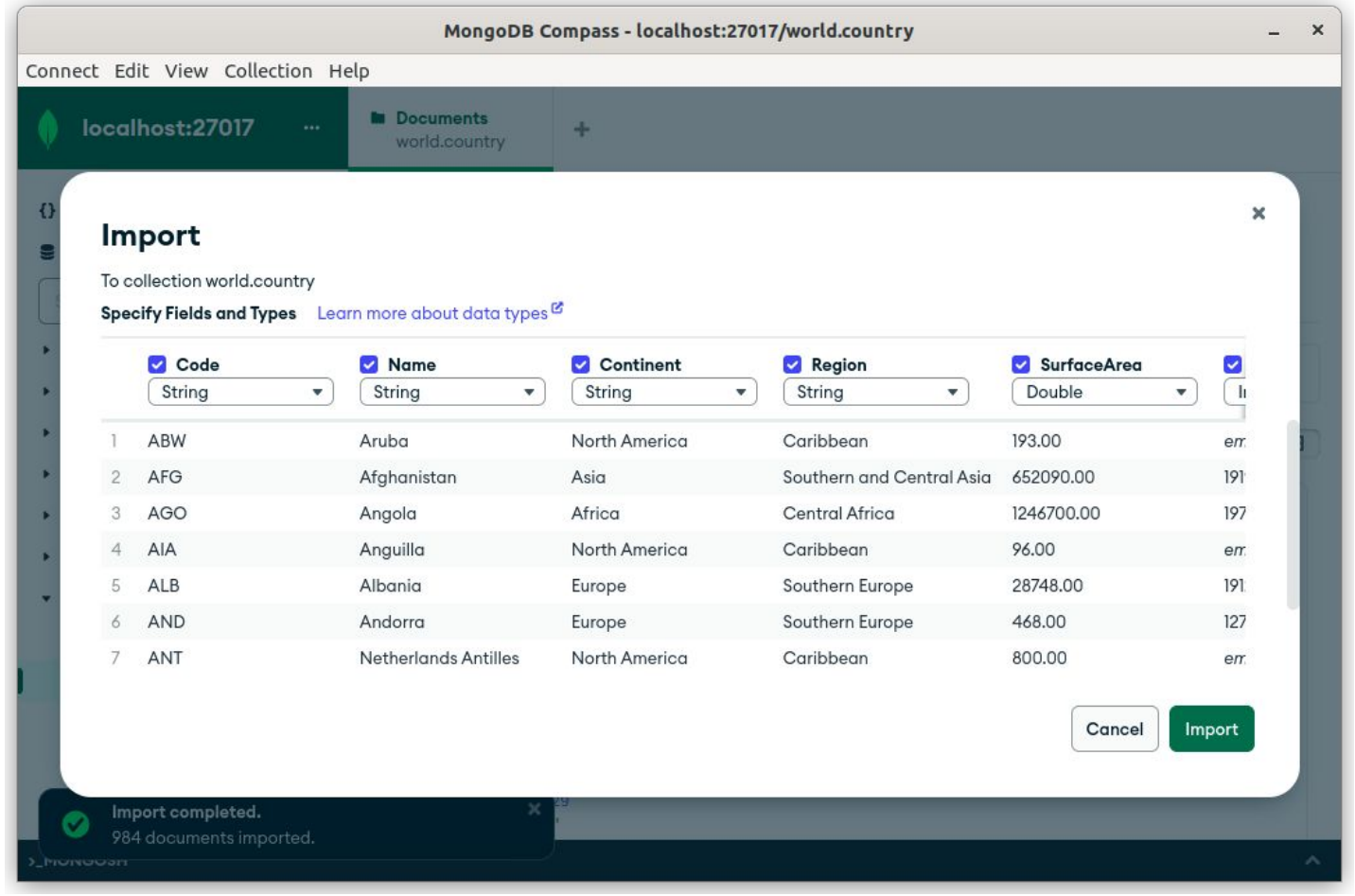
Load Data into MongoDB

- Create other collections
- Collection in mongodb = Table in MySQL
- s



Import csv data into collection

- Select the collection on left navigation to import
- To import Click on Menu Collection > Import data
- Import all data
  - Country
  - City
  - CountryLanguage



MongoDB Topics

- **Data Manipulation Language (DML)**
- Data types
- ~~Data Definition Language (DDL)~~
- ~~SQL Joins~~
- ~~Subqueries~~
- Aggregation Functions
- Constraints -> Schema Validation
- Indexes
- Transactions
- Views
- ~~Triggers~~
- ~~Stored Procedures~~

Example

- Query in mysql

```
SELECT * FROM country;
SELECT * FROM country limit 1;
SELECT name, code, population FROM country;
```

- Query in MongoDB

```
db.country.find()  or
db.getCollection('country').find()
db.getCollection('country').findOne()
db.country.find({},{Name:1,Code:1,Population:1})
```

Query Filter or WHERE clause

`db.country.find(<query filter>, <projection>)`

Query Filter takes following operators to compare against the data

- Operators for Comparison

$eq: matches values that are equal to a specified value.

$ne: matches values that are not equal to a specified value.

$gt: matches values that are greater than a specified value.

$gte: matches values that are greater than or equal to a specified value.

$lt: matches values that are less than a specified value.

$lte: matches values that are less than or equal to a specified value.

Query Filter or WHERE clause

`db.country.find(<query filter>, <projection>)`

Query Filter takes following operators to compare against the data

- Logical Operators:

$and: requires all conditions to be true.

$or: requires at least one condition to be true.

$not: matches documents that do not meet the specified condition.

- Element Operators:

$exists: matches documents that contain the specified field.

$type: matches documents that have a field of a specific BSON data type.

Query Filter or WHERE clause

`db.country.find(<query filter>, <projection>)`

Query Filter takes following operators to compare against the data

- Array Operators:

$in: matches any of the values specified in an array.

$nin: matches none of the values specified in an array.

$all: matches arrays that contain all the specified elements.

$elemMatch: matches documents has an array element with all specified values

- Regular Expression Operator:

$regex: matches documents that have a field matching the regular expression.

filter documents

```
SELECT name, population, SurfaceArea
FROM country
WHERE Code = 'IND';
```

```
db.city.find( {CountryCode: "IND"},
{Name:1,Population:1,SurfaceArea:1}
)
```

```
db.country.find( { Population: {"$gt": 100000}, SurfaceArea: {"$lt": 200} },
{Name:1, Population:1, SurfaceArea: 1}
)
```

$and

```
SELECT name, population, district
FROM city
WHERE population > 100000 and population < 500000;
```

```
 db.city.find( {"$and": [  {Population: {"$gt": 100000 }},
                {Population: {"$lt": 500000 }}
                ]},
     {Name:1,Code:1,Population:1}
)
```

$or

```
SELECT name, population, district
FROM city
WHERE population < 100000 or population > 500000;
```

```
 db.city.find( {"$or": [  {Population: {"$gt": 500000 }},
                {Population: {"$lt": 100000 }}
                ]},
     {Name:1,Code:1,Population:1}
)
```

$gte, $lte

```
SELECT name, population, district
FROM city
WHERE population between 100000  and 500000;
```

- No equivalent operator. Use less-then-equal and greater-then-equal since the start and end values are inclusive for between clause

```
db.city.find( {"$and": [  {Population: {"$gte": 100000 }},
                {Population: {"$lte": 500000 }}
                ]},
    {Name:1,Code:1,Population:1}
)
```

$in

```
SELECT Name,Code,Population
FROM country
WHERE Code IN ("IND","NPL","USA","AUS");
```

```
db.country.find(
{Code: {"$in":["IND","NPL","USA","AUS"]}},
{Name:1,Code:1,Population:1}
)
```

$regex

```
SELECT Name,Code,Population
FROM country
WHERE Code LIKE ‘A%’;
```

```
db.country.find(
{Code: {"$regex":"^A.*"}},
{Name:1,Code:1,Population:1}
)
```

$regex

```
SELECT Name,Code,Population
FROM country
WHERE Name LIKE ‘%dia%’;
```

```
db.country.find(
{Name: {"$regex":".*dia.*"}},
{Name:1,Code:1,Population:1}
)
```

sort

```
SELECT CountryCode, Name, Population
FROM city
ORDER BY CountryCode, Name;
```

```
db.city.find( {},
{CountryCode:1, Name:1, Population:1}
).sort({CountryCode: "asc",  Name: "asc"})
```

Working with NULL Values

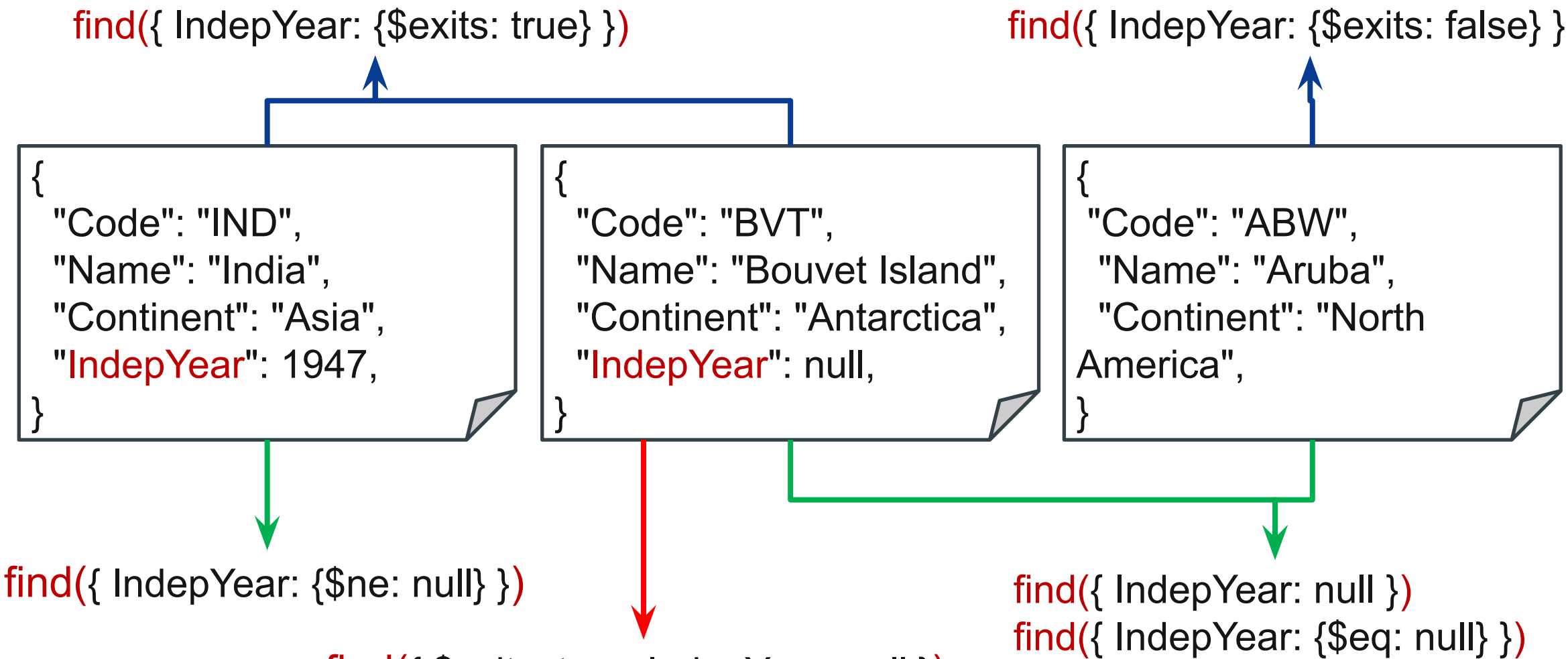
```
SELECT name, IndepYear
FROM country WHERE IndepYear IS NULL;
```

```
db.country.find( { IndepYear: null },
{ Name: 1, Code: 1, Population: 1, IndepYear: 1 } )
```

```
db.country.find( { IndepYear: { $eq: null } },
{ Name: 1, Code: 1, Population: 1, IndepYear: 1 } )
```

```
db.country.find( { IndepYear: { $eq: null, $exists: true } },
{ Name: 1, Code: 1, Population: 1, IndepYear: 1 } )
```

Working with NULL Values



Insert new document

```
db.city.insertOne({
  "ID": 1, "Name": "Ukhrul", "CountryCode": "IND",  "District": "Manipur",
  "Population": 56000
})
db.city.insertOne({
"Name": "Mao", "CountryCode": "IND",  "District": "Manipur"
})
db.city.insertOne({
"Name": "Jiribam", "CountryCode": "IND",  "District": "Manipur", "Population": 56000,
insert_date: new Date("2023-08-08")
})
```

ObjectId

- A 4-byte timestamp, representing the ObjectId's creation, measured in seconds since the Unix epoch.
- A 5-byte random value generated once per process. This random value is unique to the machine and process.
- A 3-byte incrementing counter, initialized to a random value.
- ObjectId(**"00000020f51bb4362eee2a4d"**)

4 byte timestamp	5 byte random	3 byte increment counter
00000020	f51bb4362e	ee2a4d

https://www.mongodb.com/docs/manual/reference/method/ObjectId/

Insert new document - bulk insert

```
db.city.insertMany( [
{ "ID": 2, "Name": "Thoubal", "CountryCode": "IND", "District": "Manipur",
"Population": 32000},
{ "ID": 3, "Name": "Moreh", "CountryCode": "IND", "District": "Manipur",  "Population":
23490},
{ "ID": 4, "Name": "CCpur", "CountryCode": "IND", "District": "Manipur",  "Population":
98000}
] )
```

Update new document

```
db.city.updateOne( { District: "Manipur" },
{
  $set: {
    area: 12000
  },
  $currentDate: { lastUpdated: true }
})
```

Update new document - bulk update

```
db.city.updateMany( { District: "Manipur" },
{
  $set: {
    area: 15000,
    location: [24.8170, 93.9368]
  },
  $currentDate: { lastUpdated: true }
})
```

Delete

- Delete first matching document

```
db.getCollection('city').deleteOne( { Name: "TEST03" } )
```

- Delete all matching documents

```
db.getCollection('city').deleteMany( { Name: "TEST03" } )
```

- Delete ALL documents

```
db.getCollection('city').deleteOne( { District: "Manipur" } )
```

LIMIT OFFSET

```
SELECT code, name, population
FROM country
order by name
limit 11 offset 10;
```

```
db.country.find({},
{Code:1,Name:1,Population:1}
).skip(10).limit(3)
```

Projection min, max

```
SELECT min(population), max(population)
FROM country;
```

```
db.getCollection('country').aggregate( [ {
$group: { _id:null,
    minPopulation: { $min: "$Population" },
    maxPopulation: { $max: "$Population" }
    }
} ])
```

min, max with condition

```
db.getCollection('country').aggregate( [
{
    $match: {
        "Continent": { $eq: "Europe" }       //   ~= { "Continent":  "Europe" }
    }
},
{
    $group: { _id:null,
        minPopulation: { $min: "$Population" },
        maxPopulation: { $max: "$Population" }
    }
} ] )
```

count, average, sum

```
db.getCollection('country').aggregate( [
{
    $group: { _id:null,
        countryCount: { $count: {} },
        avgPopulation: { $avg: "$Population" },
        sumSurfaceArea: { $sum: "$SurfaceArea" }
    }
} ] )
```

MySQL CASE

```
SELECT code, name, population,
CASE
    WHEN population < 1000000 THEN 'SMALL POPULATION'
    WHEN population < 5000000 THEN 'MEDIUM POPULATION'
    ELSE 'LARGE POPULATION'
END AS population_size
FROM country;
```

Similar to IF condition in other programming languages

$switch

```
db.country.aggregate( [ {
    $project: {
      Name: "$Name",
      result: {
        $switch: {
          branches: [
            { case: { $lt: [ "$Population", "active" ] }, then: "SMALL POPULATION" },
            { case: { $lt: [ "$Population", "inactive" ] }, then: "MEDIUM POPULATION" }
          ],
          default: "LARGE POPULATION"
        }
      }
    }
  } ] )
```

MongoDB Topics

- Data Manipulation Language (DML)
- **Data types**
- Schema Validation / Constraint
- Working with Arrays
- Aggregation Functions
- Indexes
- Transactions
- Views -> Virtual Collections

DataTypes

- String
- Number – can store both integer and float values
- Boolean
- ISODate
- Int32 - 32 bit signed integer
- Long - 64 bit signed integer
- Decimal128 – for scientific and high precision floating point numbers

Datatypes

- String

```
db.person.insertOne({first_name:"Test", last_name: "User"})
```

- Date

```
db.person.updateOne( { first_name:"Test", last_name: "User" },
{ "$set": { dob: new Date("2000-01-01") } } )
```

- Number

```
db.person.updateOne( { first_name:"Test", last_name: "User" },
{ "$set": { age: 23 } } )
```

Datatypes

- Datetime

```
db.person.updateOne( { first_name:"Test", last_name: "User" },
{ "$set": { update_dt: new Date() } } )
```

- Boolean

```
db.person.updateOne( { first_name:"Test", last_name: "User" },
{ "$set": { married: false } } )
```

- Number vs Decimal with exact precision

```
db.person.updateOne( { first_name:"Test", last_name: "User" },
{ "$set": { height: 175.5, weight: Decimal128("70.2") } } )
```

Datatypes

- array

```
db.person.updateOne( { first_name:"Test", last_name: "User" },
{ "$set": { hobbies: ["reading", "movies", "trekking"] } } )
```

- nested object

```
db.person.updateOne({ first_name: "Test", last_name: "User" },
{ "$set": { address: { street: "Neilit, Akampat",
        city: "Imphal",
        country: "Manipur"
        }
    }
})
```

Find by datatype

```
db.person.find( { } )

db.person.find( { "weight": { $type: "decimal" } } )

db.person.find( { "weight": { $type: "number" } } )
```

- Insert 2-3 additional Person document with hobbies?

MongoDB Topics

- Data Manipulation Language (DML)
- Data types
- **Schema Validation / Constraint**
- Working with Arrays
- Aggregation Functions
- Views
- Indexes
- Transactions

Collection with validation schema

```
db.createCollection("products", {  validator: {
   "$jsonSchema": {
      bsonType: "object",
      properties: {
        name: { bsonType: "string" },
        price: { bsonType: "decimal", minimum: 0 },
        sku: { bsonType: "int", minimum: 0 },
        quantity: { bsonType: "number", minimum: 0 },
      },
      required: ["name", "price","sku"]
   }
 } ,   validationLevel: "strict",   validationAction: "error"  } ) ;
```

Example Insert

- Required field missing

```
db.products.insertOne({ name: "P1", sku: 100 });
```

- Mismatched data type

```
db.products.insertOne({ name: "P1", price: 49.99, sku: 100 });
```

- Valid document

```
db.products.insertOne({ name: "P1", price: NumberDecimal("49.99"), sku: 100 });
db.products.insertOne({ name: "P2", price: NumberDecimal("49.99"), sku: 100,
quantity:23 });
db.products.insertOne({ name: "P3", price: NumberDecimal("49.99"), sku: 100,
quantity:15.15 });
```

Collection without validation schema

- Drop the product collection

```
db.getCollection("products").drop()
```

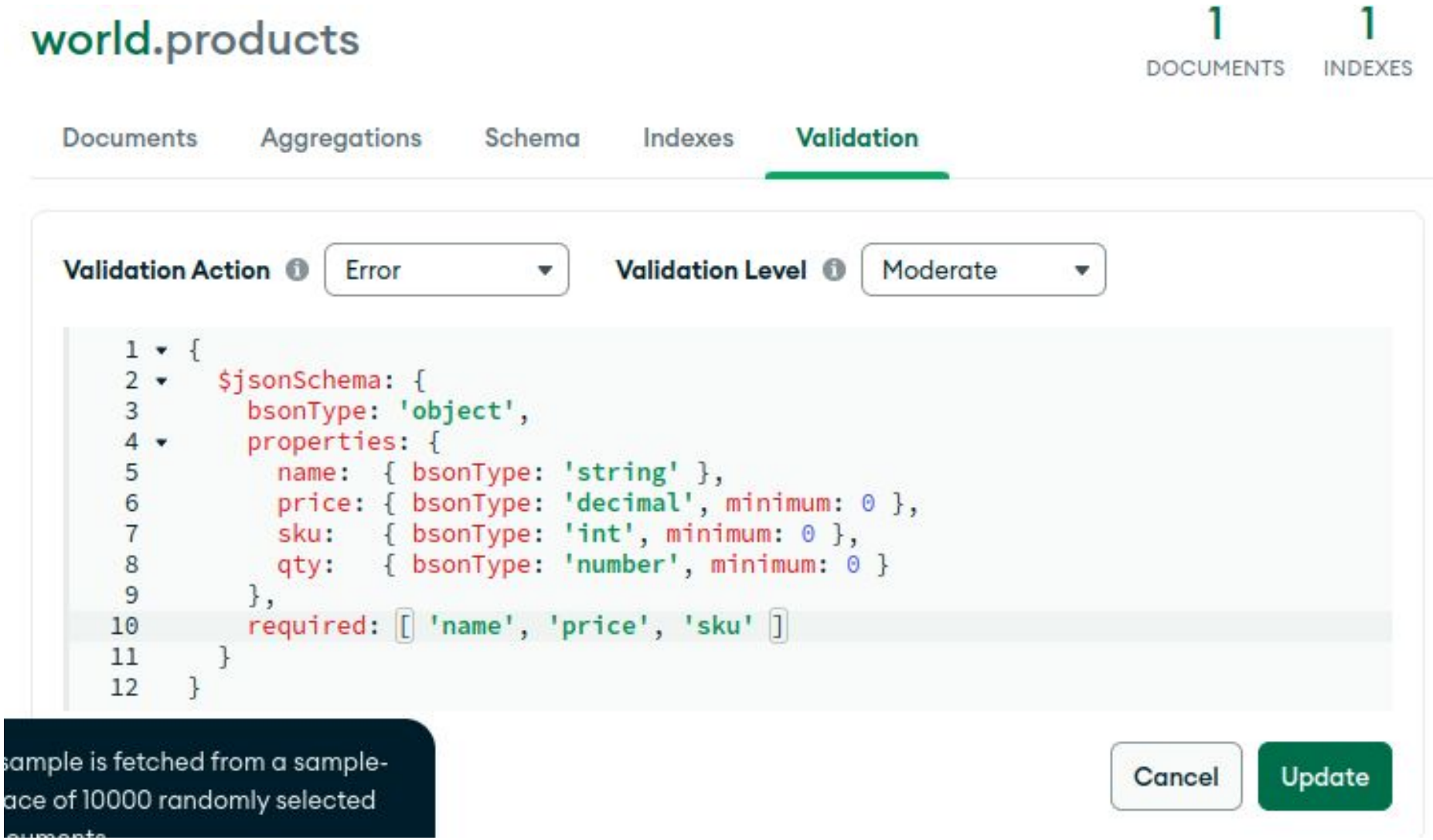
- create a new collection without schema

```
db.createCollection("products")
```

- Insert a document

```
db.products.insertOne( { name: "P1", price: 49.99, sku: 1, quantity: 20 } ) ;
db.products.insertOne( { name: "P2", price: 49.99, sku: "sku1", quantity: 13.2 } ) ;
```

Apply a schema



Example Insert Update

- Insert

```
db.products.insertOne({ name: "P3", price: 49.99, sku: -2,  quantity: -10});
db.products.insertOne( { name: "P4", price: NumberDecimal("49.99"), sku: 4,
quantity: 20 } ) ;
db.products.insertOne({ name: "P3", price: NumberDecimal("49.99"), sku: 2 });
```

- Update

```
db.products.updateOne( {name:"P2"}, {"$set": { sku : -2 } } )
db.products.updateOne( {name:"P2"}, {"$set": { sku : 2 } } )
db.products.updateOne( {name:"P2"}, {"$set": {sku:2, price:NumberDecimal("45.30")
} } )
```

MongoDB Topics

- Data Manipulation Language (DML)
- Data types
- Schema Validation
- **Working with Arrays**
- Aggregation Functions
- Views
- Indexes
- Transactions

$all

- Compares input with an array field

- Checks if reading is present in the array field hobbies

```
db.person.find({hobbies: {$all: ["reading"] } })
```

- Checks if both reading and movies is present in the array field hobbies

```
db.person.find({hobbies: {$all: ["reading", "movies"] } })
```

- Checks if reading, movies and singing is present in the array field hobbies

```
db.person.find({hobbies: {$all: ["reading", "movies", "singing"] } })
```

$elemMatch

- Compares input with an array field

- Find persons with a ‘reading’ hobby

```
db.person.find({hobbies: {$elemMatch: {$eq: "reading"} } })
```

- Find person without a ‘cooking’ hobby

```
db.person.find({hobbies: {$not: {$elemMatch: {$eq: "cooking"} } } } )
```

- Find married person with a ‘travel’ hobby

```
db.person.find({hobbies: {$elemMatch: {$eq: "travel"} }, married:true })
```

Array of Objects

- Update 3 person document with following membership attribute.

```
db.person.updateOne( { "_id": ObjectId("...") }, { $set: { membership: [
{name: "Science Congress", year: 2015},
{name: "Engineer Forum", year: 2018},
{name: "Manipur Trekker", year: 2018}
] } } )
```

[{name: "Readers Club", year: 2015}, {name: "Manipur Trekker", year: 2020}]
[{name: "Readers Club", year: 2015}, {name: "Debating Society", year: 2018}]

$elemMatch

- Find person who is a member of 'Manipur Trekker'

```
db.person.find({membership: {$elemMatch: {name: "Manipur Trekker"} } })
```

- Find person who is a member of 'Manipur Trekker' after year 2020

```
db.person.find({membership:
{$elemMatch: {name: "Manipur Trekker", year: {$gt: 2020} } }
})
```

- Find person which any membership in year 2018

```
db.person.find({membership: {$elemMatch: {year: {$eq: 2018} } } })
```

Operators revisted

$eq	equals
$ne	not equals
$gt	greater than
$gte	greater than equal
$lt	less than
$lte	less than equal
$and	logical and
$or	logical or
$not	logical not

$exists	field availability check
$type	field data type check
$in	present in given array
$nin	not present in given array
$all	array field contains all values
$elemMatch	array field contains a value
$regex	regular expression match

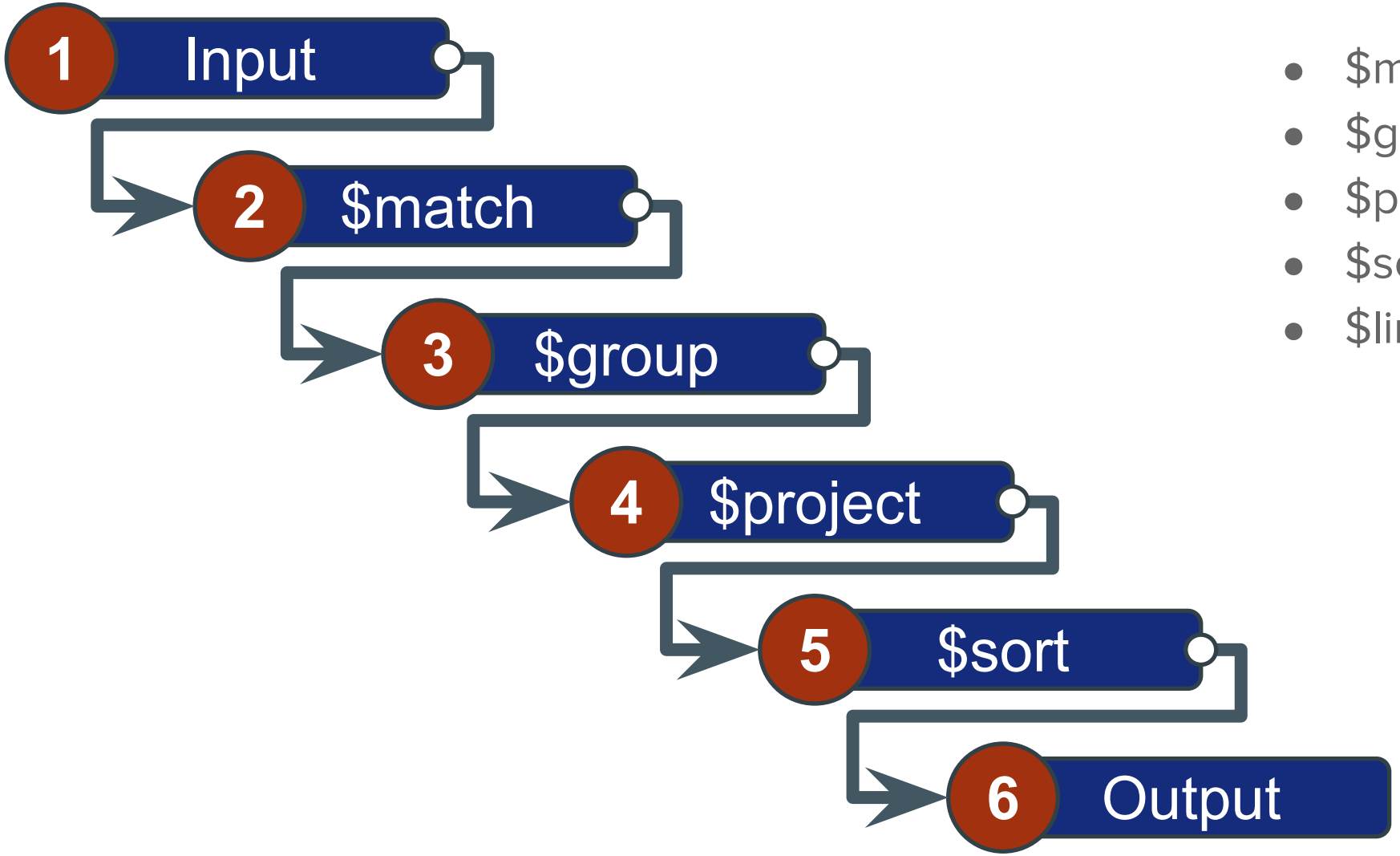
Exercise

1. Return all city document where the CountryCode = IND
2. Find all persons in address.city = Moreh
3. Find persons who joined ‘Readers Club’ before year 2017
4. Find persons who loves movies and trekking
5. Find persons born after 1995. sample date value new Date("2000-12-31")
6. Find persons whose age is between 20 and 30
7. Find persons with hobbies travel OR reading
8. Find persons who joined any club after year 2017
9. Sort persons by height
10. Find persons that do not have address field

MongoDB Topics

- Data Manipulation Language (DML)
- Data types
- Schema Validation / Constraint
- Working with Arrays
- **Aggregation Functions**
- Views
- Indexes
- Transactions

Aggregation Pipeline



- $match
- $group
- $project
- $sort
- $limit
- $lookup
- $out
- $unwind
- $addFields
- $filter

$match

world.country

240 DOCUMENTS 1 INDEXES

Documents | Aggregations | Schema | Indexes | Validation

Pipeline | $match | Explain | Export | Run | More

Stage 1 $match

```
1  /**
2   * query: The query in MQL.
3   */
4  {
5    Population: {$gt: 250000}
6  }
```

By Population size

Stage 1 $match

```
1  /**
2   * query: The query in MQL.
3   */
4  {
5    Population: {$gt: 250000},
6    Continent: {$eq: 'Europe'}
7  }
```

By Population size & Continent

$match

- By Population size

```
db.country.aggregate([
{ $match: { Population: { $gt: 250000 } } }
])
```

- By Population size & Continent

```
db.country.aggregate([
{ $match: { Population: { $gt: 250000 } ,  Continent: {$eq: "Europe"}  } }
])
```

$project

- Project or select only the Name, Population and Continent field

```
db.country.aggregate([
{ $project: { Name: 1, Population: 1, Continent: 1  } }
])
```

$sort

- Sort Ascending

```
/**
 * query: The query in MQL.
 */
{
  Continent: 1,
  Population: 1,
}
```

- Sort Descending

```
/**
 * query: The query in MQL.
 */
{
  Continent: -1,
  Population: -1
}
```

$sort

- Sort by Continent ascending and Population **ascending**

```
db.country.aggregate([
{ $sort: { Continent:1, Population: 1  } }
]);
```

- Sort by Continent ascending and Population **descending**

```
db.country.aggregate([
{ $sort: { Continent:1, Population: -1  } }
]);
```

$group

- Count grouped by Country Name

```
/**
 * query: The query in MQL.
 */
{
  _id: "$Name",
  out_count: {$count: {}}
}
```

- Count the collection

```
/**
 * query: The query in MQL.
 */
{
  _id: null,
  out_count: {$count: {}}
}
```

$group

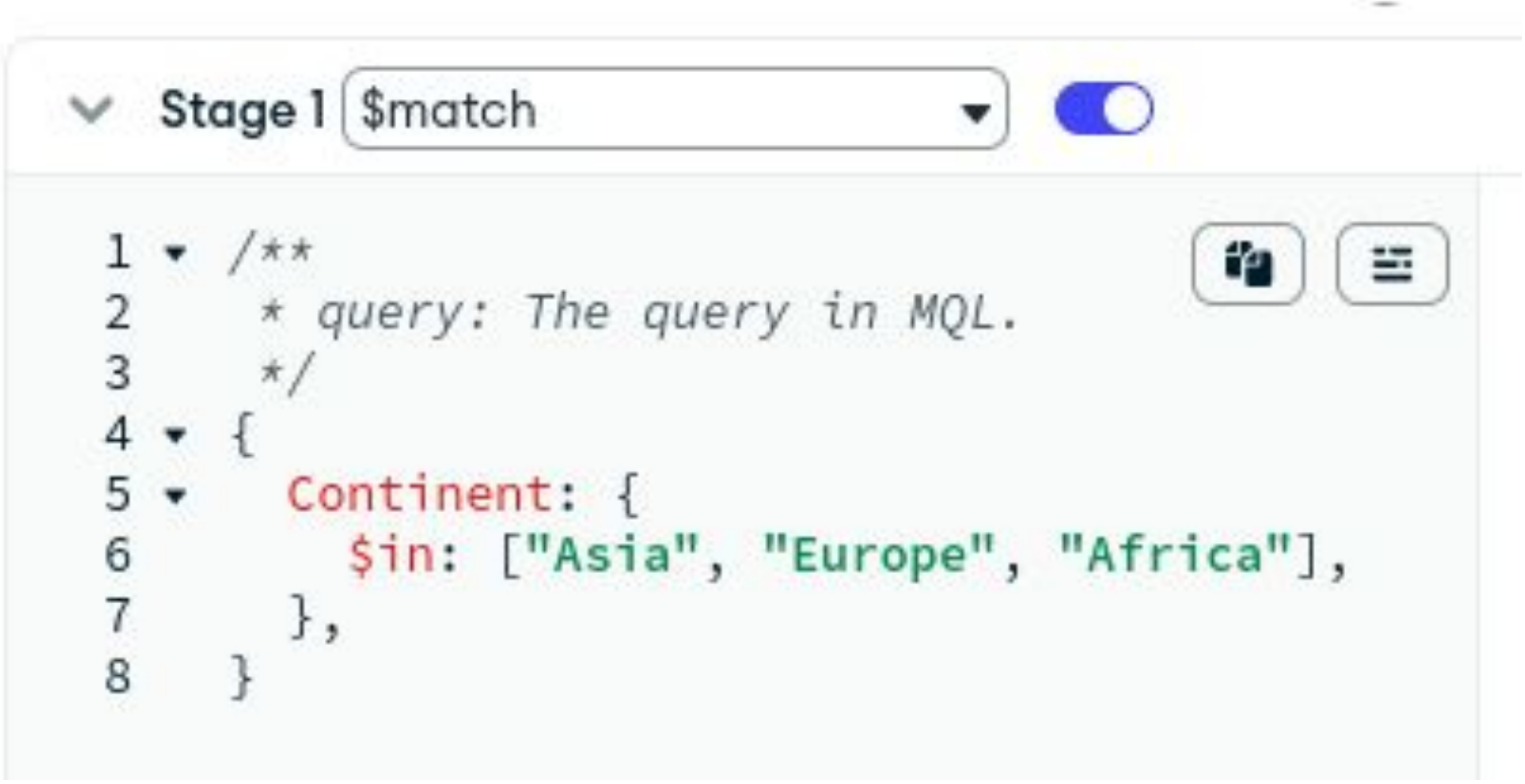
- Count grouped by Country Name

```
db.country_one.aggregate([
{ $group: { _id: "$Name", doc_count: { $count: {} } } }
] );
```

- Count of collection

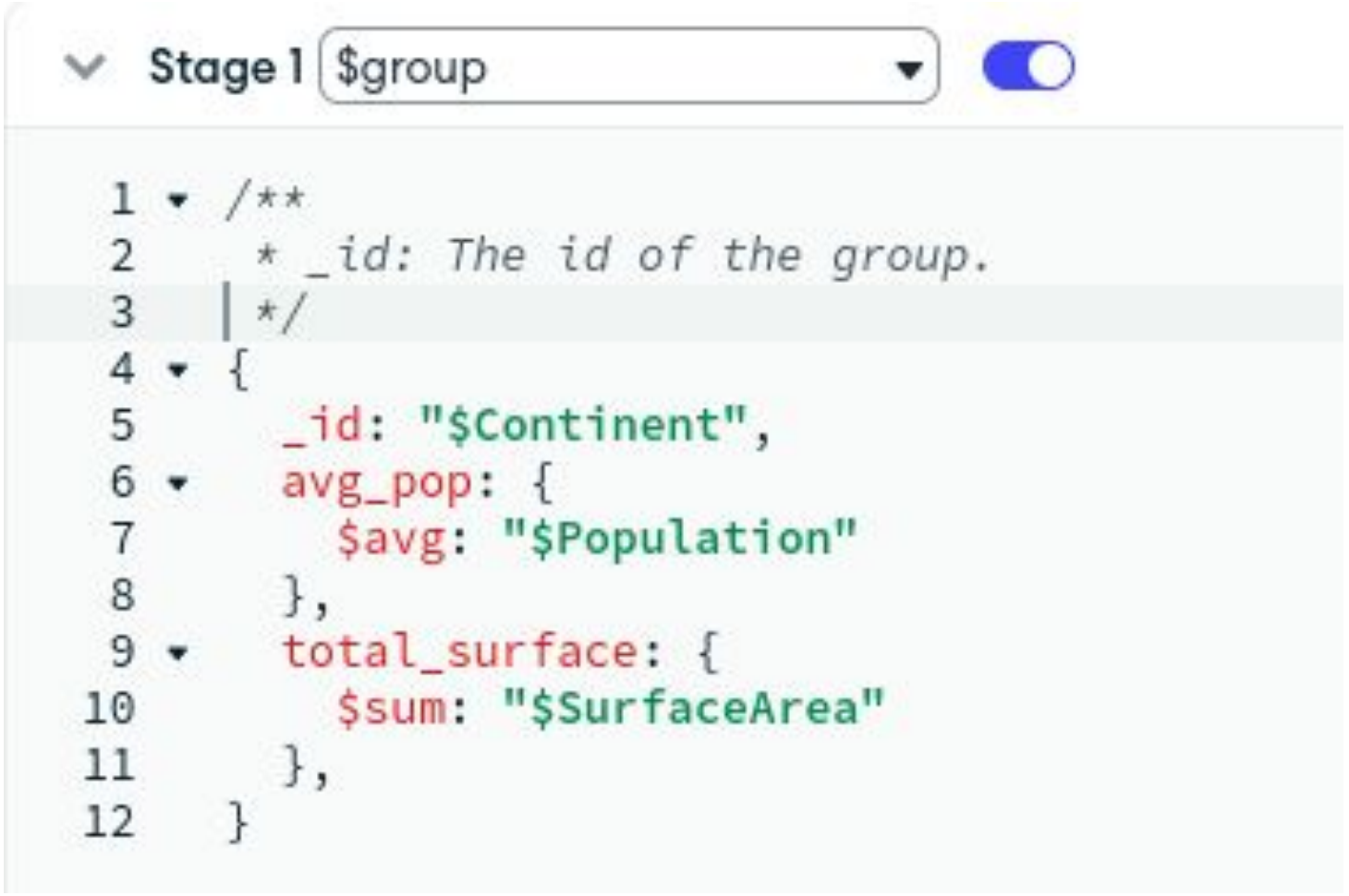
```
db.country_one.aggregate([
{ $group: { _id: null, doc_count: { $count: {} } } }
] );
```

Aggregation Pipeline Stage 1: $match



Aggregation Pipeline Stage 2: $project

Aggregation Pipeline Stage 3: $group



Aggregation Pipeline Stage 4: $sort

Aggregation Pipeline Combine All stages

```
db.country.aggregate( [
{ $match: ... },
{ $project: ... },
{ $group: ... },
{ $sort: ... }
] );
```

Step1: Merge Multiple Document

- Copy country collection into another collection

```
db.country.aggregate()

db.country.aggregate([
    {
        $project: { "_id": 1 }
    }
])
```

Step2: Merge Multiple Document

- Include the child collection

```
db.country.aggregate([
    {
        $lookup: {
          from: "city", as: "cities",
          localField: "Code", foreignField: "CountryCode",
        }
    } ,
    {
        $project: { "_id": 1, "cities._id": 0, }
    }
])
```

Step3: Merge Multiple Document

- Include another child collection

```
db.country.aggregate([
    { $lookup: { from: "city",...      } ,
    {
        $lookup: { from: "countrylanguage", as: "languages"
          localField: "Code", foreignField: "CountryCode",
        }
    },
    {    $project: { "_id": 1, "cities._id": 0, "languages._id": 0 }    }
])
```

Step4: Merge Multiple Document

- Write out the result into another document

```
db.country.aggregate([
    { $lookup: { from: "city",...     } ,
    { $lookup: { from: "countrylanguage", ...   } ,
    { $project: { "_id": 1, "cities._id": 0, "languages._id": 0 }    } ,
    { $out: "country_one" }
])
```

country_one document

```
Code: "AFG"
Name: "Afghanistan"
Continent: "Asia"
Region: "Southern and Central Asia"
SurfaceArea: 652090
IndepYear: 1919
Population: 22720000
LifeExpectancy: 45.9
GNP: 5976
LocalName: "Afganistan/Afqanestan"
GovernmentForm: "Islamic Emirate"
HeadOfState: "Mohammad Omar"
Capital: 1
Code2: "AF"
▸ cities: Array (4)
▸ languages: Array (5)
```

- Cities belonging to the country are added to the respective country document
- Similarly language spoken in the country as well

Example

- Find number of countries in each Continent

```
db.country_one.aggregate( [
    { '$group': { '_id': '$Continent', 'country_count': { '$count': {} } } }
] )
```

- Include number of cities as well

```
db.country_one.aggregate( [
    { '$group': {
            '_id': '$Continent',
            'country_count': { '$count': {} },
            'city_count': {$sum: {$size: "$cities"} }
    }  }
] )
```

Exercise

- For the continents below get the average population of the countries, the total surface area, sorted by number of cities descending
  “Europe”,”Asia”,”Africa”,”South America”

- Hint: Create a aggregation pipeline on the country_one collection
- Use match to filter by the continent names
- Then group them by Continent, and apply
  - the $avg population of the countries
  - $sum of surface area
  - city_count from previous example
- Finally use $sort to order the records by city_count

Exercise

- Count how many countries speak 2 or 3 languages
- Stages
  - $project the $size to get count of languages of each country
  - $match to list only 2,3 speaking countries
  - $group to count the countries speaking 2,3 languages

$unwind

- Find cities having population greater than 250000 in ‘Oceania’ Continent

```
db.country_one.aggregate([
  { '$match': { 'Continent': 'Oceania' } },
  { '$match': { 'cities': { '$elemMatch': { 'Population': { '$gt': 250000 } } } } }
])
```

What is the result? Is it as expected?

- Above condition is same as below

```
db.country_one.aggregate([
  { '$match': {  'Continent': 'Oceania',
                 'cities': { '$elemMatch': { 'Population': { '$gt': 250000 } } }
  } }
])
```

$unwind

- Find cities having population greater than 250000 in ‘Oceania’ Continent

```
db.country_one.aggregate([
  { '$match': { 'Continent': 'Oceania' } },
  { '$unwind': {
      'path': '$cities',
      'includeArrayIndex': 'city_index'
    } },
  { '$match': { 'cities.Population': { '$gt': 250000 } } }
])
```

Exercise

- Find the Official language of all the countries in Asia
  - $match to filter by Continent
  - $unwind to create separate document for each language
  - $match to identify official language (see field IsOfficial)
  - $project to get only the country name and the language field

MongoDB Topics

- Data Manipulation Language (DML)
- Data types
- Schema Validation / Constraint
- Working with Arrays
- Aggregation Functions
- **Views**
- Indexes
- Transactions

Create a View

- View, a READ-ONLY collection

```
db.createView("country_city_language", "country_one", [
 {  $project: {
    _id: 0,
    country: "$Name",
    cities: "$cities.Name",
    languages: "$languages.Language"
    } }
] );
```

- Query the collection: db.country_city_language.find()

Create a View

- From multiple documents (not recommended)

```
db.createView("country_city_language2", "country", [
{  $lookup: {  from: "city", as: "cities", ...  }  } ,
{  $lookup: {  from: "countrylanguage", as: "languages", ...   }  } ,
{  $project: {  _id: 0,
     country: "$Name",
     cities: "$cities.Name",
     officialLanguages: "$languages.Language"
    }
} ])
```

- Query the collection: db.country_city_language2.find()

Create a View

- Include computed values in the view & show only the documents that match certain criteria

```
db.createView("high_density_country", "country_one", [
    { $addFields: { density: { $floor: { $divide: ["$Population", "$SurfaceArea"] } } } },
    { $match: { "density": { "$gt": 10000 } } }
]);
```

- Query the collection: db.high_density_country.find()

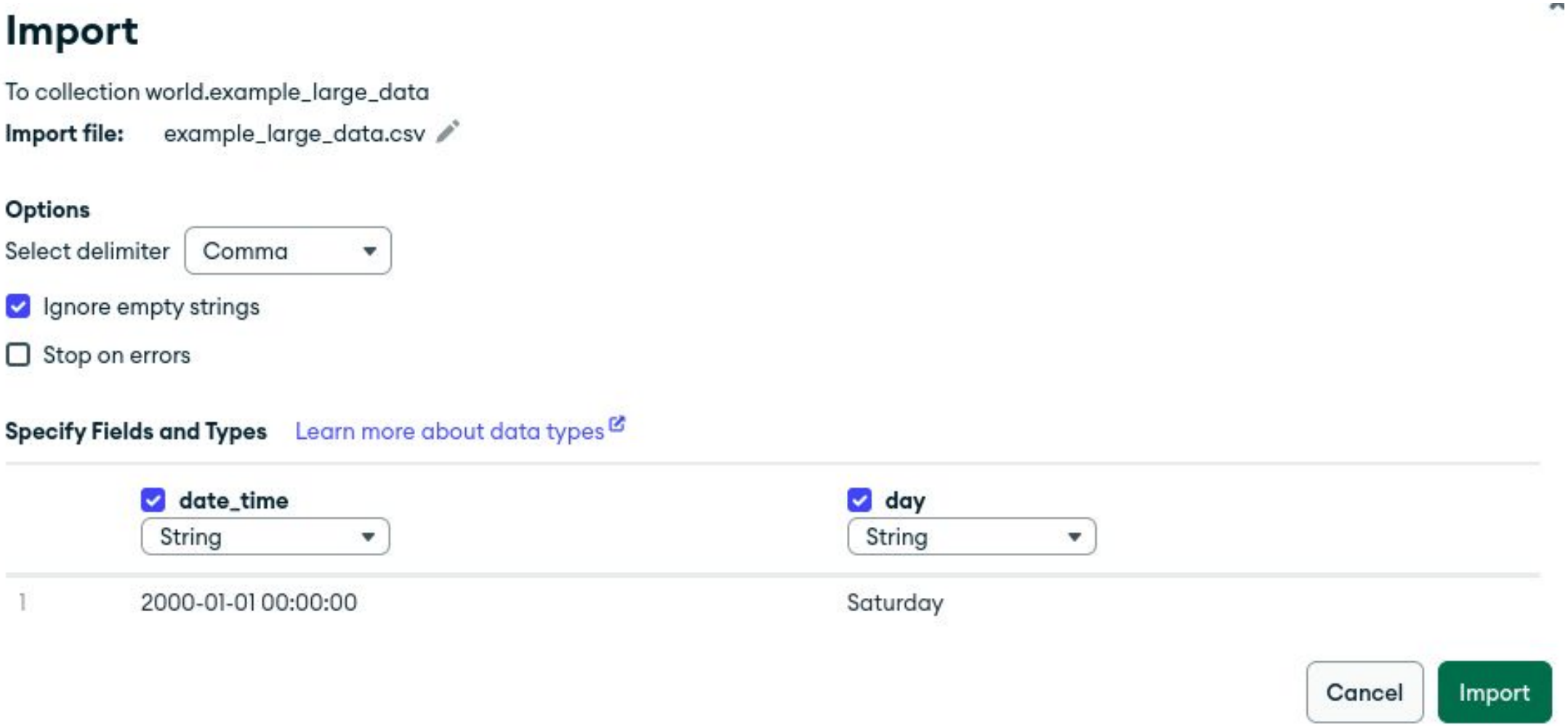
Exercise

1. Find all the countries where ‘Hindi’ is spoken?
2. Find population of Imphal or any other city in District Manipur?
3. Find the District in India with most cities?
4. Find average population of Indian District/states and sort it by highest value?
5. Find all the languages spoken in India?
6. List names of all the countries where ‘Dutch’ is spoken?
7. Which countries has maximum language spoken in it?
8. Which language is spoken in maximum countries?
9. List cities in District Manipur?

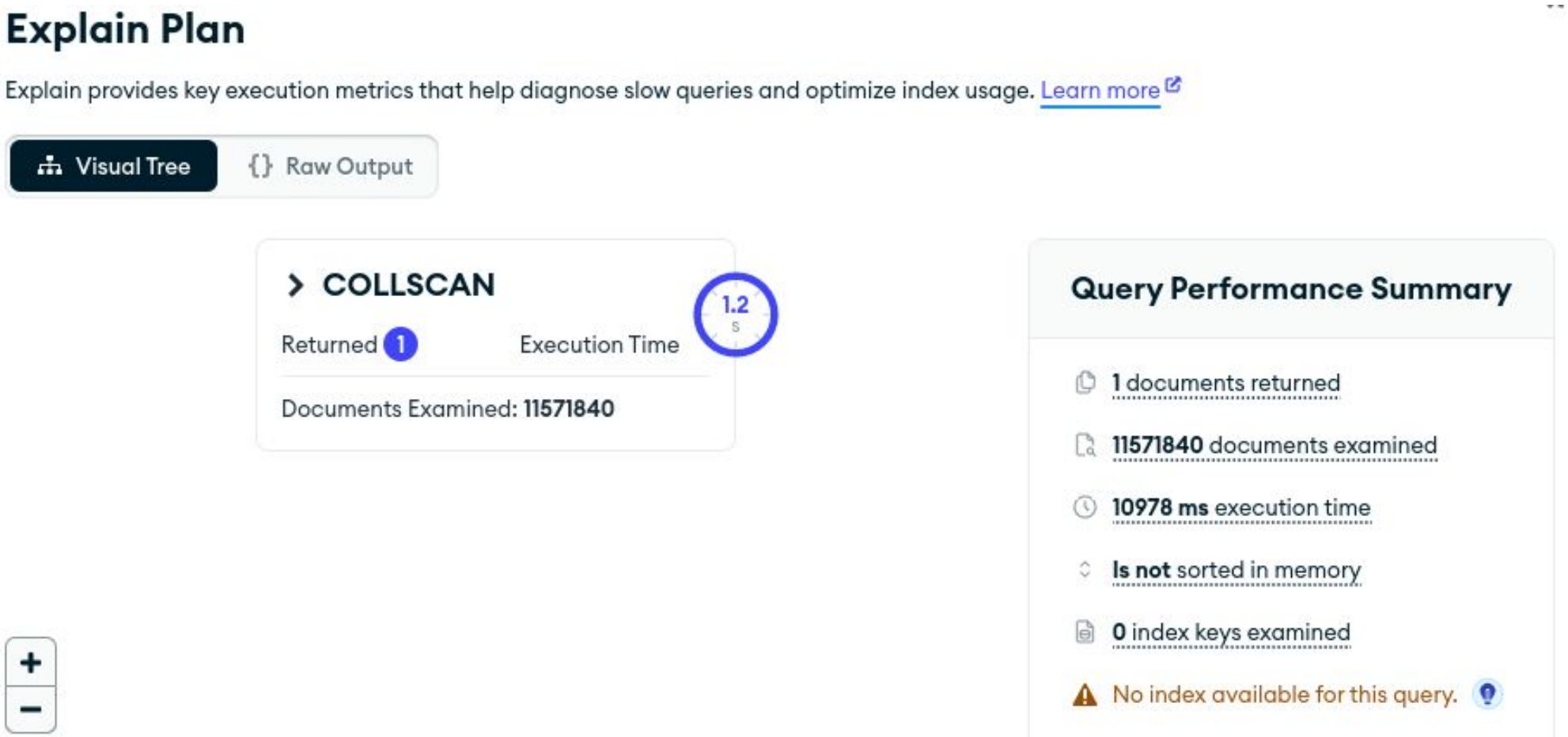
MongoDB Topics

- Data Manipulation Language (DML)
- Data types
- Schema Validation / Constraint
- Working with Arrays
- Aggregation Functions
- Views
- **Indexes**
- Transactions

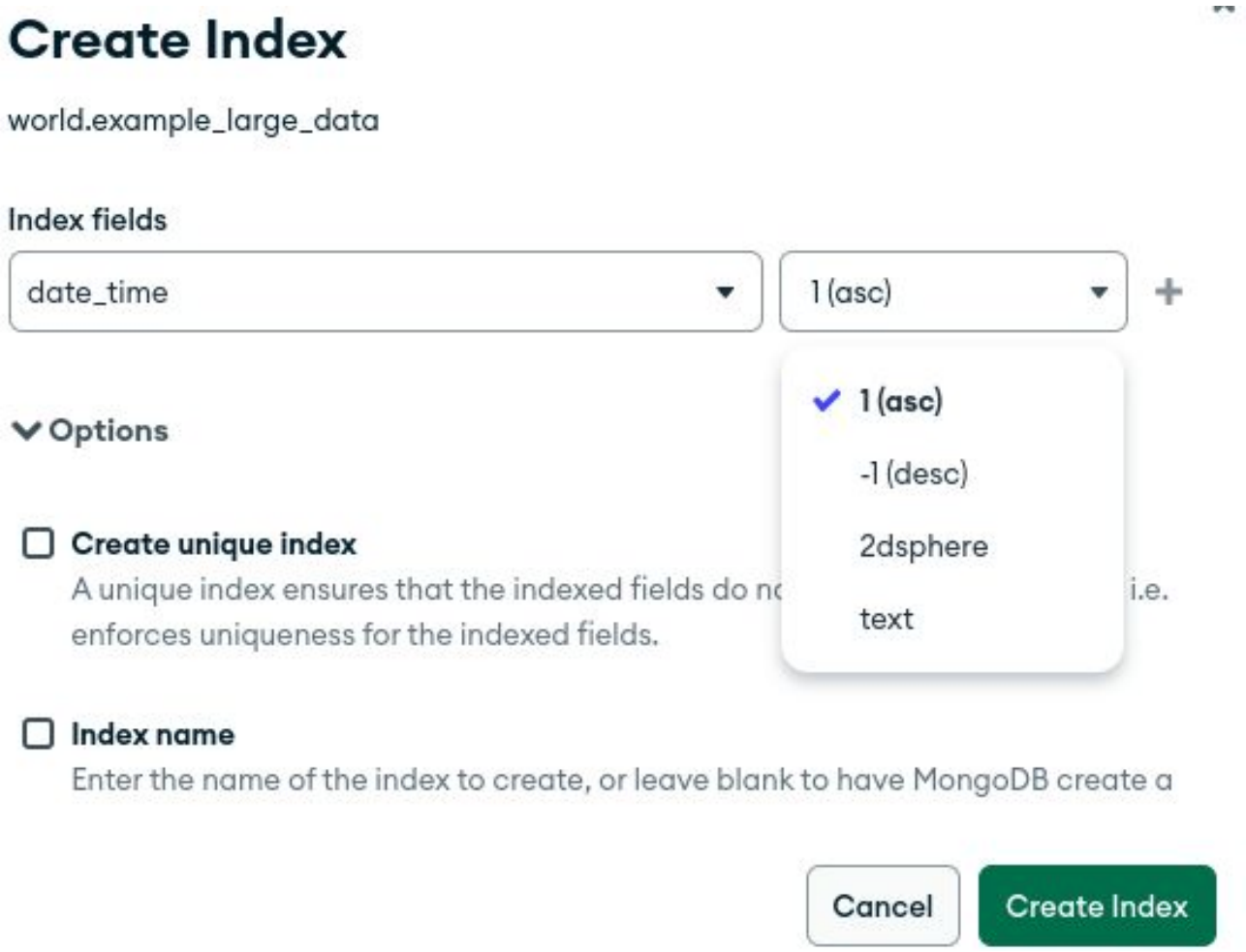
Import the large data example file



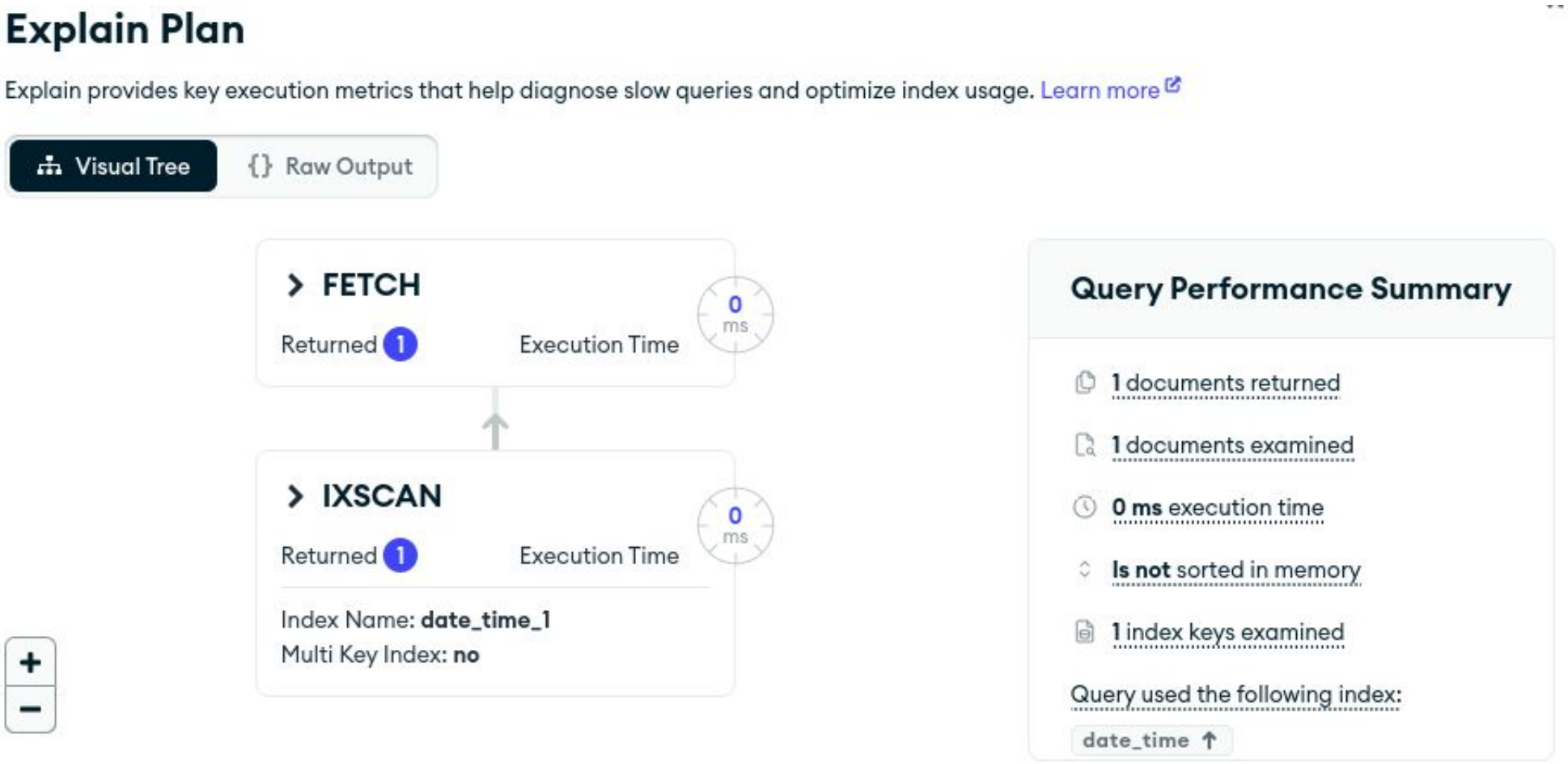
Explain Plan: Before applying index



Create index on the date_time field



After applying index

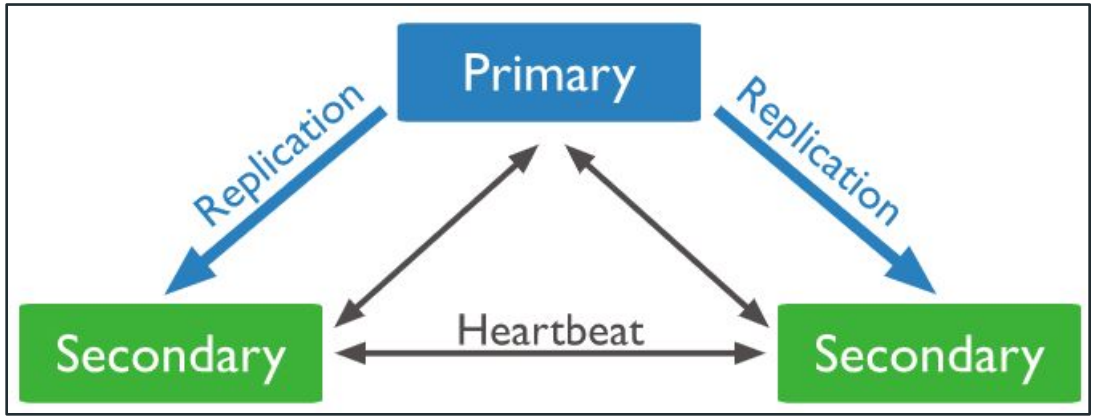


MongoDB Topics

- Data Manipulation Language (DML)
- Data types
- Schema Validation / Constraint
- Working with Arrays
- Aggregation Functions
- Views
- Indexes
- **Transactions**

MongoDB Transaction

- Mongo must run in ReplicaSet mode
- Each replicaset has a name
- Each replicaset ideally runs in a different machine




```
# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1

replication:
  replSetName: localOne
```

ReplicaSet status

```
rs.status()
{
  set: 'localOne',
  myState: 1,
  heartbeatIntervalMillis: Long("2000"),
  votingMembersCount: 1,
  members: [ {
      name: '127.0.0.1:27017',
      health: 1,
      stateStr: 'PRIMARY',
   } ],
```

myState values

0: Starting up
1: Primary
2: Secondary
3: Recovering
4: Fatal error
5: Starting up (phase 2)
6: Unknown state
7: Arbiter
8: Down
9: Rollback

Transaction Rollback

```
var r_session = db.getMongo().startSession();

var worldDb = r_session.getDatabase("world");
var personCollection = worldDb.getCollection("person");

r_session.startTransaction();
personCollection.insertOne({first_name: "Transaction", last_name: "Rollback"});
r_session.abortTransaction();

r_session.endSession();
```

unlike MySQL,
SAVEPOINT is NOT available

Transaction Commit

```
var c_session = db.getMongo().startSession();

var worldDb = c_session.getDatabase("world");
var personCollection = worldDb.getCollection("person");

c_session.startTransaction();
personCollection.insertOne({first_name: "Transaction", last_name: "Commit"});
c_session.commitTransaction();

c_session.endSession();
```

unlike MySQL,
SAVEPOINT is NOT available

DEMO APPLICATION

Fix the Mongo Query and make the application work!