MONGO DB: REPORT 2

ADD UPDATE & DELETE:

A Quick Grasp from Report 1-

* MongoDB is a NoSQL Database
* Each Row can be associated with Documents
* Documents can be represented as JSON
* Each table can be called as Collection

//Importing Data in MongoDB:

MongoDB Compass can import data into a collection from either a **JSON** or **CSV** file.

**Limitations**

* Importing data into a collection is not permitted in **MongoDB Compass Readonly Edition**.
* Importing data is not available if you are connected to a [Data Lake.](https://www.mongodb.com/docs/atlas/data-lake/)

Importing Using JSON ->

When importing data from a **JSON** file, you can format your data as:

* Newline-delimited documents, or
* Comma-separated documents in an array

EXAMPLE ->

The following newline-delimited .json file is formatted correctly:

{ "type": "home", "number": "212-555-1234" }

{ "type": "cell", "number": "646-555-4567" }

{ "type": "office", "number": "202-555-0182"}

The following comma-separated .json array file is also formatted correctly:

[{ "type": "home", "number": "212-555-1234" }, { "type": "cell", "number": "646-555-4567" }, { "type": "office", "number": "202-555-0182"}]

Compass ignores line breaks in JSON arrays.

Compass automatically generates [ObjectIDs](https://www.mongodb.com/docs/manual/reference/method/ObjectId/" \t "_self) for these objects on import since no ObjectIDs were specified in the initial JSON.

### Procedure

To import your formatted data into a collection:

**1 .**

#### **Connect to the deployment containing the**[**collection**](https://www.mongodb.com/docs/compass/current/collections/)**you wish to import data into.**

To learn how to connect to a deployment, see [Connect to MongoDB.](https://www.mongodb.com/docs/compass/current/connect/#std-label-connect-run-compass)

**2**

#### **Navigate to your target collection.**

You can either select the collection from the [Collections](https://www.mongodb.com/docs/compass/current/collections/) tab or click the collection in the left-hand pane.

**3**

#### **Click the**Add Data**dropdown and select**Import JSON or CSV file**.**

**4**

#### **Select the appropriate file type.**

Select either a JSON or CSV file to import and click **Select**.

If you are importing a CSV file, you may specify fields to import and the types of those fields under **Specify Fields and Types**.

To exclude a field from a CSV file you are importing, uncheck the checkbox next to that field name. To select a type for a field, use the dropdown menu below that field name.

**5**

#### **Configure import options.**

Under **Options**, configure the import options for your use case.

If you are importing a CSV file, you may select how your data is delimited.

For both JSON and CSV file imports, you can toggle **Ignore empty strings** and **Stop on errors**:

* If checked, **Ignore empty strings** drops fields with empty string values from your imported documents. The document is still imported with all other fields.
* If checked, **Stop on errors** prevents any data from being imported in the event of an error. If unchecked, data is inserted until an error is encountered and successful inserts are not rolled back. The import operation will not continue after encountering an error in either case.

**6**

#### **Click**Import**.**

A progress bar displays the status of the import. If an error occurs during import, the progress bar turns red and an error message appears in the dialog. To see all errors, click **View Log**.

After successful import, the dialog closes and Compass displays the collection page containing the newly imported documents.

Importing Using CSV ->

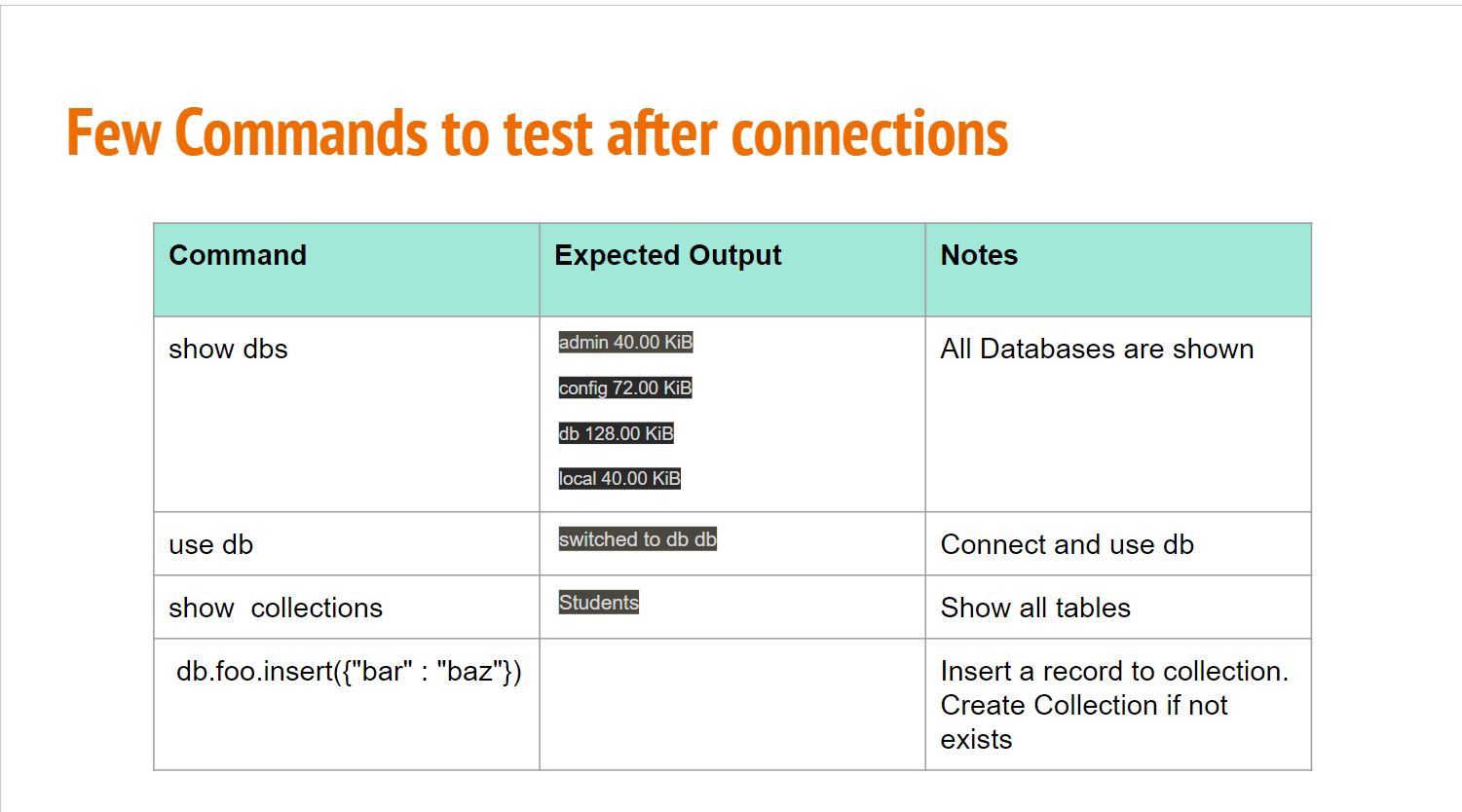
When importing data from a **CSV** file, the first line of the file must be a comma-separated list of your document field names. Subsequent lines in the file must be comma-separated field values in the order corresponding with the field order in the first line.

## EXAMPLE

The following .csv file imports three documents:

|  |
| --- |
| name,age,fav\_color,pet |
| Jeff,25,green,Bongo |
| Alice,20,purple,Hazel |
| Tim,32,red,Lassie |

MongoDB Compass automatically generates [ObjectIDs](https://www.mongodb.com/docs/manual/reference/method/ObjectId/" \t "_self) for these objects on import since no ObjectIDs were specified in the initial CSV file.

* After importing the file it must be able to be used in mongo compass after connection with the Mo



DOCUMENTS, COLLECTIONS & DATABASE:

DOCUMENTS->

* Database is the main important of the MongoDB.
* The ordered set of keys with associated values.
* The representation of documents varies by programming language, but most languages has data structures that is a natural fit, such as map, hash, or dictionary.

{“greeting”:“Hello,World!!!”}

COLLECTIONS->

* A Collection is a group of document.
* If the document is the MongoDB analog of a row in relational database, then a collection can be thought of as the analog to the table.

DATABASE->

* MongoDB groups collections into databases.
* A single instance of MongoDB can host several databases, each grouping together zero or more collections.
* A database has its own permissions, and each database is stored in separate files on disk.
* A good rule of thumb is to store all data for a single application in the same database.

DATATYPE->

## **Date**

mongosh provides various methods to return the date, either as a string or as a Date object:

* Date() method which returns the current date as a string.
* new Date() constructor which returns a Date object using the ISODate() wrapper.
* ISODate() constructor which returns a Date object using the ISODate() wrapper.

## **Int32**

If a number can be converted to a 32-bit integer, mongosh will store it as Int32. If not, mongosh defaults to storing the number as a Double. Numerical values that are stored as Int32 in mongosh would have been stored by default as Double in the mongo shell.

The [Int32()](http://mongodb.github.io/node-mongodb-native/3.6/api/Int32.html) constructor can be used to explicitly specify 32-bit integers.

db.types.insertOne(

{

"\_id": 1,

"value": Int32("1"),

"expectedType": "Int32"

}

)

## **Decimal128**

[Decimal128()](http://mongodb.github.io/node-mongodb-native/3.6/api/Decimal128.html) values are 128-bit decimal-based floating-point numbers that emulate decimal rounding with exact precision.

This functionality is intended for applications that handle [monetary data](https://www.mongodb.com/docs/manual/tutorial/model-monetary-data/), such as financial, tax, and scientific computations.

The Decimal128 [BSON type](https://www.mongodb.com/docs/manual/reference/bson-types/) uses the IEEE 754 decimal128 floating-point numbering format which supports 34 decimal digits (i.e. significant digits) and an exponent range of −6143 to +6144.

db.types.insertOne(

{

"\_id": 5,

"value": Decimal128("1"),

"expectedType": "Decimal128"

}

)

## **Timestamp**

MongoDB uses a [BSON Timestamp](https://www.mongodb.com/docs/manual/reference/bson-types/#timestamps) internally in the [oplog](https://www.mongodb.com/docs/manual/reference/glossary/" \l "std-term-oplog" \t "_self). The Timestamp type works similarly to the Java Timestamp type. Use the [Date](https://www.mongodb.com/docs/mongodb-shell/reference/data-types/#std-label-mongo-shell-date-type) type for operations involving dates.

A Timestamp signature has two optional parameters.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Timestamp( { **"t"**: <integer>, **"i"**: <integer> } ) **Long** The [Long()](http://mongodb.github.io/node-mongodb-native/3.6/api/Long.html) constructor can be used to explicitly specify a 64-bit integer.   |  | | --- | | db.types.insertOne( | | { | | **"\_id"**: 3, | | **"value"**: Long(**"1"**), | | **"expectedType"**: **"Long"** | | } | | ) | |