The best way to fully grasp the basics of MongoDB is to compare the essential structures with their counterparts in the normal SQL universe. There are three basic comparisons to make that will give you the essentials to get rolling. Alongside that, we will also show you the basic commands to use in your MongoDB shell (which is another name for the terminal program we are running to connect to our DBs).

Database Type:	**SQL**	**Mongo**
Database	Schema	Database (db)
Collection of related records	Tables	Collections
Each one record in the collection of records	Row / Record	Document

MySQL Database Schema == MongoDB Database (db)

No surprises here; the database is still the unit we use to hold an entire project's data. MongoDB is crazy, but not crazy enough to not need to use the term database! Note that from our Mongo shell, we have access to all the databases stored on our Mongo server.

Show all databases available on our current MongoDB server	Example: **show dbs**
Show current database selected	Example: **db**
Change to another database Note: If the database you're trying to switch to does not exist, Mongo shell will create a new database and switch to it.	Pattern: **use DB_NAME** Example: **use message_board_db**
Delete database Note: db.dropDatabase() will delete the current database in use.	Example: **use message_board_db** **db.dropDatabase()**

SQL: Tables == MongoDB: Collections

A SQL database is comprised of tables. Tables contain groups of similarly-structured pieces of data. This shouldn't be new to you, but it is important to take a step back and review what you've learned through a more theoretical lens. MongoDB databases are comprised of collections. A collection is the MongoDB analog to a SQL table.

View all collections in a MongoDB	Example: **show collections**

Create a new collection in the current database	Pattern:
	db.createCollection("COLLECTION_NAME")
	
	Example:
	db.createCollection("ninjas")

Destroy a collection	Pattern:
	db.COLLECTION_NAME.drop()
	
	Example:
	db.ninjas.drop()

There's not much to say about collections right now. That's because **collections are really just in place to corral individual records**. There is no structure to a collection, which is part of why MongoDB is powerful, but also part of why a NoSQL database is not a one-size-fits-all solution. This is very different than tables in a SQL database; **tables are normally where we define all the structure of a particular facet of our database in a SQL database**. Column names and their data types are very important in the world of relational databases. In NoSQL land, they are much less important, as you will soon find out.

SQL: Row / Record == MongoDB: Document (JSON object)

This is it. This is why we use MongoDB. SQL table rows have the rigid structure: every entry in a database (i.e. every row) has the same fields. This is not the case in MongoDB. Each document is a JSON object and is able to have any number of key-value pairs you so desire. And just like in regular Javascript objects, we can add key-value pairs to objects on the fly. When you use a NoSQL database, you gain speed but lose rigidity; you trade structure for flexibility. This should make sense. Remember, people use Mongo for its speed. If we can't write to objects on the fly, we might slow down our transactions and that would be counterproductive.

Technically, we've lied. MongoDB doesn't use JSON objects to store your data. They use what's called BSON (Binary JSON). BSON is friendlier to store (due to it being binary; don't stress about this) and can support a few more things (like dates, which are not part of JSON, but crucial for databases). But for all intents and purposes, MongoDB documents are to be treated and used as if they were ordinary JSON. Trust us! The next tab will delve into working with documents, so let's get moving!