Models

Models are fancy constructors compiled from `Schema` definitions. An instance of a model is called a document. Models are responsible for creating and reading documents from the underlying MongoDB database.

Compiling your first model

When you call `mongoose.model()` on a schema, Mongoose *compiles* a model for you.

```
const schema = new mongoose.Schema({ name: 'string', size: 'string' });
const Tank = mongoose.model('Tank', schema);
```

The first argument is the *singular* name of the collection your model is for. **Mongoose automatically looks for the plural, lowercased version of your model name.** Thus, for the example above, the model Tank is for the **tanks** collection in the database.

Note: The `.model()` function makes a copy of `schema`. Make sure that you've added everything you want to `schema`, including hooks, before calling `.model()`!

Constructing Documents

An instance of a model is called a document. Creating them and saving to the database is easy.

```
const Tank = mongoose.model('Tank', yourSchema);

const small = new Tank({ size: 'small' });
small.save(function (err) {
  if (err) return handleError(err);
  // saved!
});

// or

Tank.create({ size: 'small' }, function (err, small) {
  if (err) return handleError(err);
  // saved!
});

// or, for inserting large batches of documents
Tank.insertMany([{ size: 'small' }], function(err) {
```

```
});
```

Note that no tanks will be created/removed until the connection your model uses is open. Every model has an associated connection. When you use `mongoose.model()`, your model will use the default mongoose connection.

```
mongoose.connect('mongodb://localhost/gettingstarted');
```

If you create a custom connection, use that connection's `model()` function instead.

```
const connection = mongoose.createConnection('mongodb://localhost:27017/test');
const Tank = connection.model('Tank', yourSchema);
```

Querying

Finding documents is easy with Mongoose, which supports the rich query syntax of MongoDB. Documents can be retrieved using a `model`'s find, findById, findOne, or where static methods.

```
Tank.find({ size: 'small' }).where('createdDate').gt(oneYearAgo).exec(callback);
```

See the chapter on queries for more details on how to use the Query api.

Deleting

Models have static `deleteOne()` and `deleteMany()` functions for removing all documents matching the given `filter`.

```
Tank.deleteOne({ size: 'large' }, function (err) {
  if (err) return handleError(err);
  // deleted at most one tank document
});
```

Updating

Each `model` has its own `update` method for modifying documents in the database without returning them to your application. See the API docs for more detail.

```
Tank.updateOne({ size: 'large' }, { name: 'T-90' }, function(err, res) {
  // Updated at most one doc, `res.modifiedCount` contains the number
  // of docs that MongoDB updated
});
```

If you want to update a single document in the db and return it to your application, use findOneAndUpdate instead.

Change Streams

New in MongoDB 3.6.0 and Mongoose 5.0.0

Change streams provide a way for you to listen to all inserts and updates going through your MongoDB database. Note that change streams do **not** work unless you're connected to a MongoDB replica set.

```
async function run() {
  // Create a new mongoose model
  const personSchema = new mongoose.Schema({
    name: String
  });
  const Person = mongoose.model('Person', personSchema);

  // Create a change stream. The 'change' event gets emitted when there's a
  // change in the database
  Person.watch().
    on('change', data => console.log(new Date(), data));

  // Insert a doc, will trigger the change stream handler above
  console.log(new Date(), 'Inserting doc');
  await Person.create({ name: 'Axl Rose' });
}
```

The output from the above async function will look like what you see below.

```
2018-05-11T15:05:35.467Z 'Inserting doc'
2018-05-11T15:05:35.487Z 'Inserted doc'
2018-05-11T15:05:35.491Z { _id: { _data: ... },
  operationType: 'insert',
  fullDocument: { _id: 5af5b13fe526027666c6bf83, name: 'Axl Rose', __v: 0 },
  ns: { db: 'test', coll: 'Person' },
  documentKey: { _id: 5af5b13fe526027666c6bf83 } }
```

You can read more about change streams in mongoose in this blog post.

Yet more

The API docs cover many additional methods available like count, mapReduce, aggregate, and more.

Next Up

Now that we've covered `Models`, let's take a look at Documents.