In hand salry,

30% hike,

Notice period, backoff option, how many days you are in this project,

SQL and Angular imp

ORM: imp

Example :

One customer has ordered Product A, B and C today

Tomorrow the same customer orders Product A and B again.

How will you create optimized tables?

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Syntax** | **Example** | **RDBMS Equivalent** |
| Equality | {<key>:<value>} | db.mycol.find({"by":"tutorials point"}).pretty() | where by = 'tutorials point' |
| Less Than | {<key>:{$lt:<value>}} | db.mycol.find({"likes":{$lt:50}}).pretty() | where likes < 50 |
| Less Than Equals | {<key>:{$lte:<value>}} | db.mycol.find({"likes":{$lte:50}}).pretty() | where likes <= 50 |
| Greater Than | {<key>:{$gt:<value>}} | db.mycol.find({"likes":{$gt:50}}).pretty() | where likes > 50 |
| Greater Than Equals | {<key>:{$gte:<value>}} | db.mycol.find({"likes":{$gte:50}}).pretty() | where likes >= 50 |
| Not Equals | {<key>:{$ne:<value>}} | db.mycol.find({"likes":{$ne:50}}).pretty() | where likes != 50 |

db.products.find( { qty: { $gt: 25 } } )

Object Relational Mapping (ORM) is the process of mapping between objects and relational database systems. So it acts like an interface between two systems hiding details about an underlying mechanism. In this versatile world, database systems are also not 100% alike—the way of accessing data differs. When it comes to migration between databases, ORM could be an option if you want to avoid wasting time and effort. Here are some advantages of ORM over traditional query approach:

npm **install** *--save sequelize*

npm **install** *--save pg pg-hstore*

npm **install** *--save ejs*

**var** sequelize = **new** **Sequelize**('postgres://username:password@localhost:5432/db\_name');

Models are the objects which represent tables in a database. They are the heart of ORM and we can define them with sequelize.define. Our User model looks like this:

var User = sequelize.**define**('user', {

firstName: {

**type**: DataTypes.STRING,

allowNull: false,

unique: 'compositeIndex'

},

lastName: {

**type**: DataTypes.STRING,

unique: 'compositeIndex'

},

.........

.........

.........

dateJoined: {

**type**: DataTypes.DATE,

defaultValue: DataTypes.NOW

}

}, {

getterMethods : {

address: function() { **return** this.state + ', ' + this.country }

},

setterMethods : {

address: function(value) {

var names = value.split(', ');

this.setDataValue('country', names**[**0**]**);

this.setDataValue('state', names**[**1**]**);

},

}

});

### Example

**userId**: {type: Sequelize.STRING, unique: true},

**fullName**: { type: Sequelize.STRING, unique: 'compositeIndex'},

**dob**: { type: Sequelize.DATE, unique: 'compositeIndex'}

### Getter and setter methods

In Sequelize, we can define pseudo properties on a model. These properties are not an actual part of the database schema, they are just for the models. In the example above, "address" is a pseudo-property, and its value is initialized through getter and setter methods. When state and country are fetched from db, Sequelize merges them with a comma to populate the "address" property (getterMethods). Similarly, when we set the address, it gets split into state and country (setterMethods). Thus, the "address" seems like a column in db but actually it's not.

## Preparing sample data

There are different approaches to create a new entry in DB. The first approach is to build a non persistent object and then call save() to persist the data.

**var** newUser = user.build({

firstName: 'John',

lastName: 'Doe',

age: 28,

country: 'US',

state: 'Indiana',

email: 'johndoe@example.com'

});

newUser.save().then(**function**() {

*// Do stuffs after data persists*

})

**Sequelize**

[Sequelize](http://docs.sequelizejs.com/en/v3/) is a promise-based ORM for Node.js and io.js. It supports PostgreSQL, MySQL, MariaDB, SQLite and MSSQL and features transaction support, relations, read replication and more. Starting from 4.0.0 Sequelize will only support Node v4 and above to use ES6 features.

**Example**:

|  |
| --- |
| **var** Sequelize = require('sequelize');  **var** sequelize = **new** Sequelize('database', 'username', 'password');    **var** User = sequelize.define('user', {  username: Sequelize.STRING,  birthday: Sequelize.DATE  });    sequelize.sync().then(**function**() {  **return** User.create({  username: 'janedoe',  birthday: **new** Date(1980, 6, 20)  });  }).then(**function**(jane) {  console.log(jane.get({  plain: **true**  }));  }); |