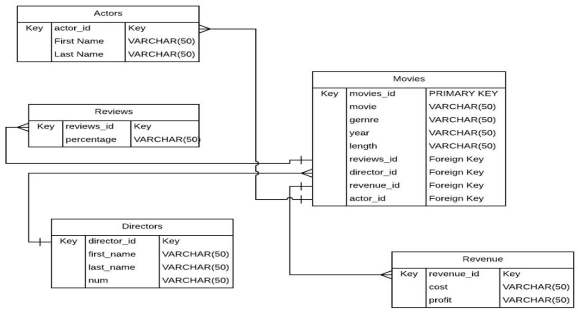
George Rooney, Patrick Koppen, Zachary Delong, Connor Aguilar
Mr. Clark
Software Application Development

Database Diagram



Database Code:

```
--login with: mysql -u root -p

DROP DATABASE moviedata;

CREATE DATABASE moviedata;

USE moviedata;
```

https://www.lucidchart.com/invitations/accept/5c67116c-89e6-4611-9cd4-a6d2861448e1

```
CREATE TABLE actors (
      actor_id SMALLINT NOT NULL,
      first_name VARCHAR(50) NOT NULL,
      last_name VARCHAR(50) NOT NULL,
      movie_id SMALLINT NOT NULL,
      last_update TIMESTAMP NOT NULL,
      PRIMARY KEY (actor_id)
);
```

```
INSERT INTO actors (movie_id, actor_id, first_name, last_name, last_update)
VALUES (1, 1, 'Mark', 'Hamill', NOW());

INSERT INTO actors (movie_id, actor_id, first_name, last_name, last_update)
VALUES (1, 2, 'Harrison', 'Ford', NOW());


INSERT INTO actors (movie_id, actor_id, first_name, last_name, last_update)
VALUES (3, 10, 'Seth', 'Rogen', NOW());

INSERT INTO actors (movie_id, actor_id, first_name, last_name, last_update)
VALUES (3, 11, 'James', 'Franco', NOW());


INSERT INTO actors (movie_id, actor_id, first_name, last_name, last_update)
VALUES (2, 3, 'Sam', 'Neill', NOW());

INSERT INTO actors (movie_id, actor_id, first_name, last_name, last_update)
VALUES (2, 4, 'Laura', 'Dern', NOW());

INSERT INTO actors (movie_id, actor_id, first_name, last_name, last_update)
VALUES (2, 5, 'Jeff', 'Goldblum', NOW());


INSERT INTO actors (movie_id, actor_id, first_name, last_name, last_update)
VALUES (4, 6, 'Charlie', 'Hunnam', NOW());

INSERT INTO actors (movie_id, actor_id, first_name, last_name, last_update)
VALUES (4, 7, 'Idris', 'Elba', NOW());

INSERT INTO actors (movie_id, actor_id, first_name, last_name, last_update)
VALUES (4, 8, 'Rinko', 'Kikuchi', NOW());

INSERT INTO actors (movie_id, actor_id, first_name, last_name, last_update)
VALUES (4, 9, 'Charlie', 'Day', NOW());


CREATE TABLE reviews(
      Movie_id SMALLINT NOT NULL,
      reviews_id SMALLINT NOT NULL,
      metacritic VARCHAR(50) NOT NULL,
      rotten VARCHAR(50) NOT NULL,
      last_update TIMESTAMP NOT NULL,
      PRIMARY KEY (reviews_id)
);
INSERT INTO reviews(movie_id, reviews_id, metacritic, rotten, last_update)
VALUES (4, 1,'Metacritic: 64','Rotten Tomatoes: 71%', NOW());
```

```
INSERT INTO reviews(movie_id, reviews_id, metacritic, rotten, last_update)
VALUES (3, 2, 'Metacritic: 64','Rotten Tomatoes: 68%', NOW());

INSERT INTO reviews(movie_id, reviews_id, metacritic, rotten, last_update)
VALUES (1, 3,'Metacritic: 92','Rotten Tomatoes: 93%', NOW());

INSERT INTO reviews(movie_id, reviews_id, metacritic, rotten, last_update)
VALUES (2, 4,'Metacritic: 68','Rotten Tomatoes: 93%', NOW());


CREATE TABLE directors (
      director_id SMALLINT NOT NULL,
      first_name VARCHAR(50) NOT NULL,
      last_name VARCHAR(50) NOT NULL,
      movie_id SMALLINT NOT NULL,
      last_update TIMESTAMP NOT NULL,
      PRIMARY KEY (director_id)
);
INSERT INTO directors(movie_id, director_id, first_name, last_name,
last_update)
VALUES(1, 1, 'George', 'Lucas', NOW());

INSERT INTO directors(movie_id, director_id, first_name, last_name,
last_update)
VALUES(2, 2, 'Stephen', 'Spielberg', NOW());

INSERT INTO directors(movie_id, director_id, first_name, last_name,
last_update)
VALUES(4, 3, 'Guillermo', 'Del Torro', NOW());

INSERT INTO directors(movie_id, director_id, first_name, last_name,
last_update)
VALUES(3, 4, 'David' , 'Green', NOW());

CREATE TABLE revenue (
      revenue_id SMALLINT NOT NULL,
      revenue VARCHAR(50) NOT NULL,
      cost VARCHAR(50) NOT NULL,
      profit VARCHAR(50) NOT NULL,
      last_update TIMESTAMP NOT NULL,
      movie_id SMALLINT NOT NULL,
      PRIMARY KEY (revenue_id)
      );

INSERT INTO revenue(movie_id, revenue_id, revenue, cost, profit, last_update)
VALUES(4, 1, '411 Million Box Office', 'Budget: 180 Million', 'Total Profit:
231 Million', NOW());
```

```
INSERT INTO revenue(movie_id, revenue_id, revenue, cost, profit, last_update)
VALUES(1, 2, ' 775.4 Million Box Office', 'Budget: 11 Million', 'Total Profit:
764.4 Million', NOW());

INSERT INTO revenue(movie_id, revenue_id, revenue, cost, profit, last_update)
VALUES(3, 3, ' 101.6 Million Box Office', 'Budget: 27 Million', 'Total Profit:
74.6 Million', NOW());

INSERT INTO revenue(movie_id, revenue_id, revenue, cost, profit, last_update)
VALUES(2, 4, ' 1.029 Billion Box Office', 'Budget: 63 Million', 'Total Profit:
966 Million', NOW());



CREATE TABLE movies (
      movies_id SMALLINT NOT NULL,
      movies VARCHAR(50) NOT NULL,
      genre VARCHAR(50) NOT NULL,
      year VARCHAR(50) NOT NULL,
      length VARCHAR(50) NOT NULL,
      revenue_id SMALLINT NOT NULL,
      actor_id SMALLINT NOT NULL,
      director_id SMALLINT NOT NULL,
      reviews_id SMALLINT NOT NULL,
      last_update TIMESTAMP NOT NULL,
      PRIMARY KEY (movies_id),
      FOREIGN KEY (reviews_id) REFERENCES reviews(reviews_id),
      FOREIGN KEY (director_id) REFERENCES directors(director_id),
      FOREIGN KEY (revenue_id) REFERENCES revenue(revenue_id),
      FOREIGN KEY (actor_id) REFERENCES actors(actor_id)

);


INSERT INTO movies(movies_id, movies, genre, year, length, revenue_id,
actor_id, director_id, reviews_id, last_update)
VALUES (1, 'A New Hope', 'Science Fiction', '1977' , '2h 1m', 2, 1, 1, 3,
NOW());

INSERT INTO movies(movies_id, movies, genre, year, length, revenue_id,
actor_id, director_id, reviews_id, last_update)
VALUES (2, 'Jurassic Park', 'Science Fiction', '1993', '2h 7m', 4, 3, 2, 4,
NOW());

INSERT INTO movies(movies_id, movies, genre, year, length, revenue_id,
actor_id, director_id, reviews_id, last_update)
VALUES (3, 'Pineapple Express', 'Comedy', '2008', '1hr 57min', 3, 10, 4, 2,
NOW());
```

```
INSERT INTO movies(movies_id, movies, genre, year, length, revenue_id,
actor_id, director_id, reviews_id, last_update)
VALUES (4, 'Pacific Rim', 'Science Fiction/Action', '2013', '2h 12m', 1, 6, 3,
1, NOW());
```

Swift Code

```
import UIKit
import Alamofire


class PostViewController: UIViewController {

   @IBOutlet weak var txtPost: UITextField!
   @IBOutlet weak var txtDesc: UITextField!
   @IBOutlet weak var btnSend: UIButton!
   @IBOutlet weak var txtError: UILabel!
   override func viewDidLoad() {
      super.viewDidLoad()

      txtError.text = "";
   }

   @IBAction func btnSend_Tap(_ sender: UIButton) {
      let url = "http://localhost:3000/task"

      let params: Parameters = [
         "subject" : txtPost.text!,
         "description" : txtDesc.text!
      ]

      Alamofire.request(url , method: .post, parameters: params, encoding:
JSONEncoding.default).responseString { response in
         debugPrint(response)
      }
   }
}
```

Why it is Interesting

This part of the code is interesting because it take the data from the database server and uses it in the swift application. The letgeturl command in the code is what connects the server and swift application. It takes the application formatted in a JSON format and displays it in a easy to view display. If the app correctly reaches the data then it will display it in the app. If the data will be displayed in a subject and description field. If the data cannot be accessed or displayed then it will not display a message.

Node.js code

App.js

```
/*
* It's important to escape characters to avoid SQL injection
* https://github.com/mysqljs/mysql#escaping-query-values
*
* HTTP status codes
* https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
*/

var http       = require('http');
var mysql      = require('mysql');
var express    = require('express');
var bodyParser  = require('body-parser');

var connection = mysql.createConnection({
  host : 'localhost',
  user : 'root',
  password : 'hawklet',
  database : 'moviedata'
});

connection.connect();

connection.query('SELECT 1 + 1 AS solution', function(err, rows, fields) {
  if(err) throw err;
  else console.log('database connected');
});

// instance of express server
var app = express();

// makes it possible to parse JSON
app.use(bodyParser.json());
```

```
// middleware: ready to handle invalid json
app.use(function(error, req, res, next) {
  if (error instanceof SyntaxError) {
     res.status(400).send('Bad Request');
  } else {
     next();
  }
});

/* POST - CREATE */
app.post('/task', function(req, res) {
  console.log('/task req.body = ' + JSON.stringify(req.body));
  var insert = 'INSERT INTO tasks SET ?';
  connection.query(insert, req.body, function(err, rows) {
     if(err) {
        res.status(400).send('Bad Request');
     } else {
        res.status(201).send('Created');
     }
  });
});

/* GET - READ */
app.get('/tasks', function(req, res) {
  var select = 'SELECT * FROM tasks';
  connection.query(select, function(err, rows) {
     res.send({ data : rows });
  });
});

app.get('/task/:id', function(req, res) {
  var select = 'SELECT * FROM tasks WHERE task_id = ?';
  connection.query(select, [req.params.id], function(err, rows) {
     res.send({ data : rows });
  });
});

/* PUT - UPDATE */
app.put('/task/:id', function(req, res) {
  console.log('put task with id: ' + req.params.id);
});
```

```
/* DELETE */
app.delete('/task/:id', function(req, res) {
  console.log('delete task with id: ' + req.params.id);
});

http.createServer(app).listen(3000);

console.log('http://localhost:3000/');
```

Why it is Interesting

This code is interesting because it is what takes information from the database and allows it to be sent to the app. This code is vital in allowing the app to utilize the information that is on the database, and without it, the app wouldn't do much of anything.