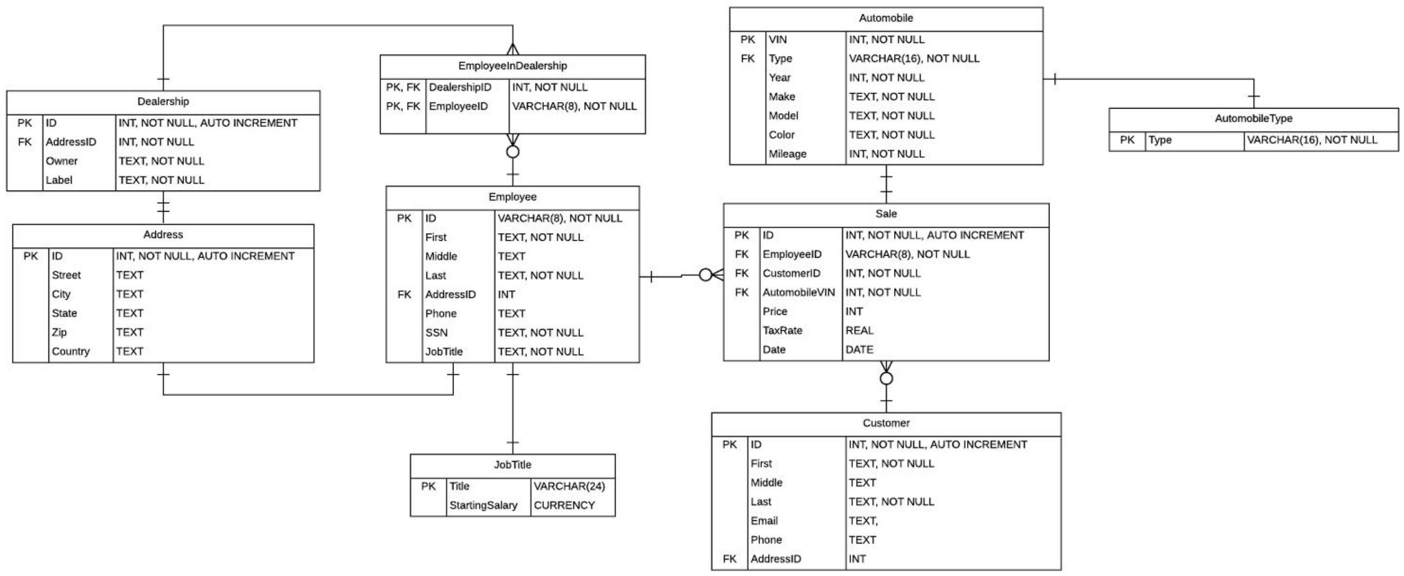
James Clark, Student1 Lastname, Student2 Lastname, Student3 Lastname
Mr. James Clark
Software Application Development
December 12, 2016

Final Report

Entity Relationship Diagram (Database Diagram)



Dealership

PK	ID	INT, NOT NULL, AUTO INCREMENT
FK	AddressID	INT, NOT NULL
	Owner	TEXT, NOT NULL
	Label	TEXT, NOT NULL

EmployeeInDealership

PK, FK	DealershipID	INT, NOT NULL
PK, FK	EmployeeID	VARCHAR(8), NOT NULL

Automobile

PK	VIN	INT, NOT NULL
FK	Type	VARCHAR(16), NOT NULL
	Year	INT, NOT NULL
	Make	TEXT, NOT NULL
	Model	TEXT, NOT NULL
	Color	TEXT, NOT NULL
	Mileage	INT, NOT NULL

AutomobileType

PK	Type	VARCHAR(16), NOT NULL

Address

PK	ID	INT, NOT NULL, AUTO INCREMENT
	Street	TEXT
	City	TEXT
	State	TEXT
	Zip	TEXT
	Country	TEXT

Employee

PK	ID	VARCHAR(8), NOT NULL
	First	TEXT, NOT NULL
	Middle	TEXT
	Last	TEXT, NOT NULL
FK	AddressID	INT
	Phone	TEXT
	SSN	TEXT, NOT NULL
	JobTitle	TEXT, NOT NULL

Sale

PK	ID	INT, NOT NULL, AUTO INCREMENT
FK	EmployeeID	VARCHAR(8), NOT NULL
FK	CustomerID	INT, NOT NULL
FK	AutomobileVIN	INT, NOT NULL
	Price	INT
	TaxRate	REAL
	Date	DATE

JobTitle

PK	Title	VARCHAR(24)
	StartingSalary	CURRENCY

Customer

PK	ID	INT, NOT NULL, AUTO INCREMENT
	First	TEXT, NOT NULL
	Middle	TEXT
	Last	TEXT, NOT NULL
	Email	TEXT,
	Phone	TEXT
FK	AddressID	INT

Interesting Swift code:

```
override func viewDidLoad() {
    super.viewDidLoad()

    lblSubject.text = ""
    lblDescription.text = ""

    let geturl = "http://localhost:3000/tasks"

    Alamofire.request(geturl , method: .get, encoding: JSONEncoding(options: [])).responseJSON {
response in
        debugPrint(response)

        if let result = response.result.value {
            let json = JSON(result as! NSDictionary)
            let tasks = json["data"].arrayValue

            // display only the last task
            let subject = tasks[tasks.count - 1]["subject"].string
            let description = tasks[tasks.count - 1]["description"].string
            self.lblSubject.text = subject
            self.lblDescription.text = description

        } else {
            self.lblSubject.text = "Error loading data"
            self.lblDescription.text = "Error loading data"
        }
    }
}
```

Why is it interesting?

This snippet of code combines two Swift frameworks: Alamofire, and SwiftyJSON. Alamofire makes it possible to efficiently write easy to read network code, and SwiftyJSON parses the response data in a more natural way. Previously I wrote basic Swift code for both these tasks and it required many more lines. This is more elegant. It's easy to read and understand once you grasp what a json data structure really is.

Interesting Node.js code:

```
/* 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');
        }
    });
});
```

Why is it interesting?

This code is interesting because if you pay close attention you'll notice that the ? in the sql query is automatically escaped and then completed according to the properties of any json object. In the query connection, req.body is a json object sent from the client connection. This design makes it possible to write dynamic query systems. For example, I can create forms that require only two fields then send partially complete forms to the back end without having to construct json strings of incomplete or null data. For mobile applications, it's important to write energy efficient code.