

# Easy Auto: Used car trading system

Group 1.

**Zheng Wang (zw5sx), Longchang Li (ll8fn),**

**Evaristo Koyama (ek4ks), Bicheng Fang (bf5ef)**

## 1. Introduction:

In this project, we designed an online used-car trading system, which integrated MySQL database as backend to manipulate data, and a web site as interface for users and dealers to interact. The core purpose of this project is to consolidate our understanding of database system designing. Due to the privacy policy, we are unable to download actual data from website, thus we generated our own data for displaying.

Obviously, the back end database is a support of our website, which stores all the user data and some meta data of our website. There are three different roles in our database:

1. Database manager (administrator) who has the super privilege to manipulate all data, grant and revoke privileges to other users and modify the structure of the database.
2. Dealers: a subgroup of actual user of our website. Dealers may put (new post) their products (used cars) on the website, delete some specific items, make transaction with customers, and review buyers' profile.
3. Customers: The majority of users in such scenario in the real world are general customers. (Though any temporal visitors can access the product list on our website, we strongly suggest users register on our website to acquire specific services) Here, customers can browse all the products on our website, and customize searching criteria to narrow the demanding results, as well as check the related cars' reports, dealer's information and so on. Customers can also add some products to their "wish list" so that they can follow the specific cars quickly.

## 2. User requirements:

After looking into some actual used-car trading systems, we found out that car information retrieval is the basic function. Users should be able to customize their own searching criteria to narrow down the results range, and rank the results by price, years,

mileages, etc. While just like all e-shopping websites, users should also be able to check the comments and reviews of dealers, to determine which dealer may be more patient or helpful. Below is a list of functions we have implemented in our demo website:

**User side:**

1. Sign on and sign in.
2. Car searching. Users can specify model, make, type, price range, and some other features to find their ideal candidate.
3. Car information browsing. After searching out several results, users can have a detailed information list of each car, as well as the dealer's information.
4. Follow! Interested in some car, and want to save them for later use or comparison? No problem, we have a follow list in our website, and each user can add as many cars they are interested in as they want.
5. Make appointments. When you're really interested with a car, you may want to make an appointment with the dealer to check the situation of this car in person.
6. Purchase.
7. Rate a transaction or dealer.

**Dealer side:**

1. Posting cars' information. Dealers can update, delete and modify their inventory list.
2. Check customers' information.
3. Make appointment and sell.

### 3. Database designs

To reach these purposes, we designed 14 tables to store our data. All tables are connected and constrained by some rules. Below are our detailed table designs (schema):

Table Name	Features
User	Username, password, name, phone, email, zipcode, authority
Appointment	Appointment_id, c_name, d_name, car_id, appointment_time
Appointment_service	Appointment_id, service_type
Car	Car_id, model_name, type_name, brand_name, place_of_production, year, mileage, imageURL, price
Customers	Username, register_date, balance
Dealer	Username, register_date, star_count
Following	Username, car_id
Inventory	Username, car_id, location
Model	Name, brand_name, year
Report	Report_id, car_id, check_date, owner, ownership_length, mileage, accident, use_type, title
Review	Review_id, c_name, d_name, release_time, comment, star_count
Services	Service_type, price, description
Transaction	Car_id, c_name, d_name, transaction_price, transaction_date
Type	Name, description

The ER-Diagram is shown in figure 1.

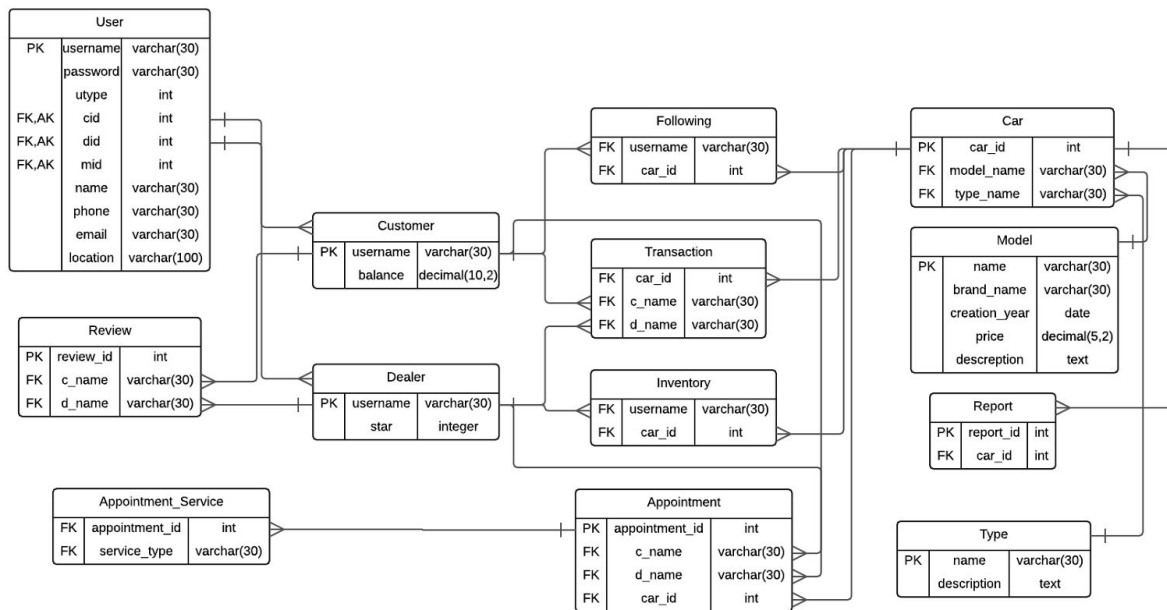


Figure 1. ER-Diagram

3NF proof for main tables:

<pre> CREATE TABLE `User` (   `username` varchar(30) NOT NULL,   `password` varchar(30) DEFAULT NULL,   `name` varchar(30) DEFAULT NULL,   `phone` varchar(30) DEFAULT NULL,   `email` varchar(30) DEFAULT NULL,   `zipcode` decimal(5,0) DEFAULT NULL,   `authority` enum ('Customer','Dealer','DBManager') DEFAULT NULL,   PRIMARY KEY (`username`) )         </pre>	<p>Proof:</p> <p>Username -&gt; password, name, phone, email, zipcode, authority.</p> <p>A -&gt; B, C, D, E, F, G</p> <p>Becomes ABCDEFG, thus it's a good design.</p>
<pre> CREATE TABLE `Transaction` (   `car_id` int(11) DEFAULT NULL,   `c_name` varchar(30) DEFAULT NULL,   `d_name` varchar(30) DEFAULT NULL,         </pre>	<p>Proof:</p> <p>Car_id, c_name, d_name -&gt; transaction_price, transaction_date</p>

<pre> `transaction_price` decimal(10,2) DEFAULT NULL, `transaction_date` date DEFAULT NULL, KEY `transaction_ibfk_1` (`car_id`), KEY `transaction_ibfk_2` (`c_name`), KEY `transaction_ibfk_3` (`d_name`), CONSTRAINT `transaction_ibfk_1` FOREIGN KEY (`car_id`) REFERENCES `Car` (`car_id`) ON DELETE NO ACTION ON UPDATE CASCADE, CONSTRAINT `transaction_ibfk_2` FOREIGN KEY (`c_name`) REFERENCES `Customer` (`username`) ON DELETE NO ACTION ON UPDATE CASCADE, CONSTRAINT `transaction_ibfk_3` FOREIGN KEY (`d_name`) REFERENCES `Dealer` (`username`) ON DELETE NO ACTION ON UPDATE CASCADE ) </pre>	<p>A, B, C -&gt; D, E</p> <p>Which becomes ABCDE, and not splittable. Thus it turns out that this is good design.</p>
--	---

## 4. Implementation:

- Database:

We built our database using MySQL. Figure 1 has shown some constraints in our actual implementation of the database. To point out that we store encrypted password, rather than plain text password for users. Besides, we've added some triggers and functions to make the database more smart and secure. For example, when a user rated a dealer, it will triggers a function to calculate the average rate of this dealer and then update the number stored in database.

- Website

The website was implemented using Javascript. The design is beautiful and clean, so that users may not get tired after long time searching. To point out that we designed our implementations of the website, parameterize the input first and then pass the parsed input to database, rather than original input, to prevent SQL injections. We believe that after encrypting users password and preventing SQL injection, our website is secure enough for most attach.

## 5. Sample queries:

### I. Login:

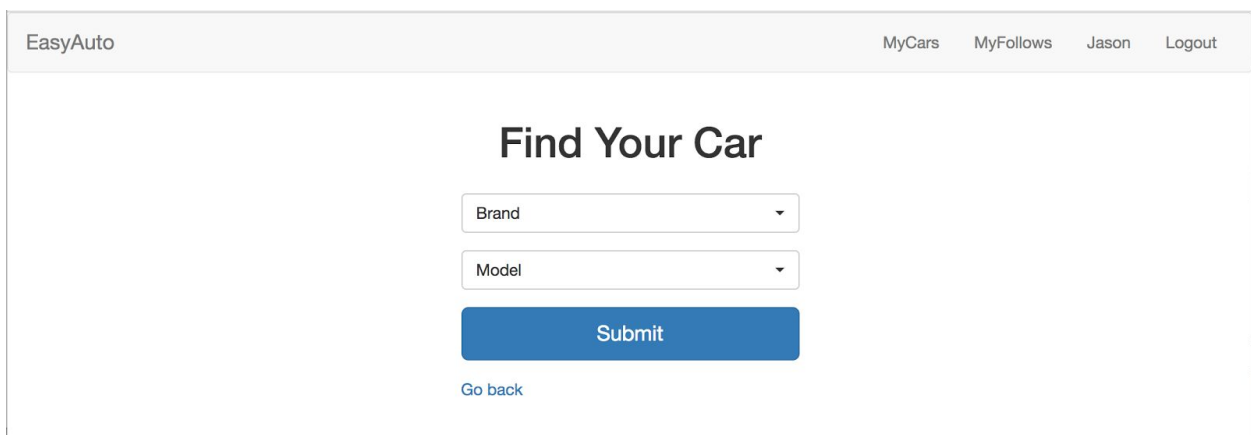
```
txtUserId = getQueryString("UserId");
txtpsw = getQueryString("Password");
txtSQL = "SELECT * FROM Users WHERE UserId = @0 AND password = @1";
db.Execute(txtSQL,txtUserId, txtpsw);
```

### II. Car searching:

```
carmake = getQueryString("Make");
carmodel= getQueryString("Model");
Carprice = getQueryString("Price");
txtSQL = "SELECT * FROM Cars WHERE Make = @0 and Model = @1 and Price < @2";
db.Execute(txtSQL,carmake,carmodel,Carprice);
```

## 6. Screen shots:

### I. Searching



The screenshot shows the 'EasyAuto' website header with navigation links: 'MyCars', 'MyFollows', 'Jason', and 'Logout'. The main content area is titled 'Find Your Car' and contains a search form with two dropdown menus labeled 'Brand' and 'Model', a blue 'Submit' button, and a 'Go back' link.

## II. Searching results

EasyAuto


MyCarsMyFollowsJasonLogout

# Welcome to EasyAuto

Find a car you love


Search

Cars you may like




GMC

More Info




GMC

More Info




Jeep


More Info





Ford

More Info









## III. Detailed information.

Model_name	Yukon
Brand_name	GMC
Type	SUV
Year	2010
Mileage	6000
Price	90200
Dealer	CarFAX
Contact	carfax@db.com
<div>Buy It! Follow</div>	



Latest Report						
Date	Owner	Ownership	Mileage	Accident	UseType	Title
Thu Jul 23 2015 00:00:00 GMT-0400 (EDT)	J*****	2.4	6000	None	personal	Clean