**Database Design Topic: Car Rental**

**CS6360.002**

Team Members : Renchi Liu (rxl161430)

Jinglin Li (jxl163530)

Yanjiao Jia (yxj160130)

1. **Project Description and Data Requirement:**

In this project, we design a database system for the Car Rental. This system required several entities: Customer, Account, Car, Order, Store and Store Representative (The staff who is in charge of the store). One Customer is allowed to have only once Account. One Car is allowed to be supervised by only one Store Representative. A Customer can rent more than one cars. One Car can be rented by more than one Customer. For the Order Process, only one Customer could appear at a single Order Place, and more cars can be exist in a single Order Place. Many cars are allowed to be maintained in a store. Therefore, there are two 1-to-1 binary relationship, two 1-to-n binary relationship, two n-to-n binary relationship.

Before this final design, we also perform the 3-NF normalization. At first, the Customer has the following attributes:

**Customer**

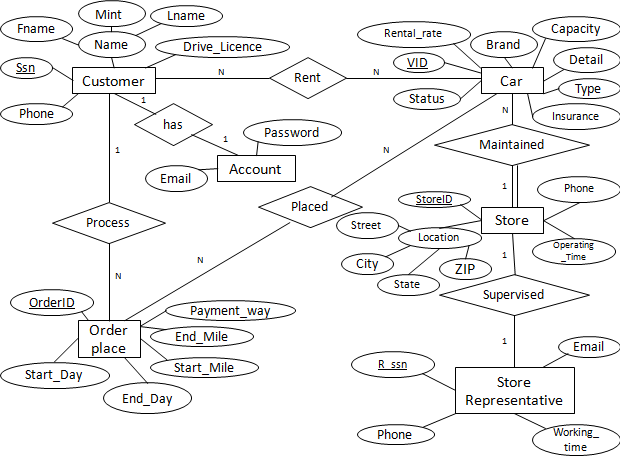
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Ssn | Fame | Mint | Lname | Drive\_Licence | Phone | Email | Password |

The Customer has Ssn as the key and Fname, Mint, Lname… Phone, Email and Password. However, Email could be the candidate attribute to the password in another table. By performing the 3-Nf normalization, Email and Password were removed from this table and constitute the table Account.

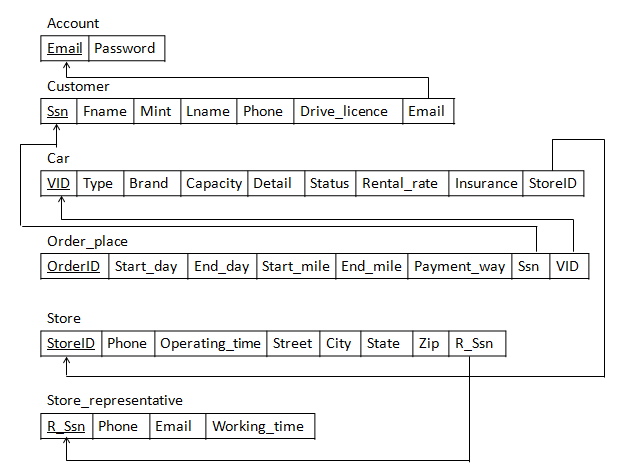
Because the car can be available in the store or not, this system could check the status of the car. If the car is available to be rent, the start mile of the car is turned to be the end mile of the previous order, the start day will be modified to be today’s date. This is the idea of the design of one Procedure. For another Procedure, because the living expense is increasing, the rental rate is also increased. Then for the cars which located in Dallas, the rental rate in increased by 25%.

We also design two triggers for this system. On of the trigger display the rental rate difference between the old values and new values. Another trigger keep a log file containing data from rows that have been deleted from Car table. This means that if a car is rented by someone, the log could keep a track on these cars, which could be inserted Car table with a different location if applicable when returned.

1. Final ER Diagram



1. Final Relational Schema



1. The code for two Procedures and Two Triggers.

use Car\_Rental

-- Procedure 1

-- update the rental rate (increase by 25%) of the cars which located in dallas

CREATE OR REPLACE PROCEDURE

UpdateRental\_Rate25 AS

CURSOR DallasCarID IS

SELECT CAR.VID

FROM car, store

WHERE car.StoreID = store.StoreID

and store.City = 'Dallas';

thisCarID Car.VID%TYPE;

BEGIN

OPEN DallasCarID;

LOOP

FETCH DallasCarID INTO thisCarID;

EXIT WHEN (DallasCarID%NOTFOUND);

UPDATE car SET Rental\_rate = Rental\_rate \* 1.25

WHERE VID = thisCarID;

END LOOP;

CLOSE DallasCarID;

END;

/

BEGIN

UpdateRental\_Rate25();

END;

/

-- Procedure 2

-- update the start day and start mile of the car

-- when the status of the car is "2" (pending for rental)

CREATE OR REPLACE PROCEDURE

UpdateStart\_Day\_And\_Mile AS

CURSOR PENDING\_OrderID IS

SELECT order\_place.OrderID

FROM order\_place, car

WHERE car.VID = order\_place.VID

AND car.Status = '2';

thisOrderID order\_place.OrderID%TYPE;

BEGIN

OPEN PENDING\_OrderID;

LOOP

FETCH PENDING\_OrderID INTO thisOrderID;

EXIT WHEN (PENDING\_OrderID%NOTFOUND);

UPDATE order\_place SET (Start\_day = '24-APR-17' and Start\_mile = End\_mile)

WHERE OrderID = thisOrderID;

END LOOP;

CLOSE PENDING\_OrderID;

END;

/

BEGIN

UpdateStart\_Day\_And\_Mile();

END;

/

-- Trigger 1

-- create a row-level trigger for the car table that would fire for INSERT

-- or UPDATE operations performed on Car table

-- this trigger will display the rental rate difference between the old values and new values.

CREATE OR REPLACE TRIGGER display\_rentalrate\_changes

BEFORE INSERT OR UPDATE ON CAR

FOR EACH ROW

WHEN (NEW.VID > 0)

DECLARE

rentalrate\_diff number;

BEGIN

rentalrate\_diff := :NEW.Rental\_rate - :OLD.Rental\_rate;

dbms\_output.put\_line('Old Rental Rate: ' || :OLD.Rental\_rate);

dbms\_output.put\_line('NEW Rental Rate: ' || :NEW.Rental\_rate);

dbms\_output.put\_line('Rental Rate difference: ' || rentalrate\_diff);

END;

/

-- trigger 2

-- keep a log file containing data from rows that have been deleted from car table

CREATE TRIGGER Car\_Delete

AFTER DELETE ON Car

REFERENCING OLD ROW AS Old

FOR EACH ROW

INSERT INTO Car\_Deleted\_Log

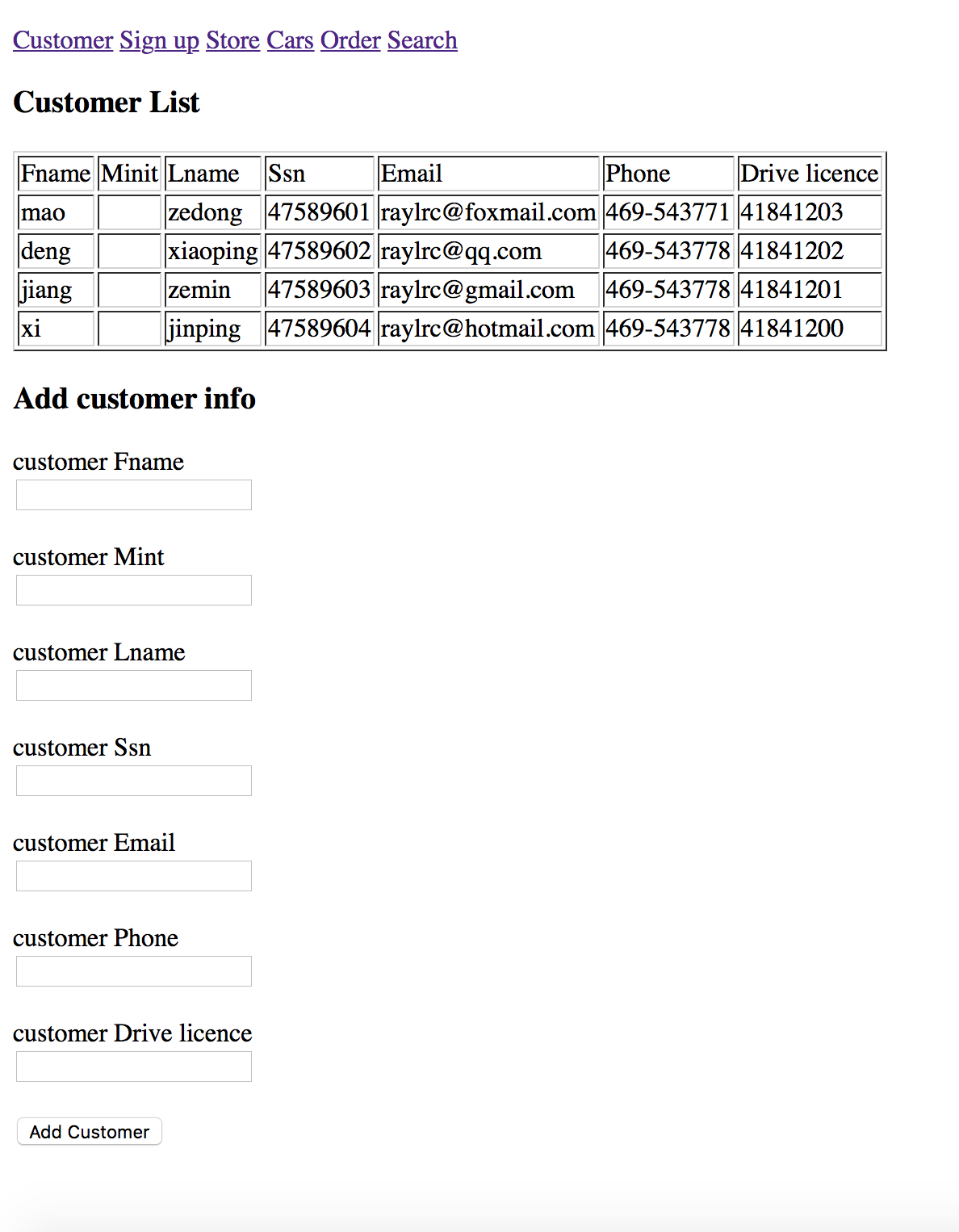
VALUES(old.VID);

1. Implement CRUD operations for the tables using PHP.

(First using the ‘addDate.sql’ file to load some test Data. )

1. Create

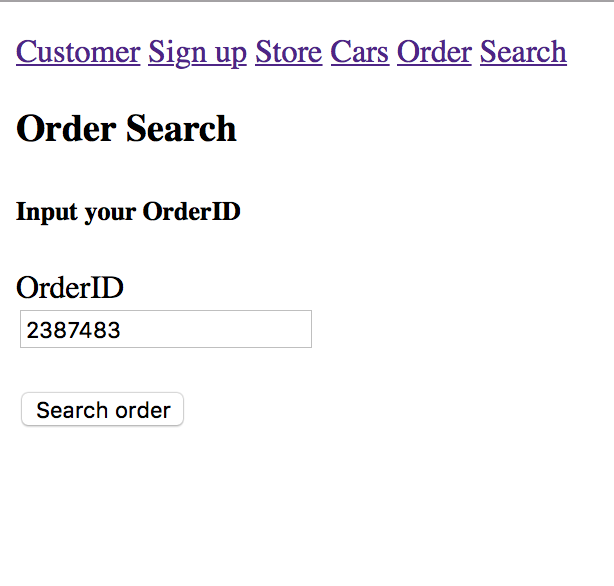
We use the Customer table to illustrate this operation.

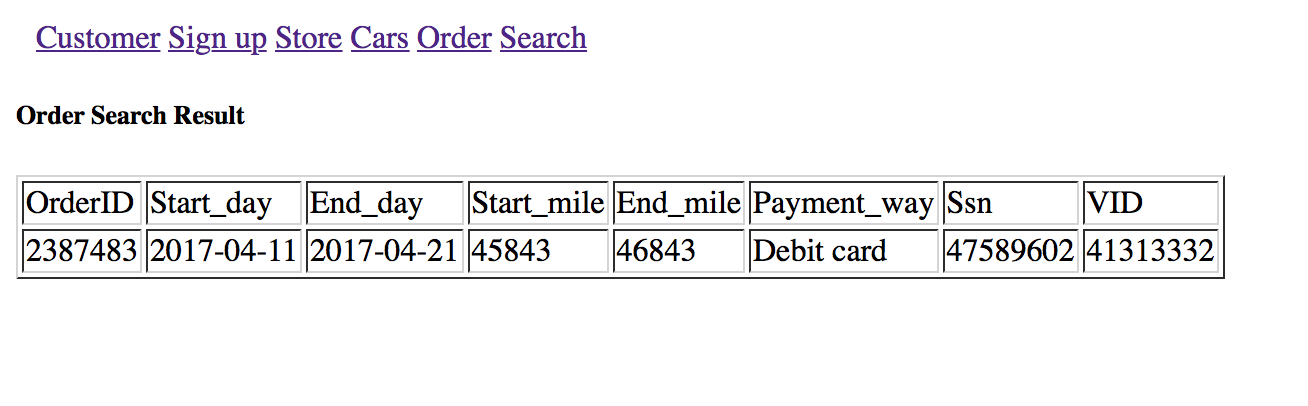


This function will insert one customer data into the table. And display in the webpage.

1. Read

We use Order Table to illustrate this operation.

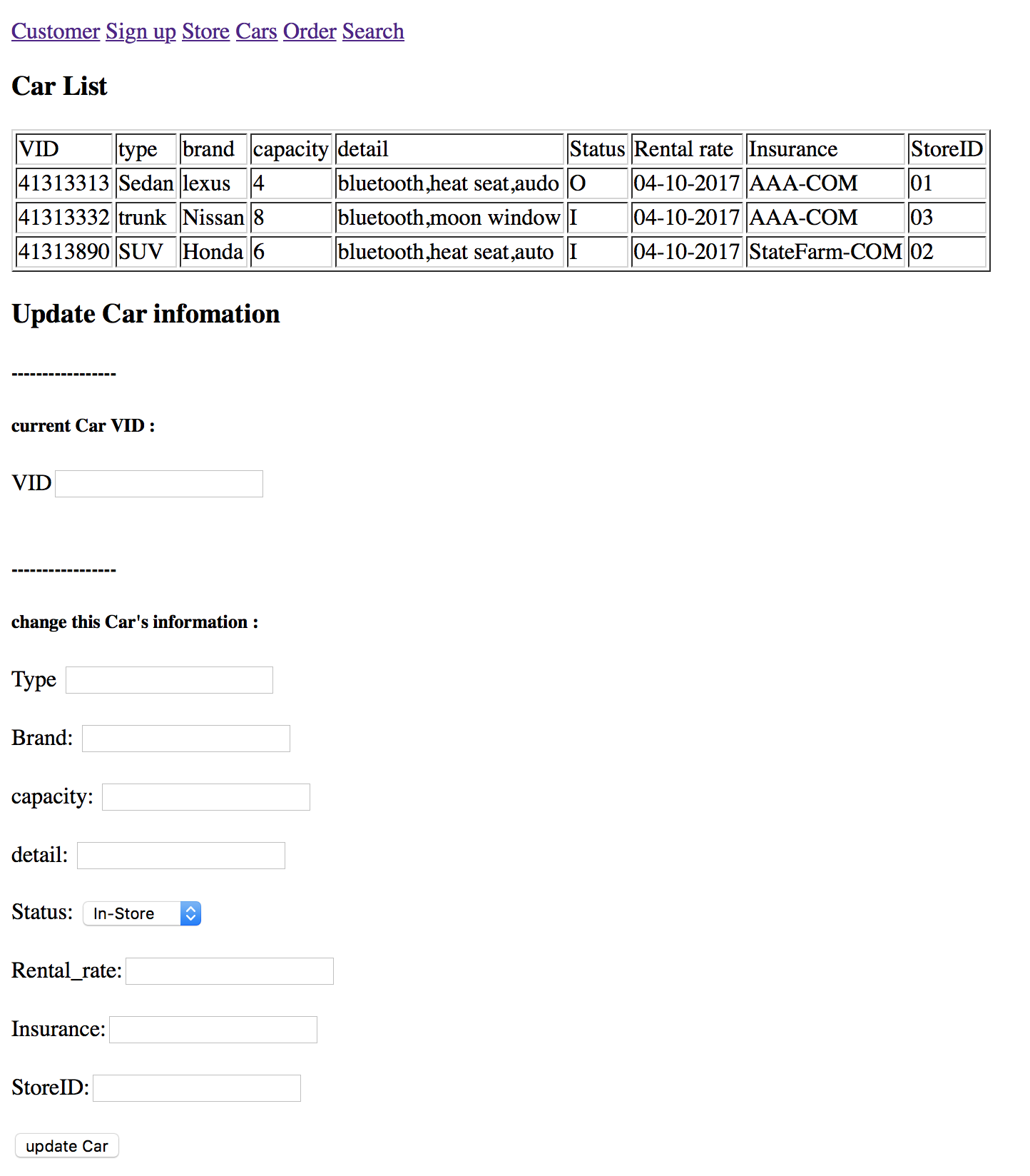




In the search page, input one Order ID which already exists in table. Then it will show the detail of this Order.

1. Update

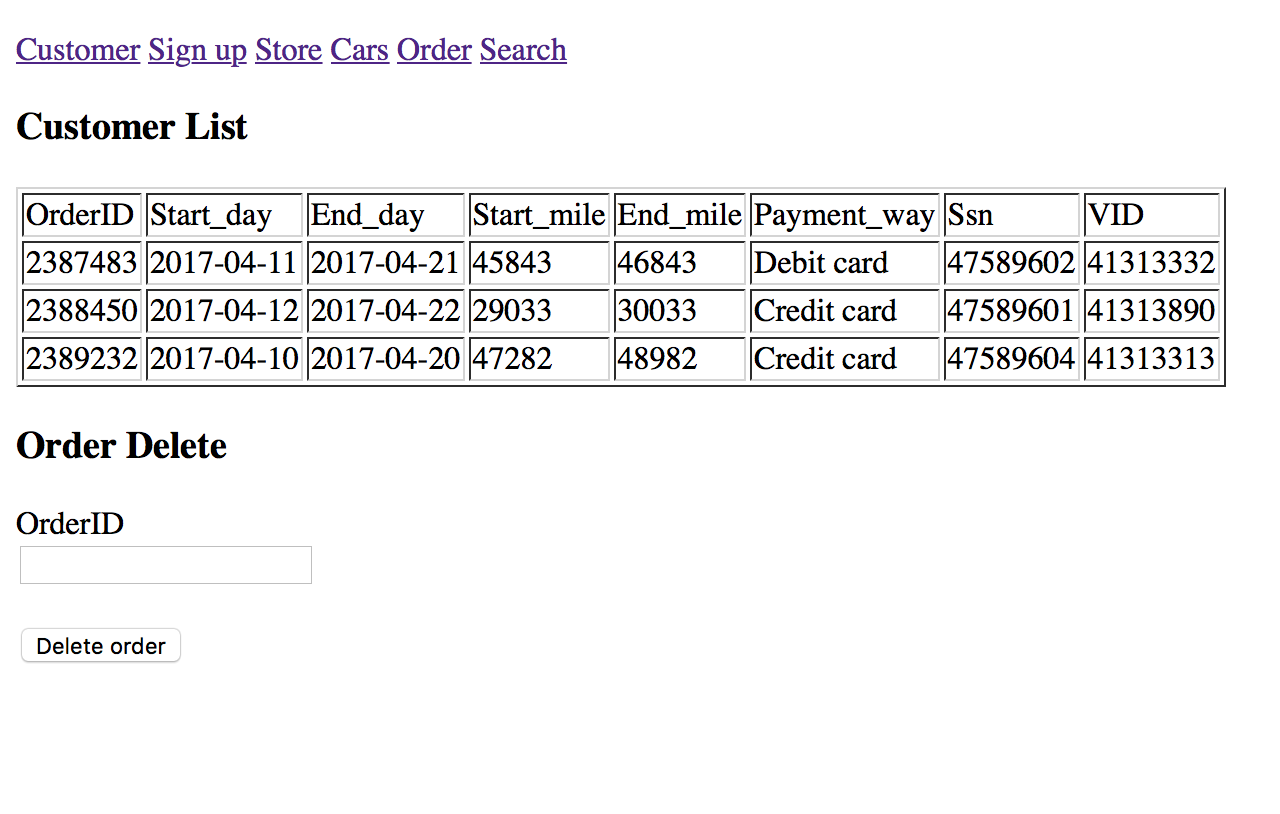
We use Car Table to illustrate this operation.



In this page, the Car data will show in the table. Input one Car VID (VID cannot be changed ,because it is the primary key), and change other attributes of the car. After update, the data will change and update in the table.

1. Delete

We use Order Table to illustrate this operation.



In this page, the order data which exist will show in the table .Input one Order ID to delete the whole data. After delete the table will update and only show data left.