

# 数据库系统实验

## 实验八：数据库设计与应用开发大作业—— 赛车游戏数据库的设计与开发

16 级计科教务 2 班

16337327      郑映雪

16337328      郑宇骁

16337341      朱志儒

# 目录

I.	<b>实验题目</b>	2
II.	<b>数据库设计</b>	2
	灵感来源&数据来源	2
	数据库概念设计	2
	数据库逻辑结构设计	3
	实体	3
	实体间的关系	6
	触发器设计	9
III.	<b>数据库开发及功能实现的代码及解释</b>	10
	创建基本表	10
	触发器设置	10
	应用程序功能实现	17
IV.	<b>UI 开发</b>	26
	实现技术	26
	功能特性	26
	功能	26
	特性	27
V.	<b>实验结果展示</b>	27
	登录/注册	27
	商城界面	28
	模糊查找	29
	个人物品界面	29
	成绩录入界面	30
VI.	<b>实验总结</b>	31

# 实验题目

## 数据库设计与应用开发大作业——赛车游戏数据库的设计与开发

### 数据库设计

#### 灵感来源&数据来源

数据库设计的灵感来源于当下时兴的一款手机游戏——QQ 飞车。我们小组考虑到可以为这次开发增加一些趣味性，所以想到设计一个类似于 QQ 飞车的**赛车游戏的数据库**。

在本次设计中，我们根据当下许多游戏的体验，总结设计开发出注册、登录、商城系统、车队系统、段位系统、添加比赛数据、搜索等数据库相关功能。

作为参考数据，本次开发中某些数据表中用到的赛道、赛车及宠物数据，爬取自 QQ 飞车手游的官方网站。[https://speedm.qq.com/web201712/car\\_list.shtml](https://speedm.qq.com/web201712/car_list.shtml)

#### 数据库概念设计

##### 1、实体

在本数据库中，设计了 6 个实体：

①player：玩家实体，包含玩家昵称、玩家等级、玩家经验值、玩家魅力值、玩家车队、玩家段位、玩家金钱等属性。

②car：赛车实体，包含赛车名、赛车级别、赛车魅力值、赛车价格等属性。

③pet：宠物实体，包含宠物名称、宠物等级、宠物魅力值、宠物价格等属性。

④map：地图实体，包含地图的名称和地图的当前记录等属性。

⑤clothes：服装实体，包含服装名称，服装魅力值，服装价格等属性。

⑥fleet：车队实体，包含车队名称、车队队长、车队段位等属性。

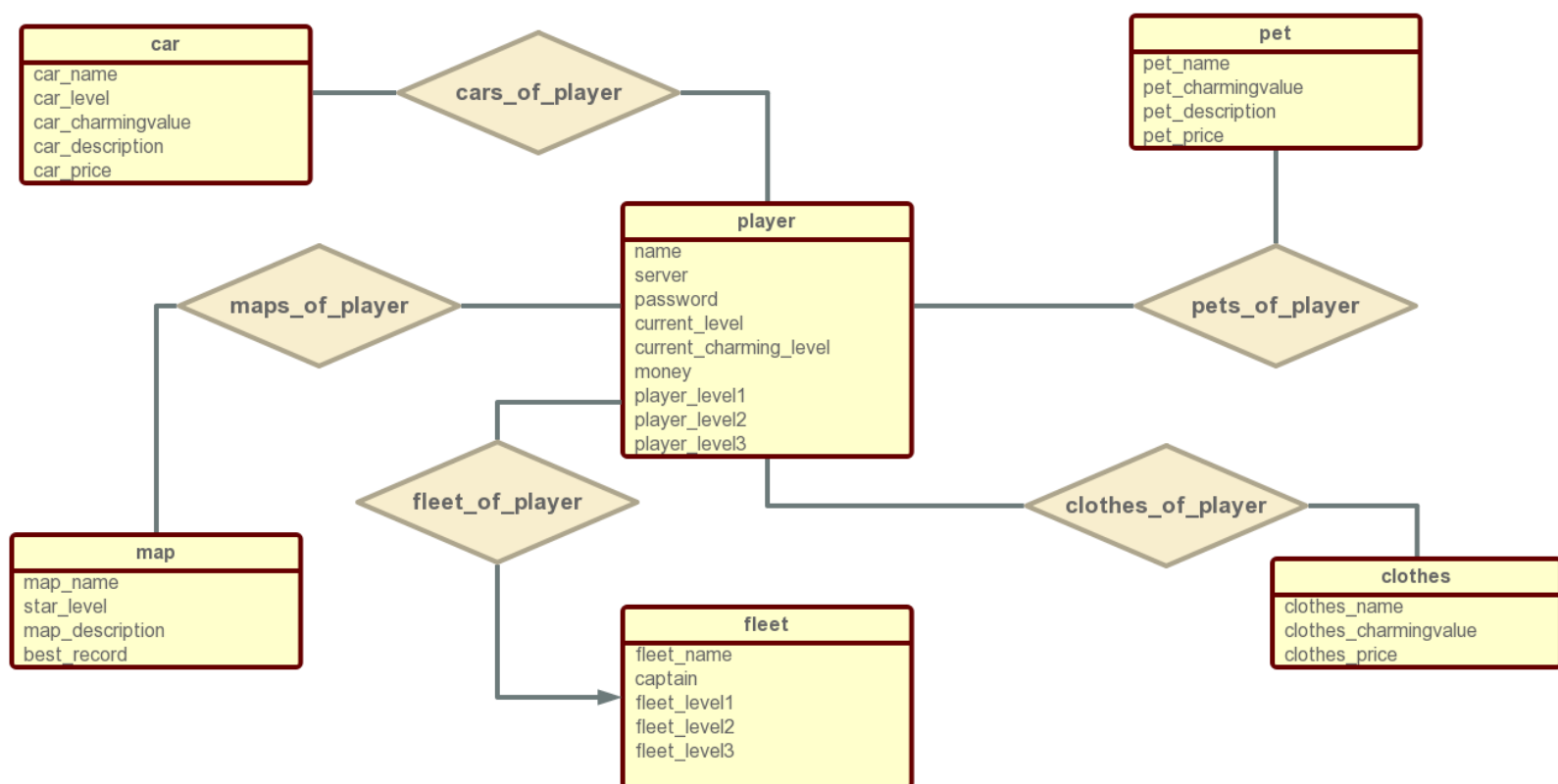
##### 2、关系

在本数据库中，设计了 5 个关系：

- ①cars\_of\_player: 表示玩家-赛车的从属关系，为一对多的关系。
- ②pets\_of\_player: 表示玩家-宠物的从属关系，为一对多的关系。
- ③maps\_of\_player: 玩家-地图的记录，为一对多的关系。
- ④clothes\_of\_player: 玩家-衣服的从属关系，为一对多的关系。
- ⑤fleet\_of\_player: 玩家-车队的关系，为多对一的关系。

### 3、E-R 图

由上面的概念设计，画出 E-R 图如下所示：



### 数据库逻辑结构设计

由数据库的概念结构，设计出数据库的逻辑结构如下所示：

#### 实体

##### ①赛车 car

属性名	类型	备注解解释
<u>car_name</u>	varchar(100)	为实体的主键，玩家-赛

		车从属关系中 car_name 的外键
car_level	varchar(10)	赛车等级（四级：ABCD）
car_charmingvalue	int	赛车本身给玩家增加的魅力值
car_description	varchar(2000)	赛车描述
car_price	int	赛车价格

## ②地图 map

属性名	类型	备注解释
<u>map_name</u>	varchar(100)	为实体的主键，玩家-地图记录关系中 map_name 的外键
star_level	int	星级，随赛道难度上升
map_description	varchar(2000)	赛道描述
best_record	int	赛道全服记录。

## ③宠物 pet

属性名	类型	备注解释
<u>pet_name</u>	varchar(100)	为实体的主键，玩家-宠物从属关系中 pet_name 的外键
pet_charmingvalue	int	宠物本身给玩家增加的魅力值

<b>pet_description</b>	varchar (2000)	宠物技能描述
<b>pet_price</b>	int	宠物价格

#### ④服装 clothes

属性名	类型	备注解解释
<b><u>clothes_name</u></b>	varchar(100)	为实体的主键，玩家-时装从属关系中 <b>clothes_name</b> 的外键
<b>clothes_charmingvalue</b>	int	服装本身给玩家增加的魅力值
<b>clothes_price</b>	int	服装价格

#### ⑤车队 fleet

属性名	类型	备注解解释
<b><u>fleet_name</u></b>	varchar(100)	为实体的主键，玩家-车队从属关系中 <b>fleet_name</b> 的外键
<b>captain</b>	varchar(100)	队长
<b>fleet_level1</b>	varchar (20)	大段位（青铜、白银、黄金等）
<b>fleet_level2</b>	int	小段位（当前大段位的几级，5级最低，1级最高）
<b>fleet_level3</b>	int	当前段位的积分

## ⑥玩家 player:

属性名	类型	备注解释
<u>name</u>	varchar(100)	昵称，由于不允许重名，故可作为主键
password	varchar(100)	玩家密码（此处简化为明文存储，实际应用中需要哈希）
current_experience	int	玩家当前经验
current_level	int	玩家当前等级
current_charmng_level	int	玩家当前魅力值
player_level1	varchar (20)	大段位（青铜、白银、黄金等）
player_level2	int	小段位（当前大段位的几级，5级最低，1级最高），
player_level3	int	当前段位的积分
money	int	玩家拥有的点券

## 实体间的关系

### ①cars\_of\_player 赛车和玩家的从属关系

属性名	类型	备注解释
<u>player_name</u>	varchar(100)	双属性主键，必须参照 player 中的主键
<u>car_name</u>	varchar (100)	必须参照 car 中的主键
is_driving	int	是否为当前携带车辆

## ②pets\_of\_player 宠物和玩家的从属关系

属性名	类型	备注解释
<u>player_name</u>	varchar(100)	双属性主键，必须参照 player 中的主键
<u>pet_name</u>	varchar(100)	必须参照 pet 中的主键
pet_experience	int	宠物经验
pet_level	int	宠物等级
is_taking	int	是否为当前携带宠物

## ③maps\_of\_player 玩家在某地图的记录

属性名	类型	备注解释
<u>player_name</u>	varchar(100)	双属性主键，必须参照 player 中的主键
<u>map_name</u>	varchar(100)	必须参照 map 中的主键
record	int	玩家在当前地图的记录

## ④clothes\_of\_player 玩家拥有的时装

属性名	类型	备注解释
<u>player_name</u>	varchar(100)	双属性主键，必须参照 player 中的主键
<u>clothes_name</u>	varchar(100)	必须参照 clothes 中的主键
iswearing	int	是否穿在玩家身上。因为玩家大可能会穿好几件衣服，通过判断

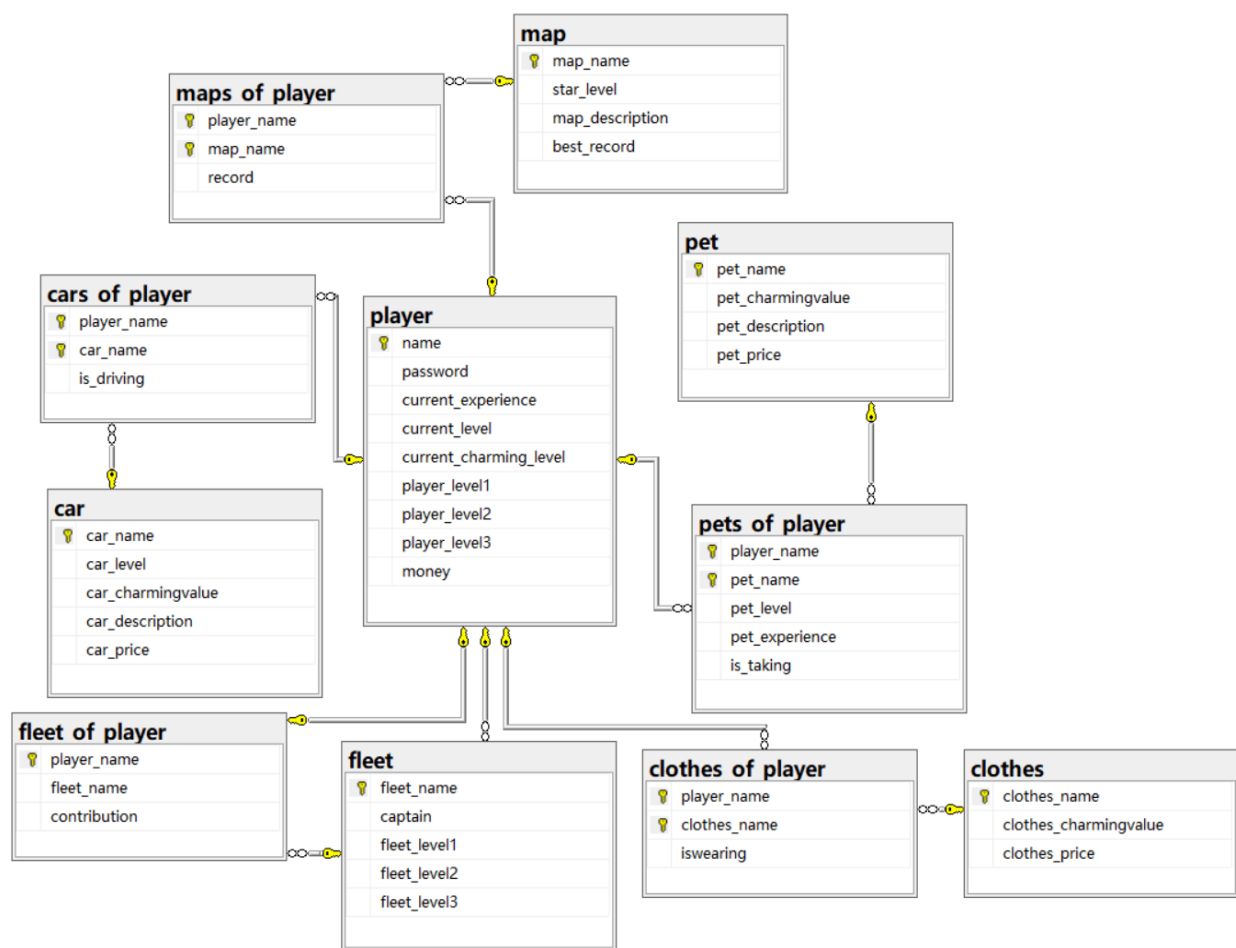


		iswearing 方便计算玩家当前的魅力值。
--	--	-------------------------

### ⑤fleet\_of\_player 玩家所在的车队

属性名	类型	备注解解释
<u>player_name</u>	varchar(100)	双属性主键，必须参照 player 中的主键
<u>fleet_name</u>	varchar(100)	必须参照 fleet 主键
contribution	int	玩家贡献值

由如上的逻辑设计，我们可以得到各个关系的关系图如下所示：



## 触发器设计

光是设计好逻辑关系还是不够的，仍然需要设计触发器来使得这个数据库的结构更加严密，更加具有完备性。触发器设计描述如下：

1、玩家在玩家-地图关系中，如果在某一地图创造新纪录（更新记录），且这个记录超过地图实体中的全服记录，则需同时更新地图实体中的记录。

2、当玩家更换当前携带的宠物时，需要更新 `is_taking` 的值，并根据宠物的新的魅力值重新计算玩家的当前魅力值。

3、当玩家更换当前驾驶的赛车时，需要更新 `is_driving` 的值，并根据赛车的新的魅力值重新计算玩家的当前魅力值。

4、当玩家更换当前穿着的衣服时，需要更新 `is_wearing` 的值，并根据衣服的新的魅力值重新计算玩家的当前魅力值。

（注：当前玩家魅力值=当前驾驶赛车魅力值+当前携带宠物魅力值+当前穿着所有服装魅力值）

5、宠物-玩家关系中，宠物的上限为 30 级，限制不能更新到 30 级以上，且不能更新一个比当前等级要小的值。

6、排位系统进行了简化，默认一段为 5 小级，每小级积 50 分。如果当前达到 50 分，则进行小级升级（注意升级是从 5 到 1，即 1 级是最高的）。如果当前 1 级达到 50 分，则升段。此处稍微繁琐一些，手动改变青铜到白银、白银到黄金、黄金到铂金、铂金到钻石、钻石到星耀，星耀到车神。

7、车队排位关系同 6 所示，默认一段为 5 小级，每小级积 50 分。如果当前达到 50 分，则进行小级升级（注意升级是从 5 到 1，即 1 级是最高的）。如果当前 1 级达到 50 分，则升段。此处稍微繁琐一些，手动改变青铜到白银、白银到黄金、黄金到铂金、铂金到钻石、钻石到星耀，星耀到车神。

8、玩家升级也需要触发器。此处简化升级系统——鉴于等级越高，玩家升一级越难，则设立触发器——当当前经验达到某一分时（可设为函数  $g(x)$ ，其中  $x$  为当前等级），则归零当前经验，升上一级。

9、玩家-服装关系、玩家-宠物关系、玩家-赛车关系添加新元组时，要根据服装里的点券数对玩家实体的点券数进行减操作。

## 数据库开发及功能实现的代码及解释

---

注：本次实验使用 python 的 `pyodbc` 库建立数据库。

### 创建基本表

在 QQ\_Speed 数据库中创建 11 个基本表，为不同的列选择合适的数据类型，创建列级完整性约束。

创建 car 表：

```
1. create table car(  
2.     car_name varchar(100) primary key,  
3.     car_level varchar(10),  
4.     car_charmingvalue int,  
5.     car_description varchar(7000),  
6.     car_price int  
7. );
```

其余的关系表建立与其相似，故在这里就不赘述了。详情请看附件的代码。

### 触发器设置

1、在 maps\_of\_player 表上定义 after update 触发器，用户更新某个赛道的比赛记录后，自动判断该记录与赛道的最好记录，如果该记录小于最好记录，则将赛道的最好记录更新为该记录。

```
1. create trigger TRI_Record_Update  
2. on maps_of_player  
3. after update  
4. as  
5.     if (update(record))  
6.     begin  
7.         declare @new_record int, @current_best_record int, @new_map_name v
```

```

    varchar(100);
8.     select @new_record = record, @new_map_name = map_name
9.     from inserted;
10.    select @current_best_record = best_record
11.    from map
12.    where map_name = @new_map_name;
13.    if (@new_record < @current_best_record)
14.        update map
15.        set best_record = @new_record
16.        where map_name = @new_map_name;
17.    end

```

2、在 maps\_of\_player 表上定义 after insert 触发器，用户插入某个赛道的比赛记录后，自动判断该记录与赛道的最好记录，如果该记录小于最好记录，则将赛道的最好记录更新为该记录。

```

1. create trigger TRI_Record_Insert
2. on maps_of_player
3. after insert
4. as
5.     declare @new_record int, @current_best_record int, @new_map_name varchar
        ar(100);
6.     select @new_record = record, @new_map_name = map_name
7.     from inserted;
8.     select @current_best_record = best_record
9.     from map
10.    where map_name = @new_map_name;
11.    if (@new_record < @current_best_record)
12.        update map
13.        set best_record = @new_record
14.        where map_name = @new_map_name;

```

3、在 pets\_of\_palyer 表上定义 instead of update 触发器，当更新用户宠物的经验值时，自动判断是否符合升级条件，如果该经验值大于或等于宠物当前等级 \*100，则宠物将升级，同时将限制宠物的最高等级为 30 级。

```

1. create trigger TRI_Pets_Of_Player_Update
2. on pets_of_player
3. instead of update
4. as
5.     if (update(pet_experience))
6.     begin
7.         declare @new_player_name varchar(100), @new_pet_level int, @new_pe
            t_name varchar(100), @new_pet_experiece int;
8.         select @new_player_name = player_name, @new_pet_level = pet_level,

```

```

    @new_pet_name = pet_name, @new_pet_experience = pet_experience
9.      from inserted;
10.     if (@new_pet_experience < @new_pet_level * 100)
11.         update pets_of_player
12.         set pet_experience = @new_pet_experience
13.         where player_name = @new_player_name and pet_name = @new_pet_name;
14.
15.     while @new_pet_experience >= @new_pet_level * 100
16.     begin
17.         if (@new_pet_level < 30)
18.             update pets_of_player
19.             set pet_level += 1, pet_experience = @new_pet_experience -
    @new_pet_level * 1000
20.             where player_name = @new_player_name and pet_name = @new_pet_name;
21.         else
22.             update pets_of_player
23.             set pet_experience = 30 * 100
24.             where player_name = @new_player_name and pet_name = @new_pet_name;
25.         select @new_pet_experience -
    = @new_pet_level * 1000, @new_pet_level += 1;
26.     end
27. end
28.
29. if (update(is_taking))
30. begin
31.     declare @new_n_player_name varchar(100), @new_n_pet_name varchar(
    100), @new_taking int;
32.     select @new_n_player_name = player_name, @new_n_pet_name = pet_name, @new_taking = is_taking
33.     from inserted;
34.     update pets_of_player
35.     set is_taking = @new_taking
36.     where @new_n_player_name = player_name and @new_n_pet_name = pet_name;
37. end

```

4、在 pets\_of\_palyer 表上定义 after pets\_of\_player 触发器，当用户携带某个宠物时，自动增加该用户的魅力值；当用户不携带某个宠物时，自动减少该用户的魅力值。

```

1. create trigger TRI_Change_Pet_Taking
2. on pets_of_player
3. after update
4. as
5.     if (update(is_taking))
6.     begin
7.         declare @new_is_taking int, @new_pop_pl_peopayecpr_name varchar(10
    0), @new_pett_name varchar(100);
8.         select @new_is_taking = is_taking, @new_pop_pl_peopayecpr_name = p

```

```

layer_name, @new_pett_name = pet_name
9.      from inserted;
10.     if (@new_is_taking = 1)
11.         update player
12.         set current_charming_level += (select pet_charmingvalue
13.                                         from pet
14.                                         where pet_name = @new_pett_name
15.                                         where name = @new_pop_pl_peopayecpr_name;
16.     else
17.         update player
18.         set current_charming_level -= (select pet_charmingvalue
19.                                         from pet
20.                                         where pet_name = @new_pett_name
21.                                         where name = @new_pop_pl_peopayecpr_name;
22.     end

```

5、在 pets\_of\_player 表上定义 after insert 触发器，用户购买某个宠物后，自动减少该用户的金币。

```

1. create trigger TRI_Pets_Of_Player_Insert
2. on pets_of_player
3. after insert
4. as
5.     declare @new_plpayer_name varchar(100), @new_pet_nname varchar(100);
6.     select @new_plpayer_name = player_name, @new_pet_nname = pet_name
7.     from inserted;
8.     update player
9.     set money -= (select pet_price
10.                  from pet
11.                  where pet_name = @new_pet_nname)
12.     where name = @new_plpayer_name;

```

6、在 cars\_of\_player 表上定义 after update 触发器，当用户驾驶某辆车时，自动增加该用户的魅力值；当用户不驾驶某辆车时，自动减少该用户的魅力值。

```

1. create trigger TRI_Change_Car_Driving
2. on cars_of_player
3. after update
4. as
5.     if (update(is_driving))
6.     begin
7.         declare @new_is_driving int, @newcp_player_name varchar(100), @new
8.         _car_name varchar(100);
9.         select @new_is_driving = is_driving, @newcp_player_name = player_n
10.        ame, @new_car_name = car_name

```

```

9.      from inserted;
10.     if (@new_is_driving = 1)
11.         update player
12.         set current_charming_level += (select car_charmingvalue
13.                                         from car
14.                                         where car_name = @new_car_name)
15.         where name = @newcp_player_name;
16.     else
17.         update player
18.         set current_charming_level -= (select car_charmingvalue
19.                                         from car
20.                                         where car_name = @new_car_name)
21.         where name = @newcp_player_name;
22.     end

```

7、在 cars\_of\_player 表上定义 after insert 触发器，用户购买某辆车后，自动减少该用户的金币。

```

1. create trigger TRI_Cars_Of_Player_Insert
2. on cars_of_player
3. after insert
4. as
5.     declare @new_player_name_cop varchar(100), @new_car_name varchar(100);
6.     select @new_player_name_cop = player_name, @new_car_name = car_name
7.     from inserted;
8.     update player
9.     set money -= (select car_price
10.                  from car
11.                  where car_name = @new_car_name)
12.     where name = @new_player_name_cop;

```

8、在 clothes\_of\_player 表上定义 after update 触发器，当用户穿戴某件服装时，自动增加用户的魅力值；当用户脱下某件服装时，自动减少用户的魅力值。

```

1. create trigger TRI_Change_Clothes_Wearing
2. on clothes_of_player
3. after update
4. as
5.     if (update(iswearing))
6.     begin
7.         declare @new_iswearing int, @copnew_player_name varchar(100), @new
8.         _clothes_name varchar(100);
9.         select @new_iswearing = iswearing, @copnew_player_name = player_na
10.         me, @new_clothes_name = clothes_name
11.         from inserted;
12.         if (@new_iswearing = 1)
13.             update player

```

```

12.         set current_charming_level += (select clothes_charmingvalue
13.                                         from clothes
14.                                         where clothes_name = @new_clothes_name)
15.         where name = @copnew_player_name;
16.     else
17.         update player
18.         set current_charming_level -= (select clothes_charmingvalue
19.                                         from clothes
20.                                         where clothes_name = @new_clothes_name)
21.         where name = @copnew_player_name;
22.     end

```

9、在 clothes\_of\_player 表上定义 after insert 触发器，当用户购买某件服装时，自动减少该用户的点券。

```

1. create trigger TRI_Clothes_Of_Player_Insert
2. on clothes_of_player
3. after insert
4. as
5.     declare @new_playercop_name varchar(100), @new_clothes_name varchar(100);
6.     select @new_playercop_name = player_name, @new_clothes_name = clothes_name
7.     from inserted;
8.     update player
9.     set money -= (select clothes_price
10.                  from clothes
11.                  where clothes_name = @new_clothes_name)
12.     where name = @new_playercop_name;

```

10、在 player 表上定义 instead of player 触发器，当用户更新其经验值时，自动判断是否符合更新条件，如果当前经验值大于或等于当前等级 x 1000，则用户将升级；当用户更新其段位积分时，自动判断是否符合升段条件，如果当前段位积分大于 50，则用户将提升小段位（从 5 到 1），如果当前小段位为 1，则用户将提升大段位（“青铜”、“白银”、“黄金”、“铂金”、“钻石”、“星耀”、“车神”）。

```

1. create trigger TRI_Player_Update
2. on player
3. instead of update
4. as
5.     declare @new_name varchar(100);
6.     if (update(money))
7.     begin

```



```

8.      declare @new_money int;
9.      select @new_name = name, @new_money = money
10.     from inserted;
11.     update player
12.     set money = @new_money
13.     where name = @new_name;
14. end
15.
16. if (update(current_experience))
17. begin
18.     declare @new_experience int, @new_level int;
19.     select @new_experience = current_experience, @new_level = current_
level, @new_name = name
20.     from inserted;
21.     if (@new_experience < @new_level * 1000)
22.         update player
23.         set current_experience = @new_experience
24.         where name = @new_name;
25.     while @new_experience >= @new_level * 1000
26.     begin
27.         update player
28.         set current_level += 1, current_experience = @new_experience -
@new_level * 1000
29.         where name = @new_name;
30.         select @new_experience =
= @new_level * 1000, @new_level += 1;
31.     end
32. end
33.
34. if (update(current_charming_level))
35. begin
36.     declare @new_charming int;
37.     select @new_charming = current_charming_level, @new_name = name
38.     from inserted;
39.     update player
40.     set current_charming_level = @new_charming
41.     where name = @new_name;
42. end
43.
44. if (update(player_level3))
45. begin
46.     declare @new_level3 int, @new_level2 int, @new_level1 varchar(20),
@next_level1 varchar(20);
47.     select @new_name = name, @new_level3 = player_level3, @new_level2
= player_level2, @new_level1 = player_level1
48.     from inserted;
49.     if (@new_level3 < 50)
50.         update player
51.         set player_level3 = @new_level3
52.         where name = @new_name;
53.     while @new_level3 >= 50
54.     begin
55.         select @next_level1 =
56.             case @new_level1
57.                 when '青铜' then '白银'

```

```

58.         when '白银' then '黄金'
59.         when '黄金' then '铂金'
60.         when '铂金' then '钻石'
61.         when '钻石' then '星耀'
62.         else '车神'
63.     end
64.     if (@new_level2 - 1 < 1)
65.         if (@new_level1 != '车神')
66.             begin
67.                 update player
68.                 set player_level2 = 5, player_level3 = @new_level3 - 5
69.                 where name = @new_name;
70.                 select @new_level2 = 5, @new_level1 = @next_level1;
71.             end
72.         else
73.             update player
74.             set player_level3 = 50
75.             where name = @new_name;
76.         else
77.             begin
78.                 update player
79.                 set player_level2 -= 1, player_level3 = @new_level3 - 50
80.                 where name = @new_name;
81.                 select @new_level2 -= 1;
82.             end
83.             select @new_level3 -= 50;
84.         end
85.     end

```

11、在 fleet 表上定义的 instead of update 触发器，当更新车队的段位积分时，自动判断是否符合升段条件，如果当前段位积分大于 50，则车队将提升小段位（从 5 到 1），如果当前小段位为 1，则车队将提升大段位（“青铜”、“白银”、“黄金”、“铂金”、“钻石”、“星耀”、“车神”）。由于此触发器实现与上一条相似，所以就不赘述了，详情请看代码。

## 应用程序功能实现

### 1、验证登录信息

根据 username 在 player 表中查询 password，确认输入的密码是否正确。

```

1. def verifyLogin(username, password):
2.     cursor.execute("""
3.         select password
4.         from player

```

```

5.         where name = ?
6.         """ , username)
7.     return password == cursor.fetchone()[0]

```

## 2、添加新用户

在 player 表中新增用户 username。

```

1. def addplayer(username, password):
2.     cursor.execute("""
3.         insert into player
4.         values (?, ?, 0, 0, 0, '青铜', 5, 0, 100000)
5.         """, username, password)
6.     cnxn.commit()
7.     return True

```

## 3、查找车辆信息

如果 name 为 None，则查找 car 表中所有车辆的信息；如果 name 为车辆名称，则查找该车辆的相关信息；支持模糊查找，即查找名字包含 name 的所有车辆的信息。

```

1. def getCar(name = None):
2.     cars = []
3.     if name == None:
4.         cursor.execute("""
5.             select *
6.             from car
7.             """)
8.         rows = cursor.fetchall()
9.         for row in rows:
10.            cars.append({
11.                'name': row.car_name,
12.                'level': row.car_level,
13.                'charming': row.car_charmingvalue,
14.                'description': row.car_description,
15.                'price': row.car_price
16.            })
17.     else:
18.         cursor.execute("""
19.             select *
20.             from car
21.             where car_name like '%{name}%'
22.             """.format(name=name))
23.         rows = cursor.fetchall()
24.         for row in rows:
25.            cars.append({
26.                'name': row.car_name,
27.                'level': row.car_level,

```

```

28.         'charming': row.car_charmingvalue,
29.         'description': row.car_description,
30.         'price': row.car_price
31.     })
32.     return cars

```

#### 4、查找宠物信息

如果 name 为 None，则查找 pet 表中所有宠物的信息；如果 name 为宠物的名称，则查找该宠物的相关信息，支持模糊查找，即查找名字包含 name 的所有宠物信息。代码与查找车辆信息类似，故不赘述。

#### 5、查找服装信息

如果 name 为 None，则查找 clothes 表中所有的服装信息；如果 name 为服装名称，则查找该服装的相关信息；支持模糊查找，即查找名字包含 name 的所有服装信息。代码与查找车辆信息类似，故不赘述。

#### 6、查找地图信息

如果 name 为 None，则查找 map 表中所有地图的信息；如果 name 为地图名字，则查找该地图的相关信息；支持模糊查找，即查找名字包含 name 的所有地图信息。代码与查找车辆信息类似，故不赘述。

#### 7、获取用户信息

查询 player 表，获取名字为 name 的用户的所有信息。

```

1. def getUserInfo(name):
2.     cursor.execute("""
3.         select *
4.         from player
5.         where name = ?
6.     """, name)
7.     row = cursor.fetchone()
8.     userinfo = {
9.         'name': name,
10.        'current_experience': row.current_experience,

```

```

11.         'experience_to_upgrade': row.current_level * 1000 - row.current_ex
           perience,
12.         'current_level': row.current_level,
13.         'current_charming_level': row.current_charming_level,
14.         'player_level_1': row.player_level1,
15.         'player_level_2': row.player_level2,
16.         'player_level_3': row.player_level3,
17.         'money': row.money
18.     }
19.     cursor.execute("""
20.         select *
21.         from fleet_of_player
22.         where player_name = ?
23.     """, name)
24.     fleet_name = cursor.fetchone()
25.     if fleet_name != None:
26.         userinfo['team'] = fleet_name.fleet_name
27.     return userinfo

```

## 8、获取车队信息

查询 fleet\_of\_player 表、fleet 表、player 表，获取车队信息以及车队成员的所有信息。此部分与第 7 部分代码相似，所以不贴在报告里了，详情请见具体代码。

## 9、获取物品信息

如果 type 为 user-car，则查询用户 name 当前驾驶的车辆信息以及他所拥有车辆的信息；如果 type 为 user-pet，则查询用户 name 当前携带的宠物的信息以及他所拥有宠物的信息；如果 type 为 user-cloth，则查询用户 name 当前穿戴的服装的信息以及他所拥有服装的信息。

```

1. def getItem(name, type):
2.     current = None
3.     item_list = []
4.     if type == 'user-car':
5.         cursor.execute("""
6.             select *
7.             from cars_of_player, car
8.             where player_name = ? and car.car_name = cars_of_player.car_na
           me
9.         """, name)
10.        rows = cursor.fetchall()
11.        for row in rows:
12.            if row.is_driving == 1:
13.                current = {
14.                    'name': row.car_name,
15.                    'level': row.car_level,

```

```

16.         'charming': row.car_charmingvalue,
17.         'description': row.car_description,
18.         'price': row.car_price
19.     }
20.     else:
21.         item_list.append({
22.             'name': row.car_name,
23.             'level': row.car_level,
24.             'charming': row.car_charmingvalue,
25.             'description': row.car_description,
26.             'price': row.car_price
27.         })
28.     elif type == 'user-pet':
29.         cursor.execute("""
30.             select *
31.             from pet, pets_of_player
32.             where player_name = ? and pet.pet_name = pets_of_player.pet_na
me
33.             """, name)
34.         rows = cursor.fetchall()
35.         for row in rows:
36.             if row.is_taking == 1:
37.                 current = {
38.                     'name': row.pet_name,
39.                     'charming': row.pet_charmingvalue,
40.                     'description': row.pet_description,
41.                     'price': row.pet_price,
42.                     'level': row.pet_level,
43.                     'experience': row.pet_experience
44.                 }
45.             else:
46.                 item_list.append({
47.                     'name': row.pet_name,
48.                     'charming': row.pet_charmingvalue,
49.                     'description': row.pet_description,
50.                     'price': row.pet_price,
51.                     'level': row.pet_level,
52.                     'experience': row.pet_experience
53.                 })
54.     elif type == 'user-cloth':
55.         cursor.execute("""
56.             select *
57.             from clothes, clothes_of_player
58.             where player_name = ? and clothes.clothes_name = clothes_of_pl
ayer.clothes_name
59.             """, name)
60.         rows = cursor.fetchall()
61.         for row in rows:
62.             if row.iswearing == 1:
63.                 current = {
64.                     'name': row.clothes_name,
65.                     'charming': row.clothes_charmingvalue,
66.                     'price': row.clothes_price
67.                 }
68.             else:

```

```

69.         item_list.append({
70.             'name': row.clothes_name,
71.             'charming': row.clothes_charmingvalue,
72.             'price': row.clothes_price
73.         })
74.     if current == None and len(item_list) == 0:
75.         return {}
76.     elif current == None:
77.         return {'item-list': item_list}
78.     return {'current': current, 'item-list': item_list}

```

## 10、删除成员

从 fleet\_of\_player 表中删除 fleet\_name 车队的 player\_name 成员。

```

1. def removePlayer(fleet_name, player_name):
2.     cursor.execute("""
3.         select player_name
4.         from fleet_of_player
5.         where fleet_name = ?
6.     """, fleet_name)
7.     rows = cursor.fetchall()
8.     notin = True
9.     for row in rows:
10.        if row.player_name == player_name:
11.            notin = False
12.    if notin:
13.        return False
14.    cursor.execute("""
15.        delete from fleet_of_player
16.        where fleet_name = ? and player_name = ?
17.    """, fleet_name, player_name)
18.    cnxn.commit()
19.    return True

```

## 11、购买物品

购买类型为 type（车辆、宠物、服装）的商品，即在相应的表（cars\_of\_player、pets\_of\_player、clothes\_of\_player）中插入新的元组，插入后将触发相应的触发器以更新用户 user 的点券。

```

1. def buy(type, name, user):
2.     cursor.execute("""
3.         select money
4.         from player
5.         where name = ?
6.     """, user)
7.     user_money = cursor.fetchone().money

```

```

8.     if type == 'car':
9.         cursor.execute("""
10.             select car_price
11.             from car
12.             where car_name = ?
13.         """, name)
14.         car_price = cursor.fetchone().car_price
15.         if user_money < car_price:
16.             return False
17.         cursor.execute("""
18.             insert into cars_of_player
19.             values (?, ?, 0)
20.         """, user, name)
21.     elif type == 'cloth':
22.         cursor.execute("""
23.             select clothes_price
24.             from clothes
25.             where clothes_name = ?
26.         """, name)
27.         clothes_price = cursor.fetchone().clothes_price
28.         if user_money < clothes_price:
29.             return False
30.         cursor.execute("""
31.             insert into clothes_of_player
32.             values (?, ?, 0)
33.         """, user, name)
34.     elif type == 'pet':
35.         cursor.execute("""
36.             select pet_price
37.             from pet
38.             where pet_name = ?
39.         """, name)
40.         pet_price = cursor.fetchone().pet_price
41.         if user_money < pet_price:
42.             return False
43.         cursor.execute("""
44.             insert into pets_of_player
45.             values (?, ?, 0, 0, 0)
46.         """, user, name)
47.     cnxn.commit()
48.     return True

```

## 12、装备或卸下物品

如果 name 为 None，则表示卸下物品；如果 name 为物品名字，则表示装备物品；类型 type（车辆、宠物、服装）表示装备或卸下物品的类型。

```

1. def dress(type, name, user):
2.     if type == 'car':
3.         is_driving = 1
4.         if name == None:

```



```

5.         is_driving = 0
6.         cursor.execute("""
7.             select car_name
8.             from cars_of_player
9.             where player_name = ? and is_driving = 1
10.        """, user)
11.         name = cursor.fetchone().car_name
12.         cursor.execute("""
13.             update cars_of_player
14.             set is_driving = ?
15.             where player_name = ? and car_name = ?
16.        """, is_driving, user, name)
17.     elif type == 'pet':
18.         is_taking = 1
19.         if name == None:
20.             is_taking = 0
21.             cursor.execute("""
22.                 select pet_name
23.                 from pets_of_player
24.                 where player_name = ? and is_taking = 1
25.            """, user)
26.             name = cursor.fetchone().pet_name
27.             cursor.execute("""
28.                 update pets_of_player
29.                 set is_taking = ?
30.                 where player_name = ? and pet_name = ?
31.            """, is_taking, user, name)
32.     elif type == 'cloth':
33.         iswearing = 1
34.         if name == None:
35.             iswearing = 0
36.             cursor.execute("""
37.                 select clothes_name
38.                 from clothes_of_player
39.                 where player_name = ? and iswearing = 1
40.            """, user)
41.             name = cursor.fetchone().clothes_name
42.             cursor.execute("""
43.                 update clothes_of_player
44.                 set iswearing = ?
45.                 where player_name = ? and clothes_name = ?
46.            """, iswearing, user, name)
47.     cnxn.commit()
48.     return True

```

### 13、添加比赛记录

在 maps\_of\_player 表中添加或更新元组表示添加新的比赛记录。每次添加比赛记录将根据赛道的星级增加用户的经验值、金币、段位积分、宠物经验值、车队贡献和车队段位积分，同时将触发相应的触发器以更新用户的等级、段位、宠物等级以及车队段位。

```

1. def addRace(lane, user, score):
2.     cursor.execute("""
3.         select *
4.         from maps_of_player
5.         where map_name = ? and player_name = ?
6.     """, lane, user)
7.     row = cursor.fetchone()
8.     if row == None:
9.         cursor.execute("""
10.             insert into maps_of_player
11.             values (?, ?, ?)
12.         """, user, lane, score)
13.     elif score < row.record:
14.         cursor.execute("""
15.             update maps_of_player
16.             set record = ?
17.             where player_name = ? and map_name = ?
18.         """, score, user, lane)
19.     cursor.execute("""
20.         select *
21.         from map
22.         where map_name = ?
23.     """, lane)
24.     map_star_level = cursor.fetchone().star_level
25.     money = map_star_level * 500
26.     player_experience = map_star_level * 1000
27.     player_victory = map_star_level * 25
28.     fleet_contribution = map_star_level * 100
29.     fleet_victory = map_star_level * 10
30.     pet_experience = map_star_level * 100
31.     cursor.execute("""
32.         update player
33.         set money += ?, current_experience += ?, player_level3 += ?
34.         where name = ?
35.     """, money, player_experience, player_victory, user)
36.     cursor.execute("""
37.         select *
38.         from fleet_of_player
39.         where player_name = ?
40.     """, user)
41.     row = cursor.fetchone()
42.     if row != None:
43.         fleet_name = row.fleet_name
44.         cursor.execute("""
45.             update fleet_of_player
46.             set contribution += ?
47.             where player_name = ?
48.         """, fleet_contribution, user)
49.         cursor.execute("""
50.             update fleet
51.             set fleet_level3 += ?
52.             where fleet_name = ?
53.         """, fleet_victory, fleet_name)
54.     cursor.execute("""
55.         select *

```

```

56.         from pets_of_player
57.         where player_name = ? and is_taking = 1
58.     """ , user)
59.     row = cursor.fetchone()
60.     if row != None:
61.         pet_name = row.pet_name
62.         cursor.execute("""
63.             update pets_of_player
64.             set pet_experience += ?
65.             where player_name = ? and pet_name = ?
66.         """ , pet_experience, user, pet_name)
67.     cnxn.commit()
68.     return True

```

## UI 开发

### 实现技术

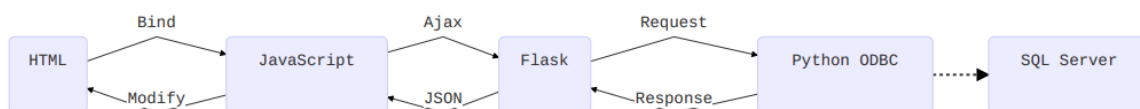
客户端本体是一个 Web 应用，分为前后端。前端与后端之间通过 ajax 实现异步通信，以 JSON 为标准数据交换格式。后端向一个 Python ODBC 应用提交数据请求并获取返回值。

前端布局：HTML5+CSS

前端逻辑：JavaScript

后端逻辑：Flask（基于 Python 的 Web 框架）

如下图所示：



### 功能特性

#### 功能

- 用户可以登录、注册。

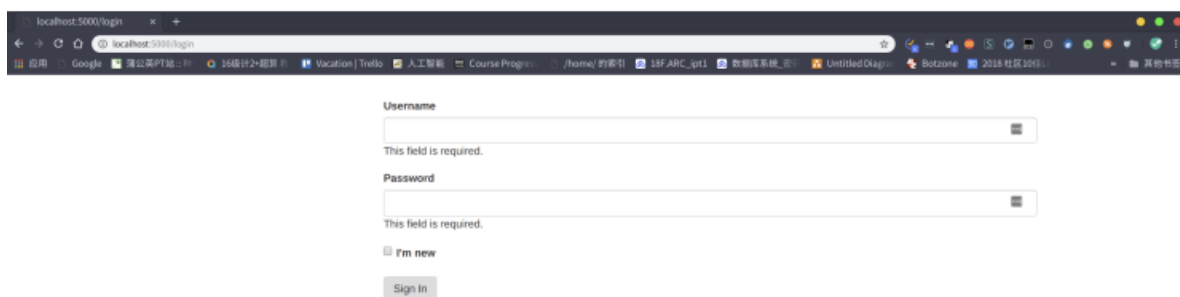
- 用户有等级、魅力值与金钱属性，并随用户的操作动态变化。
- 用户可以购买车辆、配饰与宠物。
- 用户可以在已经购买的车辆、配饰和宠物中选择装备。
- 用户可以加入车队，车队队长可以踢出玩家。
- 用户可以选择地图并录入自己的成绩。
- 所有实体均支持模糊查找搜索。

## 特性

- 前端通过模板引擎与 JavaScript 动态渲染，交互过程全程不重新加载，用户体验具有连贯性。
- 前后端通信全部异步化，用户操作提交不影响用户的正常浏览，操作提交后页面元素实时改变，用户体验具有一致性。
- 逻辑高度解耦合，后端模块化设计，可以方便地增减特性并无需大幅修改代码和数据库系统的结构。

## 实验结果展示

### 登录/注册



Username

This field is required.

Password

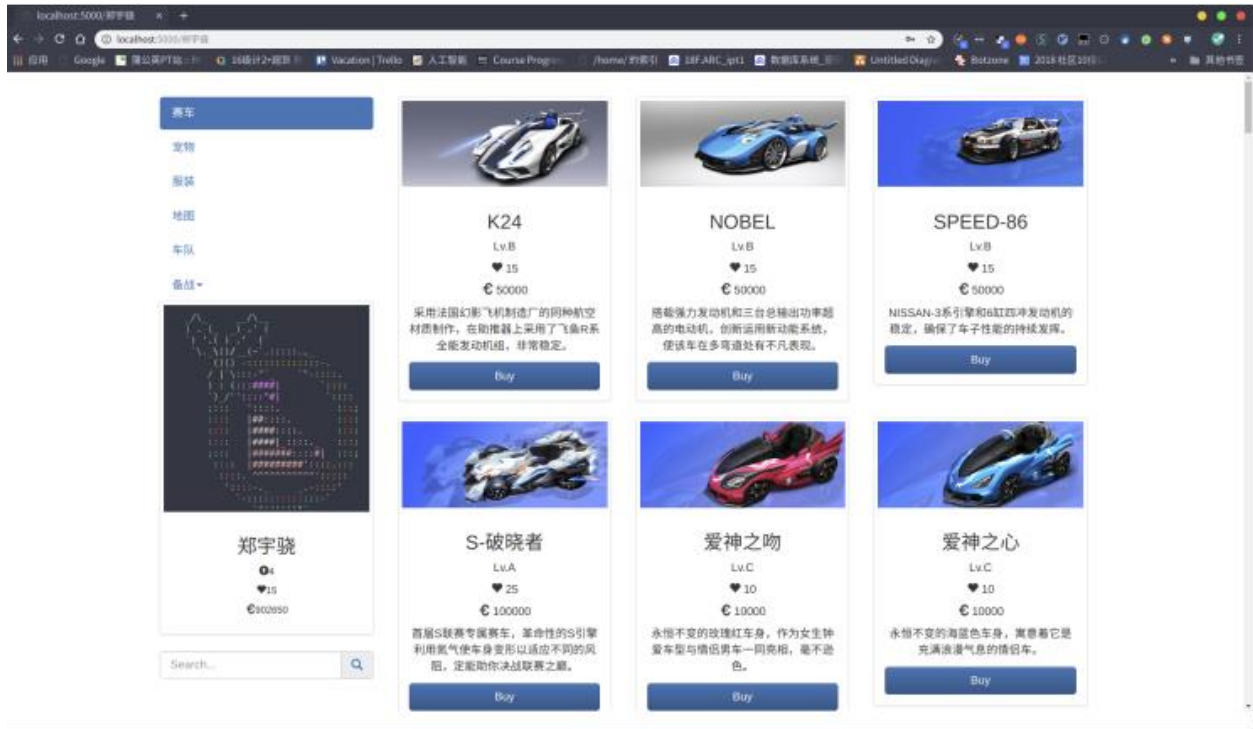
This field is required.

☐ I'm new

Sign In

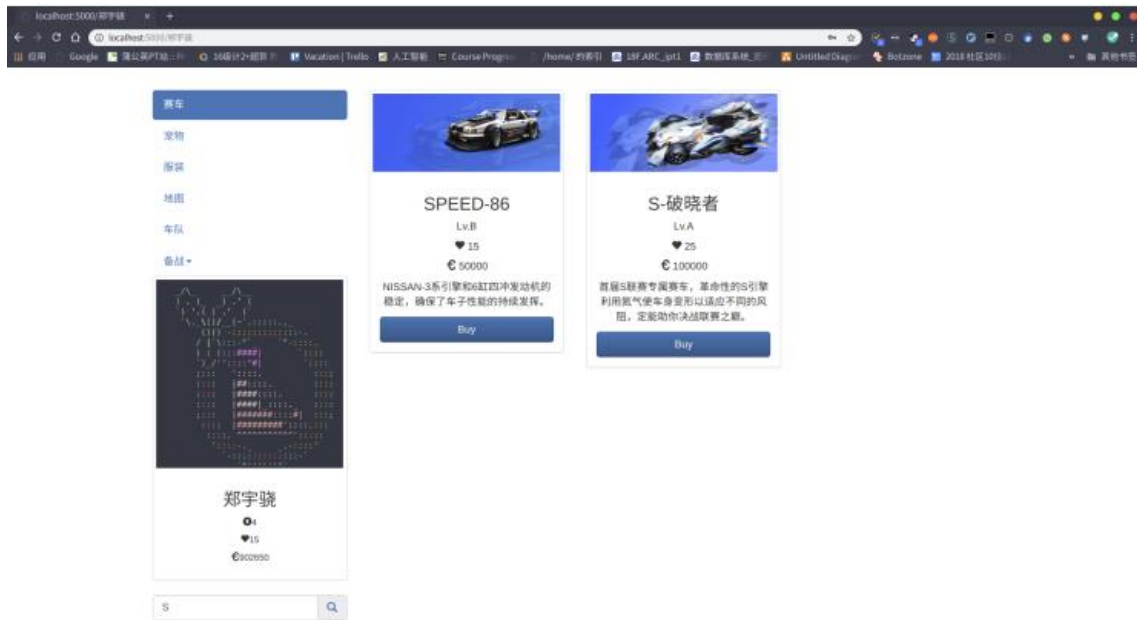
## 商城界面

左侧是类别标签选择，下方是玩家信息（头像、昵称、等级、魅力值与货币）



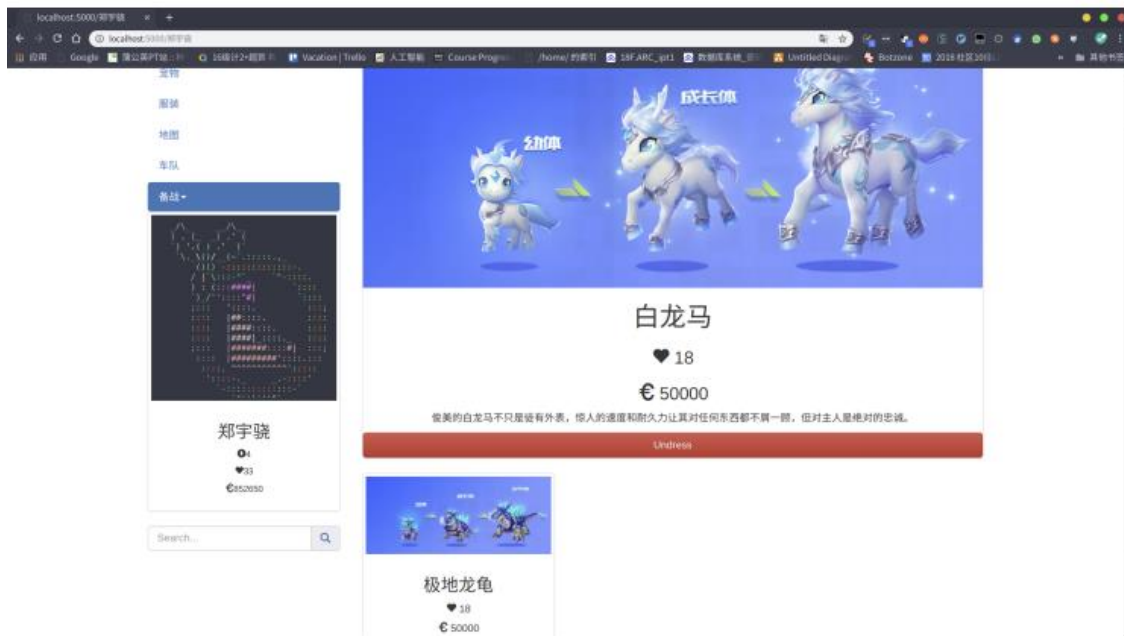
## 模糊查找

查找所有名称含有‘S’的赛车：



## 个人物品界面

可以通过点击 Dress/Undress 来换上/脱下物品，同时自身的魅力值会改变（见图一与图二魅力值的不同）





## 成绩录入界面

录入成绩后，自身的经验值和金钱会改变。如果达到升级要求，等级会改变。（见图一与图二金钱的不同）







## 实验总结

### 一、在数据库结构设计和功能设计方面：来自队员郑映雪

就设计来说，因为想尽量往一个真实的游戏数据库方向靠拢，而一个游戏数据库的结构是复杂的，实体较多，关系也较多，一不小心就会造成属性的冗余。所以，在初版设计之后，还进行了细细斟酌，仔细思考某一个属性是否必要，尤其是关系中的某一个属性是否必要，删去了不少冗余的属性，才最终形成了现在这个数据库的结构。

触发器的设计方面需要更加细心，因为游戏在等级、段位、金钱、魅力值上都是有严格的设定的，尤其在升级模块，要严防各种 bug，所以触发器是比较多的。

功能方面，我考虑到数据库的基本功能都需要得到体现——增、删、改、查。增则是通过设计商城功能，以购入赛车、宠物或衣服的功能体现；删是通过车队队长可以把车队队员踢出来体现；改是通过等级升级、魅力值增加等自身数据的修改（来源是触发器）来进行的；查则是实现了所有模块都可以使用的搜索框来体现的，此时的搜索框支持模糊搜索，按照游戏里真实的搜索框来实现。如此设计，既能完善数据库的功能，也能完整地体现基本的增删改查的功能。



## 二、在数据库后端搭建方面：来自队员朱志儒：

通过这次大作业，我掌握了基于 ODBC 驱动的数据库应用开发方法。在这次大作业中，我使用的是 python 的 pyodbc 库，使用游标创建基本表和设计好的触发器，创建基本表时，借鉴了实验一中的经验；在创建触发器时，借鉴了实验四的经验，使用 `instead of` 代替 `before`，从而创建 `before` 类型的触发器；使用 `inserted` 表示在插入完成后存储所插入行的值，使用 `deleted` 存储已经更新或删除行的旧值，从而替代 `referencing new row as` 语句和 `referencing old row as` 语句。根据前端的需要和之前的设计，我实现相应的应用功能以便前端使用，在实现这些功能时，我使用了预备语句，这既便于实现相应的功能，又能防止 SQL 注入对数据库造成不可挽回的破坏。

## 三、在数据库客户端前端设计方面：来自队员郑宇骁

这次项目是对所有已学知识的一次综合应用，通过这次项目的实践，我对以下几点数据库的知识又有了更好的理解。

数据库设计不能生搬硬套，需要结合情况。一板一眼完全死板地按照范式要求进行设计未必能设计出最合适的数据库，需要结合数据库应用的实际情况，如频繁的应用场景和应用程序构建的实际，设计开发上方便、使用上高效的数据库结构。

数据库系统的安全性应当注意。实际交互时，后端逻辑应当与数据库访问逻辑分离，应用逻辑不应直接操作数据库，以避免通过客户端对数据库进行恶意破坏。

充分的测试是必要的。在开发过程中，很多设计时没有预料到的隐藏问题都是在测试过程中暴露出来的。早发现问题才能早解决问题。

应当留意不同数据库系统软件的不同实现。并非所有数据库系统软件都严格遵守 SQL 的所有规定，有部分数据库系统软件也会实现 SQL 没有规定的功能。因此在进行数据库设计时，应考虑到目标数据库系统软件的行为，以期达到预期的目的并简化开发过程。