

---

# Trading Card Game

## 系统实现总结报告

12061036 李明浩

---

## 目录

一、	实现环境.....	1
二、	系统功能结构图.....	1
三、	数据库设计.....	1
(一)	tbl_User 表.....	1
1.	定义基本表.....	1
2.	定义完整性约束.....	2
3.	定义索引.....	2
(二)	tbl_Card 表.....	2
1.	定义基本表.....	2
2.	定义完整性约束.....	3
3.	定义索引.....	3
(三)	tbl_HoldCard 表.....	3
1.	定义基本表.....	3
2.	定义完整性约束.....	3
3.	定义索引.....	3
(四)	tbl_Pack 表.....	4
1.	定义基本表.....	4
2.	定义完整性约束.....	4
3.	定义索引.....	4
(五)	tbl_Rank 表.....	4
1.	定义基本表.....	4
2.	定义完整性约束.....	5
3.	定义索引.....	5
四、	系统安全性设计.....	5
(一)	基本表的访问.....	5
1.	tbl_User 表.....	5
2.	tbl_Card 表.....	5
3.	tbl_HoldCard 表.....	5
4.	tbl_Rank 表.....	6
5.	tbl_Pack 表.....	6
(二)	数据库安全性控制.....	6
五、	可编程性代码说明.....	6
(一)	触发器.....	6
1.	trg_User_I.....	6

---

2.	trg_Card_I .....	6
3.	trg_Card_D .....	7
4.	trg_HoldCard_I .....	7
5.	trg_Rank_I .....	7
(二)	函数.....	8
1.	fun_NormsDist.....	8
2.	fun_Integral.....	8
3.	fun_Integral.....	9
(三)	存储过程.....	9
1.	prd_BuyCard.....	9
2.	prd_CalcPowerSum .....	10
3.	prd_Event_GetCard .....	10
4.	prd_Event_LoseCard .....	11
5.	prd_GenCardShop.....	11
6.	prd_GenRand .....	12
7.	prd_GenHoldCard.....	12
8.	prd_InitUser .....	12
9.	prd_NewCard.....	12
10.	prd_OpenPack.....	12
11.	prd_RarityDistribution.....	13
12.	prd_SaveCard.....	13
13.	prd_SellCard .....	14
六、	主要技术.....	14
七、	运行示例.....	14
(一)	注册与登录.....	14
1.	注册.....	14
2.	登录.....	14
(二)	游戏本体.....	15
1.	游戏主界面.....	15
2.	开包.....	15
3.	医院.....	15
4.	比赛.....	16
5.	随机事件.....	16
6.	排行榜.....	16
7.	罕贵分布图.....	17

---

8.    卡片管理.....	17
八、    源程序简要说明.....	18
(一)  游戏主界面 Form1.cs.....	18
(二)  注册与登录界面 Login.cs.....	19
(三)  与数据库的交互.....	20
1.    ExecuteNonQuery 方法.....	20
2.    ExecuteReader 方法结合 SqlDataReader.....	21
3.    应用 DataAdapter 以及 DataSet .....	22
九、    感想与收获.....	22
十、    游戏说明书.....	23
(一)  游戏目标.....	23
(二)  游戏方法.....	23
1.    注册与登录.....	23
2.    卡片的交易.....	23
3.    事件的发生.....	23
4.    开包的惊喜.....	23
5.    比赛与医院.....	24
(三)  罕贵解释.....	24
(四)  其他.....	24
参考文献.....	24

## 一、 实现环境

操作系统: Windows 7 SP1

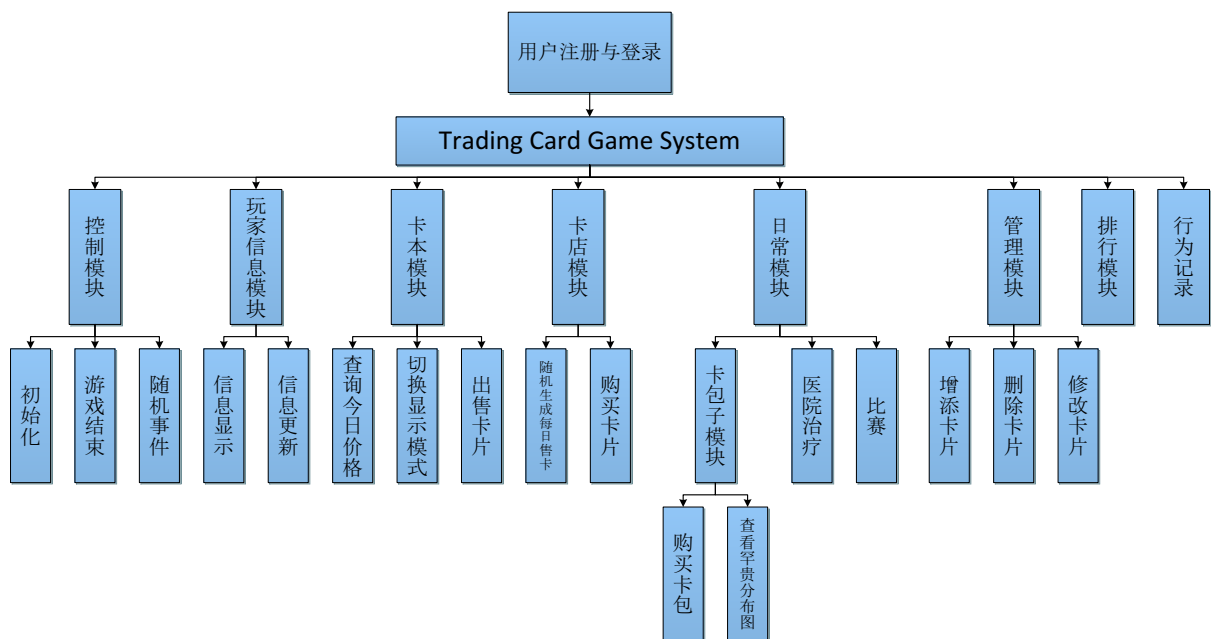
Microsoft SQL Server 版本: 12.0.2000.8

使用 IDE: Microsoft Visual Studio Ultimate 2012

开发语言: C#

框架: Microsoft .NET Framework 4.0

## 二、 系统功能结构图



## 三、 数据库设计

### (一) tbl\_User 表

#### 1. 定义基本表

tbl_User			
	列名	简洁类型	可以为 Null
	User_ID	int	否
	User_Name	varchar(20)	否
	User_Password	varchar(32)	否
	User_HP	int	否
	User_Money	int	否
	User_GameDate	int	否
	User_RegDate	datetime	否
	User_ManageLev...	int	否

## 2. 定义完整性约束

PK: User\_ID

CHECK:

CK\_tbl\_User: User\_ID > 0

CK\_tbl\_User\_1: User\_HP >= 0

CK\_tbl\_User\_2: User\_Money >= 0

CK\_tbl\_User\_3: User\_GameDate >= 0 AND User\_GameDate <= 52

## 3. 定义索引

PK\_\_tbl\_User\_\_C5B19602CEBD4720: Clustered on User\_ID

## (二) tbl\_Card 表

### 1. 定义基本表

tbl_Card			
	列名	简洁类型	可以为 Null
	Card_ID	int	否
	Card_Name	varchar(30)	否
	Card_Pow...	int	否
	Card_Price	int	否
	Card_Pack	int	否
	Card_Rarity	varchar(3)	否

## 2. 定义完整性约束

PK: Card\_ID

FK: Card\_Pack References tbl\_Pack(Pack\_ID)

CHECK:

CK\_tbl\_Card: Card\_ID > 0

CK\_tbl\_Card\_1: Card\_Power >= 0


CK\_tbl\_Card\_2: Card\_Price >= 0

## 3. 定义索引

PK\_\_tbl\_Card\_\_C1F8DC594D201248: Clustered on Card\_ID

## (三) tbl\_HoldCard 表

### 1. 定义基本表

tbl_HoldCard			
	列名	简洁类型	可以为 Null
	HoCa_UserID	int	否
	HoCa_CardID	int	否
	HoCa_Quantity	int	否

### 2. 定义完整性约束

PK: (HoCa\_UserID, HoCa\_CardID)

FK:

HoCa\_UserID References tbl\_User(User\_ID)

HoCa\_CardID References tbl\_Card(Card\_ID)

CHECK:

CK\_tbl\_HoldCard: HoCa\_Quantity > 0

### 3. 定义索引

PK\_\_tbl\_Hold\_\_4E2E6D3990681D9E: Clustered on (HoCa\_UserID, HoCa\_CardID)

#### (四) tbl\_Pack 表

##### 1. 定义基本表

tbl_Pack			
	列名	简洁类型	可以为 Null
	Pack_ID	int	否
	Pack_Name	varchar(...	否
	Pack_Date	int	否
	Pack_Price	int	否

##### 2. 定义完整性约束

PK: Pack\_ID

CHECK:

CK\_tbl\_Pack: Pack\_ID > 0

CK\_tbl\_Pack\_1: Pack\_Price >= 0

##### 3. 定义索引

PK\_\_tbl\_Pack\_\_AA6923075CA0D32A: Clustered on Pack\_ID

#### (五) tbl\_Rank 表

##### 1. 定义基本表

tbl_Rank			
	列名	简洁类型	可以为 Null
	Rank_Key	int	否
	Rank_Number	int	否
	Rank_ID	int	否
	Rank_Money	int	否
	Rank_HP	int	否
	Rank_Time	datetime	否



## 2. 定义完整性约束

PK: Rank\_Key

FK: Rank\_ID References tbl\_User(User\_ID)

CHECK:

CK\_tbl\_Rank: Rank\_Number  $\geq 0$

CK\_tbl\_Rank\_1: Rank\_Money  $\geq 0$

CK\_tbl\_Rank\_2: Rank\_HP  $\geq 0$

## 3. 定义索引

PK\_tbl\_Rank: Clustered on Rank\_Key

# 四、 系统安全性设计

## (一) 基本表的访问

### 1. tbl\_User 表

sa 以外的用户对于此表有增加、修改、查询的权限。

增加只能靠注册界面进行，同时用户所填信息（用户名、密码）将经过系统前端的合法性检查（用户名不能为空、不能重复，密码长度至少 6 位）后，插入此表中。密码在前端被 MD5 加密，保证了玩家信息的安全。若不进行注册与登录，则无法进行任何后继行为。

修改与查询是针对游戏主界面玩家信息的实时更新进行的，相关方法均为 private，可以拒绝非法的访问。

综合来看，系统前端的合法性检查、限制，以及后端相应触发器行为及完整性约束，可以保证表中数据的合法性及安全性。

### 2. tbl\_Card 表

普通玩家对于此表只有查询的权限，不会对表中信息安全造成影响；含有管理权限的管理员用户则可以对此表进行增加、删除、更新、查询的所有操作，但是一方面管理员用户是由 sa 指定的，作为此表的维护者，有较高的信用度；另一方面系统前端也对数据合法性进行了检查与限制，因此安全性也是有保障的。

### 3. tbl\_HoldCard 表

玩家对于此表将会进行频繁的增加、删除、更新与查询，所有行为均通过相应的存储过程进行，封装性良好。

#### 4. tbl\_Rank 表

玩家对于此表有增加、查询的权限。触发器的作用保证了数据的有序性以及合法性。

#### 5. tbl\_Pack 表

所有用户对此表均只有查询的权限，只有 sa 具有所有权限。

## (二) 数据库安全性控制

本系统使用了用户标识与鉴别、较为简单的合法权限检查以及数据加密技术，未采用自主存取控制、视图、审计等机制。

## 五、 可编程性代码说明

### (一) 触发器

#### 1. trg\_User\_I

```
CREATE trigger [dbo].[trg_User_I] on [dbo].[tbl_User] instead of insert as
declare @id int, @name varchar(20), @password varchar(32), @i int;
select @name = User_Name, @password = User_Password from inserted;

if not exists(select * from tbl_User)
    set @id = 1;
else begin
    set @i = 1;
    while exists (select * from tbl_User where User_ID = @i)
        set @i = @i + 1;
    set @id = @i;
end

insert into tbl_User(User_ID, User_Name, User_Password, User_HP, User_Money, User_GameDate, User_RegDate, User_ManageLevel)
values (@id+1, @name, @password, 100, 2500, 0, getDate(), 0);

GO
```

本触发器主要用于自动生成合法的用户 ID 以及用户游戏数据的初始化。由于 User\_ID 是主键，在确定值前无法进行插入操作，因此是 instead of 触发器。

ID 的生成原理是从最小值 1 开始逐渐递增向后寻找，找到的第一个不存在的 ID 即作为新用户的 ID（若初始表中无任何用户，则 ID 置为 1）。通过此方法创建的用户管理权限为 0。

#### 2. trg\_Card\_I

```

CREATE trigger [dbo].[tgr_Card_I] on [dbo].[tbl_Card]
instead of insert
as
begin
declare @card_id int, @card_name varchar(30), @card_power int, @card_price int, @card_rarity varchar(3), @card_pack int, @i int;
select @card_name = Card_Name, @card_power = Card_Power, @card_price = Card_Price, @card_rarity = Card_Rarity, @card_pack = Card_Pack from inserted;
if not exists(select * from tbl_Card)
set @card_id = 1;
else begin
set @i = 1;
while exists (select * from tbl_Card where Card_ID = @i)
set @i = @i + 1;
set @card_id = @i;
end
insert into tbl_Card values(@card_id, @card_name, @card_power, @card_price, @card_pack, @card_rarity);
end
GO

```

本触发器原理同上，作用是为自动生成合法的卡片 ID。触发器的效果可以清晰地 在卡片管理模块显示出来。

### 3. trg\_Card\_D

```

CREATE trigger [dbo].[tgr_Card_D] on [dbo].[tbl_Card]
instead of delete
as
begin
declare @card_id int, @i int;
select @card_id = Card_ID from deleted;
delete from tbl_HoldCard where HoCa_CardID = @card_id;
delete from tbl_Card where Card_ID = @card_id;
end
GO

```

由于卡片的直接删除会导致 tbl\_HoldCard 表内产生外键冲突，因此要先从 tbl\_HoldCard 表内删除所有有关此卡的条目，再从 tbl\_Card 中删除此卡。

### 4. trg\_HoldCard\_I

```

CREATE trigger [dbo].[trg_HoldCard_I] on [dbo].[tbl_HoldCard] instead of insert
as
declare @card_id int, @user_id int, @quantity int;
declare mycursor cursor for select HoCa_UserID, HoCa_CardID, HoCa_Quantity from inserted;
open mycursor;
fetch next from mycursor into @user_id, @card_id, @quantity
while @@fetch_status = 0
begin
if exists(select * from tbl_HoldCard where @card_id = HoCa_CardID and @user_id = HoCa_UserID)
begin
update tbl_HoldCard set HoCa_Quantity = HoCa_Quantity + @quantity where @card_id = HoCa_CardID and @user_id = HoCa_UserID;
end
else
begin
insert into tbl_HoldCard values(@user_id, @card_id, @quantity);
end
fetch next from mycursor into @user_id, @card_id, @quantity
end
close mycursor;
deallocate mycursor;
GO

```

本触发器的作用主要是重复性判断。对于插入的卡片表，通过游标的方式逐一在 tbl\_HoldCard 中查询是否该用户已经持有此卡，来决定是执行 update 还是 insert 操作。

### 5. trg\_Rank\_I

```

CREATE trigger [dbo].[tgr_Rank_I] on [dbo].[tbl_Rank]
instead of insert as
begin
    declare @user_id int, @user_hp int, @user_money int, @key int;
    select @user_id = Rank_ID from inserted;
    if exists(select * from tbl_Rank)
        select @key = MAX(Rank_Key) + 1 from tbl_Rank;
    else
        set @key = 1;
    select @user_hp = User_HP, @user_money = User_Money from tbl_User where @user_id = User_ID;
    insert into tbl_Rank values(@key, 0, @user_id, @user_money, @user_hp, GETDATE());
    declare @i int, @j int, @rank int, @money int;
    set @j = 1;
    select @i = count(*) from tbl_Rank;
    DECLARE mycursor CURSOR FOR SELECT Rank_Key, Rank_Money FROM tbl_Rank;
    OPEN mycursor;
    while @j <= @i
    begin
        FETCH NEXT FROM mycursor into @key, @money;
        select @rank = COUNT(*) + 1 from tbl_Rank where @money < Rank_Money;
        update tbl_Rank set Rank_Number = @rank where @key = Rank_Key;
        set @j = @j + 1;
    end
    CLOSE mycursor;
    DEALLOCATE mycursor;
end
GO

```

本触发器目的是在新的记录加入排行榜时，根据 Rank\_Money 的大小更新所有项的排名(Rank\_Number); 同时自动生成合法的 Rank\_ID，作为主键的同时可以用于游标的访问。更新排名的原理是通过游标访问 tbl\_Rank 表中每一项，将 Rank\_Num 更新为比这一项 Rank\_Money 大的数目加 1，即为排名。

## (二) 函数

### 1. fun\_NormsDist

```

CREATE function [dbo].[fun_NormsDist] (@x float, @miu float, @delta float)
returns float
as
begin
    declare @result float;
    set @result = EXP((-1)*(POWER(@x-@miu, 2)/(2*POWER(@delta, 2))))/(@delta*SQRT(2*PI()));
    return @result;
end
GO

```

本函数对应的公式为：

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

通过给定的 $x$ 、 $\mu$ 和 $\sigma$ 值，计算相应的正态分布概率密度。

### 2. fun\_Integral

```

CREATE function [dbo].[fun_Integral] (@x float, @delta float)
returns float
as
begin
    declare @y float;
    declare @i float;
    set @y = 0;
    set @i = -10;
    while @i < @x
    begin
        set @y = @y + 0.01 * dbo.fun_NormsDist(@i, 0.5, @delta);
        set @i = @i + 0.01;
    end
    return @y;
end
GO

```

本函数目的是计算当 $\mu = 0.5$ 时，在给定的 $\sigma$ 下自变量取得 $x$ 值的概率，即实现如下公式：

$$F(x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x e^{-\frac{(t-0.5)^2}{2\sigma^2}} dt$$

不过由于 SQL 语句功能限制，无法实现积分功能，因此通过代码中的循环累加方式进行积分的模拟（由于 $f(-10)$ 在 $\mu = 0.5$ 时已经足够小，因此从-10 开始循环至  $x$ ）。

### 3. fun\_Integral

```

create function [dbo].[fun_CalcFactor] (@min float, @max float, @rand float)
returns float
as
begin
    declare @factor float
    declare @r float;
    set @factor = @min + (@max-@min)*dbo.fun_Integral(@rand, 0.2);
    return @factor;
end
GO

```

本函数的目的是计算卡片价格浮动的因子。原理是通过 $@rand$  给定一个 0 至 1 之间的浮点数，通过 fun\_Integral 函数计算出 $\sigma = 0.2$ 时此随机数对应的概率，再调整至 $@min$ 和 $@max$  之间。即通过 $[0,1]$ 上的均匀分布（将 rand()视为均匀分布）映射至 $[@min, @max]$ 上的正态分布，保证了较高概率价格波动幅度小，较低概率价格波动幅度大——即达到产生每日卡片单价浮动的效果。

## (三) 存储过程

### 1. prd\_BuyCard

```

CREATE proc [dbo].[prd_BuyCard]
@card_id int,
@buy_num int,
@buy_price int,
@user_id int
as
begin
declare @now_money int, @new_money int;
declare @now_num int, @new_num int;
select @now_money = User_Money from tbl_User where @user_id = User_ID;
set @new_money = @now_money - @buy_price * @buy_num;
update tbl_User set User_Money = @new_money where User_ID = @user_id;

if exists (select * from tbl_HoldCard where HoCa_UserID = @user_id and HoCa_CardID = @card_id)
begin
select @now_num = HoCa_Quantity from tbl_HoldCard where HoCa_UserID = @user_id and HoCa_CardID = @card_id;
set @new_num = @now_num + @buy_num;
update tbl_HoldCard set HoCa_Quantity = @new_num where HoCa_UserID = @user_id and HoCa_CardID = @card_id;
end
else
begin
insert into tbl_HoldCard values(@user_id, @card_id, @buy_num);
end
end
GO

```

本存储过程应用于购买卡片。先从 tbl\_User 表的 User\_Money 扣除相应的金钱（已在前端保证完整性），再根据 tbl\_HoldCard 表中玩家是否已拥有本种类卡片来决定 insert 还是 update。

## 2. prd\_CalcPowerSum

```

CREATE proc [dbo].[prd_CalcPowerSum]
@user_id int
as
declare @multpower int;
set @multpower = 0;
create table tbl_Temp (Temp_Power int, Temp_Quantity int, Temp_Mult int);
insert into tbl_Temp
select Card_Power, HoCa_Quantity, Card_Power*HoCa_Quantity from tbl_Card, tbl_HoldCard where @user_id = HoCa_UserID and Card_ID = HoCa_CardID;
select @multpower = SUM(Temp_Mult) from tbl_Temp;
drop table tbl_Temp;
return @multpower;
GO

```

本存储过程应用于求某用户所持卡片战斗力总和，将此值返回。实现中建立了临时表用于存放单卡战斗力与卡片张数的乘积，用以求和。

## 3. prd\_Event\_GetCard

```

CREATE proc [dbo].[prd_Event_GetCard]
@user_id int,
@card_name varchar(30) out,
@card_num int out,
@card_rarity varchar(3) out
as
begin
declare @card_id int;
set @card_num = floor(rand() * 5) + 1;
Select top 1 @card_id = Card_ID, @card_name = Card_Name, @card_rarity = Card_Rarity From tbl_Card Order By newid();
insert into tbl_HoldCard values (@user_id, @card_id, @card_num);
end
GO

```

本存储过程用于随机事件中的获得卡片。原理是用 newid()随机从 tbl\_Card 表中选出 1 种卡片加入 tbl\_HoldCard 表中（含防重复的插入触发器）。

#### 4. prd\_Event\_LoseCard

```
CREATE proc [dbo].[prd_Event_LoseCard]
@user_id int,
@card_name varchar(30) out,
@losecard_num int out,
@card_rarity varchar(3) out
as
begin
    if exists(select * from tbl_HoldCard where @user_id = HoCa_UserID)
    begin
        declare @card_id int, @card_num int;
        set @losecard_num = floor(rand() * 5) + 1;
        select top 1 @card_id = HoCa_CardID, @card_num = HoCa_Quantity From tbl_HoldCard where HoCa_UserID = @user_id Order By newid();
        if (@losecard_num > @card_num) begin set @losecard_num = @card_num end
        if (@losecard_num = @card_num)
        begin
            delete from tbl_HoldCard where HoCa_UserID = @user_id and @card_id = HoCa_CardID;
        end
        else
        begin
            update tbl_HoldCard set HoCa_Quantity = HoCa_Quantity - @losecard_num where HoCa_UserID = @user_id and @card_id = HoCa_CardID;
        end
        select @card_name = Card_Name, @card_rarity = Card_Rarity from tbl_Card where Card_ID = @card_id;
    end
end
GO
```

本存储过程用于随机事件中的丢失卡片。通过随机选出要丢失的卡片种类后，要进行判断是否丢失数量大于或者等于拥有数量，决定是 delete 还是 update。

#### 5. prd\_GenCardShop

```
CREATE proc [dbo].[prd_GenCardShop]
as
begin
    declare @card_type_num_sum int, @card_type_num_sell int, @card_type_num_upper_ratio float, @card_type_num_lower_ratio float,
    @card_type_num_sell_upper float, @card_type_num_sell_lower float, @i int;
    declare @card_id int, @card_name varchar(30), @card_power int, @card_price int, @card_rarity varchar(3);
    set @card_type_num_upper_ratio = 0.7; set @card_type_num_lower_ratio = 0.25;
    set @card_type_num_sum = (select COUNT(*) from tbl_Card);
    set @card_type_num_sell_upper = @card_type_num_upper_ratio * @card_type_num_sum;
    set @card_type_num_sell_lower = @card_type_num_lower_ratio * @card_type_num_sum;
    exec prd_GenRand @card_type_num_sell_upper, @card_type_num_sell_lower, @card_type_num_sell out;
    set @i = 0;
    declare @multi int, @quantity int, @new_price int, @min_ratio float, @max_ratio float, @min_quan float, @max_quan float;
    create table tbl_Temp1(id int, name varchar(30), power int, rarity varchar(3), price int, quantity int);
    while @i < @card_type_num_sell
    begin
        select top 1 @card_id = Card_ID, @card_name = Card_Name, @card_power = Card_Power, @card_price = Card_Price, @card_rarity = Card_Rarity
        From tbl_Card Order By newid();
        if not exists (select * from tbl_Temp1 Where @card_id = id) begin
            if @card_rarity = 'N' begin set @min_ratio = 0.3; set @max_ratio = 2.5; set @min_quan = 5; set @max_quan = 100; end else begin
                if @card_rarity = 'R' begin set @min_ratio = 0.4; set @max_ratio = 2.0; set @min_quan = 5; set @max_quan = 75; end else begin
                    if @card_rarity = 'SR' begin set @min_ratio = 0.6; set @max_ratio = 1.45; set @min_quan = 1; set @max_quan = 10; end else begin
                        if @card_rarity = 'UR' begin set @min_ratio = 0.5; set @max_ratio = 1.4; set @min_quan = 1; set @max_quan = 15; end else begin
                            if @card_rarity = 'UTR' begin set @min_ratio = 0.65; set @max_ratio = 1.4; set @min_quan = 1; set @max_quan = 5; end else begin
                                if @card_rarity = 'SER' begin set @min_ratio = 0.65; set @max_ratio = 1.5; set @min_quan = 1; set @max_quan = 5; end else begin
                                    if @card_rarity = 'HR' begin set @min_ratio = 0.7; set @max_ratio = 1.5; set @min_quan = 1; set @max_quan = 3; end
                                    else begin set @min_ratio = 0.75; set @max_ratio = 1.5; set @min_quan = 1; set @max_quan = 2; end
                                end
                            end
                        end
                    end
                end
            end
        end
        set @multi = 1;
        set @quantity = 1;
        set @new_price = ROUND(dbo.fun_CalcFactor(@min_ratio, @max_ratio, rand() * @card_price, 0);
        insert into tbl_Temp1 values(@card_id, @card_name, @card_power, @card_rarity, @new_price, @quantity);
        set @i = @i + 1;
    end
    select * from tbl_Temp1;
    drop table tbl_Temp1;
end
GO
```

本存储过程作为核心功能之一，用于产生每天卡店的出售列表。  
@card\_type\_num\_upper\_ratio 以及 @card\_type\_num\_lower\_ratio 控制出现的卡片种类的范围；决定出现的种类数量后通过循环语句逐一向临时表中添加卡片，根据罕贵度设定价格因子浮动区间 @min\_ratio 以及 @max\_ratio（用上述函数 fun\_CalcFactor 根据卡片基准价格决定今日价格），同时设定 @min\_quan 以及 @max\_quan 用以控制卡片出现的张数。

## 6. prd\_GenRand

```
CREATE proc [dbo].[prd_GenRand]
@Upper float,
@Lower float,
@Result int out
as
SET @Result = ROUND(((@Upper - @Lower - 1) * RAND() + @Lower), 0);
GO
```

用于在给定范围内产生随机整数。

## 7. prd\_GenHoldCard

```
CREATE proc [dbo].[prd_GetHoldCard]
@user_id int
as
begin
select Card_Name, Card_Power, Card_Rarity, HoCa_Quantity, Card_ID
from tbl_Card, tbl_HoldCard where @user_id = HoCa_UserID and Card_ID = HoCa_CardID;
end
GO
```

用于更新卡本。

## 8. prd\_InitUser

```
CREATE proc [dbo].[prd_InitUser]
@user_id int
as
begin
update tbl_User set User_Money = 2500, User_HP = 100, User_GameDate = 0 where User_ID = @user_id;
delete from tbl_HoldCard where HoCa_UserID = @user_id;
end
GO
```

用于玩家 Gameover 后的数据初始化。

## 9. prd\_NewCard

```
CREATE proc [dbo].[prd_NewCard]
@card_name varchar(30),
@card_power int,
@card_price int,
@card_rarity varchar(3),
@pack_name varchar(4)
as
begin
declare @pack_id int;
select @pack_id = Pack_ID from tbl_Pack where Pack_Name = @pack_name;
insert into tbl_Card(Card_Name, Card_Power, Card_Price, Card_Rarity, Card_Pack)
values(@card_name, @card_power, @card_price, @card_rarity, @pack_id);
end
GO
```

用于新建卡片，主要是根据 Pack\_Name 查询 Pack\_ID。

## 10. prd\_OpenPack



```

CREATE proc [dbo].[prd_OpenPack]
@user_id int,
@pack_id int
as
begin
create table tbl_Temp(Temp_Card_ID int, Temp_Card_Name varchar(30), Temp_Card_Rarity varchar(3));
insert into tbl_Temp Select top 4 Card_ID, Card_Name, Card_Rarity From tbl_Card where Card_Rarity = 'N' and Card_Pack = @pack_id Order By newid();
declare @random float;
declare @rate_r float, @rate_sr float, @rate_ur float, @rate_other float;
set @rate_r = 0.7333;
set @rate_sr = 0.1;
set @rate_ur = 0.1333;
set @random = rand();
if (@random <= @rate_r)
insert into tbl_Temp Select top 1 Card_ID, Card_Name, Card_Rarity From tbl_Card where Card_Rarity = 'R' and Card_Pack = @pack_id Order By newid();
else if (@random <= @rate_sr + @rate_r)
insert into tbl_Temp Select top 1 Card_ID, Card_Name, Card_Rarity From tbl_Card where Card_Rarity = 'SR' and Card_Pack = @pack_id Order By newid();
else if (@random <= @rate_ur + @rate_sr + @rate_r)
insert into tbl_Temp Select top 1 Card_ID, Card_Name, Card_Rarity From tbl_Card where Card_Rarity = 'UR' and Card_Pack = @pack_id Order By newid();
else
insert into tbl_Temp Select top 1 Card_ID, Card_Name, Card_Rarity From tbl_Card
where Card_Rarity != 'UR' and Card_Rarity != 'SR' and Card_Rarity != 'R' and Card_Rarity != 'N' and Card_Pack = @pack_id Order By newid();
insert into tbl_HoldCard select @user_id, Temp_Card_ID, 1 from tbl_Temp;
select * from tbl_Temp;
drop table tbl_Temp;
end
GO

```

用于开包。一包必定有 4 张 N 卡，因此先从 tbl\_Card 表中随机选出 4 种 N 卡。之后设置其他罕贵出现的概率，根据 rand() 的值决定剩余的一张卡的罕贵度，再随机选出相应罕贵的卡。

## 11. prd\_RarityDistribution

```

CREATE proc [dbo].[prd_RarityDistribution]
@pack_id int
as
begin
declare @n_num int, @r_num int, @sr_num int, @ur_num int, @utr_num int, @ser_num int, @hr_num int, @other_num int;
select @n_num = COUNT(*) from tbl_Card where Card_Pack = @pack_id and Card_Rarity = 'N';
select @r_num = COUNT(*) from tbl_Card where Card_Pack = @pack_id and Card_Rarity = 'R';
select @sr_num = COUNT(*) from tbl_Card where Card_Pack = @pack_id and Card_Rarity = 'SR';
select @ur_num = COUNT(*) from tbl_Card where Card_Pack = @pack_id and Card_Rarity = 'UR';
select @utr_num = COUNT(*) from tbl_Card where Card_Pack = @pack_id and Card_Rarity = 'UTR';
select @ser_num = COUNT(*) from tbl_Card where Card_Pack = @pack_id and Card_Rarity = 'SER';
select @hr_num = COUNT(*) from tbl_Card where Card_Pack = @pack_id and Card_Rarity = 'HR';
select @other_num = COUNT(*) from tbl_Card
where Card_Pack = @pack_id and Card_Rarity != 'N' and Card_Rarity != 'R' and Card_Rarity != 'SR'
and Card_Rarity != 'UR' and Card_Rarity != 'UTR' and Card_Rarity != 'SER' and Card_Rarity != 'HR';
create table tbl_Temp(Temp_Rarity varchar(5), Temp_Num int);
insert into tbl_Temp values('N', @n_num);
insert into tbl_Temp values('R', @r_num);
insert into tbl_Temp values('SR', @sr_num);
insert into tbl_Temp values('UR', @ur_num);
insert into tbl_Temp values('UTR', @utr_num);
insert into tbl_Temp values('SER', @ser_num);
insert into tbl_Temp values('HR', @hr_num);
insert into tbl_Temp values('Other', @other_num);
select * from tbl_Temp;
drop table tbl_Temp;
end
GO

```

用于显示罕贵分布图。将不同罕贵分别应用 COUNT()，并生成临时表。

## 12. prd\_SaveCard

```

CREATE proc [dbo].[prd_SaveCard]
@card_id int,
@card_name varchar(30),
@card_power int,
@card_price int,
@card_rarity varchar(3),
@pack_name varchar(4)
as
begin
declare @pack_id int;
select @pack_id = Pack_ID from tbl_Pack where Pack_Name = @pack_name;
update tbl_Card set Card_Name = @card_name, Card_Power = @card_power, Card_Price = @card_price,
Card_Rarity = @card_rarity, Card_Pack = @pack_id where Card_ID = @card_id;
end
GO

```

用于更新卡片，主要是根据 Pack\_Name 查询 Pack\_ID。

### 13. prd\_SellCard

```
CREATE proc [dbo].[prd_SellCard]
@card_id int,
@sell_num int,
@sell_price int,
@user_id int
as
declare @now_num int, @new_num int, @now_money int, @new_money int;
select @now_num = HoCa_Quantity from tbl_HoldCard where HoCa_CardID = @card_id and HoCa_UserID = @user_id;
if @now_num <= @sell_num begin
delete from tbl_HoldCard where HoCa_CardID = @card_id;
end
else begin
set @new_num = @now_num - @sell_num;
update tbl_HoldCard set HoCa_Quantity = @new_num where HoCa_CardID = @card_id and HoCa_UserID = @user_id;
end
select @now_money = User_Money from tbl_User where @user_id = User_ID;
set @new_money = @now_money + @sell_price * @sell_num;
update tbl_User set User_Money = @new_money where @user_id = User_ID;
GO
```

用于卖出卡片。先根据卖出数量是否等于持有数量来决定 delete 还是 update，再进行金钱的更新。

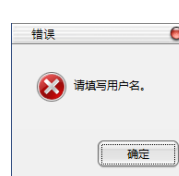
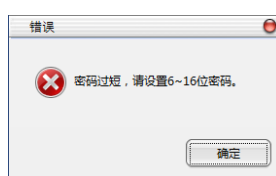
## 六、 主要技术

本系统前端基于 Microsoft .NET Framework 4.0 框架，使用 C#语言，进行 Winform 模式下的开发。后端使用 SQL 语句进行访问。

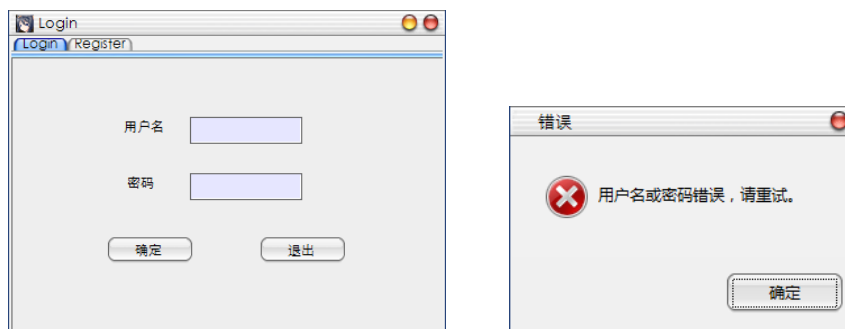
## 七、 运行示例

### (一) 注册与登录

#### 1. 注册

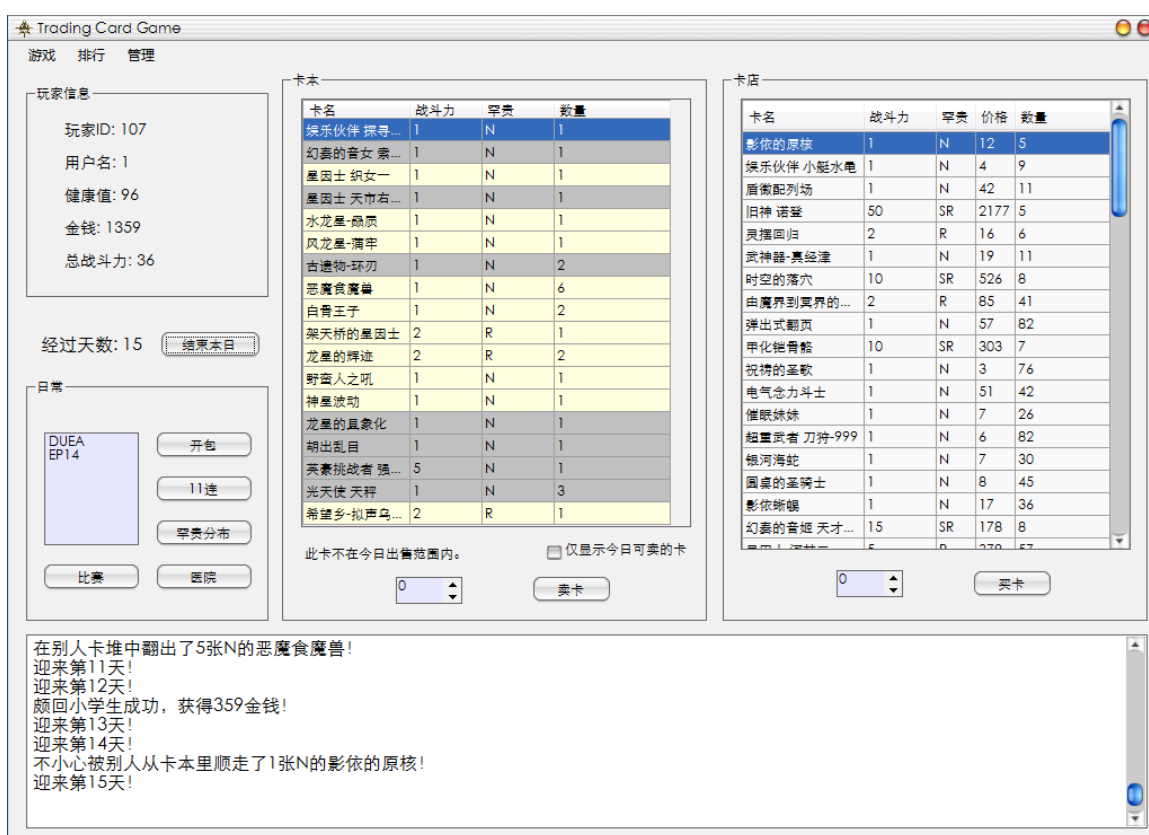


#### 2. 登录

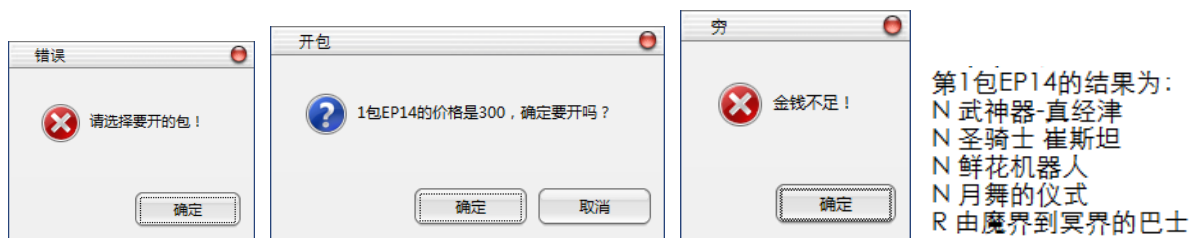


## (二) 游戏本体

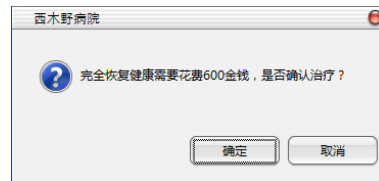
### 1. 游戏主界面



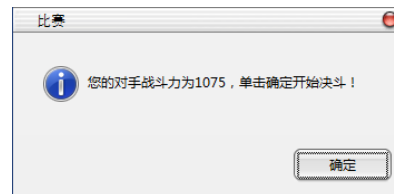
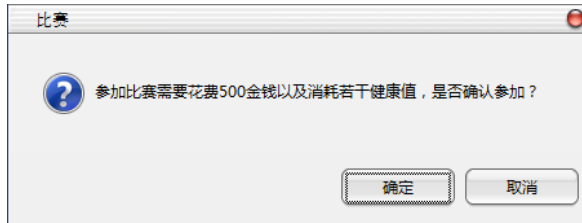
### 2. 开包



### 3. 医院



#### 4. 比赛



参加比赛不幸落败...

#### 5. 随机事件

迎来第5天！

卡商大炒SR的圣骑士 鲍斯，其单价猛增至353！

迎来第29天！

SER的杰拉的天使没什么人采用，单价跌至163！

迎来第10天！

在别人卡堆中翻出了5张N的恶魔食魔兽！

迎来第6天！

不小心被别人从卡本里顺走了1张SR的魔力之泉！

迎来第2天！

图谋不轨的玩家来翻你卡本，可你的卡本里什么也没有。

迎来第12天！

颇回小学生成功，获得359金钱！

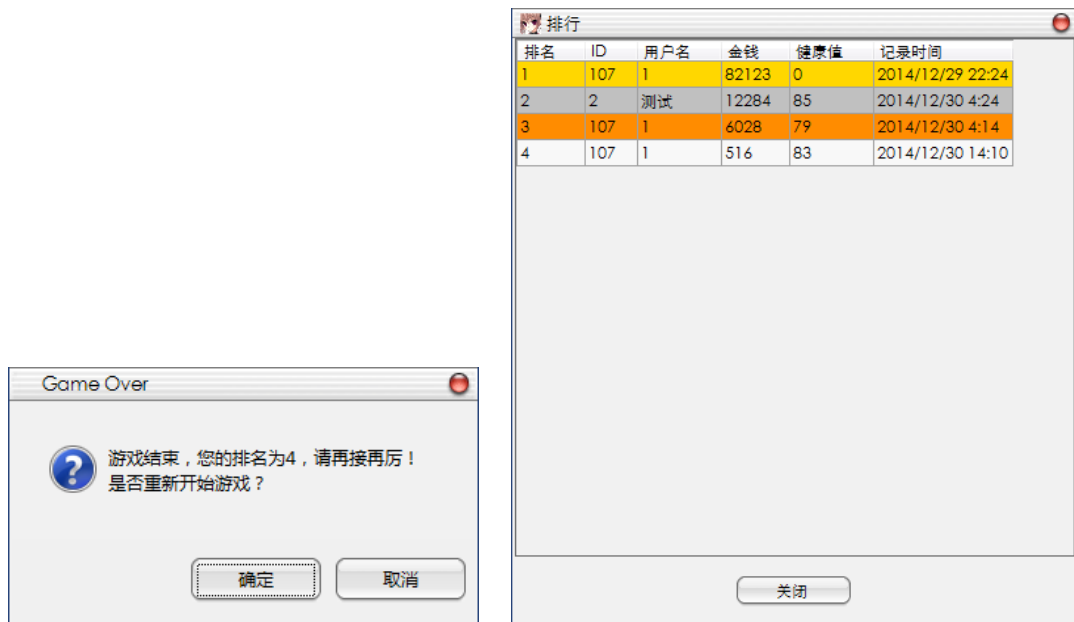
迎来第21天！

在卡店时钱被人摸走，损失154金钱！

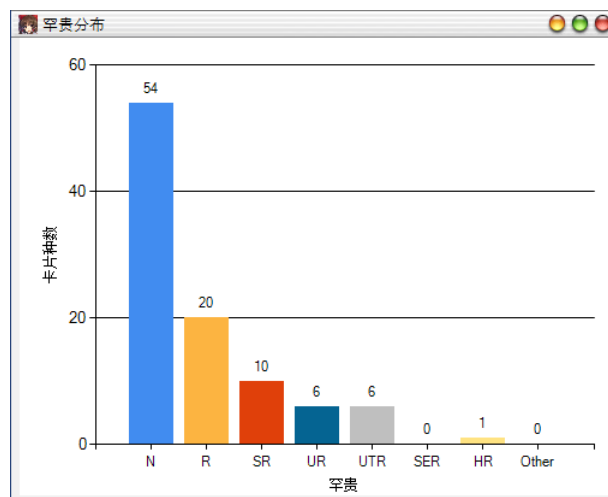
迎来第8天！

去卡店途中不小心受伤，损失4健康值！

#### 6. 排行榜



## 7. 罕贵分布图



## 8. 卡片管理

卡片总数: 156

ID	卡名	罕贵
1	闪光之骑士	R
2	傅科魔炮石	R
3	玄化武装龙	N
4	异色眼灵摆龙	UR
5	异色眼灵摆龙	UTR
6	异色眼灵摆龙	HR
7	娱乐伙伴 小艇水龟	N
8	娱乐伙伴 鞭子蛇	N
9	娱乐伙伴 宝剑鱼	N
10	娱乐伙伴 探寻河马	N
11	娱乐伙伴 万花筒蝎	R
12	娱乐伙伴 颠倒蛙	R
13	超量武者 鬼若-O2	N

卡名: 异色眼灵摆龙

战斗力: 50

价格: 1512

罕贵: HR

卡包: DUEA

保存修改 删除所选

清空信息 添加新卡

## 八、 源程序简要说明

### (一) 游戏主界面 Form1.cs

```
/*----- 访问DB-----*/
private const string ConnectionString = "server = local;
private static SqlConnection objSqlConnection = null;
private static SqlCommand objSqlCommand = null;
private static DataSet objDataSet = null;
private static SqlDataAdapter objDataAdapter = null;
private static SqlDataReader objSqlReader = null;
private static string sql = null;
private static string proc = null;
/*----- 访问DB end-----*/
```

上图在类中定义了与数据库连接相关的属性。

```
/*----- 常量-----*/
private const int EVENT_GETMONEY = 10;           //事件：获得金钱概率
private const int EVENT_LOSEMONEY = 10;          //事件：损失金钱概率
private const int EVENT_LOSEHP = 10;             //事件：损失健康值概率
private const int EVENT_GETCARD = 10;            //事件：获得卡片概率
private const int EVENT_LOSECARD = 10;           //事件：丢失卡片概率
private const int EVENT_PRICEUP = 10;            //事件：涨价概率
private const int EVENT_PRICEDOWN = 10;          //事件：降价概率
private const int HOSPITAL_PRICE = 150;          //医院单价
private const int DUEL_PRICE = 500;              //比赛单价
/*----- 常量 end-----*/
```

游戏中部分常量的调整。关于卡片售价浮动方面的调整则需要修改存储过程。

```
/*----- 玩家信息变量-----*/
private static int sm_nUserID;
private static string sm_sUserName;
private static int sm_nUserHP;
private static int sm_nUserMoney;
private static int sm_nGameDate;
private static int sm_nUserManageLevel;
/*----- 玩家信息变量 end-----*/
```

用于记录玩家信息，与数据库同步更新。

```
public Form1()
{
    InitializeComponent();
    this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
    objSqlConnection = new SqlConnection(ConnectionString);
    objSqlConnection.Open();
}
```

构造器内与数据库建立连接并打开。

```
public void PassDay()...
```

此方法用于处理一天结束后的信息变化、卡店生成以及随机事件发生。

```
/*-----玩家信息-----*/
private void UpdateUserName()...
private void UpdateUserHP()...
private void UpdateUserMoney()...
private void UpdateGameData()...
private void UpdatePowerSum()...
/*-----玩家信息 end-----*/
```

此处方法皆用于从数据库获取相应数据并更新于窗体上。

```
private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
private void dataGridView1_Sorted(object sender, EventArgs e)
```

用于更新当前选中项目所允许的出售最大值并更新于数量选择控件上，防止数目超限。（同时存储过程里有保护措施）

```
private bool isInShop(int nID)...
```

用来判断某张卡是否在今日的商店中（决定是否可以卖出）。

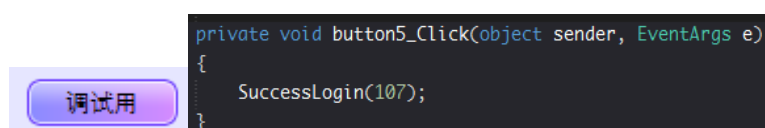
## (二) 注册与登录界面 Login.cs

```
private SqlConnection objSqlConnection = null;
private string sql = null;
private Form1 form1;
```

这里的 SqlConnection 对象直接从 Form1（游戏主窗体类）获取（往后的每个窗体均如此）。

```
private void SuccessLogin(int nID)
{
    this.Hide();
    this.form1.Show();
    this.form1.ShowInTaskbar = true;
    this.form1.Focus();
    this.form1.InitData(nID);
    this.form1.PassDay();
}
```

登录成功后，此窗体隐藏，主窗体出现，获取焦点并初始化信息。



以测试 ID（107）登录，调试时方便。

```
private void Login_Load(object sender, EventArgs e)
{
    button5.Visible = false;
}
```

在 Load 事件里将此按钮隐藏。

```
string sNewPassword = System.Web.Security.FormsAuthentication.HashPasswordForStoringInConfigFile(sPassword, "MD5").ToLower();
objSqlConnection = Form1.GetSqlConnection();
```

这里对密码使用了 MD5 加密，增强安全性。

### (三) 与数据库的交互

其余窗体所涉及代码基本为执行 SQL 查询、调用存储过程等，在此将常用查询方式总结。

#### 1. ExecuteNonQuery 方法

```
private void WriteMoney(int nMoney)
{
    int nNewMoney = 0;
    if (nMoney + sm_nUserMoney > 0)
    {
        nNewMoney = nMoney + sm_nUserMoney;
        sql = "update tbl_User set User_Money = " + nNewMoney.ToString()
            + "where User_ID = '" + sm_nUserID.ToString() + "'";
        objSqlCommand = new SqlCommand(sql, objSqlConnection);
        objSqlCommand.ExecuteNonQuery();
        UpdateUserMoney();
    }
}
```

一般用于简单的非 SELECT 的单条 SQL 语句。



```

proc = "prd_CalcPowerSum";
objSqlCommand = new SqlCommand(proc, objSqlConnection);
objDataAdapter = new SqlDataAdapter();
objDataAdapter.SelectCommand = objSqlCommand;
objDataAdapter.SelectCommand.CommandType = CommandType.StoredProcedure;
IDataParameter[] parameters = { new SqlParameter("@User_id", SqlDbType.Int) };
parameters[0].Direction = ParameterDirection.Input;
parameters[0].Value = nOppoID;
objSqlCommand.Parameters.Add(parameters[0]);
parameters[1].Direction = ParameterDirection.ReturnValue;
objSqlCommand.Parameters.Add(parameters[1]);
objSqlCommand.ExecuteNonQuery();
nOppoPower = Convert.ToInt32(parameters[1].Value);
parameters[0].Value = sm_nUserID;
objSqlCommand.ExecuteNonQuery();
nUserPower = Convert.ToInt32(parameters[1].Value);

```

也用于只需要返回值的存储过程。（图中连续调用 2 次此过程）

关于存储过程，可以通过把参数写入 SQL 语句直接调用，也可以像图中这样建立参数数组，自行设置类型、返回与否后推入 SqlCommand，这样语句直接写”Exec prd\_xxx”即可，很方便维护。

## 2. ExecuteReader 方法结合 SqlDataReader

```

private void UpdateUserName()
{
    sql = "SELECT User_Name FROM tbl_User WHERE User_ID = '" + sm_nUserID.ToString() + "'";
    objSqlCommand = new SqlCommand(sql, objSqlConnection);
    objDataAdapter = new SqlDataAdapter();
    objDataAdapter.SelectCommand = objSqlCommand;
    objSqlReader = objSqlCommand.ExecuteReader();
    if (objSqlReader.Read())
        sm_sUserName = objSqlReader.GetValue(objSqlReader.GetOrdinal("User_Name")).ToString();
    objSqlReader.Close();
    this.UserNameLabel.Text = "用户名: " + sm_sUserName;
}

```

常用于简单的 SELECT 语句，而且返回内容通常是单值或者是简单的表，用 SqlDataReader 类提供的方法很容易访问并提取结果。

```

private void UpdatePack()
{
    if (listBox1.Items.Count > 0)
        listBox1.Items.Clear();
    sql = "SELECT Pack_Name FROM tbl_Pack WHERE Pack_Date <= " + sm_nGameDate.ToString() + ";";
    objSqlCommand = new SqlCommand(sql, objSqlConnection);
    objDataAdapter = new SqlDataAdapter();
    objDataAdapter.SelectCommand = objSqlCommand;
    objSqlReader = objSqlCommand.ExecuteReader();
    while (objSqlReader.Read())
        listBox1.Items.Add(objSqlReader.GetValue(objSqlReader.GetOrdinal("Pack_Name")).ToString());
    objSqlReader.Close();
}

```

这里通过 while 语句循环 Read 方法，获得了当前时间能开的卡包名。

### 3. 应用 DataAdapter 以及 DataSet

```
sql = "EXEC prd_GetHoldCard " + sm_nUserID.ToString() + ";";
objDataSet = new DataSet();
DataTable objDataTable = new DataTable();
objDataTable.Columns.Add("Name", typeof(System.String));
objDataTable.Columns.Add("Power", typeof(System.Int32));
objDataTable.Columns.Add("Rarity", typeof(System.String));
objDataTable.Columns.Add("Quantity", typeof(System.Int32));
objDataTable.Columns.Add("ID", typeof(System.Int32));
DataSet objDataSet2 = new DataSet();
objDataSet2.Tables.Add(objDataTable);
objDataAdapter = new SqlDataAdapter(sql, objSqlConnection);
objDataAdapter.Fill(objDataSet);
```

DataSet 类是 .NET 里提供的数据库表的数据结构，非常适合查询返回结果为较为复杂的表时使用。同时，DataSet 可以直接与 DataGridView 和 Chart 控件绑定，将接收数据以表或图的方式直观显示，省去许多麻烦。

在这里并没有将 DataGridView 与数据源绑定，全部使用 SQL 语句来访问。

## 九、感想与收获

从主题的制订、规划，到数据库、程序的建立，直至最后的完善，前后只有一周多的时间，但是在这一周内我收获了很多。

首先是数据库方面，我体会到了从需求出发进行设计，一步一步由 E-R 图、关系模式到建立完整数据库的流程。把课堂上学到的知识用在实际应用中，更加体会了整体设计的重要性。一旦上层设计出了问题，会给底层的代码带来不小的麻烦（比如规范化程度不够带来的许多冲突）。

其次是 .NET Framework 和 C# 的使用方面。以前在 Visual Basic 上接触过浅层的 .NET Framework 开发，这次通过不断在技术博客、书中查资料、寻求帮助，在 C# 中一边学一边摸索，将游戏开发了出来。从整体看，架构完整性可能并不完善（类的方面），一些细节处理也不够到位（控件的命名等），不过因为时间精力有限，不能一一修改。在以后的程序开发中，这将会成为很好的教训与经验。

游戏的平衡性方面作出了不少努力，从价格的正态分布浮动到随机事件的设计，这都是提高可玩性的必经之路，可能仍有些许不足，欢迎测试者、玩家们提出建议与意见。

展望未来，可以再强化安全性后把数据库放到服务器上，此游戏就成了一款“网游”，各玩家可以挑战排行榜。再进一步，可以去分析相关网站的网页，实现实时更新价格与

卡包，能增添许多趣味性。

纵观整个开发，付出与收获是成正比的：挥洒了时间与汗水，收获了各种方面的经验（从前端至后端，包括 C#语言方面、.NET Framework 的了解与使用、Winform 的开发、SQL 的嵌入、后端数据库的实现等），同时完成一个自己的游戏也产生一定成就感。因此，这是一次非常有意义的课程设计。

## 十、 游戏说明书

### (一) 游戏目标

游戏的唯一目标就是收集尽可能多的金钱！在 52 天内，玩家需要发挥自己的经营头脑，规划金钱，通过交易等方式获取最大利益。时间到达 52 天后，将会根据玩家现有金钱数来决定在排行榜上的位置。

### (二) 游戏方法

#### 1. 注册与登录

第一次玩本游戏的玩家需要进行注册，用户名不能为空，最长 20 个英文字符；密码为 6 至 16 位（字符不限）。已经拥有账号的玩家可以直接登录。目前暂不提供密码查询与修改功能，请妥善保管。

若需要以管理员身份登录，用户名为 1，密码为 123123123。

#### 2. 卡片的交易

每天都可以随时进行卡片交易，选中相应卡片，调整数量即可买入或卖出。请注意，卡片有各自的基准价格，这个价格与罕贵有一定关系，但也不排除例外（此处参考了 <http://oreneturn.com/> 上 2014 年 12 月 29 日游戏王 DUEA 与 EP14 所有卡的真实价格）。如何做到低买高卖呢？这就需要观察各卡片价格走势，分析其基准价格，做出合理判断，以获取最大收益。

#### 3. 事件的发生

每天都有一定概率发生随机事件，事件也分好与坏，有可能天降神卡，也有可能被盗走金钱，这些在一定程度上会影响策略。

#### 4. 开包的惊喜

卡包所有卡的设计基于游戏王 DUEA 与 EP14.1 包固定出 4 张 N 与 1 张其他罕贵，这里的其他罕贵代表 73.3%的可能性是 R，其余是更高的罕贵。通常来说，4 张 N1 张 R 的售价是很难捞回成本的，但各卡包里也都有能卖极高价的卡片，因此开与不开，需要做出一定规划，可以根据卡包的罕贵分布来决定。

## 5. 比赛与医院

花报名费参加比赛后，会从其他所有玩家内任意选一名与玩家比赛，玩家可以知道对方的战斗力，而最终胜率与自己对方战斗力的比值、自己的健康度都有关系，获得的奖金与这些也有关系。所以在交易卡片时，有针对性地选购一些高战斗力卡片，用比赛赚钱也是一种策略。

健康度为 0 时会立刻游戏结束，因此可以去医院治疗。随机事件、比赛会导致健康度下降，这里需要留心。

### (三) 罕贵解释

N: Normal 平卡

R: Rare 银字

SR: Super Rare 面闪

UR: Ultra Rare 金闪

UTR: Ultimate Rare 3D

SER: Secret Rare 银碎

HR: Holographic Rare 全息

ESR: Extra Secret Rare

CR: Collector's Rare

### (四) 其他

游戏中的每一步都会被保存，因此没有后悔的机会。游戏支持断点保存，但是每次登录后会自动进入下一天，所以请确保本日所有该做的事情做完后再退出游戏。

## 参考文献

[1]王珊,萨师煊. 数据库系统概论(第四版) [M]. 北京: 高等教育出版社,2006:174-176.