Ticket Sales Rapport

Beskriv programmet:

Jag har i princip skrivit två program, de har samma funktionalitet men det som skiljer sig är att den ena använder sig av en List för att lagra data och sparar allting lokalt medans den andra använder sig av en databas. De är uppdelade i branches i mitt GIT Repo; master = med lista; DB = med databas. Anledning till varför jag valde att dela upp dem var för att det var en sån drastisk skillnad i hur de två systemen är uppbyggda. En annan anledning är att du antagligen inte kommer kunna köra databas programmet om du inte laddar ned MySQL och sätter upp det med samma inställningar som jag har.

Database branch:

Databasen är byggd iMySQL i databas språket SQL. Det första jag gjorde var att ladda ned MySQL.Data denpendencien genom NuGet.

Det första programmet måste göra är att upprätta en anslutning till databasen. Detta görs genom att skapa en sträng med Servern, user id, password, och databas namnet. Det var väldigt enkelt då jag är på localhost så kunde jag bara skriva localhost som server och jag är på standard usern “root”. Sedan använder jag MySQL connection och sätter den till open för att ansluta. I databasen så kommer datan sparas i en tabell. Tabellen definierade jag i MySQL programmet men det skulle gå att göra via Visual Studio men det skulle komplicera allt så jag ansåg att det var mer effektivt att göra så här.

Connect

```
public void Connect()
  {
    string cs = @"server=localhost;userid=root;password=Robin789;
    database=ticketsales";
    con = new MySqlConnection(cs);
    con.Open();
  }
```

Tabellen innehåller: - ID - Detta är en int och ett primary key(den måste vara unik) Den har även en inställning som heter automatic index vilket gör att den skapar numret näst på tur automatiskt. - NAME - Detta är en sträng eller mer exakt en VARCHAR(45) - aTickets, cTickets, sTickets - Detta är INT värden för antalet biljetter av varje typ. - refunded - En boolean som avgör om den är refunded eller ej (refunded=1; !refunded = 0) Default värdet är 0.

När man köper någonting så så kollar den först satt inputen är giltig och allt sånt och skickar det man skrev till en annan metod. I den metoden finns en sträng som heter sql. I den strängen finns SQL koden för att lägga till objektet. Variablerna matas in med hjälp av lite string interpolation. Sedan avänder jag ExecuteScalar metoden från MySQL dependencein för att skicka strängen till databasen och utföra den.

Add

```
public void Add(string name, string aTickets, string cTickets, string sTickets)
    {
        var stm = $"insert into purchases (name,aTickets,cTickets,sTickets)
        values ('{name}', {aTickets},{cTickets},{sTickets});";
        var cmd = new MySqlCommand(stm, con);
        cmd.ExecuteScalar();
    }
```

När man refundar någonting fungerar det ungefär likadant fast man använder en annan SQL kod i strängen som i det här fallet ändrar refunded till 1.

Refund

```
public void Refund(string[] refund)
    {
        var stm = $"UPDATE `ticketsales`.`purchases` SET `refunded` = 1
        WHERE(`NAME` = '{refund[1]}');";
        var cmd = new MySqlCommand(stm, con);
        cmd.ExecuteScalar();
        Console.WriteLine("Refunded");
    }
```

När man vill skriva ut allting så är det likadant men man använder SQL koden för att skriva ut allt. Om man vill hitta ett individuellt element så finns det ett jätte simpelt sätt att göra det

Load all

```
public void Load()
    {
        string sql = "SELECT * FROM purchases;";
        var cmd = new MySqlCommand(sql, con);
        MySqlDataReader rdr = cmd.ExecuteReader();
        while (rdr.Read())
        {
            Console.Write("ID: {0}, Name: {1}, Adult Tickets: {2},
            Child Tickets: {3}, Senior Tickets {4}", rdr.GetInt32(0),
            rdr.GetString(1), rdr.GetInt32(2), rdr.GetInt32(3),
            rdr.GetInt32(4));
            if (rdr.GetInt32(5) == 1)
            {
                Console.Write(", !REFUNDED!");
            }
            Console.WriteLine("");

        }
        rdr.Close();
    }
```

Load one

```
 string sql = $"SELECT* FROM purchases where name = '{search[1]}';";
var cmd = new MySqlCommand(sql, con);
MySqlDataReader rdr = cmd.ExecuteReader();
bool found = false;
while (rdr.Read())
{
    Console.WriteLine("ID: {0}, Name: {1}, Adult Tickets: {2}, Child Tickets: {3},
    Senior Tickets {4}", rdr.GetInt32(0), rdr.GetString(1), rdr.GetInt32(2),
    rdr.GetInt32(3), rdr.GetInt32(4));
    found = true;
}
```

För att skriva ut det totala antalet biljetter samt hur många biljetter av varje typ som sålts så kan vi använda oss av SQL:s SUM som tar alla värden i en kollumn och adderar dem

Print sum

```
public void Sum()
    {
        string sql = "select refunded, sum(aTickets), sum(cTickets),
        sum(sTickets) from purchases group by 1;";
        var cmd = new MySqlCommand(sql, con);
        MySqlDataReader rdr = cmd.ExecuteReader();
        while (rdr.Read())
        {
            var refunded = rdr.GetBoolean(0);
            if (refunded)
            {
                Console.WriteLine("refunded:");
            } else
            {
                Console.WriteLine("purchased:");
            }
            Console.WriteLine("Adult Tickets: {0}, Child Tickets: {1},
            Senior Tickets {2}, Total {3}", rdr.GetInt32(1), rdr.GetInt32(2),
            rdr.GetInt32(3), rdr.GetInt32(1+2+3));
        }
        rdr.Close();
    }
```

Sammanfattning

Det är i princip hela programmet. Någonting som man kan se är att det är betydligt smidigare att använda sig av databaser än listor och det blir mycket färre rader kod. Det bästa med databasen enligt mig (förutom att man kan komma åt den lättare) är att man slipper skapa en klass med alla värden. SQL tabellerna är mycket smidigare och lättare att hantera. SQL är även ett väldigt kraftfullt språk så det är många saker som går att

göra betydligt lättare. Ett exempel är om man vill hitta ett visst element. I c# skulle jag behöva skriva en kod som söker igenom hela listan och kollar om elementet stämmer överens med inputen medans i SQL kan man bara skriva en kort rad med kod som uppfyller samma funktion.
Det finns säkerhetsbrister med SQL som SQL injections men i det här fallet motverkas dem till viss del genom att jag splittar inputen med ett mellanslag

Beskriv Utväcklingsprocessen

Jag började med att skriva en lista på de funktionella krav som behövdes så att jag kunde ha den som en typ av checklista. Från början planerade jag att välja Grafiskt användargränssnitt fördjupningen samt databaser. Jag började med att skapa ett windows forms dokument i c# och började jobba på det och skapa knappar. Jag kände mig inte helt nöjd med hur det var och det uppstod en massa problem där lösningen alltid verkade vara att bara göra ett konsol program istället. Till slut kom jag fram till att skapa ett Grafiskt användargränssnitt inte var lika roligt som jag hade tänkt mig så jag struntade i den iden och bestämde mig för att fokusera på databaser. Då började jag om från scratch men som tur var så kunde jag använda mig av mycket av jag redan skrivit i myrstack uppgiften så jag kunde återanvända mycket kod. På grund av detta gick det ganska snabbt att skriva klart den. Vid det här tillfället kände jag att jag ville ha med två av fördjupningarna så jag kollade igenom listan och tog det jag ansåg skulle vara lättast att implementera vilket var att kunna spara och hitta specifika transaktioner. Min kod kunde i princip redan göra det här, det var bara ett fall av att skriva en metod som faktiskt gjorde det. De enda problem som uppstod här var att jag inte kunde lista ut vissa saker, som hur man fick den att ladda in datan från en fil ordentligt eller hur jag skulle skriva ut det totala antalet som sålts. Men båda de problemen löste jag genom att be om hjälp.

Planen hade från början varit att skriva koden normalt och sedan bara ersätta spara och hämta delarna med någon typ av databas. När jag började söka på databaser och pratade med min pappa om dem insåg jag att det inte skulle vara så enkelt. Det visade sig att jag skulle behöva skriva om majoriteten av koden. Men när jag kollade vidare så insåg jag att det inte var så komplicerat. SQL koden var betydligt lättare att använda och mer effektiv än den kod jag redan skrivit så jag hann skriva om det på bara några timmar. Här hade jag lite problem med variabler som inte kunde upptäckas av de metoder som behövde använda dem men det var relativt lätt löst när man väl klurade ut det. Annars uppstod inte så många problem utom ett. Problemet var enbart ett kommande kunde köras varje gång programmet startades alla inputs efter det gav en exception. Exceptionen visade att problemet var att MySqlDataReader öppnade en connection till databasen när den kallades och eftersom connectionen aldrig stängdes kunde inte de andra kommandona öppna nya. Jag sökte lite på problemet och löste det genom att helt enkelt sätta en rdr.Close(); i slutet av varje metod som stänger connectionen.

Beskriv hur arbetet gått

I helhet skulle jag säga att arbetet gått ganska bra. Jag hade det väldigt segt att komma igång och jag visste inte riktigt hur jag skulle börja. Men tack vare att jag fick lite hjälp med kravlistan så kunde jag komma igång. Det jag spenderade mycket tid på först var windows forms och att lära mig det vilket jag ändå inte använde i slutändan så det var inte särskilt bra. Men när jag väl bestämde mig för att ta tag i det så gick det ganska snabbt att skriva klart det. Det som är negativt med att jag gjorde det snabbt är att jag begränsar min kreativitet lite och om jag hade gett mig själv mer tid så skulle jag kanske kommit på bättre lösningar på problem och bättre sätt att göra saker. Jag skulle även ha mer tid att testa min kod för eventuella buggar.

Någonting jag ska tänka på att förbättra till nästa gång är att verkligen komma igång i tid och försöka göra lite många gånger istället för väldigt mycket några få gånger. Jag tror att arbetet kommer vara mer effektivt nästa gång då jag under den här tiden har lärt mig mycket mer om git och speciellt branches.

Vad har du lärt dig?

Jag har lärt mig väldigt mycket under det här projektet. Jag har lärt mig om windows forms existens och lite om hur man använder det. Jag har lärt mig väldigt mycket om hur databaser är uppbyggda samt om SQL språket (och lite om SQL injections). Jag har lärt mig mer om hur man använder GitHub och några användbara Git-Bash kommandon. Jag har även lärt mig om branches och säkert en massa annat som jag inte kan komma att tänka på.

Vad skulle du vilja lära dig?

Jag skulle vilja lära mig mer om hur databaser fungerar, speciellt hur jag får dem att fungera över internet och inte bara på den lokala datorn