4.4. Vizualizarea stocurilor

Posibilitatea de a vizuliza stocurile reprezintă o necesitate importantă pentru a putea afla în cantități sunt disponibile produsele. Dacă se cunosc cantitățile atunci se poate deduce care sunt produsele care sunt în cantitate mică pentru a se semnala necesitatea lor.

Adăugarea de produse în stoc nu se face direct, aceste produse trebuie să provină de undeva, nu este posibil pur și simplu să introducem o cantitate dintr-un anumit produs pe stoc. Crearea stocurilor se face prin odată cu introducerea unei facturi de intrare, care se ocupă cu introducerea de stocuri noi. Operația de adăugare de stoc **(Figura 4.19)** este executată de funcția Create, procedura stocată care este apelată din această funcție este:

```
CREATE Procedure [dbo].[sp_Stoc_Insert]
	@IdProdus int,
	@IdPozitieFacturaIntrare int,
	@Cantitate decimal(18,2)
As
Begin
	Insert Into Stoc
		([IdProdus],[IdPozitieFacturaIntrare],[Cantitate])
	Values
		(@IdProdus,@IdPozitieFacturaIntrare,@Cantitate)

	Declare @ReferenceID int
	Select @ReferenceID = @@IDENTITY

	Return @ReferenceID

End
```

Practic poziția factură de intrare este o cheie străină care folosește la operația de cuplare cu facturile de intrare.

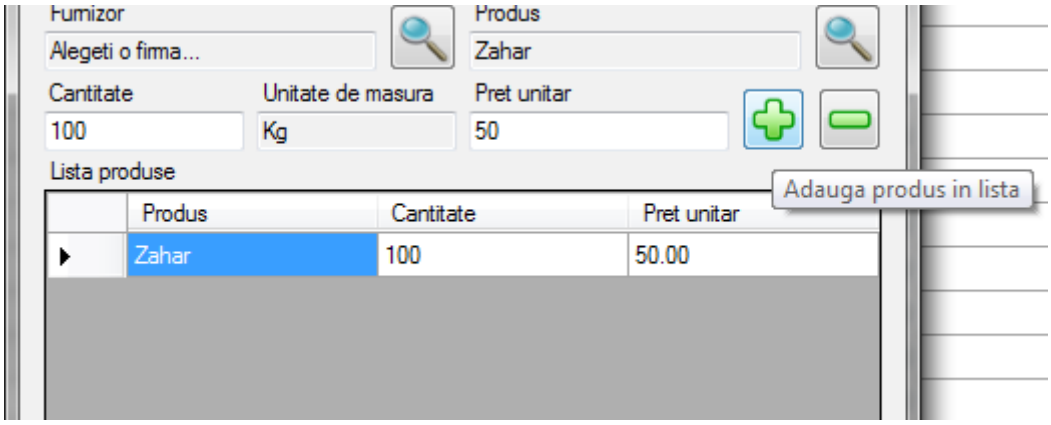

Figura 4.19 *Adăugarea unui produs în lista de stocuri*

Vânzarea produselor de pe stoc este iarăși o operație indirectă, stocurile nu se pot goli pur și simplu, se scade din stocuri prin intermediul unei facturi de ieșire. Adăugarea unui produs în limita cantității disponibile în lista de produse a unei facturi de ieșire conduce, la salvarea facturii de ieșire, la scăderea cantității de produse vândute din stocurile actuale se apelează funcția Update:

```
public void Update(List<DbObject> dbObjects, int id)
    {
      try
      {
        using (SqlConnection con = ConnectionHelper.Connection)
        {
          try
          {
            using (SqlCommand cmd =
                                  new SqlCommand(updateStoredProcedureName, con))
            {
              con.Open();
              cmd.CommandType = System.Data.CommandType.StoredProcedure;
              if (id != 0) cmd.Parameters.AddWithValue("@Id", id);
              foreach (var dboObject in dbObjects)
```

```
                {
                    cmd.Parameters.AddWithValue(dboObject.Name, dboObject.Value);
                }
                cmd.ExecuteNonQuery();
                con.Close();
            }
        }
        catch (Exception ex)
        {
            throw new Exception(StringProcedureFail + updateStoredProcedureName, ex);
        }
    }
}
catch (Exception ex)
{
    throw new Exception(StringDatabaseFail, ex);
}
}
```

Această funcție apelează procedura stocată pentru update din baza de date, care actualizează datele din stocul din care s-au vândut produsele. Mai importantă este procedura stocată care salvează legătura stocului cu tabela facturilor de ieșire:

```
CREATE Procedure [dbo].[sp_PozitiiFacturiIesire_Insert]
    @IdFacturaIesire int,
    @IdStoc int,
    @Cantitate decimal(18,2),
    @PretUnitar decimal(18,2)
As
Begin
    Insert Into PozitiiFacturiIesire
```

```
            ([IdFacturaIesire],[IdStoc],[Cantitate],[PretUnitar])
        Values
            (@IdFacturaIesire,@IdStoc,@Cantitate,@PretUnitar)

        Declare @ReferenceID int
        Select @ReferenceID = @@IDENTITY

        Return @ReferenceID

End
```

Poziția facturii de ieşire reține stocul din care s-au vândut produsele, factura asociată acestei vânzări, cantitatea produsului vândut şi prețul unitar pe produs. Iar printr-o interogare ce implică două operații de cuplare cu tabela facturilor de ieşire şi cu tabela stocurilor se poate afla uşor stocul din care s-au vândut produsele pentru o anumită factură.

Funcția de salvare a unei facturi de ieşire primeşte ca parametru o listă de poziții pentru factura de ieşire, pentru fiecare poziție în parte se actualizează stocul aferent poziție respective. Salvarea este o tranzacție, deoarece nu se permite salvarea doar a unei părți dintr-o factură. Se încercă salvarea întregii facturi, dacă nu se reuşeşte se dă un mesaj corespunzător şi se face rollback.

```
public PersistenceResult Save(List<PozitieFacturaIesire> pozitiiList)
    {
        var result = new PersistenceResult();
        try
        {
            if (pozitiiList.Count == 0)
            {
                return new PersistenceResult
                {
                    Status = Enums.StatusEnum.Errors,
```

```
                Message = "Nu ati adaugat niciun produs in factura!"
            };
        }
        using (TransactionScope scope = new TransactionScope())
        {
            var facturaPR = this.Save(); // salvez factura
            if (facturaPR.Status == Enums.StatusEnum.Errors)
            {
                throw new Exception(facturaPR.Message, facturaPR.ExceptionOccurred);
            }
            var facturaId = this.ID;
            foreach (var item in pozitiiList) // se adaugă fiecare poziție și se face update la
stoc
            {
                var stocPR = item.StocObject.Save();
                if (stocPR.Status == Enums.StatusEnum.Errors)
                {
                    throw new Exception(stocPR.Message, stocPR.ExceptionOccurred);
                }
                item.IdFacturaIesire = facturaId;
                var pozitiePR = item.Save();
                if (pozitiePR.Status == Enums.StatusEnum.Errors)
                {
                    throw new Exception(pozitiePR.Message, pozitiePR.ExceptionOccurred);
                }
            }

            scope.Complete();
        }
        result.Message = StringSaveSuccess;
        result.Status = Enums.StatusEnum.Saved;
```

```
        }
        catch (Exception ex)
        {
            result.Message = StringSaveFail;
            result.Status = Enums.StatusEnum.Errors;
            result.ExceptionOccurred = ex;
        }
        return result;
    }
```

Vizualizarea stocurilor se face din tree-ul aplicației (**Figura 4.20.**), pentru utilizator nu prezintă interes să poată vizualiza aceleași produse în mai multe stocuri. Astfel înainte de afișarea stocurilor, ele trebuie să treacă printr-o operație care pregătește datele. Funcția care se ocupă se numște GetAllGroupedByProdus și primește ca parametru numele unui produs, parametru care este opțional, dacă se dorește aflarea cantității unui anumit produs din stoc.

```
    public static List<Stoc> GetAllGroupedByProdus(string produsName = "")
    {
        var query = from s in Stoc.GetAll()
                    where s.NumeProdus.ToLower().Contains(produsName.ToLower())
                    group s by s.IdProdus into s_nou
                    select new Stoc
                    {
                        ID = s_nou.Max(p => p.ID),
                        Cantitate = s_nou.Sum(p => p.Cantitate),
                        IdPozitieFacturaIntrare =
                                    s_nou.Select(p => p.IdPozitieFacturaIntrare).First(),
                        IdProdus = s_nou.Select(p => p.IdProdus).First()
                    };
        return query.ToList();
    }
```

Funcția GetAllGroupedByProdus grupează toate stocurile pe produse, iar pentru fiecare grup rezultat se face însumarea cantităților stocurilor, astfel fiecare grup devine un stoc nou în care cantitatea este cantitatea totală pe produs.

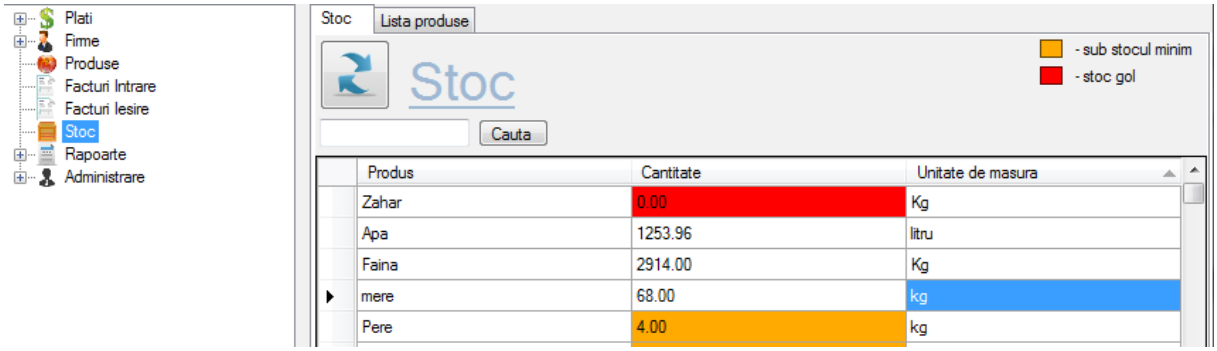

Figura 4.20. *Vizualizarea stocurilor*

Fiecare produs are un atribut numit stoc minim care precizează care este stocul minim recomandat pentru un produs. În **Figura 4.20** stocurile fiind grupate pe produse se poate printr-o operație de cuplare cu tabela de produse să se afle care este cantitatea minimă și să se compare cu stocul curent, utilitatea este exemplificată în **Figura 4.20** prin colorarea celulelor în funcție de cantitate:

```
if (stocGrupat.ProdusObject.StocMinim > stocGrupat.Cantitate)
{
   if (stocGrupat.Cantitate == 0)
   {
      e.CellStyle.BackColor = Color.Red;
      e.CellStyle.SelectionBackColor = Color.DarkRed;
   }
   else
   {
      e.CellStyle.BackColor = Color.Orange;
      e.CellStyle.SelectionBackColor = Color.DarkOrange;
   }
}
```