

**T.C.
ONDOKUZ MAYIS ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**



Veri Tabanı Laboratuvarı

Deney Föyü-4

Konu: Stored Procedure(Saklı Yordamlar), Fonksiyonlar, Cursor

STORED PROCEDURE

T-SQL (sql server da yazdığımız komutların tamamına verilen addır.) programlama dili veri tabanı ile program arasındaki ilk basamaktır.

Hazırladığımız programda veri tabanına işlem yaptırmanın iki yöntemi vardır.

1. İşlem için gerekli komutlar programdan gönderilir
2. İşlemler saklı yordamlarda saklanır. Programdan saklı yordam(stored procedure) çağrılır.

Prosedür ne demektir?

Her bir prosedür, belli bir işlevi yerine getirmek için özenle yapılandırılmış program parçasıdır. Mesela, iki sayı alıp bunların toplamalarını hesaplayan bir kod parçasını toplayıcı adında bir prosedür içerisine paketleyebiliriz. Bir prosedür, başka bir prosedür içerisinden çağrılabilir. Bu da sık kullanılan işlemler için yazılmış kodların bir defa yazılıp çok defa kullanılmasını böylelikle de programlamayı kolaylaştırmayı amaçlar.

Saklı yordamlar, birçok gelişmiş programlama dilindeki fonksiyon yapılarına karşılık gelir. Birden fazla işlemi, paketlenmiş bir halde tek bir komut ile çalıştırmamız gerektiğinde stored procedure kullanılır. İşlemden kasıt T-SQL ile yapılabilen her şeydir.

Bir SP oluşturulduktan sonra, veri tabanı sunucusunda saklanır. Her ihtiyaç duyulduğunda aynı SP defalarca çağrılabilir. Cursor gibi oturum kapandığında silinmez.

Network bazlı çalışmalarda ağ trafiği ve sistem kaynaklarının kullanımını düzenleyerek de performans artışı sağlar. Bir dize işlem, tek bir paket içerisinde yer alır. Gerektiğinde bir tek komut ile tetiklenebilir. Paketin tamamı çalışıncaya kadar istemde bulunan terminale hiçbir şey gönderilmez. Tüm komutlar bittiğinde bir tek sonuç gönderilir. Bu da bazı durumlarda ağ trafiğini rahatlatır.

Bir SP sistem tarafından oluşturulduğu anda şu aşamalara tabi tutulur:

1. SP'nin bileşenleri parçalara ayrıştırılır.
2. Veri tabanı içerisinde table, view gibi başka nesnelere atıfta bulunan referanslar varsa, geçerli olup olmadıkları kontrol edilir. (Geçerli:1-nesne varmı, 2-izin var mı)
3. Kontrollerden geçen SP'nin adı sysobjects tablosunda, kodları ise syscomments tablosunda saklanır.
4. Bu işlemlerle birlikte derleme işlemi yapılır. Normalizasyon işlemleri olarak da anılan bu işlemler sonucunda, ağaç şeması elde edilir. Bu şema da sysprocedures tablosunda saklanır.
5. SP herhangi bir anda çağrıldığında, ilk kez çalışıyorsa bu işlemler gerçekleştirilir. İlk defa çağrılmıyorsa, kontrol, sorgulama ağacı oluşturma işlemleri yapılmaz ve oldukça hızlı bir şekilde SP'nin derlenmiş hali çalışır. Bundan dolayı SP'ler derlenen nesnelerden biri olarak anılır.

Oluşturulan veya var olan saklı yordamları kullanıldığında birçok avantaj elde edilir. Bunlardan birkaçı dolayısıyla saklı yordam kullanım sebepleri aşağıdaki gibidir:

- Saklı yordamlar, SQL Server'a esneklik ve hız kazandırır.
- Önceden derlenmiş olduğu için, normal kullanılan bir SQL sorgusunun tekrar tekrar çalıştırılmasına oranla daha fazla performans elde edilmesini sağlarlar.
- Bir kez yazılıp, tekrar tekrar kullanıldığı için modüler bir yapıda program geliştirilmiş olur.
- Aynı Transact-SQL cümleciğini birden fazla yerde kullanılacağı zaman, bunu bir saklı yordam haline getirerek, kullanımını sadece ismini çağırma ile gerçekleştirilebilir.
- Belirli girdi ve çıktı parametreleri olduğu için, saklı yordamların kullanımı ile güvenlik açısından yöneticiyi de sağlama almış olur.
- Ağ trafiğini azaltır. İstemci tarafından birçok satıra sahip SQL komutunun sunucuya gitmesinden, sadece saklı yordamın adının sunucuya gitmesi ağı daha az meşgul etmiş olur.

Stored procedure Kullanımı

- *Parametre almayan Procedure kullanımı*

```
DELIMITER //
CREATE PROCEDURE SakliYordam()
BEGIN
    -- SQL Komutları
END//
DELIMITER ;
```

CREATE PROCEDURE ibaresinden sonra Procedure'e vermek istediğimiz ismi yazarız. **BEGIN** ve **END** arasında kalan alan kapsam anlamına gelir. Çalıştırılacak kodlar bu kapsam arasına yazılır.

Örneğin Kisiler tablosunda bulunan tüm kayıtları getirmek için yazmamız gereken Procedure aşağıdaki gibi olacaktır.

```
DELIMITER //
CREATE PROCEDURE Kisiler_SelectAll()
BEGIN
    SELECT * FROM Kisiler;
END//
DELIMITER ;
```

Peki bu Store Procedure nasıl çalıştırılır? Bunu da **CALL** komutu ile yaparız.

```
CALL Kisiler_SelectAll();
```

Komutunun çalıştırılması ile bize dönen sonuç aşağıdaki gibidir.

Result Grid

Filter Rows:

Export:

| # | id | adi | soyadi | telefon | yas | |
|---|----|-------|----------|-------------|-----|--|
| 1 | 1 | Nur | Kozan | 0555xxxxxxx | 24 | |
| 2 | 2 | Omer | Yaygumus | 0553xxxxxxx | 24 | |
| 3 | 3 | Kubra | Kozan | 0552xxxxxxx | 20 | |
| 4 | 4 | Esra | Kozan | 0554xxxxxxx | 22 | |

- **Parametrelili Store Procedure kullanımı**

Yarattığımız saklı yordamlar genellikle parametrelili olurlar. Parametre ile saklı yordama deger gönderebilir, deger alabiliriz. Parametreler saklı yordamları daha esnek ve kullanılabilir kılar. Mysql de saklı yordam parametreleri üç yöntem belirtecinden birini alabilir. Bu yöntem belirteçleri IN, OUT, INOUT olabilir.

IN Yöntem Belirteci:

IN yöntem belirtecini biz parametreye sadece deger göndereceksek kullanırız. Yani IN yöntem belirteci ile gonderdiğimiz deger stored procedure'un parametre degiskenine kopyalanır ve Stored procedure'ın gövdesi içerisinde bilinir durumdadır.

OUT Yöntem Belirteci

Bu yöntem belirteci ile tanımlanmış bir parametreye dışarıdan bir değişken gönderip stored procedure'un bu değişkene bir deger atamasını sağlarız. Sonunda da stored procedure işlemesi bittiğinde bu degeri kullanabiliriz.

INOUT Yöntem Belirteci:

INOUT ismindende anlaşılacağı gibi IN ve OUT yöntem belirteçlerinin birleşimidir. Yani hem stored procedure'e deger gönderilebilir hemde stored procedureden bir deger alınabilir. Bu yöntem diğer programlama dillerindeki referans parametre tipine benzemektedir. Yani değişken stored procedure'a referansı vasıtası ile geçirilir.

```
DELIMITER //
CREATE PROCEDURE SakliYordam(parametre1,parametre2,parametreN)
BEGIN
    -- SQL Komutları
END//
DELIMITER ;
```

Yine Kisiler tablosundan bir Procedure ile örneklendirelim.

```
Query 1 x Kisiler x Kisiler x SQL File 1* x
Limit to 1000 rows
1 DELIMITER //
2
3 CREATE PROCEDURE KisiEkle
4 (
5     IN gelen_kisi_id INT, IN gelen_kisi_adi VARCHAR(40),
6     IN gelen_kisi_soyadi VARCHAR(40), IN gelen_kisi_telefon VARCHAR(20),
7     IN gelen_kisi_yas INT
8 )
9 BEGIN
10
11     INSERT INTO Kisiler(id, adi, soyadi, telefon, yas)
12     VALUES(gelen_kisi_id, gelen_kisi_adi, gelen_kisi_soyadi,
13     gelen_kisi_telefon, gelen_kisi_yas );
14
15 END//
16 DELIMITER ;
```

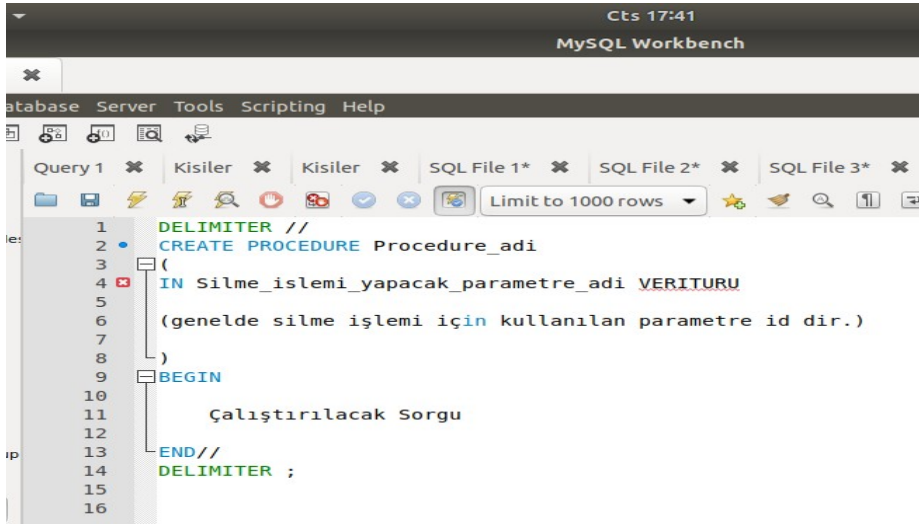
Parametrelili Stored Procedure’de parametresizden farklı olarak sorguda kullanılacak olan parametreler procedür isminden sonra () içlerinde tanımlanır ve bu tanımlama yaparken de IN anahtar sözcüğü kullanılır. Ayrıca bu parametrenin tipi tabloda kullanıldığı gibi yazılır. Parametreler istenildiği kadar artırılıp azaltılır. Daha sonra **BEGIN** ve **END** aralığına parametrelili sorgu yazılır. Çalıştırma kısmında ise parametresiz sorguda olduğu gibi **CALL** komutu ile çalıştırılır ancak burada farklı olarak procedure’de istenen parametre değerleri de buradan yollarılır.

```
Query 1 x Kisiler x Kisiler x SQL File 1* x SQL File 2* x
Limit to 1000 rows
1 CALL KisiEkle(5, 'Duygu', 'Aksehir', '0545XXXXXXX', 21);
```

Yazdığımız parametrelili Stored Procedure’deki id parametresine 5 değerini, adi parametresine ‘Duygu’ değerini, soyadi parametresine ‘Aksehir’ değerini, telefon parametresine ‘0545XXXXXXX’ değerini, yas parametresine ise 21 değerini göndererek Procedure’u çalıştırdık. Bunun sonucunda bize dönen sonuç aşağıdaki gibidir.

| Result Grid | | | | | | |
|--|----|-------|----------|-------------|-----|--|
| Filter Rows: <input type="text"/> | | | | | | |
| Export: <input type="text"/> Wrap Cell Content: <input type="text"/> | | | | | | |
| # | id | adi | soyadi | telefon | yas | |
| 1 | 1 | Nur | Kozan | 0555xxxxxxx | 24 | |
| 2 | 2 | Omer | Yaygumus | 0553xxxxxxx | 24 | |
| 3 | 3 | Kubra | Kozan | 0552xxxxxxx | 20 | |
| 4 | 4 | Esra | Kozan | 0554xxxxxxx | 22 | |
| 5 | 5 | Duygu | Aksehir | 0545XXXXXXX | 21 | |

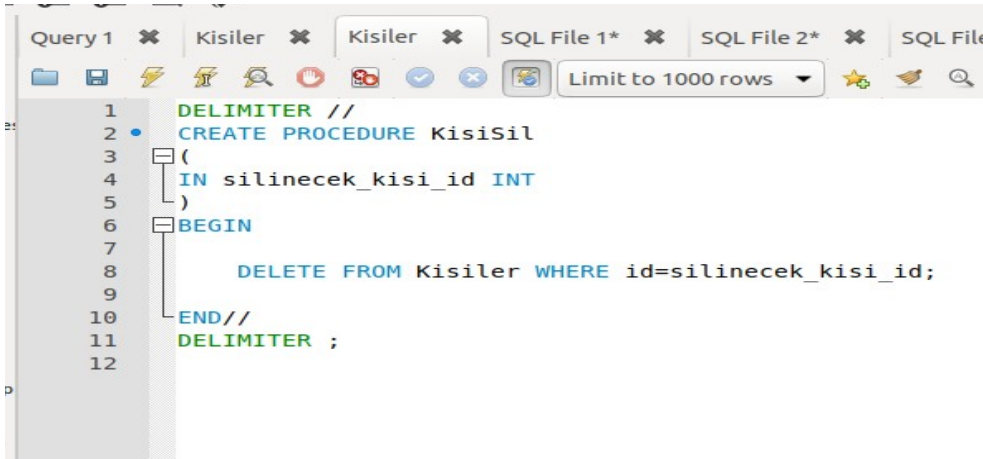
Silme işlemi yapan Procedure kullanımı



```
1 DELIMITER //
2 CREATE PROCEDURE Procedure_adi
3 (
4     IN Silme_islemi_yapacak_parametre_adi VERITURU
5 )
6     (genelde silme işlemi için kullanılan parametre id dir.)
7 )
8 BEGIN
9     Çalıştırılacak Sorgu
10
11 END//
12 DELIMITER ;
```

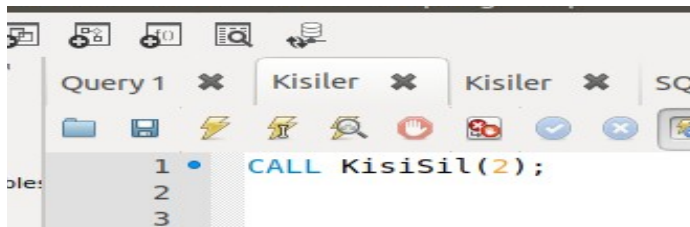
Silme işlemi yapan bir procedure yazmak istiyorsak parametre olarak Id'yi kullanırız demiştik ancak bu zorunlu bir kural olarak düşünülmemelidir. Silenecek kayda Id'sinden ulaşabiliriz mantığı ile bunu ifade ettik. Farklı bir şekilde de kullanılabilir.

Yine Kisiler tablomuzdan faydalanarak bir kaydı silme işlemi yapan procedure yazalım.



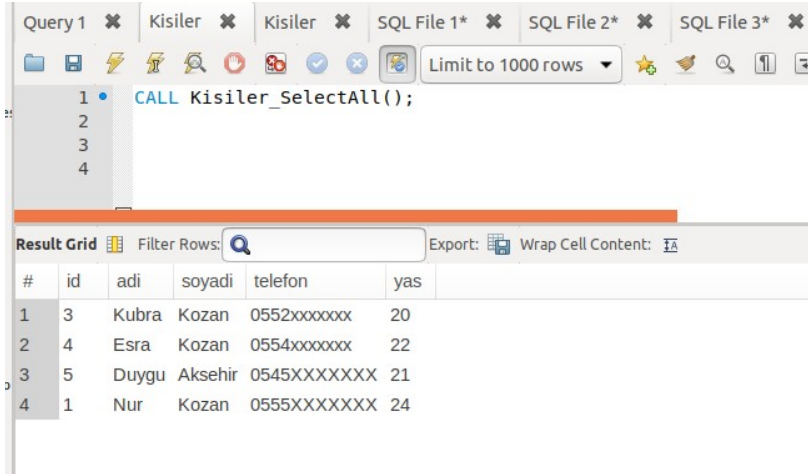
```
1 DELIMITER //
2 CREATE PROCEDURE KisiSil
3 (
4     IN silinecek_kisi_id INT
5 )
6 BEGIN
7     DELETE FROM Kisiler WHERE id=silinecek_kisi_id;
8
9 END//
10 DELIMITER ;
```

Yukarıdaki kod çalıştırılıp silme işlemi yapan Procedure oluşturulduktan sonra yine **CALL** komutu ile bu Procedure çalıştırıldı. Procedure çalıştırdıktan sonra Id'sini verdiğimiz kayıt silinir. Kayıt silindikten sonra tablonun son halini görmek için yukarıda yazmış olduğumuz Procedure **Kisiler_SelectAll**' u tekrar çalıştırabiliriz. Procedure'lerin bir kez yazılıp derlenip tekrar tekrar çalıştırılabilirliklerini zaten yukarıda belirtmiştik.



```
1 CALL KisiSil(2);
2
3
```

Yukarıda bulunan kod satırını çalıştırdığımız zaman aşağıdaki sonucu elde ederiz. Gördüğünüz gibi 2 numaralı Id' ye sahip Omer isimli kayıt silindi.



Query 1 x Kisiler x Kisiler x SQL File 1* x SQL File 2* x SQL File 3* x

Limit to 1000 rows

```
1 CALL Kisiler_SelectAll();
2
3
4
```

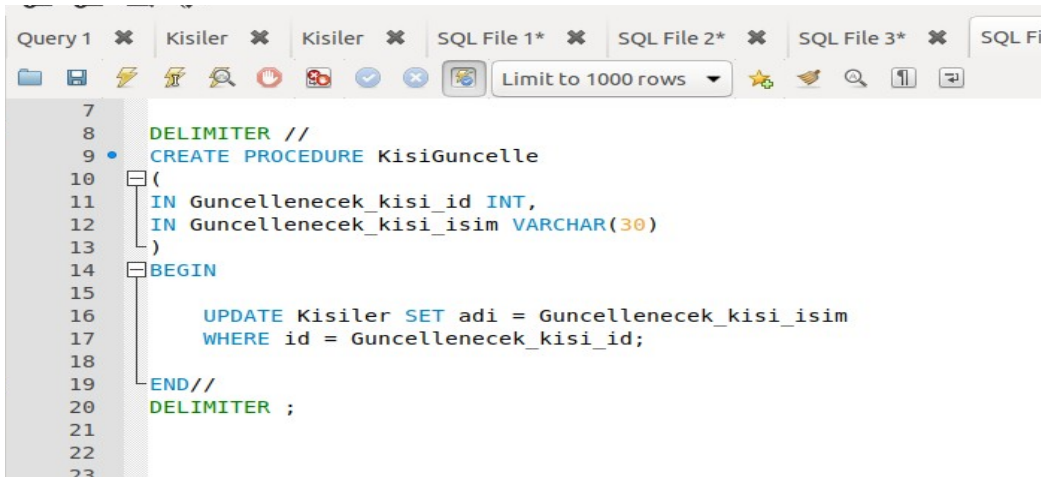
Result Grid Filter Rows: Export: Wrap Cell Content: I

| # | id | adi | soyadi | telefon | yas |
|---|----|-------|---------|-------------|-----|
| 1 | 3 | Kubra | Kozan | 0552xxxxxxx | 20 |
| 2 | 4 | Esra | Kozan | 0554xxxxxxx | 22 |
| 3 | 5 | Duygu | Aksehir | 0545XXXXXXX | 21 |
| 4 | 1 | Nur | Kozan | 0555XXXXXXX | 24 |

NOT: Silme ve güncelleme prosedürlerini oluştururken Error Code:1175 ile ilgili hata alırsanız SET SQL_SAFE_UPDATES = 0 ; komutunu yazıp çalıştırmanız yeterli olacaktır.

Güncelleme işlemi yapan Procedure kullanımı

Güncelleme yapan Procedurede' de en az iki parametre kullanılır. İlk parametre ile güncelleme yapılacak olan kayıt belirlenir. Farklı bir kullanıma gidilmediği sürece kayıt belirleyici parametre Id' dir. Diğer parametre ise kayıta değişiklik yapılacak olan kolondur. Yine Kisiler tablosu üzerinden örneklendirecek olursak;

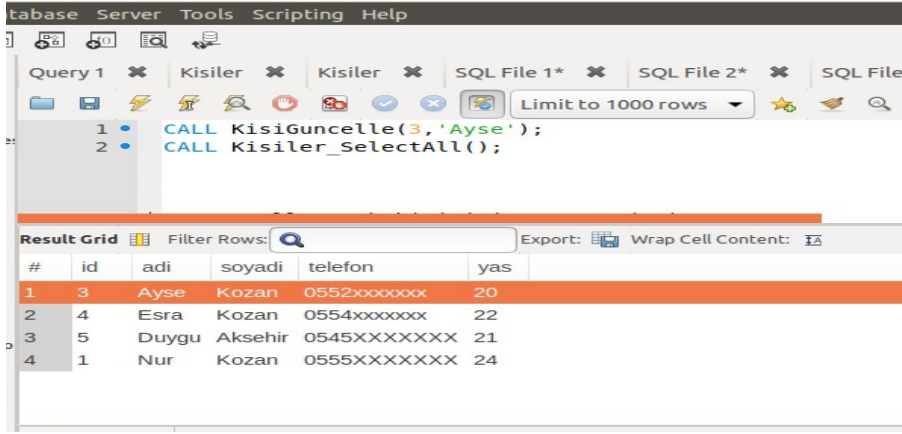


Query 1 x Kisiler x Kisiler x SQL File 1* x SQL File 2* x SQL File 3* x SQL Fi

Limit to 1000 rows

```
7
8 DELIMITER //
9 CREATE PROCEDURE KisiGuncelle
10 (
11     IN Guncellenecek_kisi_id INT,
12     IN Guncellenecek_kisi_isim VARCHAR(30)
13 )
14 BEGIN
15
16     UPDATE Kisiler SET adi = Guncellenecek_kisi_isim
17     WHERE id = Guncellenecek_kisi_id;
18
19 END//
20 DELIMITER ;
21
22
23
```

Yukarıda ki Procedure'de Id' si verilen kaydın Adi alanı parametre olarak yollanan değer ile değiştirilsin böylece o kayıt güncellensin isteniyor.



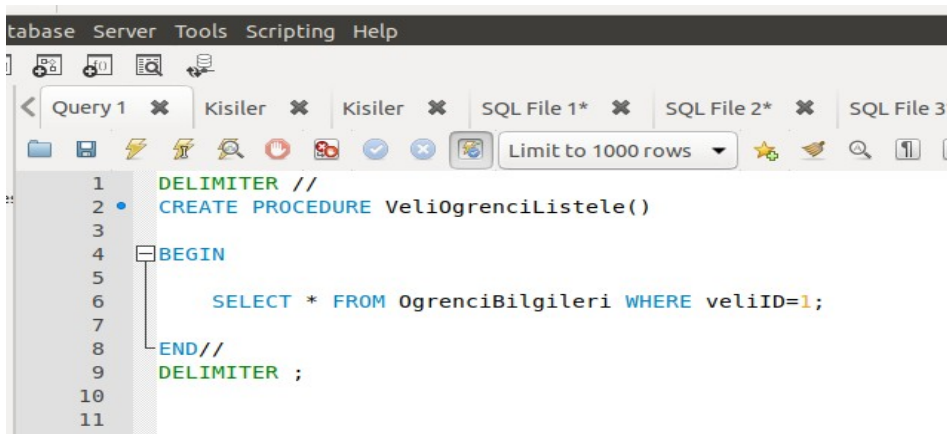
Update Procedure yukarıda bulunan komutlar ile çalıştırıldı ve yukarıda verilen çıktıda gördüğünüz gibi Id 'si 3 olan kaydın adi alanına 'Ayse' stringi yazılıyor.

Örnek

Aşağıda verilmiş olan OgrenciBilgileri tablosunda veliId'si 1 olan öğrencileri listeleyen bir stored procedure oluşturalım.

| # | ogrenciID | ogrenciNo | ogrenciAd | ogrenciSoyad | ogrenciAlan | veliID |
|---|-----------|-----------|-----------|--------------|-------------|--------|
| 1 | 1 | 121 | Ahmet | Gunes | 1 | 1 |
| 2 | 2 | 122 | Mustafa | Yildiz | 1 | 2 |
| 3 | 3 | 123 | Ayşe | Gunes | 2 | 1 |
| 4 | 5 | 200 | Fatma | Yildiz | 2 | 2 |

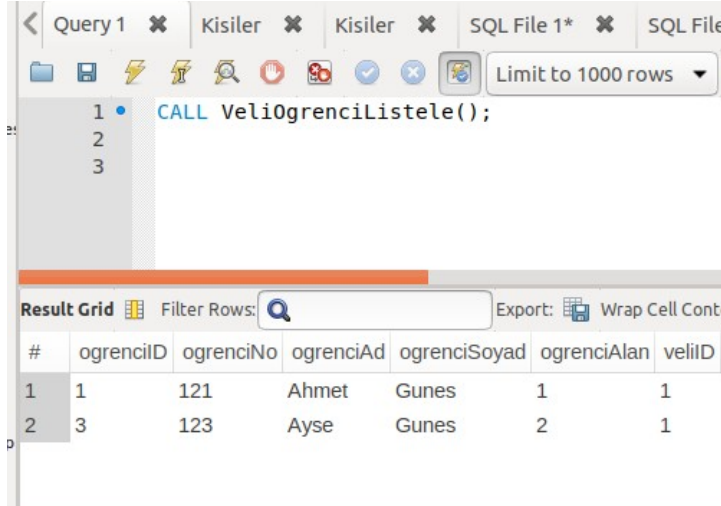
New SQL Script butonuna tıklayarak yeni bir sorgu penceresi açalım. Sorgu penceresinde SQL kod bloğumuzu aşağıdaki gibi oluşturalım.



Şimdi Execute butonuna tıklayarak sorgumuzu çalıştıralım. Çalıştırdıktan sonra sonuç penceresinde herhangi bir liste oluşmayacaktır. Biz burada sadece ağ veri tabanı yazılımımıza

yeni bir stored procedure tanımladık. Tanımlamış olduğumuz stored procedure listesine Stored Procedures düğümünden erişebiliriz.

Tanımlamış olduğumuz saklı yordamı çalıştırmak için CALL komutunu kullanırız. Şimdi tanımladığımız VeliOgrenciListe saklı yordamını çağıralım ve sonucu görelim. Yeni bir sorgu penceresi açalım. Sorgu penceremize aşağıdaki komutu yazalım ve çalıştıralım. Sonuç penceresinde aşağıdaki gibi bir liste oluşacaktır.



The screenshot shows a SQL query editor with a query window containing the command `CALL VeliOgrenciListele();`. Below the query window, the 'Result Grid' is displayed, showing the results of the stored procedure call. The grid has columns for row number, ogrenciID, ogrenciNo, ogrenciAd, ogrenciSoyad, ogrenciAlan, and veliID. Two rows of data are shown.

| # | ogrenciID | ogrenciNo | ogrenciAd | ogrenciSoyad | ogrenciAlan | veliID |
|---|-----------|-----------|-----------|--------------|-------------|--------|
| 1 | 1 | 121 | Ahmet | Gunes | 1 | 1 |
| 2 | 3 | 123 | Ayşe | Gunes | 2 | 1 |

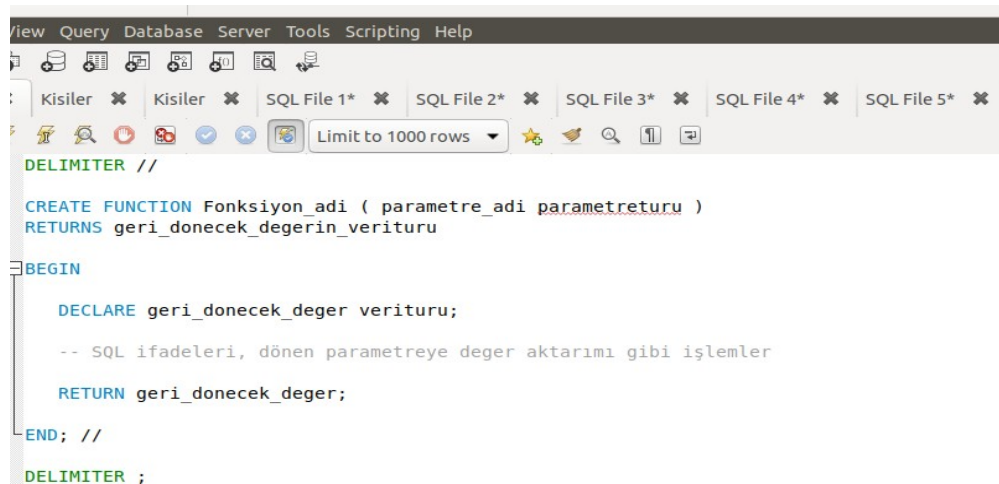
KULLANICI TANIMLI FONKSİYON KULLANIMI

Kullanıcı tanımlı fonksiyonlar, SQL'de tanımlı olan fonksiyonlar gibi aldığı parametreleri işleyerek geriye bir değer döndüren sql ifadeleridir. Özellikle procedurelerden en büyük farkı sorguların içinde direk kullanılabilmeleridir.

Fonksiyon Çeşitleri

- 1.Geriye sabit değer döndüren fonksiyonlar
- 2.Geriye Sorgu döndüren fonksiyonlar
- 3.Geriye tablo değişkeni döndüren fonksiyonlar

Geriye sabit değer döndüren fonksiyon oluşturma



The screenshot shows a SQL query editor with a query window containing the code to create a user-defined function. The code is as follows:

```
DELIMITER //

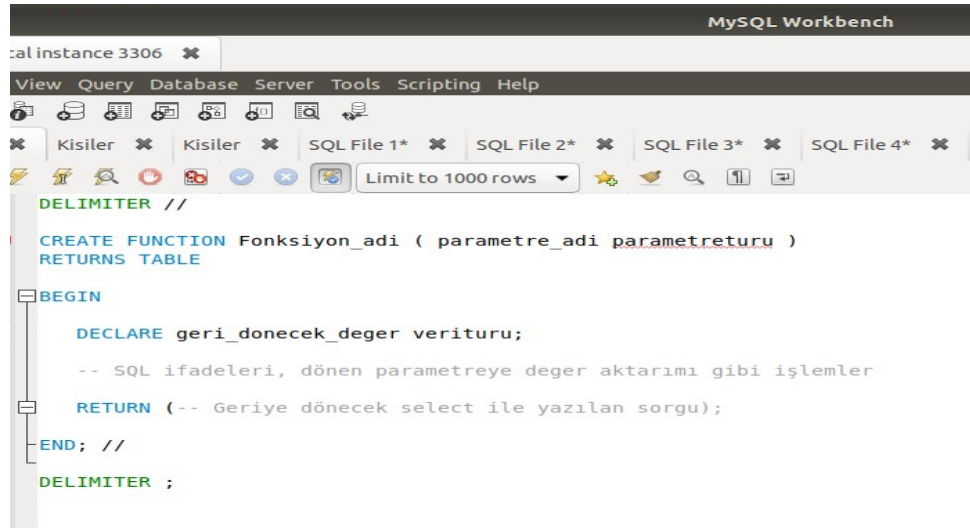
CREATE FUNCTION Fonksiyon_adi ( parametre_adi parametreturu )
RETURNS geri_donecek_degerin_verituru
BEGIN
    DECLARE geri_donecek_deger verituru;

    -- SQL ifadeleri, dönen parametreye deger aktarımı gibi işlemler

    RETURN geri_donecek_deger;
END; //

DELIMITER ;
```

Geriye Sorgu döndüren fonksiyon oluşturma



```
DELIMITER //  
  
CREATE FUNCTION Fonksiyon_adi ( parametre_adi parametreturu )  
RETURNS TABLE  
  
BEGIN  
  
    DECLARE geri_donecek_deger verituru;  
  
    -- SQL ifadeleri, dönen parametreye deger aktarımı gibi işlemler  
  
    RETURN (-- Geriye dönecek select ile yazılan sorgu);  
  
END; //  
  
DELIMITER ;
```

Geriye tablo değişkeni döndüren fonksiyon oluşturma

Geriye tablo döndüğü için select ile kullanırken **from** ifadesinden sonra fonksiyonun kullanılması gerekir.

Fonksiyon Üzerinde Değişiklik Yapma

Fonksiyon üzerinde değişiklik yapmak için yukarıdaki tanımlama şekillerinde olduğu gibi **Create function..** yerine **Alter function....** yaparak fonksiyon üzerinde değişiklik yapmak mümkündür.

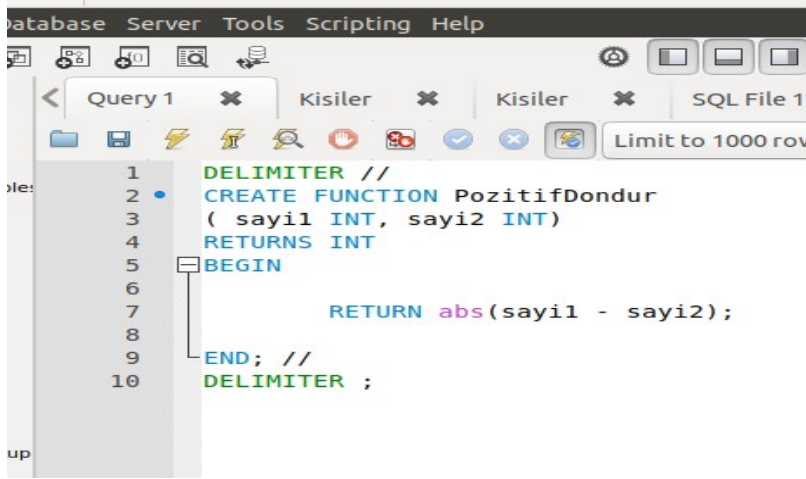
Kullanıcı Tanımlı Fonksiyonları Silme

View, Procedure' lerde olduğu gibi Drop ile siliyoruz.

```
1  
2 Drop function function_adi  
3
```

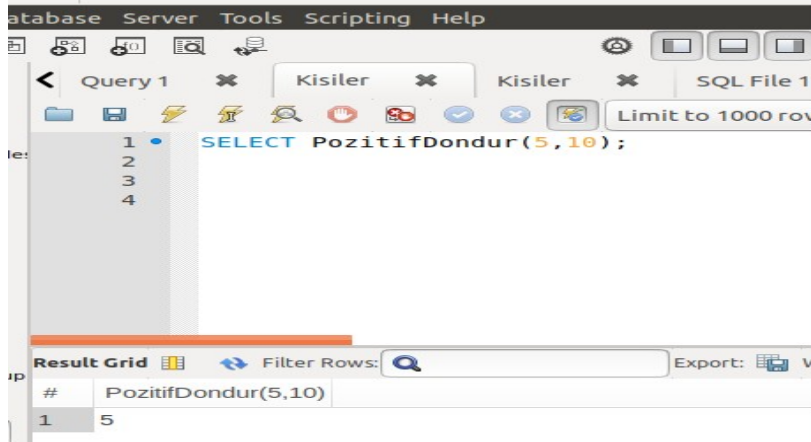
Örnek: Girilen iki parametreyi birbirinden çıkarıp geriye pozitif sonuc döndüren fonksiyon oluşturunuz.

Yöntem1:



```
1 DELIMITER //
2 CREATE FUNCTION PozitifDondur
3 ( sayi1 INT, sayi2 INT)
4 RETURNS INT
5 BEGIN
6
7     RETURN abs(sayi1 - sayi2);
8
9 END; //
10 DELIMITER ;
```

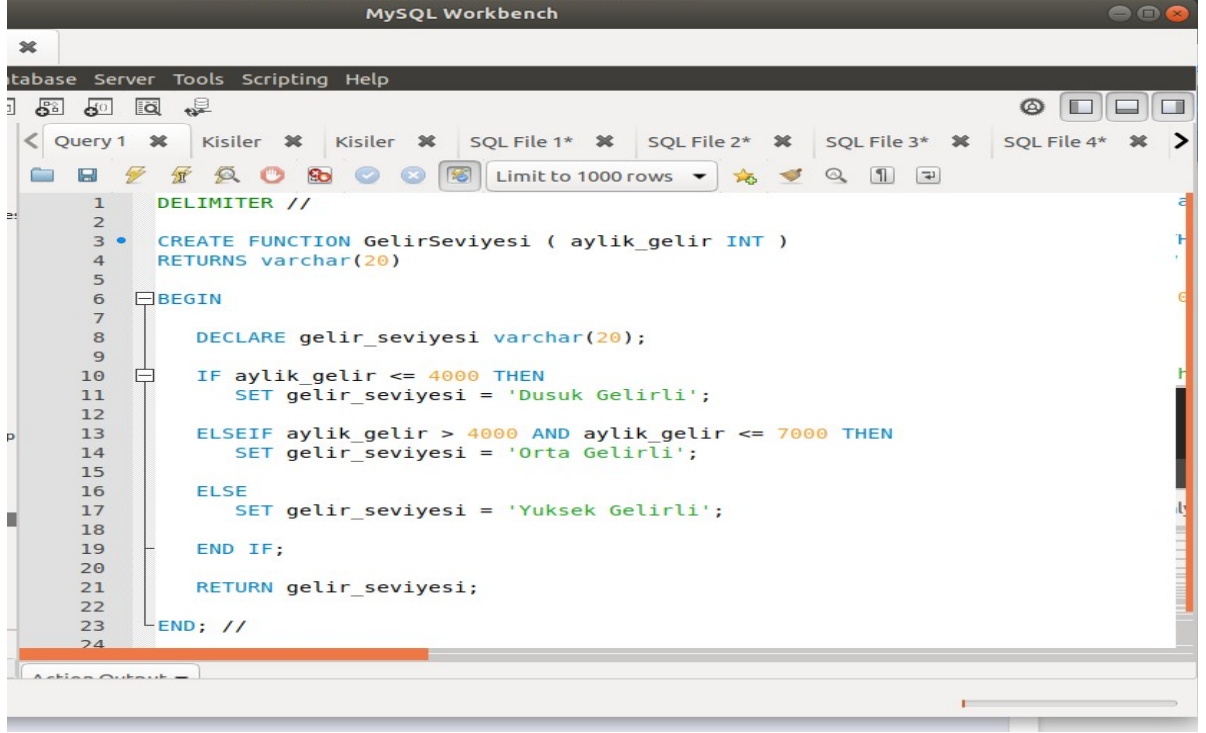
Kullanım:



```
1 SELECT PozitifDondur(5,10);
2
3
4
```

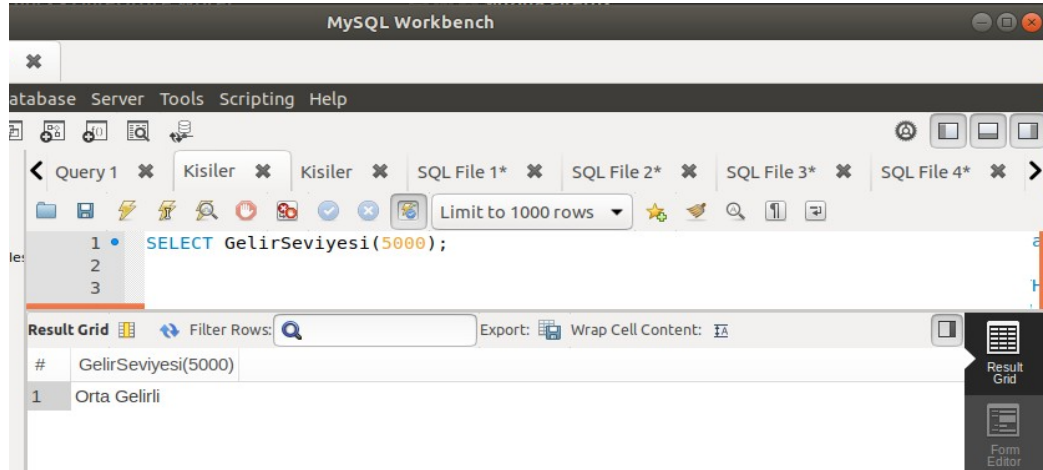
| # | PozitifDondur(5,10) |
|---|---------------------|
| 1 | 5 |

Örnek: Parametre olarak aylık gelir değerini alıp gelir seviyesinin ne olduğuna karar veren bir fonksiyon oluşturunuz.



```
1 DELIMITER //
2
3 CREATE FUNCTION GelirSeviyesi ( aylık_gelir INT )
4 RETURNS varchar(20)
5
6 BEGIN
7
8     DECLARE gelir_seviyesi varchar(20);
9
10    IF aylık_gelir <= 4000 THEN
11        SET gelir_seviyesi = 'Dusuk Gelirli';
12
13    ELSEIF aylık_gelir > 4000 AND aylık_gelir <= 7000 THEN
14        SET gelir_seviyesi = 'Orta Gelirli';
15
16    ELSE
17        SET gelir_seviyesi = 'Yuksek Gelirli';
18
19    END IF;
20
21    RETURN gelir_seviyesi;
22
23 END; //
```

Kullanım:



```
1 SELECT GelirSeviyesi(5000);
2
3
```

| # | GelirSeviyesi(5000) |
|---|---------------------|
| 1 | Orta Gelirli |

CURSOR

Cursor bir veri parçası veya parçalarını “sistematik” bir şekilde döndürmemizi sağlayan T-SQL komududur. Yapı olarak pek çok benzemesede programlama dillerinde kullandığımız for,foreach yapısına benzemektedir.

Cursor’ı daha iyi anlamanız için basit bir senaryo üzerinden gidelim. Şimdi diyelim ki üniversitede herhangi bir dersin hocasısınız. Öğrencilere ait sınav sonuçlarını da bir tabloda tuttuğunuzu varsayalım. Senaryo gereği öğrenci tablosunu bir temp tablo olarak tanımlayacağız.

```
CREATE TABLE #Ogrenci(  
id INT IDENTITY,  
ad NVARCHAR(20),  
soyad NVARCHAR(20),  
vize TINYINT,  
final TINYINT,  
Status NVARCHAR(15))
```

```
INSERT #Ogrenci (ad,soyad,vize,final) VALUES (N'Fatih' , N'ÖZGÜR' , 65, 50)  
INSERT #Ogrenci (ad,soyad,vize,final) VALUES (N'Salih' , N'KARABIYIK' , 48, 62)  
INSERT #Ogrenci (ad,soyad,vize,final) VALUES (N'Mustafa' , N'BİL' , 75, 85)  
INSERT #Ogrenci (ad,soyad,vize,final) VALUES (N'Ali' , N'ÇULOĞLU' , 35, 40)  
INSERT #Ogrenci (ad,soyad,vize,final) VALUES (N'Fatma' , N'ENGÜL' , 79, 60)  
INSERT #Ogrenci (ad,soyad,vize,final) VALUES (N'Ayşe' , N'LALOĞLU' , 62, 59)  
INSERT #Ogrenci (ad,soyad,vize,final) VALUES (N'Nilgün' , N'DALBUR' , 35, 55)  
INSERT #Ogrenci (ad,soyad,vize,final) VALUES (N'Fırat' , N'HOKKA' , 70, 35)  
INSERT #Ogrenci (ad,soyad,vize,final) VALUES (N'Filiz' , N'SAYGINLI' , 74, 40)
```

Şimdi yılsonunda şöyle bir rapor ve iş istediğinizi varsayalım. Bütünleme sınavına kalan öğrencilerin isimlerini bir tabloya yazmak ve bunları listelemek ve öğrenci tablosundaki Status alanına ise ortalamayı geçenleri ‘GEÇTİ’ kalanları ise ‘BUTE KALDI’ olarak değer atamak istiyoruz. İşte bunun gibi durumlarda Cursor imdadımıza yetişiyor.

Tekrar senaryo gereği bütünlemeye kalanlar için bir temp tablo yaratalım.

```
CREATE TABLE #ButeKalanlar(  
ad NVARCHAR(20),  
soyad NVARCHAR(20)  
)
```

Cursor’da tanımlamaya başlamadan önce bizlerin hangi değerleri döndürmemiz gerekiyor ilk önce bunlara ait değişken tanımlamamız gerekli.

```
DECLARE @id INT  
DECLARE @midTermExam INT  
DECLARE @finalExam INT
```

Daha sonrasında ise cursor tanımlamamızı gerçekleştiriyoruz. Buradaki püf nokta şu yukarıda tanımladığımız değişkenlere sorguda aldığımız alanların sırayla gitmesi. Yani @id değişkeni

öğrenci tablosundaki id alanı ile @midtermExam değişkeni öğrenci tablosunda vize alanına denk gelmektedir.

```
DECLARE CRS_SINAV CURSOR FOR
SELECT id,
       vize,
       final
FROM #Ogrenci
```

Tanımlamamız bittikten sonra ise Cursor'ı açıyoruz.

```
OPEN CRS_SINAV
```

Şimdi ise cursor'da döndürmemiz gereken değerleri FETCH komutu ile döndürüyoruz.

```
FETCH NEXT FROM CRS_SINAV INTO @id,@midTermExam,@finalExam
```

SQL Server'da FETCH komutunun durumunu @@FETCH_STATUS metodu içinde saklar. Eğer FETCH sorunsuz bir şekilde dönmüşse "0", başarısız ise "-1", FETCH edilen satır yok ise "-2" değeri alınır. Öyleyse While ile FETCH içerisinde dönebiliriz.

```
WHILE @@FETCH_STATUS =0
```

Şimdi ise bu döngü içerisinde yapacağımız işlere sıra geldi. Öncelikle Ortalama adında bir değişken oluşturalım. Burada önemli olan durum şu döngü içerisinde tanımlayacağımız her değişkenin bir ilk değeri olması gerekli yoksa yanlış sonuçlar olabilmektedir.

```
BEGIN
```

```
    DECLARE @Ortalama FLOAT
    SET @Ortalama = 0
```

```
    IF(((@midTermExam * 30) / 100 + (@finalExam * 70) /100) >= 50)
```

```
    BEGIN
```

```
        UPDATE #Ogrenci
```

```
        SET Status = 'GEÇTİ'
```

```
        WHERE id = @id
```

```
    END
```

```
    ELSE
```

```
    BEGIN
```

```
        UPDATE #Ogrenci
```

```
        SET Status = 'BUTE KALDI'
```

```
        WHERE id = @id
```

```
    INSERT INTO #ButeKalanlar
```

```
( ad, soyad )  
SELECT ad,soyad FROM #Ogrenci WHERE id = @id
```

```
END
```

```
FETCH NEXT FROM CRS_SINAV INTO @id,@midTermExam,@finalExam
```

```
END
```

Yukarda ne yaptık kısa bir özet geçelim. Eğer vizenin %30'u ve finalin %70 toplamı 50 ve üzeri ise öğrenci tablosundaki Status alanını geçti değeri set et eğer 50 aşağısında ise status alanına hem değer set et hem de kalan kişiyi ButeKalanlar tablosuna insert et. Sonrasında ise FETCH ile sıradaki veri kümesini döndürüyoruz.

Bundan sonra ise Cursor'ı kapatıyoruz.

```
CLOSE CRS_SINAV
```

Ayrıca ek olarak cursor' bellekten silmek için DEALLOCATE ile cursor'ı sonlandırıyoruz . Bir nevi Droplama işlemi diyebiliriz.

```
DEALLOCATE CRS_SINAV
```

Şimdi bakalım tablolarımızda neler var...

```
SELECT * FROM #Ogrenci  
SELECT * FROM #ButeKalanlar
```

```
DROP TABLE #Ogrenci  
DROP TABLE #ButeKalanlar
```

| | id | ad | soyad | vize | final | Status |
|---|----|---------|-----------|------|-------|------------|
| 1 | 1 | Fatih | ÖZGÜR | 65 | 50 | GEÇTİ |
| 2 | 2 | Salih | KARABİYİK | 48 | 62 | GEÇTİ |
| 3 | 3 | Mustafa | BİL | 75 | 85 | GEÇTİ |
| 4 | 4 | Ali | ÇULOĞLU | 35 | 40 | BUTE KALDI |
| 5 | 5 | Fatma | ENGÖL | 79 | 60 | GEÇTİ |
| 6 | 6 | Ayşe | LALOĞLU | 62 | 59 | GEÇTİ |
| 7 | 7 | Nilgün | DALBUR | 35 | 55 | BUTE KALDI |
| 8 | 8 | Fırat | HOKKA | 70 | 35 | BUTE KALDI |
| 9 | 9 | Filiz | SAYGINLI | 74 | 40 | GEÇTİ |

| | ad | soyad |
|---|--------|---------|
| 1 | Ali | ÇULOĞLU |
| 2 | Nilgün | DALBUR |
| 3 | Fırat | HOKKA |

RAPORDA YAPILMASI İSTENENLER

Soru 1) Girilen 1. parametrenin 2.parametreye tam bölünüp bölünmediğini yazdıran fn_BolunurMu adında fonksiyon yazınız ve bu fonksiyonunun kullanımını gösteriniz.

Soru 2) Parametre olarak verilen sayının faktöriyel değerini hesaplayan fonksiyonu yapınız.

Soru 3) ad, soyad, bolum ve maas kolonlarından meydana gelen ‘calisanlar’ isminde tablo oluşturun ve bu tabloya 3 kişi ekleyin.(SQL ifadeleri ile)

- a) Bölüm bilgisinden çalışan personellerin maaşına %25 zam yapan prosedür yazınız.
- b) Parametre olarak bölüm bilgisini alıp bu bölümdeki çalışanların maaşları toplamını döndüren prosedürü yazınız.

Soru 4) tOgrenci tablosu, ogrenciID (Primary Key), ad, soyad, dogumtarih, adres, telefon, bolumID, vizenotu, finalnotu bilgilerini içermektedir.

Cursor kullanarak adı içerisinde “a ve i” harfleri olan öğrencilerin vize notlarına 5 puan ekleme yapan bu harflerin geçmediği öğrencilerin vizenotlarından 5 puan eksiltten bir cursor uygulaması yapınız. (SQL kodları ile ifade edebilirsiniz.)