

**SQL Language Center**



Hazırlayan: Eldiana Belekova

Öğretmen Görevlisi: Mehmet Hepkorucu

Tarih: 01.05.2025

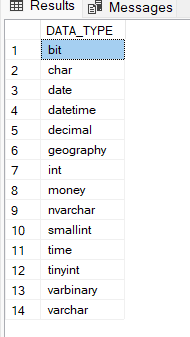
**Giriş:**

Bu proje, **LanguageCenter Dil Kursu** veritabanının yönetimini ele alır. Veritabanı tasarımı, **SQL Server** üzerinde gerçekleştirilmiş olup, **veritabanı yönetim sistemleri** kullanılarak veriler üzerinde çeşitli işlemler yapılmıştır. Veritabanında yer alan **Kurslar**, **Öğrenciler**, **Öğretmenler**, **Kayıtlar** gibi tablolar arasında **INNER JOIN**, **LEFT JOIN**, **RIGHT JOIN** ve **FULL OUTER JOIN** kullanılarak ilişkiler kurulmuş ve ilişkili veriler elde edilmiştir. **Proje, 26 tablodan oluşmaktadır.** Projede kullanılan kısıtlamalar ve ilişkiler, verilerin doğru ve tutarlı olmasını sağlamak için büyük önem taşımaktadır.

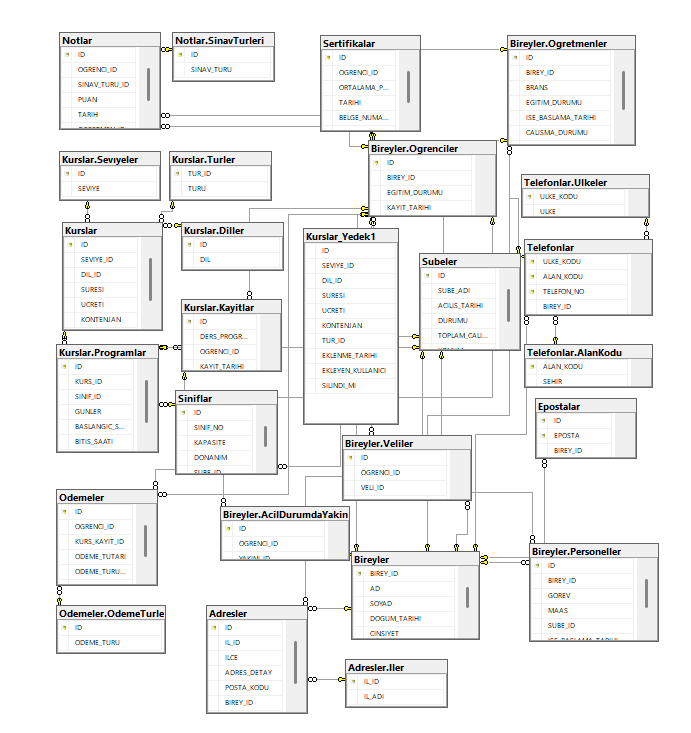
**Projenin Amacı:**

Projenin amacı, LanguageCenter Dil Kursu’nun öğrenci kayıtlarını, öğretmen bilgilerini, kurs programlarını ve ders içeriklerini düzenli bir şekilde takip edebileceği bir veri tabanı oluşturmaktır. Dil kursları; öğrenci kayıtları, kurs takvimleri ve öğretmen atamaları gibi işlemleri yoğun şekilde yürüttüğünden, bu süreçlerin elle yönetilmesi hatalara ve zaman kaybına yol açmaktadır. Bu proje sayesinde tüm veriler tek bir sistemde toplanarak, kurs yönetimi daha hızlı, düzenli ve hatasız bir şekilde yapılabilecek; yöneticiler ve çalışanlar için iş yükü azalacaktır.

**Kullandıgım Veri Tipler:**

****

**Diagram:**

****

**JOINLAR**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 02.05.2025

\* Description:

\* - Bu dört sorgu, farklı JOIN türlerinin nasıl çalıştığını gösterir.

\* - Her sorgu farklı bir veri ilişkisini analiz eder.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

--INNER JOIN

-- OGRENCINI aldıgı notların hangı sınav turu oldugunu gosteriyor

SELECT NS.ID, NS.SINAV\_TURU , N.PUAN, N.TARIH

FROM [Notlar.SinavTurleri] NS

INNER JOIN Notlar N

ON NS.ID = N.SINAV\_TURU\_ID

--LEFT JOIN

--Bu sorgu, tüm sınıfların hangi programlara bağlı olduğunu (ve bağlı değilse bile sınıfın listede görünmesini) sağlar.

SELECT S.ID, S.SINIF\_NO,S.KAPASITE, KP.ID, KP.GUNLER,KP.BASLANGIC\_SAATI,KP.BITIS\_SAATI

FROM [Siniflar] S

LEFT JOIN [Kurslar.Programlar] KP

ON KP.KURS\_ID = S.ID

--RIGHT JOIN

--Öğretmenlerin hangi şubede çalıştığı

SELECT

OG.ID AS OGRETMEN\_ID,

OG.BIREY\_ID,

OG.BRANS,

OG.EGITIM\_DURUMU,

OG.MAASI,

S.SUBE\_ADI,

S.ACILIS\_TARIHI,

S.DURUMU,

S.TOPLAM\_CALISANLAR,

S.KONUM

FROM [dbo].[Bireyler.Ogretmenler] OG

RIGHT JOIN [dbo].[Subeler] S

ON OG.SUBE\_ID = S.ID;

--FULL OUTER JOIN

--Bireylerin eposta adresi olup olmadığını kontrol et.

SELECT B.AD, B.SOYAD, E.EPOSTA

FROM [dbo].[Bireyler] B

FULL JOIN [dbo].[Epostalar] E

ON B.BIREY\_ID = E.BIREY\_ID;

**Stored Procedure: Kayıt Ekleme INSERT**

**Amaç:** Bir öğrencinin sınav notlarını kullanarak ortalama puanını hesaplar ve ardından bu ortalamayı "Sertifikalar" tablosuna ekler. Öğrencinin OGRENCI\_ID'si verildiğinde, tüm notları alınarak ortalama hesaplanır ve bu ortalama ile birlikte sertifika kaydı oluşturulur.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 02.05.2025

\* Description:

\* - Öğrencinin not ortalamasını hesaplar ve sertifikalar tablosuna ekler.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE PROCEDURE OrtalamaHesaplaVeSertifikalaraEkle

@OgrenciID INT

AS

BEGIN

DECLARE @Ortalama DECIMAL(5,2);

SELECT @Ortalama = AVG(puan) FROM Notlar

WHERE OGRENCI\_ID = @OgrenciID;

INSERT INTO Sertifikalar ([OGRENCI\_ID],[ORTALAMA\_PUAN] ,[TARIHI])

VALUES (@OgrenciID, @Ortalama, GETDATE());

END;

EXEC OrtalamaHesaplaVeSertifikalaraEkle @OgrenciID = 8;

**Stored Procedure: Kayıt Degiştirme UPDATE**

**Amaç:** Sertifikalar tablosundaki "DURUM" alanını, öğrenci puanlarına göre güncelleyerek başarı durumlarını belirlemek.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 02.05.2025

\* Description:

\* - Sertifikalar tablosundaki null olan DURUM alanını,

\* öğrenci ortalama puanına göre "Başarılı" veya "Başarısız" olarak günceller.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE PROCEDURE BasariDurumuSertifikalaraEkle

AS

BEGIN

UPDATE Sertifikalar

SET DURUM =

CASE

WHEN ORTALAMA\_PUAN >= 60 THEN 'Başarılı'

ELSE 'Başarısız'

END

WHERE DURUM IS NULL;

END;

EXEC BasariDurumuSertifikalaraEkle;

**Stored Procedure: Kayıt Silme DELETE**

**Amaç:** Belirtilen öğrenciye ait sertifikaları silmek.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 02.05.2025

\* Description:

\* - Öğrenciye ait sertifikaları silen stored procedure.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE PROCEDURE SertifikaSil

(@OgrenciID INT)

AS

BEGIN

DELETE FROM Sertifikalar

WHERE OGRENCI\_ID = @OgrenciID;

END;

EXEC SertifikaSil @OgrenciID = 1;

**Stored Procedure: 3 Tabloya Veri Ekleme TRANSACTION**

**Amaç:** Kurs, program ve kayıt ekleme işlemlerinin hatasız ve birlikte yapılmasını, hata olursa geri alınmasını sağlamak.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 02.05.2025

\* Description:

\* - Kurs, program ve kayıt ekleme işlemlerini bir transaction

\* içinde yapan stored procedure. Hata durumunda işlemi geri alır.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Her bir INSERT işlemi sonrasında **yeni satırın kimlik (identity) değerini almak** gerekiyor. Ancak burada önemli olan, bu işlemlerin **birbiriyle ilişkili** olması ve bir bütün olarak değerlendirilmesi. Bu yüzden SCOPE\_IDENTITY() her adımda kullanılıyor.

create PROCEDURE EklemeTransactionu

( @SEVIYE\_ID INT,

@DIL\_ID INT,

@SURESI VARCHAR(50),

@UCRETI DECIMAL(10,2),

@KONTENJAN INT,

@TUR\_ID INT,

@SINIF\_ID INT,

@GUNLER VARCHAR(100),

@BASLANGIC\_SAATI TIME,

@BITIS\_SAATI TIME,

@OGRETMEN\_ID INT,

@OGRENCI\_ID INT

)

AS

BEGIN

BEGIN TRANSACTION;

-- Kurs

INSERT INTO KURSLAR (SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID)

VALUES (@SEVIYE\_ID, @DIL\_ID, @SURESI, @UCRETI, @KONTENJAN, @TUR\_ID);

IF @@ERROR <> 0

BEGIN

ROLLBACK TRANSACTION;

RETURN;

END

DECLARE @KURS\_ID INT = SCOPE\_IDENTITY();

-- Program

INSERT INTO [dbo].[Kurslar.Programlar] (KURS\_ID, SINIF\_ID, GUNLER, BASLANGIC\_SAATI, BITIS\_SAATI, OGRETMEN\_ID)

VALUES (@KURS\_ID, @SINIF\_ID, @GUNLER, @BASLANGIC\_SAATI, @BITIS\_SAATI, @OGRETMEN\_ID);

IF @@ERROR <> 0

BEGIN

ROLLBACK TRANSACTION;

RETURN;

END

DECLARE @DERS\_PROGRAM\_ID INT = SCOPE\_IDENTITY();

-- Kayıt

INSERT INTO [dbo].[Kurslar.Kayitlar] (DERS\_PROGRAM\_ID, OGRENCI\_ID, KAYIT\_TARIHI)

VALUES (@DERS\_PROGRAM\_ID, @OGRENCI\_ID, GETDATE());

IF @@ERROR <> 0

BEGIN

ROLLBACK TRANSACTION;

RETURN;

END

COMMIT TRANSACTION;

END

EXEC EklemeTransactionu

@SEVIYE\_ID = 1,

@DIL\_ID = 2,

@SURESI = '3 ay',

@UCRETI = 9000,

@KONTENJAN = 30,

@TUR\_ID = 2,

@SINIF\_ID = 1,

@GUNLER = 'Pazartesi, Çarşamba',

@BASLANGIC\_SAATI = '09:00',

@BITIS\_SAATI = '11:00',

@OGRETMEN\_ID = 1,

@OGRENCI\_ID = 6;

**TRANSACTION: 3 Tabloya Veri Ekleme**

**Amaç:** Kurs, program ve kayıt işlemlerinin bir bütün olarak, hata durumunda geri alınabilecek şekilde yapılmasını sağlamak.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 02.05.2025

\* Description:

\* - Kurs, program ve kayıt işlemlerini aynı işlem bloğu içinde

\* gerçekleştiren ve hata durumunda geri alan transaction.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Her bir INSERT işlemi sonrasında **yeni satırın kimlik (identity) değerini almak** gerekiyor. Ancak burada önemli olan, bu işlemlerin **birbiriyle ilişkili** olması ve bir bütün olarak değerlendirilmesi. Bu yüzden SCOPE\_IDENTITY() her adımda kullanılıyor.

BEGIN TRANSACTION;

-- Kurs

INSERT INTO KURSLAR (SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID)

VALUES (1, 2, '3 ay', 9000, 30, 2);

DECLARE @KURS\_ID INT = SCOPE\_IDENTITY();

-- Program

INSERT INTO [dbo].[Kurslar.Programlar] (KURS\_ID, SINIF\_ID, GUNLER, BASLANGIC\_SAATI, BITIS\_SAATI, OGRETMEN\_ID)

VALUES (@KURS\_ID, 1, 'Pazartesi, Çarşamba, Cuma', '10:00', '12:00', 1);

DECLARE @DERS\_PROGRAM\_ID INT = SCOPE\_IDENTITY();

-- Kayıt

INSERT INTO [dbo].[Kurslar.Kayitlar] (DERS\_PROGRAM\_ID, OGRENCI\_ID, KAYIT\_TARIHI)

VALUES (@DERS\_PROGRAM\_ID, 5, GETDATE());

-- Hata kontrolü

IF @@ERROR <> 0

BEGIN

ROLLBACK TRANSACTION;

RETURN;

END

COMMIT TRANSACTION;

**1. Table-Valued FUNCTION**

**Amaç**: **Velisi olan öğrencileri** listelemek için tasarlanmış. Yani, **velisi bulunan** öğrencilerin bilgilerini döndürür.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 02.05.2025

\* Description:

\* - Velisi olan öğrencileri listeleyen tablo döndüren fonksiyon.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE FUNCTION dbo.fn\_VelisiOlanOgrenciler()

RETURNS TABLE

AS

RETURN

(

SELECT O.ID AS OgrenciID, O.BIREY\_ID, O.EGITIM\_DURUMU, O.KAYIT\_TARIHI, O.SUBE\_ID

FROM [dbo].[Bireyler.Ogrenciler] O

INNER JOIN [dbo].[Bireyler.Veliler] V ON O.ID = V.OGRENCI\_ID

);

select \* from dbo.fn\_VelisiOlanOgrenciler();

**2. Table-Valued FUNCTION**

**Amaç**: Bu fonksiyon, kurslara kayıtlı olduğu halde hiç ödeme yapmamış öğrencilerin listesini döndürmek için yazılmıştır. Odemeler tablosunda ödeme kaydı bulunmayan öğrencileri tespit ederek, onların temel bilgilerini içeren bir tablo döndürür.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 02.05.2025

\* Description:

\* - Ödeme yapmayan öğrencileri listeleyen tablo döndüren fonksiyon. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE FUNCTION dbo.fn\_OdemesizOgrenciler()

RETURNS TABLE

AS

RETURN

(

SELECT DISTINCT O.ID AS OgrenciID, O.BIREY\_ID, O.EGITIM\_DURUMU, O.KAYIT\_TARIHI, O.SUBE\_ID

FROM [dbo].[Bireyler.Ogrenciler] O

INNER JOIN [dbo].[Kurslar.Kayitlar] K ON O.ID = K.OGRENCI\_ID

LEFT JOIN [dbo].[Odemeler] OD ON O.ID = OD.OGRENCI\_ID

WHERE OD.ID IS NULL);

SELECT \* FROM dbo.fn\_OdemesizOgrenciler();

**1. Scalar-Valued FUNCTION**

**Amaç**: Bu fonksiyon, belirli bir öğrenci ID'sine ait Odemeler tablosundaki toplam ödeme tutarını hesaplamak için yazılmıştır. Öğrencinin ödediği tüm tutarları toplayarak toplam ödemesini döndürür; eğer ödeme yoksa sıfır döner. Böylece öğrencinin toplam ödeme bilgisi kolayca elde edilir.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 02.05.2025

\* Description:

\* Parametre olarak verilen OGRENCI\_ID değerine göre

\* Odemeler tablosundaki ilgili öğrencinin toplam ödeme tutarını hesaplar

\* ve döndürür. Eğer ödeme kaydı yoksa sıfır değerini döndürür.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE FUNCTION fn\_OgrenciToplamOdeme (@OGRENCI\_ID INT)

RETURNS DECIMAL(10,2)

AS

BEGIN

DECLARE @Toplam DECIMAL(10,2);

SELECT @Toplam = SUM(ODEME\_TUTARI)

FROM Odemeler

WHERE OGRENCI\_ID = @OGRENCI\_ID;

RETURN ISNULL(@Toplam, 0);

END

SELECT dbo.fn\_OgrenciToplamOdeme(2) AS ToplamOdeme;

SELECT \* FROM Odemeler

**2. Scalar-Valued FUNCTION**

**Amaç**: Bu fonksiyon, bir öğrencinin kaç farklı kurs kaydı olduğunu saymak ve geri döndürmek için yazılmıştır. Belirli bir öğrenci ID’sine göre Kurslar.Kayitlar tablosundaki kayıtları sayarak, o öğrencinin kaç kursa kayıt yaptığını kolayca öğrenmeyi sağlar.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 02.05.2025

\* Description:

\* - Parametre olarak verilen OGRENCI\_ID değerine göre

\* Kurslar.Kayitlar tablosundaki ilgili öğrencinin toplam kurs kayıt

\* sayısını döndürür.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE FUNCTION fn\_OgrenciKursKayitSayisi (@OGRENCI\_ID INT)

RETURNS INT

AS

BEGIN

DECLARE @KayitSayisi INT;

SELECT @KayitSayisi = COUNT(\*)

FROM [dbo].[Kurslar.Kayitlar]

WHERE OGRENCI\_ID = @OGRENCI\_ID;

RETURN @KayitSayisi;

END;

SELECT dbo.fn\_OgrenciKursKayitSayisi(1) AS KayitSayisi;

SELECT \* FROM [Kurslar.Kayitlar]

**After Insert TRIGGER**

**Amaç:** Kurslar tablosuna yeni bir kayıt eklendiğinde, aynı kaydın Kurslar\_Yedek1 tablosuna da eklenmesini sağlar. Böylece sisteme eklenen tüm kayıtlar yedek tabloda da tutulur ve kayıtların kim tarafından ve ne zaman eklendiği izlenebilir olur. Veri bütünlüğü ve izlenebilirlik için gereklidir.

CREATE TRIGGER trg\_Kurslar\_Insert

ON Kurslar

AFTER INSERT

AS

BEGIN

INSERT INTO Kurslar\_Yedek (ID, SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID, EKLENME\_TARIHI, EKLEYEN\_KULLANICI, SILINDI\_MI)

SELECT

ID, SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID, GETDATE(), SYSTEM\_USER, 0

FROM inserted;

END;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 01.05.2025

\* Description:

\* - Bu trigger, Kurslar tablosuna yeni bir kayıt eklendiğinde,

\* ilgili kaydı Kurslar\_Yedek1 tablosuna da ekler.

\* - Eklenen kaydın eklenme tarihi ve ekleyen kullanıcı bilgisi de

\* yedek tabloda yer alır.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE TRIGGER trg\_Kurslar\_Insert\_Yedek

ON Kurslar

AFTER INSERT

AS

BEGIN

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Kurslar\_Yedek1')

BEGIN

INSERT INTO Kurslar\_Yedek1 (ID, SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID, EKLENME\_TARIHI, EKLEYEN\_KULLANICI, SILINDI\_MI)

SELECT ID, SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID, GETDATE(), HOST\_NAME(), 0

FROM INSERTED;

END

ELSE

BEGIN

CREATE TABLE Kurslar\_Yedek1 (

ID INT,

SEVIYE\_ID INT,

DIL\_ID INT,

SURESI VARCHAR(50),

UCRETI DECIMAL(10,2),

KONTENJAN INT,

TUR\_ID INT,

EKLENME\_TARIHI DATETIME DEFAULT GETDATE(),

EKLEYEN\_KULLANICI NVARCHAR(50),

SILINDI\_MI BIT DEFAULT 0

);

INSERT INTO Kurslar\_Yedek1 (ID, SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID, EKLENME\_TARIHI, EKLEYEN\_KULLANICI, SILINDI\_MI)

SELECT ID, SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID, GETDATE(), HOST\_NAME(), 0

FROM INSERTED;

END

END;

**Instead Of Delete TRIGGER**

**Amaç:** Kurslar tablosundaki bir kaydın silinmesi yerine, Kurslar\_Yedek1 tablosunda ilgili kaydın SILINDI\_MI alanını 1 yaparak silinmiş gibi işaretlenmesini sağlar. Bu yöntem, ana tablodan veri kaybolmadan silinmiş gibi gösterir ve veri kaybını önleyerek geçmiş kayıtların izlenmesine olanak tanır.

ALTER TRIGGER trg\_Kurslar\_Delete

ON Kurslar

INSTEAD OF DELETE

AS

BEGIN

UPDATE Kurslar\_Yedek

SET SILINDI\_MI = 1

WHERE ID IN (SELECT ID FROM deleted);

END;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 01.05.2025

\* Description:

\* - Kurslar tablosunda bir silme işlemi yapılmadan önce,

\* ilgili kaydın Kurslar\_Yedek1 tablosunda "silindi" olarak işaretler.

\* - Yedek tablodaki kaydın ‘SILINDI\_MI’ değeri 1 olarak güncellenir.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE TRIGGER trg\_Kurslar\_Delete

ON Kurslar

INSTEAD OF DELETE

AS

BEGIN

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Kurslar\_Yedek1')

BEGIN

UPDATE Kurslar\_Yedek1

SET SILINDI\_MI = 1

WHERE ID IN (SELECT ID FROM DELETED);

END

ELSE

BEGIN

CREATE TABLE Kurslar\_Yedek1 (

ID INT,

SEVIYE\_ID INT,

DIL\_ID INT,

SURESI VARCHAR(50),

UCRETI DECIMAL(10,2),

KONTENJAN INT,

TUR\_ID INT,

EKLENME\_TARIHI DATETIME DEFAULT GETDATE(),

EKLEYEN\_KULLANICI NVARCHAR(50),

SILINDI\_MI BIT DEFAULT 0

);

INSERT INTO Kurslar\_Yedek1 (ID, SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID, EKLENME\_TARIHI, EKLEYEN\_KULLANICI, SILINDI\_MI)

SELECT ID, SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID, GETDATE(), HOST\_NAME(), 1

FROM DELETED;

END

END;

**After Update TRIGGER**

**Amaç:** Kurslar tablosundaki bir kaydın güncellenmesi durumunda, aynı güncellemenin Kurslar\_Yedek1 tablosuna da yansıtılmasını sağlar. Böylece yedek tablo güncel tutulur ve hem geçmiş hem de mevcut değişikliklerin yedeği saklanarak veri tutarlılığı sağlanır.

CREATE TRIGGER trg\_Kurslar\_Update

ON Kurslar

AFTER UPDATE

AS

BEGIN

UPDATE Kurslar\_Yedek

SET

Kurslar\_Yedek.SEVIYE\_ID = inserted.SEVIYE\_ID,

Kurslar\_Yedek.DIL\_ID = inserted.DIL\_ID,

Kurslar\_Yedek.SURESI = inserted.SURESI,

Kurslar\_Yedek.UCRETI = inserted.UCRETI,

Kurslar\_Yedek.KONTENJAN = inserted.KONTENJAN,

Kurslar\_Yedek.TUR\_ID = inserted.TUR\_ID

FROM Kurslar\_Yedek

INNER JOIN inserted

ON Kurslar\_Yedek.ID = inserted.ID;

END;

-- KONTROL

INSERT INTO Kurslar (SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID)

VALUES (1, 5, '2 ay', 15000, 15, 1);

SELECT \* FROM Kurslar\_Yedek;

SELECT \* FROM Kurslar

UPDATE Kurslar

SET SURESI = '3 ay', UCRETI = 16000 WHERE ID = 39;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Eldiana Belekova || Date: 01.05.2025

\* Description:

\* Kurslar tablosunda bir güncelleme yapılırken, ilgili kaydın

\* Kurslar\_Yedek1 tablosunda da güncellenmesini sağlar.

\* Güncellenen kaydın tüm bilgileri (eklenme tarihi, ekleyen kullanıcı vb.)

\* yedek tabloda da güncellenir.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CREATE TRIGGER trg\_Kurslar\_Update\_Yedek

ON Kurslar

AFTER UPDATE

AS

BEGIN

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Kurslar\_Yedek1')

BEGIN

UPDATE k

SET

k.SEVIYE\_ID = i.SEVIYE\_ID,

k.DIL\_ID = i.DIL\_ID,

k.SURESI = i.SURESI,

k.UCRETI = i.UCRETI,

k.KONTENJAN = i.KONTENJAN,

k.TUR\_ID = i.TUR\_ID,

k.EKLENME\_TARIHI = GETDATE(),

k.EKLEYEN\_KULLANICI = HOST\_NAME()

FROM Kurslar\_Yedek1 k

INNER JOIN INSERTED i ON k.ID = i.ID;

END

ELSE

BEGIN

CREATE TABLE Kurslar\_Yedek1 (

ID INT,

SEVIYE\_ID INT,

DIL\_ID INT,

SURESI VARCHAR(50),

UCRETI DECIMAL(10,2),

KONTENJAN INT,

TUR\_ID INT,

EKLENME\_TARIHI DATETIME DEFAULT GETDATE(),

EKLEYEN\_KULLANICI NVARCHAR(50),

SILINDI\_MI BIT DEFAULT 0

);

INSERT INTO Kurslar\_Yedek1 (ID, SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID, EKLENME\_TARIHI, EKLEYEN\_KULLANICI, SILINDI\_MI)

SELECT ID, SEVIYE\_ID, DIL\_ID, SURESI, UCRETI, KONTENJAN, TUR\_ID, GETDATE(), HOST\_NAME(), 0

FROM INSERTED;

END

END;

SELECT \* FROM Kurslar\_Yedek1 WHERE ID = 41;

**Sonuç:**

**LanguageCenter Dil Kursu** veritabanı, ilişkisel veritabanı kısıtlamaları ile veri bütünlüğü sağlanmıştır. **Trigger'lar** ve **Stored Procedure'lar** kullanılarak işlemler güvenli bir şekilde yapılmış, **Transaction** yapıları ile olası hatalar engellenmiştir. **After Insert Trigger** ile yedekleme, **Instead Of Trigger** ile silme işlemleri kontrollü bir şekilde gerçekleştirilmiş ve **After Update Trigger** ile güncellemeler hatasız yapılmıştır.

Genel olarak, veri güvenliği ve tutarlılığı sağlanmış, ancak **sistemdeki hatalar için sürekli iyileştirmelere ihtiyaç duyulabilir**.