Proje Tanımı:

Ders yönetimi projesi öğrencilerin,öğretmenlerin ve derslerin bilgilerini içeren bir veri tabanı projesidir.Bu proje'nin amaçları;

1-)Öğrencilerin kişisel bilgileri,aldığı dersleri,derslerden alınan vize ve final notları ve ait oldukları bölüm hakkında bilgi verir.

2-)Öğretmenlerin kişisel bilgileri,anlatmakla sorumlu olduğu dersler hakkında bilgi verir.

3-)Derslerin kredi,akts bilgileri,hangi döneme ait oldukları hakkında bilgi verir.

SQL BİLEŞENLERİ :

1-) SELECT kredi,ders_ad,bolum_id, MIN(kredi) AS dusuk_kredi FROM dersler WHERE kredi=(SELECT MIN(kredi) FROM dersler) GROUP BY kredi,ders_ad,bolum_id;

Bu sorguda amaç en düşük kredili dersi bulmak.Bunun için MIN operatörünü kullanıyoruz.WHERE kredi den sonra dersler tablosundan en düşük kredili dersi buluyoruz ve o dersin kredi,bolum_id'si dersin adı ve bölüm_id'sini görmek istediğimiz için SELECT kısmında bunu yazıyoruz.

2-) SELECT final,ogrenci_id,ders_id,donem, MAX(final) AS en_yuksek_not FROM ders_kayit WHERE final=(SELECT MAX(final) FROM ders_kayit) GROUP BY final,ogrenci_id,ders_id,donem;

Bu sorguda amaç en yüksek final notuna sahip dersi bulmak.Bunun için MAX operatörünü kullanıyoruz. Bu sorgu da GROUP BY kullanarak en yüksek final notuna sahip dersin final,öğrenci_id,ders_id ve hangi döneme(güz,bahar) ait olduğunu bulmak.

3-) SELECT ogrenci_ad,ogrenci_soyad,danisman_id,bolum_id FROM ogrenci WHERE ogrenci_email LIKE "%an%" ORDER BY bolum_id DESC,danisman_id ASC;

Bu sorgu da amaç email adresinde a ve n harflerini birlikte bulunduran öğrencilerin bilgilerini göstermek a ve n harflarini tabloda taramak için LIKE operatörünü kullanıyoruz.Ayrıca iki tane yüzde kullanmamızın sebebi harfleri başta ya da sonda aramak yerine herhangi bir yerde aramaktır.Burada ORDER BY kullanarak küçükten büyüğe ya da küçükten büyüğe sıralamak.

4-) SELECT ogretmen_ad,ogretmen_soyad FROM ogretmen WHERE ogretmen_ad NOT LIKE "M%" ORDER BY ogretmen_soyad ASC;

Bu sorgu da amaç öğretmenlerden adı M harfi ile başlayanları bulmak ,bu yüzden LIKE kullanıyoruz ve vsoyadlarını azalan sıraya göre(bir nevi alfabetik sıra)düzenlemek.

5-) SELECT danisman_id, COUNT*() AS ogrenci_sayisi FROM ogrenci GROUP BY danisman_id HAVING COUNT()* > 1;

Bu sorgu da amaç danışman_id'si 1'den büyük olan öğrencileri bulmak ,burada HAVING kullanmamızın sebebi bir sütundaki toplam ifadeyi bulmak o yüzden kullanıyoruz.

6-) SELECT ders_ad,ROUND(AVG(akts),0) AS yüksek_akts FROM dersler GROUP BY ders_id HAVING ROUND(AVG(akts),0) > 4;

Bu sorgu da amaç HAVING şartıyla 4 akts'den büyük dersleri yüksek akts'li dersler olarak filtrelemek.Bunun için HAVING ve GROUP BY kullandık.Ayrıca burada ROUND kullanmamızın sebebi şu.Kredi veya AKTS bunlar tam sayıdır.Bunları tamsayı olarak ekrana yansıtmak.

7-) CREATE VIEW ogrenci_not_gorunumu AS SELECT ogrenci_id,ROUND(AVG((vize + final)/2),2) AS ortalama_not FROM ders_kayit GROUP BY ogrenci_id HAVING ROUND(AVG((vize + final)/2),2)>65; SELECT * FROM ogrenci_not_gorunumu;

Bu sorgu da amaç öğrencilerin vize final notlarının ortalaması alıp 65'ten büyük olanları öğrenci not görünümü adlı görünüm tasarlayarak erkana yansıtmak.Filtreleme için GROUP BY kullanıyoruz ve HAVING kullanarak not ortalamasına 65 şartını getiriyoruz.

8-) CREATE VIEW ders_zorluk AS SELECT ders_ad,kredi,akts, (kredi*0.6* + *akts*0.4) AS zorluk_derecesi, CASE WHEN (kredi*0.6* + *akts*0.4)>=5 THEN 'Zor' WHEN (kredi*0.6* + *akts*0.4)>=3 THEN 'Orta' ELSE 'Kolay' END AS derece FROM dersler; SELECT*FROM ders_zorluk;

Bu sorgu da amaç derslerin kredi ve akts'sini belirli yüzdelerle çarpıp derslerin zorluk seviyesini belirlemek.Burada CASE WHEN kullanmamızın amacı filtrelemek. WHERE kullanamazdık çünkü WHERE komutu filtreleme yapamaz sadece şart koymak için kullanılır.

9-) CREATE VIEW ogrenci_bilgi AS SELECT ogrenci_ad,ogrenci_soyad,ogrenci_id,ogrenci_email FROM ogrenci WHERE bolum_id>1 AND ogrenci_ad LIKE "%e%" ; SELECT * FROM ogrenci_bilgi;

Bu sorgu da amaç adında e harfi geçen öğrencileri bulmak ve bunları öğrenci bilgi adlı bir görünüm elde etmek.Harf araması için LIKE kullanıyoruz.İki tane yüzde kullanmamızın sebebi adın herhangi bir yerinde geçmesi baş veya son farketmez ,harfi adın heryerinde arıyoruz.

10-) SELECT ogrenci_ad,ogrenci_soyad FROM ogrenci WHERE ogrenci_id IN (SELECT ogrenci_id FROM ders_kayit WHERE vize>(SELECT AVG(vize) FROM ders_kayit));

Bu sorgu da amaç alt sorgu kullanarak farklı tablolardaki verileri birleştirmek ve vizesi ortalama vizenin üstünde olan öğrencileri bulmak.Alt sorgu kullanılırken birleştirme için IN operatörünü kullanıyoruz.

11-) SELECT ogretmen_ad,ogretmen_soyad FROM ogretmen WHERE ogretmen_id IN (SELECT ogretmen_id FROM dersler GROUP BY ogretmen_id HAVING COUNT(ders_id)=3);

Bu sorgu da amaç alt sorgu kullanarak ders_id'si 3 olan ve bu şarta sahip dersleri veren öğretmenleri bulmak.Burada filtreleme için GROUP BY ve şartın sağlanması için HAVING operatörünü kullanıyoruz.Tabloları birleştirmek için IN operatörünü kullanıyoruz.

12-) SELECT ogrenci.ogrenci_ad,ogrenci.ogrenci_soyad,ders_kayit.vize,ders_kayit.final,ders_kayit.donem FROM ogrenci JOIN ders_kayit ON ogrenci.ogrenci_id=ders_kayit.ogrenci_id;

Bu sorgu da amaç öğrencilerin belirli ders dönemlerinde(güz,bahar) aldıkları notları bulmak.Burada iki tabloyu birleştirmek JOIN kullanıyoruz ,alt sorguya göre daha kullanışlı ve pratik.JOIN kullanmak için seçtiğimiz tablolarda ortak parametreler olmalı yani FOREIGN KEY ile bağladığımız parametreler üzerinde gitmek gerekiyor.Aksi takdir de JOIN kullanmak yanlış olur.JOIN kullanırken SELECT kısmına parametreleri ya da ekrana yansıtmak istediğimiz değerleri yazarken iki farklı tablo kullanacağımızdan tabloadı.parametre şeklinde kullanmamız gerekiyor ya da pratik olarak o tabloya uygun harf vermek ama bunun için As Alias yani takma ad kullanmalıyız verilerin karışmaması açısından.

13-) SELECT o.ogrenci_ad,o.ogrenci_soyad,d.ders_ad,d.kredi,d.akts,b.bolum_ad FROM ogrenci AS o JOIN dersler AS d ON o.bolum_id=d.bolum_id JOIN bolum AS b ON d.bolum_id=b.bolum_id

Bu sorgu da amaç öğrencilerin okudukları bölümlerde gördükleri derslerin adları,kredi ve akts bilgilerini görmek.Burada birden fazla tablo birleştirme olduğu için JOIN kullanmak pratik. Burada tablo sayısı fazla olduğu için tablolara uygun harf verip takma ad ile kullanımını pratik hale getiriyoruz ve tablolardaki ortak parametreleri eşitlik olarak kullanıyoruz.

14-) DELIMITER //

CREATE PROCEDURE ogretmen_ara(IN harf_arama VARCHAR(50), IN bolum_no INT) BEGIN SELECT ogretmen_ad, ogretmen_email, bolum_id FROM ogretmen WHERE ogretmen_email LIKE CONCAT(harf_arama, '%') AND bolum_id >= bolum_no; END //

DELIMITER ; CALL ogretmen_ara('a',1);

Bu sorgu da amaç prosedür oluşturup adı a harfi ile başlayan öğretmenleri görmek.Prosedür kullanmamızın sebebi ileriye dönük olarak da işimize yaramasıdır ve yapısı,kullanımı gereği dinamiktir.DELIMITER diyip prosedürümüzü başlatıyoruz ve harf araması yapacağımız için LIKE kullanıyoruz.Burada yüzde işaretinin harften sonra olması a harfi ile başlayanları bulmak içindir .Sonrasında CONCAT kullanıyoruz sebebi kullanıcıdan yani CALL operatörü içine yazdığımız harfi LIKE ile aramak ve buradaki bir diğer şartımız CALL operatörü içine yazdığımız sayısal değerin tablodaki bölüm_id sinde eşit veya küçük olma şartıdır.

15-) DELIMITER // CREATE PROCEDURE dersEkle(IN p_ders_id INT ,IN p_ders_ad VARCHAR(100),IN p_kredi INT ,IN p_akts INT ,IN p_ogretmen_id INT,IN p_bolum_id INT)

BEGIN INSERT INTO dersler(ders_id, ders_ad, kredi, akts, ogretmen_id, bolum_id)
VALUES(p_ders_id,p_ders_ad,p_kredi,p_akts,p_ogretmen_id,p_bolum_id); END//

DELIMITER ;

CALL dersEkle(309,'veri yapilari',3,5,1001,1);

SELECT*FROM dersler;

Bu sorgu da amaç prosedür kullanarak ders ekleme işlemleri yapmak .Bunun için prosedür içine parametreler yazıyoruz.Eğer parametreleri biz kendimiz ekleyeceksek prosedüre parametre yazmak zorundayız ama var olan tablo veya tablolardan veri çekeceksek prosedür boş kalmalıdır.Ekleme işlemi için INSERT INTO operatörünü kullanıyoruz sonrasında prosedürü sonlandırmak için END // diyip sonlandırma işlemini yapıyoruz ve son olarak CALL ile tabloya yazılacak olan değerleri yazıyoruz.Tabloda yaptığımız sorgunun sonucunu görmek için SELECT*FROM dersler;komutu yazıyoruz.

16-) DELIMITER // CREATE PROCEDURE ogrenciDersNot()

BEGIN SELECT o.ogrenci_ad,o.ogrenci_soyad,d.ders_ad,dk.vize,dk.final FROM ogrenci AS o JOIN ders_kayit AS dk ON o.ogrenci_id=dk.ogrenci_id JOIN dersler AS d ON d.ders_id=dk.ders_id;

END // DELIMITER ; CALL ogrenciDersNot();

Bu sorgu da amaç öğrencilerin derlerden aldığı notları görmek.Burada birden fazla tabloyu birleştirmek ve ileriye dönük olarakda işlemleri yapmak için prosedür kullanıyoruz.Burada ayrıca tablodan veri çekeceğimiz için prodeüre herhangi bir parametre vermiyoruz.Yine JOIN kullanmak için tablolardaki ortak parametreleri kullanıyoruz ve pratik ve veri karmaşasını önlemek için takma ad ile tabloları tanıyoruz.İşlem bitmesi için END // diyip prosedürü sonlandırıyoruz ve CALL ile prosedürümüzü çağırıyoruz.

17-) START TRANSACTION; DELIMITER // CREATE PROCEDURE dersEkle(IN p_ders_id INT, IN p_ders_ad VARCHAR(50), IN p_kredi INT, IN p_akts INT, IN p_ogretmen_id INT, IN p_bolum_id INT) BEGIN

IF p_kredi>4 THEN ROLLBACK; SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Kredi 4 uzeri olamaz';

ELSEIF p_akts<3 THEN ROLLBACK; SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Akts en az 3 olmalidir';

ELSE INSERT INTO dersler(ders_id, ders_ad, kredi, akts, ogretmen_id, bolum_id) VALUES
(p_ders_id,p_ders_ad,p_kredi,p_akts,p_ogretmen_id,p_bolum_id);

END IF;

END // DELIMITER ;

CALL dersEkle(300,'Ayrik Matematik',4,2,1002,1);

SELECT*FROM dersler ;

Bu sorgu da amaç veri bütünlüğünü korumak ve hataları direk fark edip işlemi sonlandırmak için TRANSACTION kullanıyoruz.Bu operatörü kullanma sebebimiz işlemlerin herhangi bir hata ile karşılaşması halinde yapılan işlemleri geri alıp tabloyu eski haline getirmek.Burada ders ekleme işlemi yapıyoruz.IF VE ELSE kullanıyoruz.TRANSACTION ‘ı başlatıyoruz ve sonra prosedürü de başlatıyoruz. Veri eklemeyi kendimiz yapacağımız için prosedürün içine parametreleri kednimiz yazıyoruz.Sonrasında kredi ve akts şartı koyuyoruz eğer herhangi birisi bu şartı sağlamazsa ROLLBACK ile işlemleri geri alıyoruz ve ekrana hata mesajı veriyoru.Ama eğer şartlar sağlanırsa INSERT INTO ile ders ekleme işlemini yapıyoruz ve sonrasında prosedürümüzü sonlandırıyoruz.

18-) DELIMITER //

CREATE PROCEDURE harfnotuhesapla() BEGIN START TRANSACTION; IF EXISTS(SELECT 1 FROM ders_kayit WHERE vize<0 OR vize>100 OR final<0 OR final>100) THEN ROLLBACK; SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Geçersiz vize veya final girilemez';

ELSE INSERT INTO dersharfnotu(ogrenci_id,vize,final,ortalama,harf_notu) SELECT ogrenci_id,vize,final, (vize*0.4* + *final*0.6) AS ortalama, CASE WHEN (vize*0.4* + *final*0.6) >=90 THEN 'AA' WHEN (vize*0.4* + *final*0.6) >=80 THEN 'BB' WHEN (vize*0.4* + *final*0.6) >=70 THEN 'CC' WHEN (vize*0.4* + *final*0.6) >=60 THEN 'DD' ELSE 'FF' END AS harf_notu FROM ders_kayit; COMMIT; END IF; END // DELIMITER ;

CALL harfnotuhesapla(); SELECT*FROM dersharfnotu;

Bu sorgu da amaç öğrencilerin derslerden aldıkları notları belirli bir yüzdeyle çarptıktan sonra belirli notlara harf notu vermek.Yine prosedürümüzü başlatıyoruz ve sonrasında TRANSACTION işlemini başlatıyoruz. Burada tablodan veri çekeceğimiz için prosedürü boş bırakıyoruz,zaten burada prosedür kullanmamızın amacı ileriye dönük olarak işlemlerimizi yapmak.Sonrasında vize ve final not şartını yazıyoruz eğer bu şart sağlanmazsa ROLLBACK diyip yapılan bütün işlemleri geri alıyoruz ve ekrana hata mesajı veriyoruz. Şartlar sağlanırsa INSERT INTO ile kelyeceğiğmiz parametreleri yazıyoruz ve ayrı bir tabloda değerlendirme yapacağımız için tabloda görmek istediğimiz parametreleri yazıyoruz.Sonrasında CASE WHEN ile filtreleme yapıp bu filtrelemeyi AS ile harf notu

adı sütunda yazıyoruz.En sonda da COMMIT diyip bu değişikliği tabloya kaydediyoruz. Ayrıca burada tabloya vize ve final notlarını eklemiyoruz ,onları kullanarak takma ad ile harf notu için kulanıyoruz.

19-) DELIMITER //

CREATE PROCEDURE ogrenciEkle(IN p_ogrenci_id INT, IN p_ogrenci_ad VARCHAR(50), IN p_ogrenci_soyad VARCHAR(50), IN p_ogrenci_mail VARCHAR(50), IN p_bolum_id INT, IN p_vize INT, IN p_final INT, IN p_danisman_id INT) BEGIN START TRANSACTION;

```
IF (p_vize + p_final)/2 < 60 THEN
    ROLLBACK;
    SELECT '60 ortalamanin altinda ogrenci alinamaz' AS mesaj;
ELSE
    INSERT INTO ogrenci(
        ogrenci_id, ogrenci_ad, ogrenci_soyad, ogrenci_email, bolum_id, danisman_id
    ) VALUES (
        p_ogrenci_id, p_ogrenci_ad, p_ogrenci_soyad, p_ogrenci_mail, p_bolum_id, p_danisman_id
    );
    COMMIT;
    SELECT 'Ogrenci basariyla eklendi' AS mesaj;
END IF;
```

END // DELIMITER ;

CALL ogrenciEkle(2010, 'Ahmet', 'Yılmaz', 'ahmet@example.com', 3, 70, 80, 1004);

Bu sorgu da amaç öğrenci eklemek. Prosedür ve TRANSACTION kullanarak bu işlemi yapıyoruz.Eğer hata olursa ROLLBACK ile yapılan tüm işlemleri geri alıyoruz ve tablomuz eskisi gibi kalıyor.Eğer not ortalaması =60 şartı sağlanıyorsa öğrenciyi ekliyoruz ve COMMIT ile tabloya kaydediyoruz.Burada yine kendimiz ekleme yapacağımız için prosedürün içini dolduruyoruz.

20-)veri ekleme INSERT INTO ogrenci(ogrenci_id,ogrenci_ad,ogrenci_soyad,ogrenci_email,bolum_id,danisman_id) VALUES (2000,'Ali','Kece','ali.kece@example.com',3,1002);

SELECT * FROM ogrenci;

Bu sorgu da amaç INSERT INTO operatörünü kullanarak tabloya veri eklemek.

21-) DELETE FROM ders_kayit WHERE ders_id=301; SELECT*FROM ders_kayit;

DELETE FROM dersler WHERE ders_id=301; SELECT*FROM dersler

Bu sorgu da amaç DELETE operatörü ile silme işlemi yapmak. DELETE FROM yazıp sonrasında verinin silinecek olduğu tablo adını yazıyoruz ve sonrasında WHERE komutunu yazıyoruz.Eğer bu komutu yazmazsak yazılan tabloyu komple siler. Burada iki işlem yapmamızın sebebi sileceğimiz verinin FOREIGN KEY ile bağlantılı olması bu yüzden önce id_sini siliyoruz yani kaydını sonrasında dersi siliyoruz.

22-) UPDATE ders_kayit SET final=

CASE

WHEN vize <=70 THEN final + 15

WHEN vize >70 AND vize <=80 THEN final +5

ELSE final

END ;

SELECT*FROM ders_kayit;

Bu sorgu da amaç UPDATE operatörü ile güncelleme yapmak.UPDATE yazıp sonra güncelleme yapılacak tablo adını yazıyoruz. Sonra SET diyip hangi veri de değişim yapılacaksa onu yazıyoruz.Burada CASE WHEN filtreleme yapıyoruz.Vize notu ile final notunu güncelliyoruz.

23-) CREATE TABLE ogrenci_log (

  log_id INT AUTO_INCREMENT PRIMARY KEY,

  ogrenci_id INT,

  islem_tipi ENUM('INSERT','UPDATE','DELETE') NOT NULL,

  islem_tarihi TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

  mesaj VARCHAR(255)

);

```
DELIMITER //

CREATE TRIGGER log_ogrenci_ekle
AFTER INSERT ON ogrenci
FOR EACH ROW
BEGIN
   INSERT INTO ogrenci_log (ogrenci_id, islem_tipi, mesaj)
   VALUES (NEW.ogrenci_id, 'INSERT', CONCAT('', NEW.ogrenci_ad, '', NEW.ogrenci_soyad,'' eklendi.'));
END;
//
```

```
DELIMITER ;
```

Bu sorgu da amaç öğrenci eklemek.Bunun için log kaydı oluşturuyoruz.Log kaydı yapmamızın sebebi öğrenci tablosunda yapılan işlemleri takip etmek.Ayrıca burada trigger oluşturuyoruz .Bunu da oluşturmamızın sebebi öğrenci tablosuna öğrenci eklendiği zaman bunu otomatik olarak öğrenci_log tablosuna kaydetmek.

24-)

```
DELIMITER //

CREATE TRIGGER log_ogrenci_guncelle
AFTER UPDATE ON ogrenci
```

```
FOR EACH ROW
BEGIN
   INSERT INTO ogrenci_log (ogrenci_id, islem_tipi, mesaj)
   VALUES (NEW.ogrenci_id, 'UPDATE', CONCAT('Öğrenci ', NEW.ogrenci_ad, ' ', NEW.ogrenci_soyad, ' güncellendi.'));
END;
//

DELIMITER ;
UPDATE ogrenci
SET ogrenci_ad = 'Cengiz ', ogrenci_soyad = 'Çolak'
WHERE ogrenci_id = 2001;


SELECT * FROM ogrenci_log;
SELECT * FROM ogrenci;
```

Bu sorgu da amaç UPDATE operatörü ile var olan tablo da öğrenci güncelleme işlem, yapmak.Bu işlemi otomatik öğrenci_log tablosuna kaydetmek için trigger kullanıyoruz.Daha sonra güncelleme işleminin başarılı olduğunu anlamak için öğrenci_log tablosunu ekrana yansıtıyoruz.Daha sonra öğrenci tablosunda bu işlemi görmek için öğrenci tablosunuekrana yansıtıyoruz.

25-) DELIMITER //

```
CREATE TRIGGER log_ogrenci_sil
AFTER DELETE ON ogrenci
FOR EACH ROW
BEGIN
   INSERT INTO ogrenci_log (ogrenci_id, islem_tipi, mesaj)
   VALUES (OLD.ogrenci_id, 'DELETE', CONCAT('Öğrenci ', OLD.ogrenci_ad, ' ', OLD.ogrenci_soyad, ' silindi.'));
END;
//
```

DELIMITER ;

DELETE FROM ders_kayit

WHERE ogrenci_id = 2005;

DELETE FROM ogrenci

WHERE ogrenci_id =2005 ;

SELECT * FROM ogrenci_log;

SELECT * FROM ogrenci;

Bu sorgu da amaç öğrenci tablosundan belirlenen bir öğrenciyi silmek.Bunun için DELETE opratörünü kullanıyoruz.Bu işlemi öğrenci_log tablosuna otomatik kaydetmek için tetikleyici kullanıyoruz.Silme işlemi yapmak için kullanacığımız parametre FOREIGN KEY ile bağlı olduğu için önce ders_kayit tablosundan öğrencinin id'sini siliyoruz.FOREIGN KEY ile olan bağlantısını kopardıktan sonra öğrenci tablosundan belirlenen öğrenciyi siliyoruz.Ekrana bu mesajı vermek için önce öğrenci_log tablosunu ekrana yansıtıyoruz.Daha sonra öğrenci tablosunu yansıtıyoruz.

TABLOLARIN NORMALİZASYON KURALLARINA UYGUNLUĞU VE TABLO DETAYLARI

1-)CREATE TABLE bolum (

bolum_id INT PRIMARY KEY, -- burada her bölüm için benzersiz bir_id oluşturuyoruz

bolum_ad VARCHAR(100) NOT NULL -- her bölümün adı olmak zorundadır

Bu tabloda tüm alanlar atomik yapıdadır ve birincil anahtar tek kolon (`bolum_id`) olduğu için 1NF ve 2NF kurallarına uygundur. Anahtar olmayan `bolum_ad` kolonunun başka bir anahtar olmayan kolona bağımlılığı bulunmadığından geçişli bağımlılık yoktur. Bu nedenle tablo 3NF'e uygundur.

2-)CREATE TABLE ogretmen (

ogretmen_id INT PRIMARY KEY,

ogretmen_ad VARCHAR(50) NOT NULL,

ogretmen_soyad VARCHAR(50) NOT NULL,

ogretmen_email VARCHAR(100),

bolum_id INT,

FOREIGN KEY (bolum_id) REFERENCES bolum(bolum_id) -- öğretmenin hangi bölüme ait olduğuna ulaşmak için);

Tablodaki tüm alanlar atomiktir ve `ogretmen_id` tek kolonlu bir birincil anahtardır, dolayısıyla 1NF ve 2NF koşulları sağlanmaktadır. Anahtar olmayan kolonlar (`ogretmen_ad`, `ogretmen_soyad`, `ogretmen_email`, `bolum_id`) arasında geçişli bağımlılık bulunmamaktadır. `bolum_id` yabancı anahtar olup 3NF için sorun oluşturmaz. Bu nedenle tablo 3NF'e uygundur.

3-)
```
CREATE TABLE ogrenci (
   ogrenci_id INT PRIMARY KEY,
   ogrenci_ad VARCHAR(50) NOT NULL,
   ogrenci_soyad VARCHAR(50) NOT NULL,
   ogrenci_email VARCHAR(100) UNIQUE ,
   bolum_id INT,
   danisman_id INT,
   FOREIGN KEY (bolum_id) REFERENCES bolum(bolum_id), -- öğrencinin okuduğu bölüme ulaşmak için
   FOREIGN KEY (danisman_id) REFERENCES ogretmen(ogretmen_id) -- öğrencinin o bölümdeki danışmanı );
```

Bu tablo atomik verilerden oluşmaktadır ve tek kolonlu birincil anahtar (`ogrenci_id`) nedeniyle kısmi bağımlılık yoktur. Anahtar olmayan kolonlar (`ogrenci_ad`, `ogrenci_soyad`, `ogrenci_email`, `bolum_id`, `danisman_id`) arasında geçişli bağımlılık bulunmamaktadır. Bu nedenle tablo 3NF'e uygundur.

4-)
```
CREATE TABLE dersler (
   ders_id INT PRIMARY KEY,
   ders_ad VARCHAR(100) NOT NULL UNIQUE ,
   kredi INT NOT NULL CHECK(kredi>0) ,
   akts INT NOT NULL CHECK(akts>0) ,
   ogretmen_id INT,
   bolum_id INT,
   FOREIGN KEY (ogretmen_id) REFERENCES ogretmen(ogretmen_id), -- dersi veren öğretmene ulaşmak
   FOREIGN KEY (bolum_id) REFERENCES bolum(bolum_id) -- dersin hangi bölüme ait olduğunu bulmak);
```

Tablodaki tüm alanlar atomiktir ve birincil anahtar tek kolon olduğu için 1NF ve 2NF'e uygundur. Anahtar olmayan kolonlar (`ders_ad`, `kredi`, `akts`, `ogretmen_id`, `bolum_id`) kendi aralarında bağımlılık oluşturmaz. Yabancı anahtarlar 3NF'i etkilemediğinden tablo 3NF'e uygundur.

5-)CREATE TABLE ders_kayit (

ogrenci_id INT,

ders_id INT,

yil INT NOT NULL DEFAULT 2025,

donem ENUM('Güz', 'Bahar'), -- burada enum kullanarak sadece güz ve baharda bu işlemin yapılacağını belirtir

vize INT,

final INT, -- bu notları tutmamızın sebebi ders yönetim sistemibdeki öğrencilerin akademik ilerleyişini takip etmek

PRIMARY KEY (ogrenci_id, ders_id), -- aynı öğrenci aynı dersi aynı yıl ve dönemde sadece bir kez alabilir

FOREIGN KEY (ogrenci_id) REFERENCES ogrenci(ogrenci_id), -- ders kaydı bir öğrenciye ait olmalı

FOREIGN KEY (ders_id) REFERENCES dersler(ders_id) -- her dersin kaydı bir derse ait olacak);

Bu tablo atomik alanlardan oluşmaktadır ve birleşik birincil anahtar (`ogrenci_id`, `ders_id`) 1NF ve 2NF koşullarını sağlamaktadır; çünkü diğer kolonlar bu iki anahtar alanına birlikte bağımlıdır. Anahtar olmayan kolonlar (`yil`, `donem`, `vize`, `final`) arasında geçişli bağımlılık bulunmamaktadır. Bu nedenle tablo 3NF'e uygundur.