T.C.

SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ

FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

2023 – 2024 Güz Dönemi

Ders : Veri Tabanı Yönetim Sistemleri

Hazırlayan : Esra İclal Akyol (B221210032)

Ders Sorumlusu : Prof. Dr. Celal ÇEKEN

E-Posta : esra.akyol@ogr.sakarya.edu.tr

GitHub : https://github.com/esraiclalakyl/VTYSPROJE

SENARYO

Bir şehir bisiklet paylaşım sisteminin veri tabanı oluşturulmuştur. Kullanıcılar, bisikletler, istasyonlar ve bunlara ait bilgiler kaydedilir. 16 tablodan oluşmaktadır, 5 fonksiyon 4 trigger vardır. Uygulama üzerinden kullanıcılar, istasyonlar, bisikletler tablolarına bilgi ekleyip,silinebilir ya da güncellenip, listelenebilir.

İŞ KURALLARI

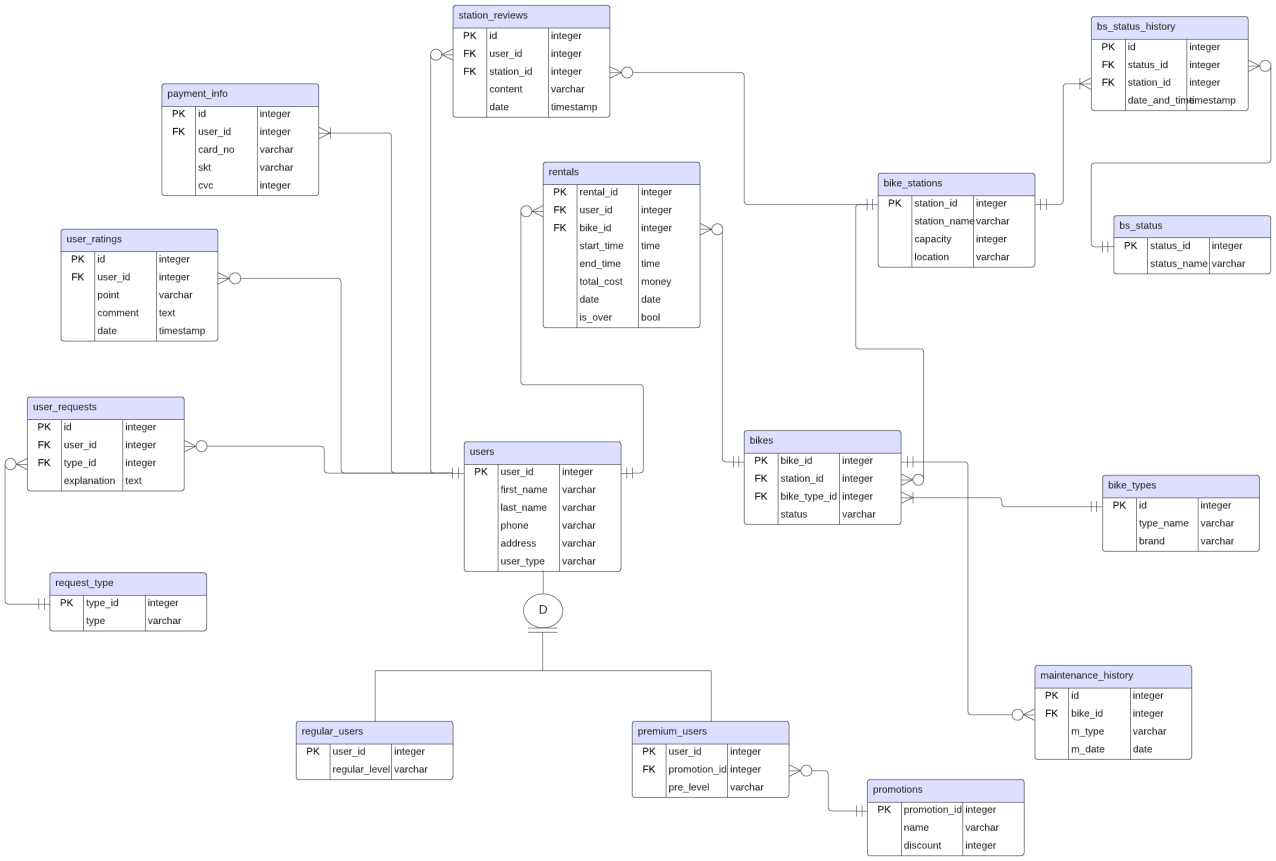
1. Kullanıcıların UserId, ad, soyad, telefon, adres, kullanıcı tipi alanları bulunmaktadır.
2. Genel ve premium olmak üzere iki tür kullanıcı bulunmaktadır.
3. Bir kullanıcı genel kullanıcı veya premium kullanıcı türlerinden birini seçmelidir, ikisi aynı anda olamaz.
4. Genel kullanıcılar tablosunda Id ve Genel kullanıcı level alanları vardır.
5. Premium kullanıcılar tablosunda Id ve Premium kullanıcı level alanları vardır.
6. Bisiklet istasyonlarının StationId, istasyon adı, konum, kapasite alanları bulunmaktadır.
7. Bisikletlerin BikeId, durumu (kullanımda, bakımda, boşta) alanları bulunmaktadır.
8. Bisiklet Türleri tablosunda Id, tür adı ve marka alanları bulunmaktadır.
9. Araç bakım geçmişi tablosunda Id, bakım türü ve bakım tarihi alanları bulunmaktadır.
10. Bisiklet istasyon durumları tablosunda Id ve istasyon durum adı, tarih ve zaman alanları bulunmaktadır.
11. Ödeme bilgileri tablosunda Id, kart no, son kullanma tarihi ve cvc alanları bulunmaktadır.
12. Kullanıcı puanları tablosunda Id, puan, yorum ve tarih alanları bulunmaktadır.
13. Kullanıcı İstekleri tablosunda Id, açıklama alanları bulunmaktadır.
14. İstek Tipleri tablosunda TypeId, tür adı bulunmaktadır.
15. Kampanyalar tablosunda PromotionId, kampanya adı ve indirim yüzdesi alanları bulunmaktadır.
16. Bir kampanyayı hiç premium kullanıcı kullanmayabilir birden fazla premium kullanıcı da kullanabilir, bir premium kullanıcı bir kampanya kullanmak zorundadır ve birden fazla kampanya kullanamaz.
17. Bir bisikletin bakım geçmişi olmayabilir birden fazla da olabilir, bir araç bakım geçmişi bir bisiklete aittir ve sadece bir bisiklete ait olabilir.
18. Bir bisiklet türünde en az bir tane bisiklet bulunmalıdır birden fazla bisiklet de bulunabilir. Bir bisiklet, bisiklet türlerinden birine sahip olmalıdır, sadece bir tane bisiklet türüne sahip olabilir.
19. Bir bisiklet istasyonunun en az bir durumu en çok da bir durumu olabilir. Bir durum hiç istasyona ait olmayabilir birden fazla istasyona da ait olabilir.
20. Bir İstasyonda hiç bisiklet olmayabilir, birden fazla bisiklet de olabilir. Bir bisiklet mutlaka bir istasyonda bulunmalıdır, birden fazla istasyonda bulunamaz.
21. Bir kullanıcının mutlaka bir ödeme bilgisi bulunmalıdır, birden fazla da bulunabilir. Bir ödeme bilgisi bir kullanıcıya ait olmalıdır ve sadece bir kullanıcıya ait olabilir.
22. Bir kullanıcının hiç puanlaması olmayabilir, birden çok da olabilir. Bir puanlama mutlaka bir kullanıcıya ait olmalıdır ve sadece bir kullanıcıya ait olabilir.
23. Bir kullanıcının hiç isteği olmayabilir, birden fazla da olabilir. Bir istek mutlaka bir kullanıcıya ait olmalıdır ve sadece bir kullanıcıya ait olabilir.
24. Bir istek tipinde hiç istek olmayabilir, birden fazla da istek olabilir. Bir istek mutlaka bir istek tipine ait olmalıdır ve sadece bir istek tipine ait olabilir.
25. Bir kullanıcı hiç istasyon incelemesi yapmamış olabilir, birden fazla kez yapmış da olabilir. Bir istasyon incelemesi bir kullanıcıya ait olmalıdır, sadece bir kullanıcıya ait olabilir.

26. Bir istasyon hiç incelemeye sahip olmayabilir, birden fazlaya da sahip olabilir. Bir istasyon incelemesi bir istasyona ait olmalıdır, sadece bir istasyona ait olabilir.
27. Bir kullanıcı hiç bisikleti kiralamamış da olabilir, birden fazla bisikleti de kiralayabilir. Bir bisiklet hiç kiralanmamış da olabilir, birden fazla kullanıcı tarafından kiralanmış da olabilir.

İLİŞKİSEL ŞEMA

- users(**user_id**:integer, first_name: varchar, last_name : varchar, phone : varchar, address : varchar, user_type:varchar)
- regular_users (**user_id**: integer, regularlevel: varchar)
- premium users (**user id** :integer, promotion_id : integer, pre_level : varchar)
- promotions (**promotions_id**:integer, name : varchar, discount : integer)
- user_requests (**id**: integer, userid : integer, type_id : integer, explanation : text)
- request_type(**type_id** : integer, type :varchar)
- user_ratings (**id**: integer, user_id : integer, point: varchar, comment : text, date : timestamp)
- payment_info (**id** : integer, user_id : integer, card_no : varchar, skt : varchar, cvc : integer)
- station_reviews (**id** : integer, user id : integer, station_id : integer, content : varchar, date : timestamp)
- rentals (**rental_id** : integer, user_id : integer, bike_id : integer, start time : time, end_time : time, total_cost : Money, date : date, is_over : bool)
- bikes (**bike_id** : integer, station_id : integer, bike_type_id : integer, status : varchar)
- bike_stations (**station_id** : integer,station_name : varchar, capacity : integer, location : varchar)
- bs_status_history (**id** : integer, status_id : integer, station_id : integer, date_and_time : timestamp)
- bs_status (**status_id** : integer, status_name :varchar)
- bike_types (**id** : integer, type_name : varchar, status : varchar)
- maintenance_history **(id** : integer, bike_id : integer, m_type : varchar, m_date : date)

VARLIK BAĞINTI DİYAGRAMI



SQL İFADELERİ

SQL ifadeleri github dosyasında da yüklüdür.

Sırasıyla Kullanıcı Puanları, İstasyon durumları, Bisiklet İstasyonları, bisiklet tipleri, kullanıcılar, ödeme bilgileri, İstasyon incelemeleri, promosyonlar, premium kullanıcılar, istek tipi, kullanıcı istekleri, bisikletler, kiralamalar, regular kullanıcılar, araç bakım geçmişi, istasyon durum geçmişi tabloları aşağıdadır.

```
CREATE TABLE "user_ratings" (
  "id" integer,
  "user_id" integer,
  "point" varchar(1),
  "comment" text,
  "date" timestamp,
  PRIMARY KEY ("id"),
  CONSTRAINT "FK_user_ratings.user_id"
    FOREIGN KEY ("user_id")
      REFERENCES "users"("user_id")
);

CREATE TABLE "bs_status" (
  "status_id" integer,
  "status_name" varchar(40),
   PRIMARY KEY ("status_id")
);

CREATE TABLE "bike_stations" (
  "station_id" integer,
  "station_name" varchar(70),
  "capacity" integer,
  "location" varchar,
   PRIMARY KEY ("station_id")
);

CREATE TABLE "station_reviews" (
  "id" integer,
```

```
CREATE TABLE "bike_types" (
  "id" integer,
  "type_name" varchar(30),
  "brand" varchar(20),
  PRIMARY KEY ("id")
);

CREATE TABLE "users" (
  "user_id" integer,
  "first_name" varchar(40),
  "last_name" varchar(40),
  "phone" varchar(10),
  "address" varchar,
  "user_type" varchar(20) NOT NULL,
  PRIMARY KEY ("user_id")
);

CREATE TABLE "payment_info" (
  "id" integer,
  "user_id" integer,
  "card_no" varchar(16),
  "skt" varchar,
  "cvc" varchar(3),
  PRIMARY KEY ("id"),
  CONSTRAINT "FK_payment_info.user_id"
    FOREIGN KEY ("user_id")
      REFERENCES "users"("user_id")
```

```
CREATE TABLE "station_reviews" (
  "id" integer,
  "user_id" integer,
  "station_id" integer,
  "content" varchar(50),
  "date" timestamp,
  PRIMARY KEY ("id"),
  CONSTRAINT "FK_station_reviews.station_id"
    FOREIGN KEY ("station_id")
      REFERENCES "bike_stations"("station_id"),
  CONSTRAINT "FK_station_reviews.user_id"
    FOREIGN KEY ("user_id")
      REFERENCES "users"("user_id")
);

CREATE TABLE "promotions" (
  "promotion_id" integer,
  "name" varchar(50),
  "discount" integer,
  PRIMARY KEY ("promotion_id")
);

CREATE TABLE "premium_users" (
  "user_id" integer,
  "promotion_id" integer,
  "pre_level" varchar(10),
  PRIMARY KEY ("user_id"),
  "promotion_id" integer,
  "pre_level" varchar(10),
  PRIMARY KEY ("user_id"),
  CONSTRAINT "FK_premium_users.promotion_id"
    FOREIGN KEY ("promotion_id")
      REFERENCES "promotions"("promotion_id"),
    CONSTRAINT "FK_premium_users.user_id"
    FOREIGN KEY ("user_id")
      REFERENCES "users"("user_id")
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE "request_type" (
  "type_id" integer,
  "type" varchar(50),
  PRIMARY KEY ("type_id")
);

CREATE TABLE "user_requests" (
  "id" integer,
  "user_id" integer,
  "type_id" integer,
  "explanation" text,
  PRIMARY KEY ("id"),
  CONSTRAINT "FK_user_requests.type_id"
    FOREIGN KEY ("type_id")
      REFERENCES "request_type"("type_id"),
```

```
CREATE TABLE "bikes" (
  "bike_id" integer,
  "station_id" integer,
  "bike_type_id" integer,
  "status" varchar(20),
  PRIMARY KEY ("bike_id"),
  CONSTRAINT "FK_bikes.bike_type_id"
    FOREIGN KEY ("bike_type_id")
      REFERENCES "bike_types"("id"),
  CONSTRAINT "FK_bikes.station_id"
    FOREIGN KEY ("station_id")
      REFERENCES "bike_stations"("station_id")
);

CREATE TABLE "rentals" (
  "rental_id" integer,
  "user_id" integer,
  "bike_id" integer,
  "start_time" time,
  "end_time" time,
  "total_cost" money,
  "date" date,
  "is_over" boolean,
  PRIMARY KEY ("rental_id"),
  CONSTRAINT "FK_rentals.bike_id"
    FOREIGN KEY ("bike_id")
      REFERENCES "bikes"("bike_id"),
```

Kalıtım burada tanımlı.

```
CREATE TABLE "regular_users" (
  "user_id" integer,
  "regular_level" varchar(10),
  PRIMARY KEY ("user_id"),
CONSTRAINT "FK_regular_users.user_id"
    FOREIGN KEY ("user_id")
      REFERENCES "users"("user_id")
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE "maintenance_history" (
  "id" integer,
  "bike_id" integer,
  "m_type" varchar(50),
  "m_date" date,
  PRIMARY KEY ("id"),
  CONSTRAINT "FK_maintenance_history.bike_id"
    FOREIGN KEY ("bike_id")
      REFERENCES "bikes"("bike_id")
);

CREATE TABLE "bs_status_history" (
  "id" integer,
  "status_id" integer,
  "station_id" integer,
  "date_and_time" timestamp,
  PRIMARY KEY ("id"),
```

ÖRNEK INSERT İŞLEMLERİ

```
-- BSStatus
INSERT INTO "bs_status" ("status_id", "status_name") VALUES
  (1, 'Bakımda, kullanılamaz'),
  (2, 'Bisiklet var, kullanılabilir'),
  (3, 'Bisiklet yok, kullanılabilir');


-- BikeStations
INSERT INTO "bike_stations" ("station_id", "station_name", "capacity", "location") VALUES
  (1, 'Kadıköy İstasyonu', 20, '123 Main St'),
  (2, 'Çayırova İstasyonu', 15, '456 Oak St'),
  (3, 'Gebze İstasyonu', 10, '456 Oak St');


-- StationReviews
INSERT INTO "station_reviews" ("id", "user_id", "station_id", "content", "date") VALUES
  (1, 5, 1, 'Memnunum!', '2023-01-22 09:45:00'),
  (2, 2, 3, 'Beğenmedim', '2022-06-05 11:20:00'),
  (3, 5, 2, 'Bir öncekinden daha kullanışlıydı', '2023-11-03 09:25:00');


-- Promotions
INSERT INTO "promotions" ("promotion_id", "name", "discount") VALUES
  (1, 'İlkbahar indirimi', 10),
  (2, 'Premium kullanıcılara özel', 30),
  (3, 'Yılbaşı indirimi', 20);
```

```
-- PremiumUsers
INSERT INTO "premium_users" ("user_id", "promotion_id", "pre_level") VALUES
  (2, 2, 'Gold'),
  (3, 2, 'Silver'),
  (5, 3, 'Silver');


-- RequestType
INSERT INTO "request_type" ("type_id", "type") VALUES
  (1, 'Uygulama Geliştirme'),
  (2, 'Bisikletlerle ilgili sorunlar'),
  (3, 'İstasyonlarla ilgili sorunlar'),
  (4, 'Üyelikle ilgili sorunlar');



-- UserRequests
INSERT INTO "user_requests" ("id", "user_id", "type_id", "explanation") VALUES
  (1, 1, 2, 'Bisikletin bakıma ihtiyacı var!'),
  (2, 3, 4, 'Üyelik ödememi yaptım ancak premium gözükmüyorum.'),
  (3, 5, 3, 'Kadıköy istasyonunda bisiklet yok.');


-- Bikes
```

```sql
-- BikeTypes
INSERT INTO "bike_types" ("id", "type_name", "brand") VALUES
  (1, 'Elektrikli Bisiklet', 'Bosch'),
  (2, 'Şehir Bisikleti', 'Trek'),
  (3, 'Elektrikli Bisiklet', 'Haibike'),
  (4, 'Şehir Bisikleti', 'Specialized');


-- Users
INSERT INTO "users" ("user_id", "first_name", "last_name", "phone", "address", "user_type") VALUES
  (1, 'Selim', 'Kaya', '1234567890', 'İnönü, Ataşehir/İstanbul', 'Regular'),
  (2, 'Büşra', 'Durgun', '9876543210', 'Suadiye, Kadıköy/İstanbul', 'Premium'),
  (3, 'Sevgi', 'Gedik', '9876543210', 'Göztepe, Kadıköy/İstanbul', 'Premium'),
  (4, 'Alican', 'Korkmaz', '9876543210', 'Osmanağa, Kadıköy/İstanbul', 'Regular'),
  (5, 'Mehtap', 'Gürsoy', '9876543210', 'Osmanağa, Kadıköy/İstanbul', 'Regular'),
  (6, 'Kumru', 'Göğebakan', '9876543210', 'Rasimpaşa, Kadıköy/İstanbul', 'Premium');


-- PaymentInfo
INSERT INTO "payment_info" ("id", "user_id", "card_no", "skt", "cvc") VALUES
  (1, 4, '1234567890123456', '03/25', '604'),
  (2, 3, '9876543210987653', '06/26', '632'),
  (3, 2, '9876543210987254', '06/26', '414'),
  (4, 1, '9876543210983654', '08/27', '223'),
  (5, 6, '9876543210917654', '07/24', '596'),
```

FONKSİYONLAR

5 fonksiyon vardır.

Resimde de yazdığı gibi id girilen kullanıcının tüm kiralama geçmişini, bisikletler tablosuyla inner joinleyerek getirir.

```sql
--1.Fonksiyon-- ID' si girilen kullanıcının kiralama geçmişini getirir
CREATE OR REPLACE FUNCTION get_user_rentals(p_user_id INTEGER)
RETURNS TABLE (
    rental_id INTEGER,
    user_id INTEGER,
    bike_id INTEGER,
    start_time TIME,
    end_time TIME,
    total_cost MONEY,
    date DATE,
    is_over BOOLEAN,
    bike_type_id INTEGER,
    status VARCHAR(20)
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        rentals.rental_id,
        rentals.user_id AS rental_user_id,
        rentals.bike_id,
        rentals.start_time,
        rentals.end_time,
        rentals.total_cost,
        rentals.date,
        rentals.is_over,
        bikes.bike_type_id,
        bikes.status
    FROM
```

UserRatings tablosundaki point değişkenine göre uygulamanın ortalama puanını integer olarak hesaplar ve döndürür.

```sql
--2.Fonksiyon-- Point değişkenine göre uygulamanın ortalama puanını getirir
CREATE OR REPLACE FUNCTION get_application_rating()
RETURNS DECIMAL AS $$
DECLARE
    total_points DECIMAL;
    total_users INTEGER;
    average_rating DECIMAL;
BEGIN
    -- Calculate total points and total users
    SELECT COALESCE(SUM(point::numeric), 0), COUNT(*) INTO total_points, total_users FROM user_ratings;

    -- Calculate average rating
    IF total_users > 0 THEN
        average_rating := total_points / total_users;
    ELSE
        average_rating := 0; -- Avoid division by zero
    END IF;

    -- Return the calculated average rating
    RETURN average_rating;
END;
$$ LANGUAGE plpgsql;
--------------------------------
```

Bisikletler tablosunun tamamını bakım geçmişi tablosuyla inner join yapar

```sql
--3.Fonksiyon-- Tüm bisikletlerin bakım geçmişini bisikletler tablosuyla inner join yaparak getirir
CREATE OR REPLACE FUNCTION get_bike_maintenance_history()
RETURNS TABLE (
    bike_id INTEGER,
    bike_type_name VARCHAR(30),
    maintenance_id INTEGER,
    maintenance_type VARCHAR(50),
    maintenance_date DATE
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        b.bike_id,
        bt.type_name AS bike_type_name,
        mh.id AS maintenance_id,
        mh.m_type AS maintenance_type,
        mh.m_date AS maintenance_date
    FROM
        bikes b
    JOIN
        maintenance_history mh ON b.bike_id = mh.bike_id
    JOIN
        bike_types bt ON b.bike_type_id = bt.id;
END;
$$ LANGUAGE plpgsql;
--------------------------------
```

Id parametresine göre istasyonun bilgilerini ve statülerini getirir

```sql
--4.Fonksiyon-- ID'si girilen istasyonun bilgilerini istasyon durumları tablosuyla inner join yaparak getirir
CREATE OR REPLACE FUNCTION get_station_info(station_id_param INTEGER)
RETURNS TABLE (
    station_name VARCHAR(70),
    location VARCHAR,
    status_name VARCHAR(40),
    content VARCHAR(50),
    review_date TIMESTAMP
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        bs.station_name,
        bs.location,
        bss.status_name,
        sr.content,
        sr.date AS review_date
    FROM
        bike_stations bs
    LEFT JOIN
        station_reviews sr ON bs.station_id = sr.station_id
    LEFT JOIN
        bs_status_history bssh ON bs.station_id = bssh.station_id
    LEFT JOIN
        bs_status bss ON bssh.status_id = bss.status_id
    WHERE
```

Id parametresine göre kullanıcının bütün kart bilgilerini getirir, birden fazla kartı varsa da getirir.

```sql
--5.fonksiyon--Id'si girilen kullanıcının kart bilgilerini getirir
CREATE OR REPLACE FUNCTION get_user_payment_info(user_id_param INTEGER)
RETURNS TABLE (
    card_id INTEGER,
    card_no VARCHAR(16),
    skt VARCHAR,
    cvc VARCHAR(3)
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        pi.id AS card_id,
        pi.card_no,
        pi.skt,
        pi.cvc
    FROM
        payment_info pi
    WHERE
        pi.user_id = user_id_param;
END;
$$ LANGUAGE plpgsql;
----------------------------------------
```

TRİGGERLAR

4 trigger vardır.

```sql
--2.Trigger--Kullanıcı tablosuna bir kullanıcı eklenirken regular ya da premium olmasına göre kalıtım alınan tablolara gönderilir
--Yani eğer premium kullanıcı oluşturulduysa bilgileri premium_users tablosuna gönderilir
-- Trigger fonksiyonu oluşturulur
CREATE OR REPLACE FUNCTION assign_user_to_user_type_table()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.user_type = 'Regular' THEN
        --Regular ise regular_users tablosuna bilgileri ekler
        INSERT INTO regular_users (user_id, regular_level) VALUES (NEW.user_id, DEFAULT);
        --Premium ise premium_users tablosuna bilgileri ekler
    ELSIF NEW.user_type = 'Premium' THEN
        INSERT INTO premium_users (user_id, promotion_id, pre_level) VALUES (NEW.user_id, DEFAULT, DEFAULT);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER assign_user_to_user_type_trigger
AFTER INSERT ON users
FOR EACH ROW
EXECUTE FUNCTION assign_user_to_user_type_table();
```

Üstteki trigger kullanıcı tablosuna bilgi eklenirken not null olarak oluşturulmuş usertype değişkenine göre çalışır, eğer regular olarak girildiyse id ile beraber regular users tablosuna, premium olarak girildiyse id ile beraber premium users tablosuna kaydedilir, kalıtımı hem foreign key ile hem bu sekilde birleştirmiş oldum.

Alttaki trigger rentals tablosuna göre çalışır, eğer bir update işleminde totalcast adlı değişken boş bırakıldıysa kendisi başlangıç ve bitiş zamanına göre ödenmesi gereken ücreti bulur ve yazdırır.

```sql
--1.trigger-- Eğer kiralama işleminin fiyatı girilmediyse dakikasına göre fiyatı kendi hesaplar

CREATE OR REPLACE FUNCTION calculate_rental_cost()
RETURNS TRIGGER AS $$
DECLARE
    rental_duration interval;
BEGIN
    -- Eğer total_cost değeri boş ise
    IF NEW.total_cost IS NULL THEN
        -- Kiralama süresini hesapla
        rental_duration := NEW.end_time - NEW.start_time;

        -- Kiralama süresini dakika cinsinden al ve ücreti hesapla
        NEW.total_cost := EXTRACT(MINUTE FROM rental_duration);

        -- Dakikaları 1 TL ile çarp
        NEW.total_cost := NEW.total_cost * 1;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER calculate_rental_cost_trigger
BEFORE INSERT ON rentals
FOR EACH ROW
EXECUTE FUNCTION calculate_rental_cost();
```

3.trigger Kullanıcı tablosuna göre çalışır, bir update ya da güncelleme işleminde ad ve soyad değişkenlerinşn baş harfleri küçük ise büyük harfe çevirir ve yazdırır.

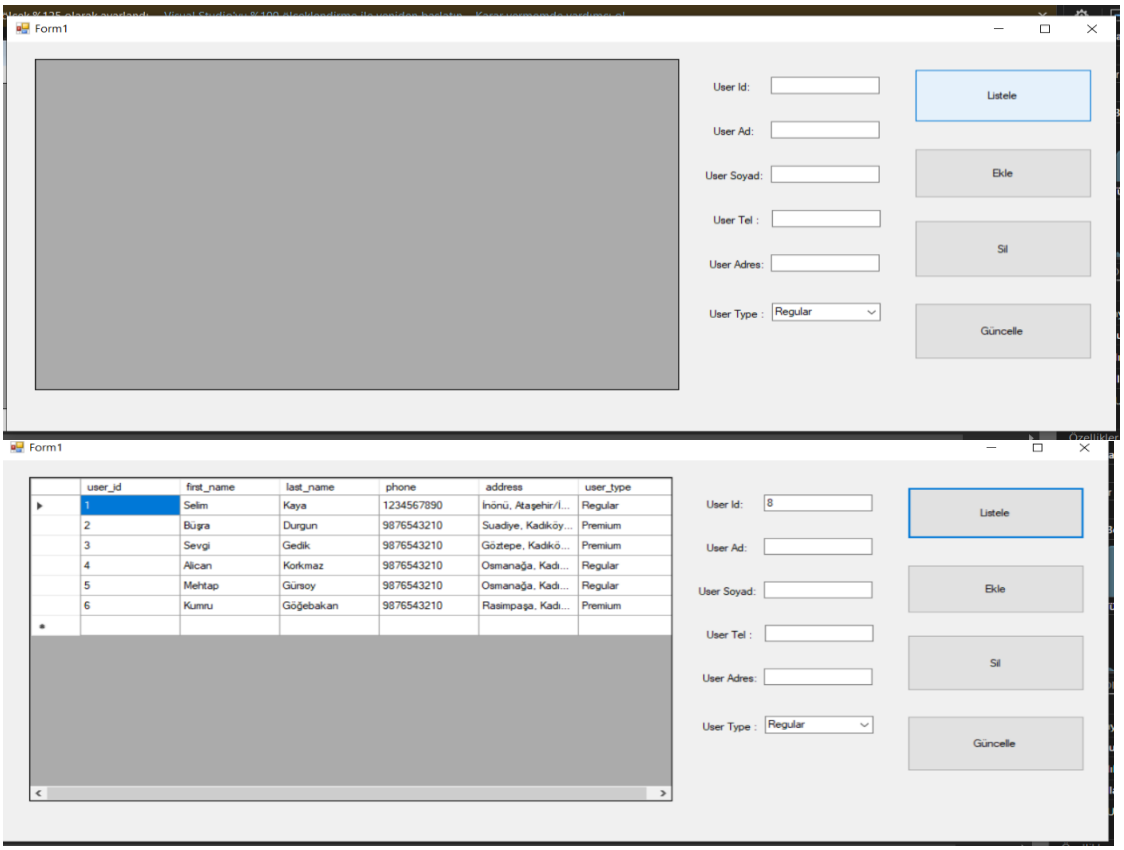
```sql
--3.Trigger-- Kullanıcı tablosuna ad soyad girildiğinde ilk harfleri küçük ise büyük yapar
CREATE OR REPLACE FUNCTION capitalize_name()
RETURNS TRIGGER AS $$
BEGIN
    -- bir kullanıcı eklenilir ise
    IF TG_OP = 'INSERT' THEN
        -- Ad ve soyadı büyük harfe çevirir
        NEW.first_name := INITCAP(NEW.first_name);
        NEW.last_name := INITCAP(NEW.last_name);
    END IF;

    -- Güncellemede de aynısını yapar
    IF TG_OP = 'UPDATE' THEN
        IF NEW.first_name IS DISTINCT FROM OLD.first_name THEN
            NEW.first_name := INITCAP(NEW.first_name);
        END IF;

        IF NEW.last_name IS DISTINCT FROM OLD.last_name THEN
            NEW.last_name := INITCAP(NEW.last_name);
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER capitalize_name_trigger
```

4.trigger Kiralama işlemlerindeki isover değerine göre çalışır, eğer değiştirildiyse buna göre kiralamada kullanılan bisikletin durumunu kullanılabilir veya kullanılamaz yapar.

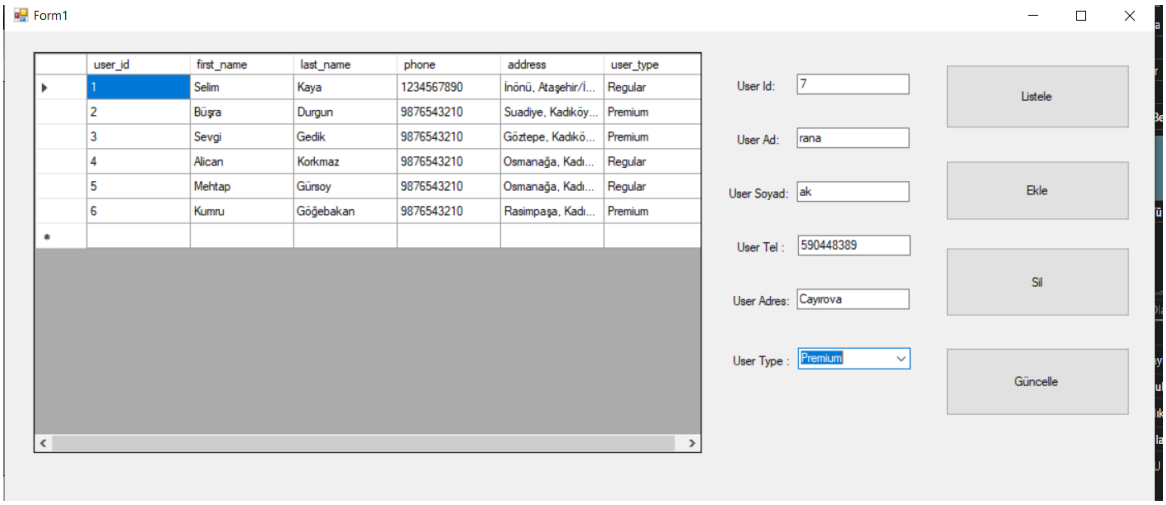
```sql
--4.trigger-- Kiralama işlemlerindeki is_over değerinin değişmesine göre bisikletin durumunu değiştirir

CREATE OR REPLACE FUNCTION update_bike_status_on_rental_change()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.is_over IS DISTINCT FROM NEW.is_over THEN
        -- Eğer is_over değiştiyse
        IF NEW.is_over THEN
            -- Eğer is_over true olduysa bisiklet kullanılabilir hale gelir
            UPDATE bikes
            SET status = 'Kullanılabilir'
            WHERE bike_id = NEW.bike_id;
        ELSE
            -- Eğer is_over false olduysa bisiklet kullanılamaz hale gelir
            UPDATE bikes
            SET status = 'Kullanılamaz'
            WHERE bike_id = NEW.bike_id;
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_bike_status_trigger
AFTER UPDATE ON rentals
FOR EACH ROW
EXECUTE FUNCTION update_bike_status_on_rental_change();
```

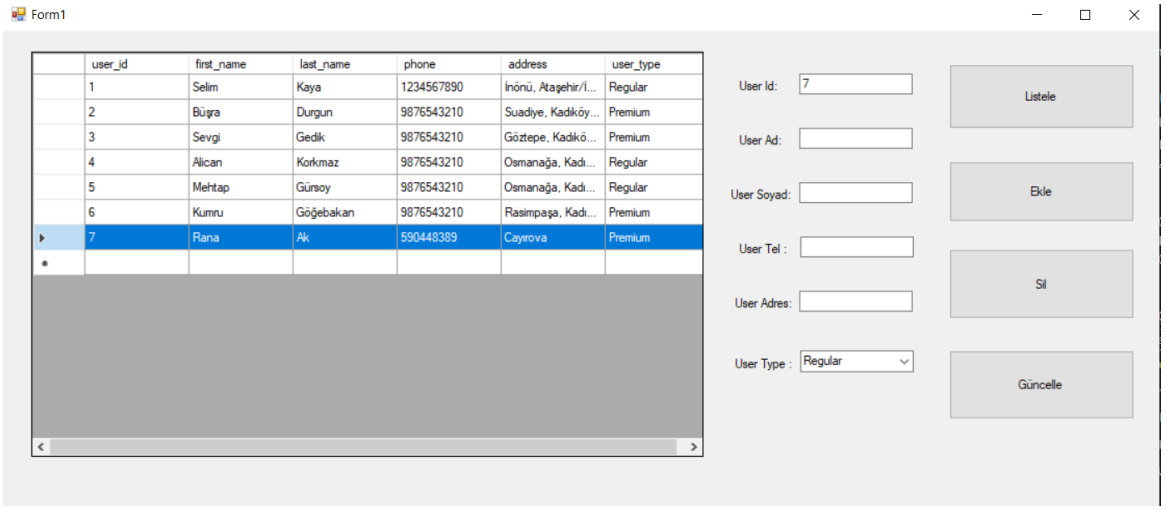
UYGULAMA ÜZERİNDE ARAMA İŞLEMİ

Uygulama Windows form üzerinden gerçeklenir.

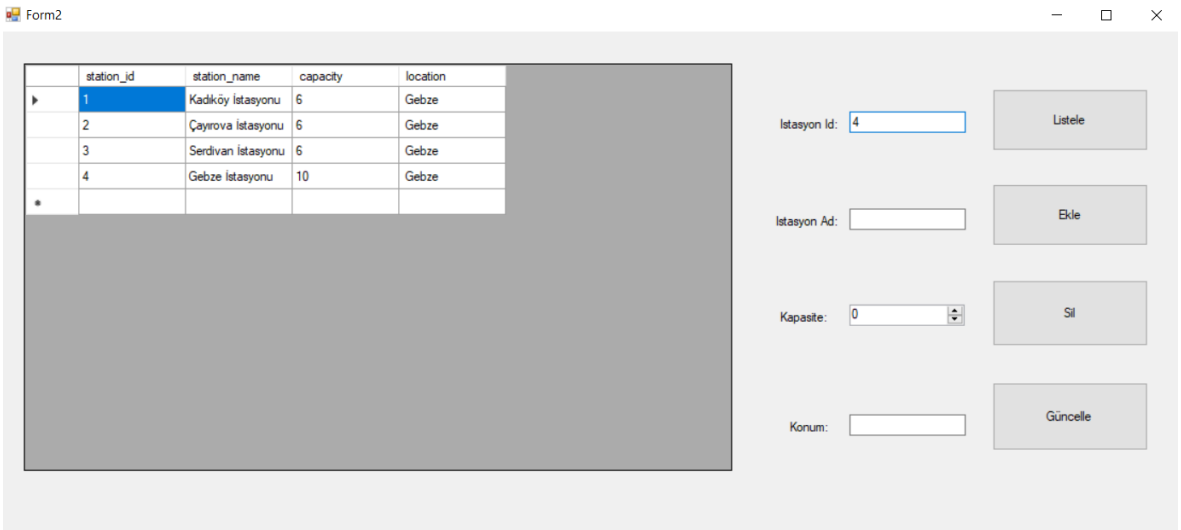


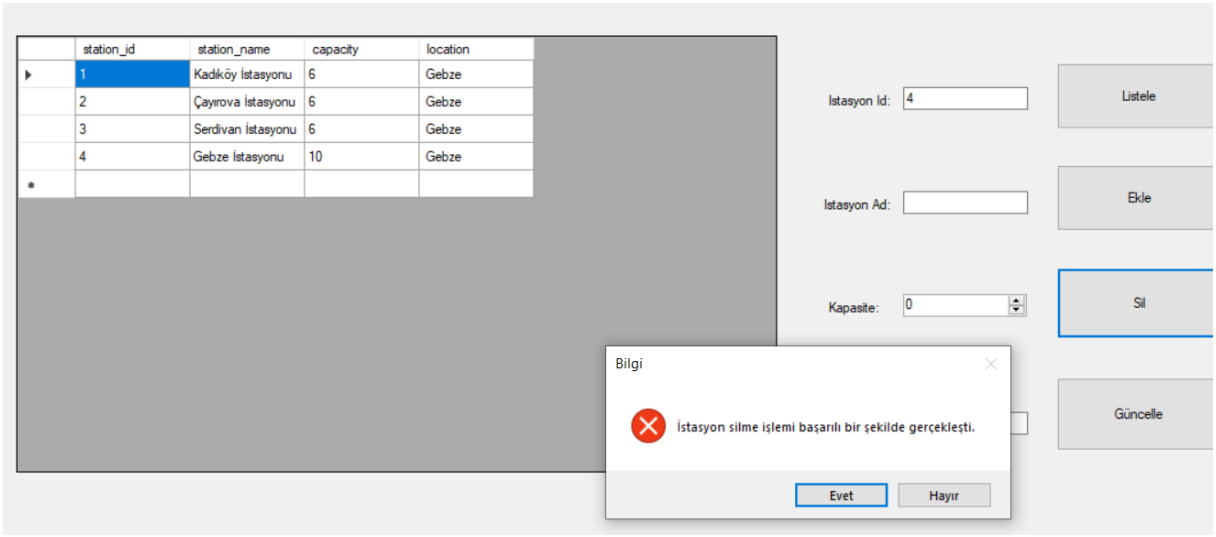
UYGULAMA ÜZERİNDE EKLEME İŞLEMİ

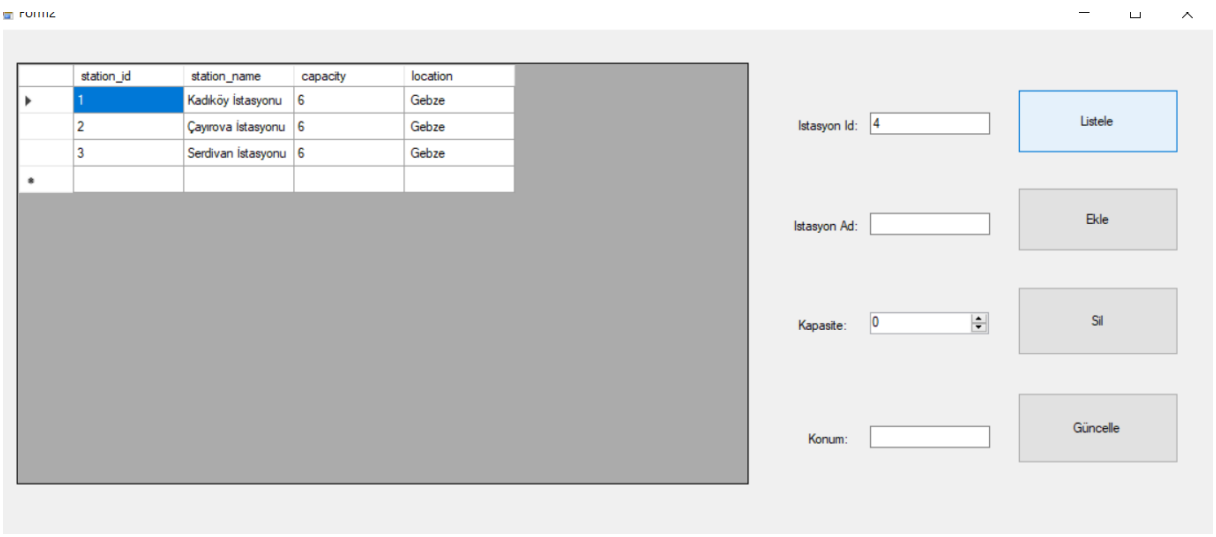




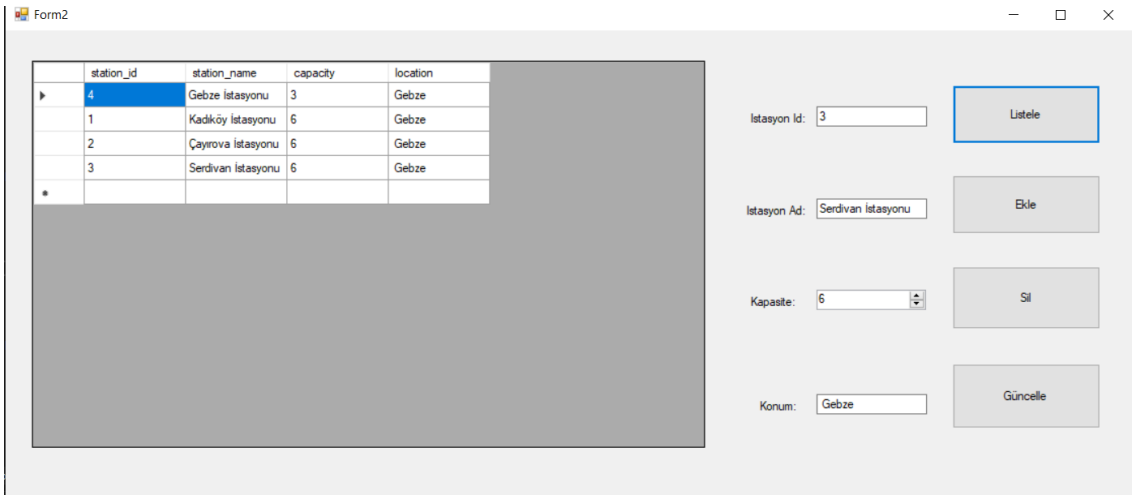
UYGULAMA ÜZERİNDE SİLME İŞLEMİ







UYGULAMA ÜZERİNDE GÜNCELLEME İŞLEMİ



Burada kapasiteyi 6 dan 10a güncelledik.