SAKARYA
ÜNİVERSİTESİ

Öğrenci Adı: Melisa

Öğrenci Soyadı: Yüksek

Öğrenci Numarası: G231210005

Öğrenci E-Postası:
melisa.yuksek@ogr.sakarya.edu.tr

Öğrenci Adı: Sümeyye

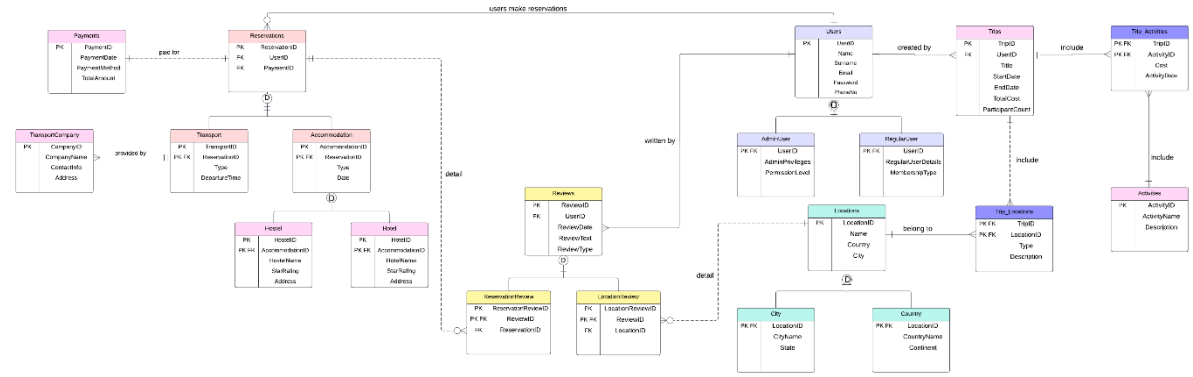
Öğrenci Soyadı: Nurlu

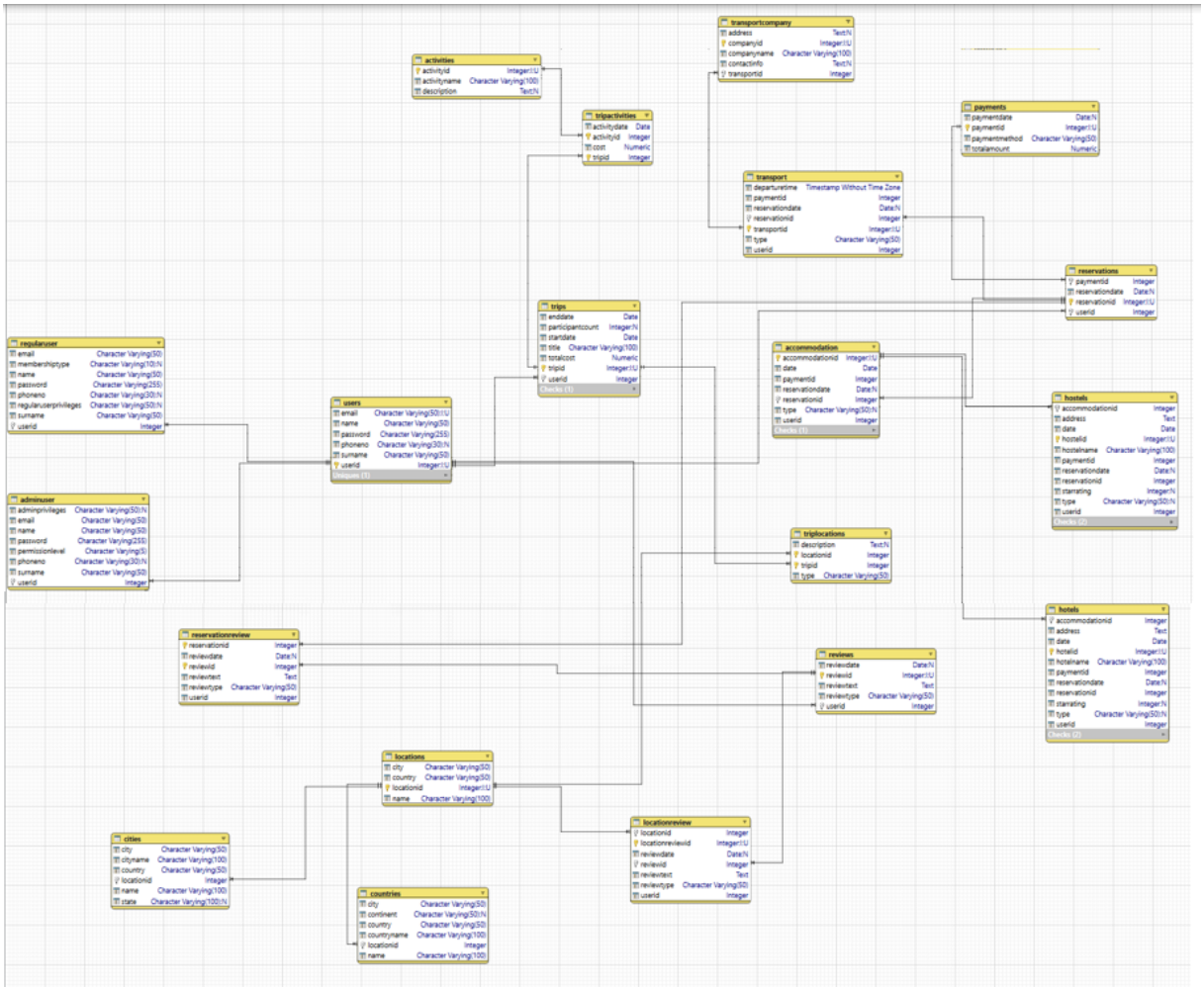
Öğrenci Numarası: G231210053

Öğrenci E-Postası:
sumeyye.nurlu@ogr.sakarya.edu.tr

Github Reposu: https://github.com/MelisaYuksek/Database-Travel

Varlık Bağıntı Modeli:





Seyahat Rehberi ve Rezervasyon Sistemi

Günümüzde seyahat eden bireylerin, planlama ve rezervasyon süreçlerini daha kolay bir şekilde yönetebilecekleri bir sisteme ihtiyaç duyulmaktadır.

Bu bağlamda geliştirilecek olan **Seyahat Rehberi ve Rezervasyon Sistemi**, kullanıcıların seyahat planlarını daha organize ve etkin bir şekilde yapmalarına olanak tanıyacaktır. Sistem, kullanıcıların ihtiyaç duyduğu bilgileri ve işlemleri merkezi bir platform üzerinden gerçekleştirebilmelerini sağlayacaktır.

İŞ KURALLARI

- Sistemdeki Userlar **AdminUser** (yönetici) ya da bir **RegularUser** (normal kullanıcı) veya her ikisi olabilir. (Overlapping)
- Her kullanıcı sisteme kayıtlı olmalıdır.
- Bir kullanıcının, birden fazla rezervasyonu ve seyahati olabilir.
- Bir rezervasyon ve seyahat mutlaka bir kullanıcıya bağlı olmalıdır. Rezervasyonlar tek başına anlamlı değildir.
- Her rezervasyon için bir ödeme yapılmalıdır, her ödeme yalnızca bir rezervasyona aittir.
- Rezervasyonlar ya bir ulaşım türünü (Transport) ya da bir konaklama birimini (Accommodation) içermelidir. (Disjoint)
- Bir rezervasyon, yalnızca bir ulaşım türü ya da bir konaklama ile ilişkilendirilebilir.
- Ödeme türü (örneğin nakit, kredi kartı) veya ödeme durumu (tamamlandı, beklemede) gibi detaylar tutulabilir.
- Her hotel ve hostel bir konaklama (Accommodation) birimidir, ancak her konaklama (accommodation) birimi otel ya da hostel olmayabilir. (Disjoint)
- Taşıma hizmetleri, bir taşımacılık şirketi (TransportCompany) tarafından sağlanır.
- Bir taşımacılık şirketi birden fazla taşıma hizmeti sağlayabilir.
- Her taşıma işlemi, yalnızca bir taşımacılık şirketine bağlı olmalıdır.
- Her lokasyon ya bir şehir (City) ya da bir ülke (Country) olmalıdır. (Disjoint ve Total Completeness)Her lokasyon ya bir şehir (City) ya da bir ülke (Country) olabilir. (Disjoint) Diyagramda total completeness (tamlık) zorunluluğu belirtilmemiş. Kuralı şu şekilde değiştirin
- Lokasyonlar seyahatlerin hedef noktalarıdır.
- Şehir ve ülke bilgileri ayrık olarak tutulur; bir lokasyon aynı anda hem şehir hem de ülke olamaz.
- Her seyahat mutlaka bir kullanıcıya bağlı olmalıdır.

- Seyahatler birden fazla lokasyon (Locations) içerebilir.
- Seyahatler bir veya daha fazla aktivite (Activities) içerebilir.
- Her seyahatin lokasyon ve aktivite bilgilerinin eksiksiz tanımlanması zorunlu değildir.
- Her aktivite bağımsız bir şekilde sistemde tanımlanabilir.
- Bir aktivite, birden fazla seyahate bağlanabilir.
- Her inceleme ya bir rezervasyona (ReservationReview) ya da bir lokasyona (LocationReview) ait olmalıdır. (Disjoint ve Total Completeness)
- İncelemeler, rezervasyonlar veya lokasyonlar hakkında kullanıcı geri bildirimlerini tutar.
- Kullanıcılar birden fazla inceleme bırakabilir.
- **Users ↔ Reservations**: Bir kullanıcı birden fazla rezervasyon yapabilir, ancak her rezervasyon yalnızca bir kullanıcıya ait olabilir.
- **Users ↔ Trips**: Bir kullanıcı birden fazla seyahate katılabilir; bir seyahate birden fazla kullanıcı katılabilir.
- **Reservations ↔ Payments**: Bir rezervasyonun yalnızca bir ödemesi olabilir, ancak her ödeme yalnızca bir rezervasyonla ilişkilidir.
- **Trips ↔ Activities**: Bir seyahat birden fazla aktivite içerebilir; bir aktivite birden fazla seyahatle ilişkilendirilebilir.

İlişkisel Şema:

1. **Reservations**: PK ReservationID, FK UserID, FK PaymentID
2. **Transport**: PK TransportID, PK FK ReservationID, Type, DepartureTime
3. **Accommodation**: PK AccommodationID, PK FK ReservationID,Type,Date
4. **Users**: PK UserID, Name, Surname, Email, Password, PhoneNo
5. **AdminUser**: PK FK UserID, AdminPrivileges, PermissionLevel
6. **RegularUser**: PK FK UserID, RegularUserDetails, MembershipType
7. **Locations**: PK LocationID, Name, Country, City
8. **City**: PK FK LocationID, CityName, State
9. **Country**: PK FK LocationID, CountryName, Continent
10. **Reviews**: PK ReviewID, FK UserID, ReviewDate, ReviewText, ReviewType
11. **ReservationReview**: PK FK ReviewID, PK ReservationReviewID FK ReservationID
12. **LocationReview**: PK FK ReviewID, FK LocationID, PK LocationReview
13. **Trips**: PK TripID, FK UserID, Title, StartDate, EndDate, TotalCost, ParticipantCount
14. **Payments**: PK PaymentID, PaymentDate, PaymentMethod, TotalAmount

15. **Activities**: PK ActivityID, ActivityName, Description
16. **TransportCompany**: PK CompanyID, CompanyName, ContactInfo, Address
17. **Trip_Locations**: PK FK TripID, PK FK LocationID, Type, Description
18. **Trip_Activities**: PK FK TripID, PK FK ActivityID, Cost, ActivityDate
19. **Hotel**: PK HotelID, PK FK AccommodationID, HotelName, StarRating, Address
20. **Hostel**: PK HostelID , PK FK AccommodationID, HostelName, StarRating, Address

1. Kullanıcılar (Users):

Her kullanıcı benzersiz bir UserID ile tanımlanır.

Kullanıcılar ad (Name), soyad (Surname), e-posta (Email), şifre (Password) ve telefon numarası (PhoneNo) bilgilerine sahiptir.

Kullanıcılar, yalnızca bir türde olabilir: AdminUser veya RegularUser (overlapping kalıtım).

Bir kullanıcı birden fazla yorum (Reviews) yazabilir ve birden fazla gezi (Trips) oluşturabilir.

Bir kullanıcı çok fazla rezervasyon (Reservations) yapabilir.

2. Yönetici Kullanıcı (AdminUser):

Yönetici kullanıcılar, UserID aracılığıyla tanımlanır ve Users tablosuyla ilişkilidir.

Her yönetici özel yetkilere (AdminPrivileges) ve izin seviyelerine (PermissionLevel) sahiptir.

3. Normal Kullanıcı (RegularUser):

Normal kullanıcılar, UserID aracılığıyla tanımlanır ve Users tablosuyla ilişkilidir.

Her normal kullanıcı üyelik türü (MembershipType) ve ek detaylara (RegularUserDetails) sahiptir.

4. Rezervasyonlar (Reservations):

Her rezervasyon benzersiz bir ReservationID ile tanımlanır.

Her rezervasyon, bir kullanıcıya (UserID) aittir ve sadece bir ödeme kaydıyla (PaymentID) ilişkilidir.

Rezervasyonlar, ulaşım (Transport) veya konaklama (Accommodation) içerebilir (disjoint kuralı).

BURAYI KNTROL ET!! LOCATAIONREVIEW VAR

5. Ulaşım (Transport):

Ulaşım kayıtları benzersiz bir TransportID ve ReservationID kombinasyonu ile tanımlanır.

Her ulaşım bir rezervasyonla ilişkilidir (ReservationID).

Ulaşım kayıtları tür (Type) ve kalkış zamanı (DepartureTime) bilgilerini içerir.

Ulaşım hizmetleri, en az bir ulaşım şirketi (TransportCompany) tarafından sağlanır.

6. Konaklama (Accommodation):

Her konaklama benzersiz bir AccommodationID ve ReservationID kombinasyonu ile tanımlanır.

Her konaklama bir rezervasyonla ilişkilidir (ReservationID).

Konaklama kayıtları tür(Type) ve tarih (Date) bilgilerini içerir.

Konaklama kayıtları otel (Hotel) veya hostel (Hostel) olabilir (disjoint kuralı).

7. Otel (Hotel):

Her otel benzersiz bir HotelID ile tanımlanır.

Oteller, ad (HotelName), yıldız derecesi (StarRating) ve adres (Address) bilgilerine sahiptir. PK FKAccommodationID ve PK FKReservationID

8. Hostel (Hostel):

Her hostel benzersiz bir HostelID ile tanımlanır.

Hosteller, ad (HostelName) yıldız derecesi (StarRating) ve adres (Address) bilgilerine sahiptir.PK FKAccommodationID ve PK FKReservationID

9. Konumlar (Locations):

Her konum benzersiz bir LocationID ile tanımlanır.

Her konumun bir adı (Name) vardır ve ülke (Country) ile şehir (City) bilgilerini içerir. (disjoint)

Her konumda birden fazla gezi (Trips) düzenlenebilir.

BURAYI KNTROL ET!! LOCATAIONREVIEW VAR

10. Şehir (City):

Her şehir bir LocationID ile tanımlanır ve bir konumla ilişkilidir.

Şehirler ad (CityName) ve eyalet (State) bilgilerini içerir.

11. Ülke (Country):

Her ülke bir LocationID ile tanımlanır ve bir konumla ilişkilidir.

Ülkeler ad (CountryName) ve kıta (Continent) bilgilerini içerir.

12. Yorumlar (Reviews):

Her yorum benzersiz bir ReviewID ile tanımlanır.

Yorumlar bir kullanıcı (UserID) tarafından yazılır ve tarih (ReviewDate), metin (ReviewText) ve tür (ReviewType) bilgilerini içerir.

Yorumlar ya rezervasyonlara (ReservationReview) ya da konumlara (LocationReview) ilişkindir (disjoint kuralı).

13. Rezervasyon Yorumları (ReservationReview):

Her rezervasyon yorumu, bir yorum (ReviewID) ve bir rezervasyonla (ReservationID) ilişkilidir.

14. Konum Yorumları (LocationReview):

Her konum yorumu, bir yorum (ReviewID) ve bir konumla (LocationID) ilişkilidir.

15. Geziler (Trips):

Her gezi benzersiz bir TripID ile tanımlanır.

Geziler, bir kullanıcıya (UserID) aittir ve başlık (Title), başlangıç tarihi (StartDate), bitiş tarihi (EndDate), toplam maliyet (TotalCost) ve katılımcı sayısı (ParticipantCount) bilgilerini içerir.

Her gezi, birden fazla konum (Trip_Locations) ve aktiviteyi (Trip_Activities) içerebilir.

16. Aktiviteler (Activities):

Her aktivite benzersiz bir ActivityID ile tanımlanır.

Aktiviteler, ad (ActivityName), açıklama (Description) bilgilerini içerir.

17. Ulaşım Şirketleri (TransportCompany):

Her ulaşım şirketi benzersiz bir CompanyID ile tanımlanır.

Şirketler, ad (CompanyName), iletişim bilgileri (ContactInfo) ve adres (Address) bilgilerini içerir.

Her ulaşım şirketi, yalnızca bir tane ulaşım hizmeti (Transport) sağlar.

18. Ödemeler (Payments):

Her ödeme benzersiz bir PaymentID ile tanımlanır.

Her ödeme yalnızca bir rezervasyon (Reservations) için yapılır.

Ödeme, tarih (PaymentDate), yöntem (PaymentMethod) ve toplam tutar (TotalAmount) bilgilerini içerir.

Fonksiyonlar (Stored Procedures/Functions):

1. add_admin_user Fonksiyonu

Bu fonksiyon, yeni bir admin kullanıcısı eklemek için kullanılır.

- Parametreler:
    - p_userid: Kullanıcının benzersiz kimliği.
    - p_firstname, p_lastname: Kullanıcının adı ve soyadı.
    - p_email, p_password: Kullanıcının e-posta adresi ve şifresi.
    - p_phoneno: Kullanıcının telefon numarası.
    - p_permission_level: Kullanıcının sahip olduğu yetki seviyesi (admin, superadmin, vb.).
- İşlem:

Kullanıcının mevcut olup olmadığının kontrol edilmesi: İlk olarak, veritabanında bu userid'ye sahip bir kullanıcının olup olmadığı kontrol edilir. Eğer kullanıcı zaten varsa, bir hata mesajı döner ve işlem yapılmaz.

Admin kullanıcıyı ekleme: Kullanıcı zaten mevcut değilse, admin kullanıcısı public.adminuser tablosuna eklenir.

Başarı bildirimi: Kullanıcı başarıyla eklendiyse, işlem sonucu bir bildirim (notice) olarak kaydedilir.

2. delete_admin_user Fonksiyonu

Bu fonksiyon, belirli bir admin kullanıcısını silmek için kullanılır.

- Parametreler:
    - p_userid: Silinecek kullanıcının benzersiz kimliği.
- İşlem:Admin kullanıcısını silme: public.adminuser tablosundan, belirtilen userid'ye sahip olan kullanıcı silinir.

Başarı bildirimi: Kullanıcı başarıyla silindiği takdirde bir başarı bildirimi (notice) gösterilir.

3. transfer_reservation Fonksiyonu

Bir kullanıcının rezervasyonunu başka bir kullanıcıya aktarmak için kullanılır.

- Parametreler:
    - p_old_userid: Rezervasyonun mevcut sahibi olan kullanıcının kimliği.
    - p_new_userid: Rezervasyonun devredileceği yeni kullanıcının kimliği.

- p_reservation_id: Aktarılacak rezervasyonun kimliği.

İşlem: Rezervasyonun varlığının kontrol edilmesi: İlk olarak, belirtilen eski kullanıcıya ait bir rezervasyon olup olmadığı kontrol edilir. Eğer rezervasyon yoksa, bir hata mesajı döndürülür.

Rezervasyon aktarımı: Eğer rezervasyon mevcutsa, bu rezervasyon yeni kullanıcıya aktarılır (userid değeri güncellenir).

Başarı bildirimi: Rezervasyon başarıyla aktarıldığında bir bildirim (notice) döndürülür.

4. update_admin_user Fonksiyonu

Bu fonksiyon, var olan bir admin kullanıcısının bilgilerini güncellemek için kullanılır.

- Parametreler:
  - p_userid: Güncellenecek admin kullanıcısının kimliği.
  - p_firstname, p_lastname, p_email, p_password, p_phoneno, p_permission_level: Güncellenecek admin kullanıcısının bilgileri.

İşlem: Admin kullanıcısının bilgilerini güncelleme: public.adminuser tablosunda, belirtilen userid'ye sahip olan kullanıcı için belirli alanlar (firstname, lastname, email, password, phoneno, permission_level) güncellenir.

Başarı bildirimi: Kullanıcı başarıyla güncellendiyse bir başarı bildirimi (notice) döndürülür.

5. update_user_phoneno Fonksiyonu

Bu fonksiyon, bir kullanıcının telefon numarasını güncellemek için kullanılır.

- Parametreler:
  - p_userid: Telefon numarası güncellenmesi gereken kullanıcının kimliği.
  - p_phoneno: Yeni telefon numarası.
- İşlem:

Kullanıcının telefon numarasını güncelleme: public.users tablosunda, belirtilen userid'ye sahip kullanıcının telefon numarası p_phoneno ile güncellenir.

Başarı bildirimi: Telefon numarası başarıyla güncellenirse bir bildirim (notice) döndürülür.

Triggerlar (Triggers)

1. admin_permission_change_trigger Trigger'ı

Bu trigger, admin kullanıcılarının bilgileri güncellendiğinde otomatik olarak tetiklenir.

- **Olay:** Admin kullanıcılarının firstname veya lastname alanlarında bir değişiklik yapıldığında tetiklenir.
- **İşlem:** Admin kullanıcılarının izin seviyelerindeki değişiklikleri loglamak için log_admin_permission_change() fonksiyonu çağrılır. Bu sayede, herhangi bir kullanıcı izni değiştiğinde sistemdeki değişiklikler kaydedilir.

2. trigger_cascade_delete_reviews Trigger'ı

Bir kullanıcı silindiğinde, o kullanıcıya ait yorumların da silinmesini sağlayan bir trigger'dır.

- **Olay:** Kullanıcı DELETE işlemi ile silindiğinde tetiklenir.
- **İşlem:** Kullanıcı silindiği zaman, ona ait tüm yorumlar da silinir. Bu işlem, cascade_delete_reviews() fonksiyonu aracılığıyla yapılır. Bu sayede veritabanındaki ilişkilendirilmiş verilerde tutarsızlık oluşmaz.

3. update_related_transport_accommodation Trigger'ı

Bir rezervasyonun tarihi değiştiğinde, bu değişiklikle ilgili diğer tabloları güncellemek için kullanılır.

- **Olay:** reservation_date üzerinde bir UPDATE işlemi yapılırsa tetiklenir.
- **İşlem:** Rezervasyon tarihi değiştiğinde, buna bağlı olarak ulaşım ve konaklama bilgileri de güncellenir. Bu işlem update_related_transport_accommodation() fonksiyonu aracılığıyla gerçekleştirilir.

4. set_default_name_trigger Trigger'ı
Bir adminuser tablosuna yeni bir kayıt eklenirken, eğer isim alanı boş bırakılmışsa otomatik olarak bir varsayılan isim atanmasını sağlayan bir trigger'dır.

- **Olay:** adminuser tablosuna yapılan INSERT (ekleme) işlemi sırasında tetiklenir.
- **İşlem:** Yeni eklenen kayıtta name alanı boş (NULL) ise, set_default_name() fonksiyonu kullanılarak bu alana 'Default Name' değeri atanır.

  Amaç: Boş isimle kaydedilen verilerin önüne geçmek. Veritabanındaki name alanında tutarlı ve anlamlı değerler bulundurmak.

  Avantaj: Varsayılan bir değer atanarak hatalı ya da eksik veri girişlerinin önüne geçilir. Veritabanındaki veri bütünlüğü korunur.

SQL Kodları:

```
-- Create tables

CREATE TABLE public.users (  --overlapping /total comp.

                "userid" SERIAL PRIMARY KEY,

                "name" VARCHAR(50) NOT NULL,

                "surname" VARCHAR(50) NOT NULL,

                "email" VARCHAR(50) UNIQUE NOT NULL,

                "password" VARCHAR(255) NOT NULL,

                "phoneno" VARCHAR(30)
```

```
);


CREATE TABLE public.adminuser (
                        "userid" INT PRIMARY KEY,
                        "adminprivileges" VARCHAR(50),
                        "permissionlevel" VARCHAR(5) NOT NULL,
                        CONSTRAINT "unique_admin_userid" FOREIGN KEY ("userid") REFERENCES
public.users("userid") ON DELETE CASCADE
) INHERITS("public"."users");


CREATE TABLE public.regularuser (
                         "userid" INT PRIMARY KEY,
                         "regularuserprivileges" VARCHAR(50),
                         "membershiptype" VARCHAR(10),
                         CONSTRAINT "regularuser_userFK" FOREIGN KEY ("userid") REFERENCES
public.users("userid")
) INHERITS("public"."users");




CREATE TABLE public.locations (
                        "locationid" SERIAL PRIMARY KEY,
                        "name" VARCHAR(100) NOT NULL,
                        "country" VARCHAR(50) NOT NULL,
                        "city" VARCHAR(50) NOT NULL
);


CREATE TABLE public.cities (
                       "locationid" INT PRIMARY KEY,
                       "cityname" VARCHAR(100) NOT NULL,
                       "state" VARCHAR(100),
```

```
                    CONSTRAINT "city_locationFK" FOREIGN KEY ("locationid") REFERENCES public.locations("locationid")
) INHERITS("public"."locations");

CREATE TABLE public.countries (
                    "locationid" INT PRIMARY KEY,
                    "countryname" VARCHAR(100) NOT NULL,
                    "continent" VARCHAR(50),
                    CONSTRAINT "country_locationFK" FOREIGN KEY ("locationid") REFERENCES public.locations("locationid")
) INHERITS("public"."locations");

CREATE TABLE public.payments (
                    "paymentid" SERIAL PRIMARY KEY,
                    "paymentdate" DATE DEFAULT CURRENT_DATE,
                    "totalamount" NUMERIC(10, 2) NOT NULL,
                    "paymentmethod" VARCHAR(50) NOT NULL
);

CREATE TABLE public.reservations (
                    "reservationid" SERIAL PRIMARY KEY,
                    "reservationdate" DATE DEFAULT CURRENT_DATE,
                    "userid" INT NOT NULL,
                    "paymentid" INT NOT NULL,
                    CONSTRAINT "reservation_userFK" FOREIGN KEY ("userid") REFERENCES public.users("userid") ON DELETE CASCADE,
                    CONSTRAINT "reservation_paymentFK" FOREIGN KEY ("paymentid") REFERENCES public.payments("paymentid") ON DELETE SET NULL
);

CREATE TABLE public.transport (
                    "transportid" SERIAL PRIMARY KEY,
```

```
                "type" VARCHAR(50) NOT NULL,
                "departuretime" TIMESTAMP NOT NULL,
                "reservationid" INT NOT NULL,
                CONSTRAINT "transport_reservationFK" FOREIGN KEY ("reservationid")
REFERENCES public.reservations("reservationid") ON DELETE CASCADE
) INHERITS("public"."reservations");

CREATE TABLE public.accommodation (
                "accommodationid" SERIAL PRIMARY KEY,
                "type" VARCHAR(50) CHECK ("type" IN ('hotel', 'hostel')),
                "date" DATE NOT NULL,
                "reservationid" INT NOT NULL,
                CONSTRAINT "accommodation_reservationFK" FOREIGN KEY ("reservationid")
REFERENCES public.reservations("reservationid") ON DELETE CASCADE
) INHERITS("public"."reservations");

CREATE TABLE public.hotels (
                "hotelid" SERIAL PRIMARY KEY,
                "hotelname" VARCHAR(100) NOT NULL,
                "starrating" INT CHECK ("starrating" BETWEEN 1 AND 5),
                "address" TEXT NOT NULL,
                "accommodationid" INT NOT NULL,
                CONSTRAINT "hotel_accommodationFK" FOREIGN KEY ("accommodationid")
REFERENCES public.accommodation("accommodationid") ON DELETE CASCADE
) INHERITS("public"."accommodation");

CREATE TABLE public.hostels (
                "hostelid" SERIAL PRIMARY KEY,
                "hostelname" VARCHAR(100) NOT NULL,
                "starrating" INT CHECK ("starrating" BETWEEN 1 AND 5),
                "address" TEXT NOT NULL,
                "accommodationid" INT NOT NULL,
```

```
		CONSTRAINT "hostel_accommodationFK" FOREIGN KEY ("accommodationid") REFERENCES public.accommodation("accommodationid") ON DELETE CASCADE

) INHERITS("public"."accommodation");


CREATE TABLE public.transportcompany (

		"companyid" SERIAL PRIMARY KEY,

		"transportid" INT NOT NULL REFERENCES public.transport("transportid") ON DELETE CASCADE ON UPDATE CASCADE,

		"companyname" VARCHAR(100) NOT NULL,

		"contactinfo" TEXT,

		"address" TEXT

);


CREATE TABLE public.reviews (

		"reviewid" SERIAL PRIMARY KEY,

		"userid" INT NOT NULL,

		"reviewdate" DATE DEFAULT CURRENT_DATE,

		"reviewtext" TEXT NOT NULL,

		"reviewtype" VARCHAR(50) NOT NULL,

		CONSTRAINT "review_userFK" FOREIGN KEY ("userid") REFERENCES public.users("userid") ON DELETE CASCADE

);


CREATE TABLE public.reservationreview (

		 "reviewid" INT NOT NULL,

		 "reservationid" INT NOT NULL,

		 CONSTRAINT "reservationreviewPK" PRIMARY KEY ("reviewid", "reservationid"),

		 CONSTRAINT "reservationreview_reservationFK" FOREIGN KEY ("reservationid") REFERENCES public.reservations("reservationid") ON DELETE CASCADE,

		 CONSTRAINT "reservationreview_reviewFK" FOREIGN KEY ("reviewid") REFERENCES public.reviews("reviewid") ON DELETE CASCADE

) INHERITS("public"."reviews");
```

```
CREATE TABLE public.locationreview (
                        "locationreviewid" SERIAL PRIMARY KEY,
                        "reviewid" INT NOT NULL,
                        "locationid" INT NOT NULL,
                        CONSTRAINT "locationreview_locationFK" FOREIGN KEY ("locationid")
REFERENCES public.locations("locationid") ON DELETE CASCADE,
                        CONSTRAINT "locationreview_reviewFK" FOREIGN KEY ("reviewid")
REFERENCES public.reviews("reviewid") ON DELETE CASCADE
) INHERITS("public"."reviews");

CREATE TABLE public.trips (
                    "tripid" SERIAL PRIMARY KEY,
                    "userid" INT NOT NULL,
                    "title" VARCHAR(100) NOT NULL,
                    "startdate" DATE NOT NULL,
                    "enddate" DATE NOT NULL,
                    "totalcost" DECIMAL(10, 2) NOT NULL,
                    "participantcount" INT CHECK ("participantcount" >= 1),
                    CONSTRAINT "trip_userFK" FOREIGN KEY ("userid") REFERENCES
public.users("userid") ON DELETE CASCADE
);

CREATE TABLE public.activities (
                        "activityid" SERIAL PRIMARY KEY,
                        "activityname" VARCHAR(100) NOT NULL,
                        "description" TEXT
);

CREATE TABLE public.tripactivities (
                        "tripid" INT NOT NULL,
                        "activityid" SERIAL,
```

```
                        "cost" DECIMAL(10, 2) NOT NULL,

                        "activitydate" DATE NOT NULL,

                        CONSTRAINT "tripactivityPK" PRIMARY KEY ("tripid", "activityid"),

                        CONSTRAINT "tripactivity_tripFK" FOREIGN KEY ("tripid") REFERENCES
public.trips("tripid") ON DELETE CASCADE,

                        CONSTRAINT "tripactivity_activityFK" FOREIGN KEY ("activityid") REFERENCES
public.activities("activityid") ON DELETE CASCADE

);


CREATE TABLE public.triplocations (

                        "tripid" INT NOT NULL,

                        "locationid" INT NOT NULL,

                        "type" VARCHAR(50) NOT NULL,

                        "description" TEXT,

                        CONSTRAINT "triplocationPK" PRIMARY KEY ("tripid", "locationid"),

                        CONSTRAINT "triplocation_tripFK" FOREIGN KEY ("tripid") REFERENCES
public.trips("tripid") ON DELETE CASCADE,

                        CONSTRAINT "triplocation_locationFK" FOREIGN KEY ("locationid") REFERENCES
public.locations("locationid") ON DELETE CASCADE

);
CREATE OR REPLACE FUNCTION add_admin_user(

   p_userid INT,

   p_name VARCHAR(50),

   p_surname VARCHAR(50),

   p_email VARCHAR(50),

   p_password VARCHAR(255),

   p_phoneno VARCHAR(30),

   p_adminprivileges VARCHAR(50),

   p_permissionlevel VARCHAR(5)

)

   RETURNS VOID AS $$

BEGIN
```

```
    -- Check if userid exists in the users table
    IF EXISTS (SELECT 1 FROM public.users WHERE userid = p_userid) THEN
        RAISE EXCEPTION 'Cannot add admin user because the UserID % already exists.', p_userid;
    END IF;

    -- Insert into the adminuser table
    INSERT INTO public.adminuser (userid, name, surname, email, password, phoneno, adminprivileges,
permissionlevel)
    VALUES (
            p_userid,
            p_name,
            p_surname,
            p_email,
            p_password,
            p_phoneno,
            p_adminprivileges,
            p_permissionlevel
        );

    -- Check for PL/pgSQL language
    PERFORM * FROM pg_language WHERE lanname = 'plpgsql';

    -- Optionally, you can raise a notice for success
    RAISE NOTICE 'Admin user % % added successfully.', p_name, p_surname;
END;
$$ LANGUAGE plpgsql;



SELECT * FROM public.users WHERE userid = 101;
------------------------------------------------------------FONK1 çalıştır
```

```
-- Foreign key kısıtlamasını geçici olarak kaldır
ALTER TABLE public.adminuser DISABLE TRIGGER ALL;

------------------>
SELECT add_admin_user( 101, 'John','Doe','john@gmail.com','password123','+904545645','full','4');


SELECT * FROM public.adminuser WHERE userid = ;

---------->
SELECT*FROM adminuser;


SELECT delete_admin_user(); ----------------FONK2 Çalıştır


-- Foreign key kısıtlamasını tekrar etkinleştir
ALTER TABLE public.adminuser ENABLE TRIGGER ALL;


-- Function to delete admin user
CREATE OR REPLACE FUNCTION delete_admin_user(p_userid INT)
    RETURNS VOID AS $$
BEGIN
    -- Delete from the adminuser table using the provided userid
    DELETE FROM public.adminuser WHERE userid = p_userid;

    -- Optionally, raise a notice
    RAISE NOTICE 'Admin user with userid % deleted successfully.', p_userid;
END;
$$ LANGUAGE plpgsql;
```

```
-- Function to update admin user
CREATE OR REPLACE FUNCTION update_admin_user(
    p_userid INT,
    p_name VARCHAR(50),
    p_surname VARCHAR(50),
    p_email VARCHAR(50),
    p_password VARCHAR(255),
    p_phoneno VARCHAR(30),
    p_adminprivileges VARCHAR(50),
    p_permissionlevel VARCHAR(5)
)
    RETURNS VOID AS $$
BEGIN
    -- Update the adminuser table based on the provided userid
    UPDATE public.adminuser
    SET
        name = p_name,
        surname = p_surname,
        email = p_email,
        password = p_password,
        phoneno = p_phoneno,
        adminprivileges = p_adminprivileges,
        permissionlevel = p_permissionlevel
    WHERE userid = p_userid;

    -- Optionally, raise a notice
    RAISE NOTICE 'Admin user with userid % updated successfully.', p_userid;
END;
$$ LANGUAGE plpgsql;
```

```
-- Function to update user phone number
CREATE OR REPLACE FUNCTION update_user_phoneno(
    p_userid INT,
    p_new_phoneno VARCHAR(30)
)
    RETURNS VOID AS $$
BEGIN
    -- Update the user's phone number
    UPDATE public.users
    SET phoneno = p_new_phoneno
    WHERE userid = p_userid;

    -- Optionally, raise a notice
    RAISE NOTICE 'Phone number for user with userid % updated successfully.', p_userid;
END;
$$ LANGUAGE plpgsql;

-- Function to transfer reservation from one user to another
CREATE OR REPLACE FUNCTION transfer_reservation(
    p_old_userid INT,
    p_new_userid INT,
    p_reservationid INT
)
    RETURNS VOID AS $$
BEGIN
    -- Check if the old user has the reservation
    IF NOT EXISTS (SELECT 1 FROM public.reservations WHERE userid = p_old_userid AND reservationid
= p_reservationid) THEN
        RAISE EXCEPTION 'Reservation with id % does not belong to user with id %.', p_reservationid,
p_old_userid;
    END IF;
```

```
    -- Update the reservation to be under the new user's name
    UPDATE public.reservations
    SET userid = p_new_userid
    WHERE reservationid = p_reservationid;

    -- Optionally, raise a notice
    RAISE NOTICE 'Reservation with id % transferred from user % to user %.', p_reservationid,
p_old_userid, p_new_userid;
END;
$$ LANGUAGE plpgsql;



-- Trigger function to cascade delete related reservations
CREATE OR REPLACE FUNCTION cascade_delete_reservation()
    RETURNS TRIGGER AS $$
BEGIN
    -- Cascade delete related transport and accommodation records
    DELETE FROM public.transport WHERE reservationid = OLD.reservationid;
    DELETE FROM public.accommodation WHERE reservationid = OLD.reservationid;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;


----------------
CREATE OR REPLACE FUNCTION log_admin_permission_change()
    RETURNS TRIGGER AS $$
BEGIN
    -- Log admin permission changes in a log table
    INSERT INTO public.users_log (userid, action, action_date)
    VALUES (NEW.userid, 'Admin Permissions Updated', CURRENT_TIMESTAMP);
```

```
    -- Return the new record so that the update happens
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TABLE public.users_log (
    log_id SERIAL PRIMARY KEY,
    userid INT NOT NULL,
    action VARCHAR(255) NOT NULL,
    action_date TIMESTAMP NOT NULL,
    FOREIGN KEY (userid) REFERENCES public.users(userid) ON DELETE CASCADE
);


CREATE TRIGGER admin_permission_change_trigger
    AFTER UPDATE ON public.users
    FOR EACH ROW
    WHEN (OLD.name IS DISTINCT FROM NEW.name OR OLD.surname IS DISTINCT FROM
NEW.surname)
    EXECUTE FUNCTION log_admin_permission_change();


----------
-- Trigger function to delete related reviews when a user is deleted
CREATE OR REPLACE FUNCTION cascade_delete_reviews()
    RETURNS TRIGGER AS $$
BEGIN
    -- Delete reviews related to the user
    DELETE FROM public.reviews WHERE userid = OLD.userid;
```

```
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Trigger definition for the 'users' table
CREATE TRIGGER trigger_cascade_delete_reviews
    AFTER DELETE ON public.users
    FOR EACH ROW
    EXECUTE FUNCTION cascade_delete_reviews();
```

```
-- Trigger function to update transport and accommodation dates when reservation date changes
CREATE OR REPLACE FUNCTION update_related_transport_accommodation()
    RETURNS TRIGGER AS $$
BEGIN
    -- Update transport date related to the reservation
    UPDATE public.transport
    SET departuretime = NEW.reservationdate
    WHERE reservationid = NEW.reservationid;

    -- Update accommodation date related to the reservation
    UPDATE public.accommodation
    SET date = NEW.reservationdate
    WHERE reservationid = NEW.reservationid;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
-- Trigger definition for the 'reservations' table
CREATE TRIGGER trigger_update_related_transport_accommodation
   AFTER UPDATE OF reservationdate ON public.reservations
   FOR EACH ROW
   WHEN (OLD.reservationdate IS DISTINCT FROM NEW.reservationdate)
   EXECUTE FUNCTION update_related_transport_accommodation();

CREATE OR REPLACE FUNCTION set_default_name()
RETURNS TRIGGER AS $$
BEGIN
   -- Eğer isim verilmemişse, varsayılan bir isim belirle
   IF NEW.name IS NULL THEN
      NEW.name := 'Default Name';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER set_default_name_trigger
BEFORE INSERT
ON public.adminuser
FOR EACH ROW
EXECUTE FUNCTION set_default_name();



INSERT INTO public.users ("userid", "name", "surname", "email", "password", "phoneno")
VALUES (6, 'Alex', 'Morgan', 'amorgan7@abc.com', 'Xy7!pZ3b', '+1 (312) 485-2938');

INSERT INTO public.users ("userid", "name", "surname", "email", "password", "phoneno")
VALUES (2, 'Sam', 'Tanner', 'stanner2@xyz.org', 'Qw3@RtY9', '+1 (561) 307-8234');
```

INSERT INTO public.users ("userid", "name", "surname", "email", "password", "phoneno")
VALUES (3, 'Jordan', 'Lee', 'jlee8@company.net', 'Pq2@Jh8o', '+1 (415) 652-9726');

INSERT INTO public.users ("userid", "name", "surname", "email", "password", "phoneno")
VALUES (4, 'Chris', 'Henderson', 'chenderson5@demo.edu', 'Lx4!Fg7v', '+1 (555) 107-4829');

INSERT INTO public.users ("userid", "name", "surname", "email", "password", "phoneno")
VALUES (7, 'Taylor', 'Davis', 'tdavis1@service.com', 'Bf9!TuX3', '+1 (408) 305-2013');