T.C
KOCAELİ SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR/YAZILIM MÜHENDİSLİĞİ

VERİ TABANI YÖNETİM SİSTEMLERİ PROJESİ

Muhammed Yusuf Kaya
210501007

Talha Tuna
220502015

DERS SORUMLUSU:
PROF. DR. NEVCİHAN DURU

TARİH:
05/05/2024

1 GİRİŞ

1.1 Projenin amacı

- Bu ödevin amacı bir gemi yönetim sistemi oluşturmaktır. Bu sistemde gemiler, limanlar, gemi seferleri ve mürettebat gibi çeşitli verileri yönetmek için bir veritabanı kullanılmıştır. Ayrıca PyQt5 kütüphanesi kullanılarak bir arayüz oluşturulmuştur. Gemi yönetimiyle ilgili verilerin etkili bir şekilde saklanması, güncellenmesi ve görüntülenmesidir. Ayrıca gemi seferlerinin ve liman ziyaretlerinin yönetilmesi için bir arayüz sunarak kullanıcıların bu verilere kolayca erişmesini sağlamaktır.

2 GEREKSİNİM ANALİZİ

2.1 Arayüz gereksinimleri

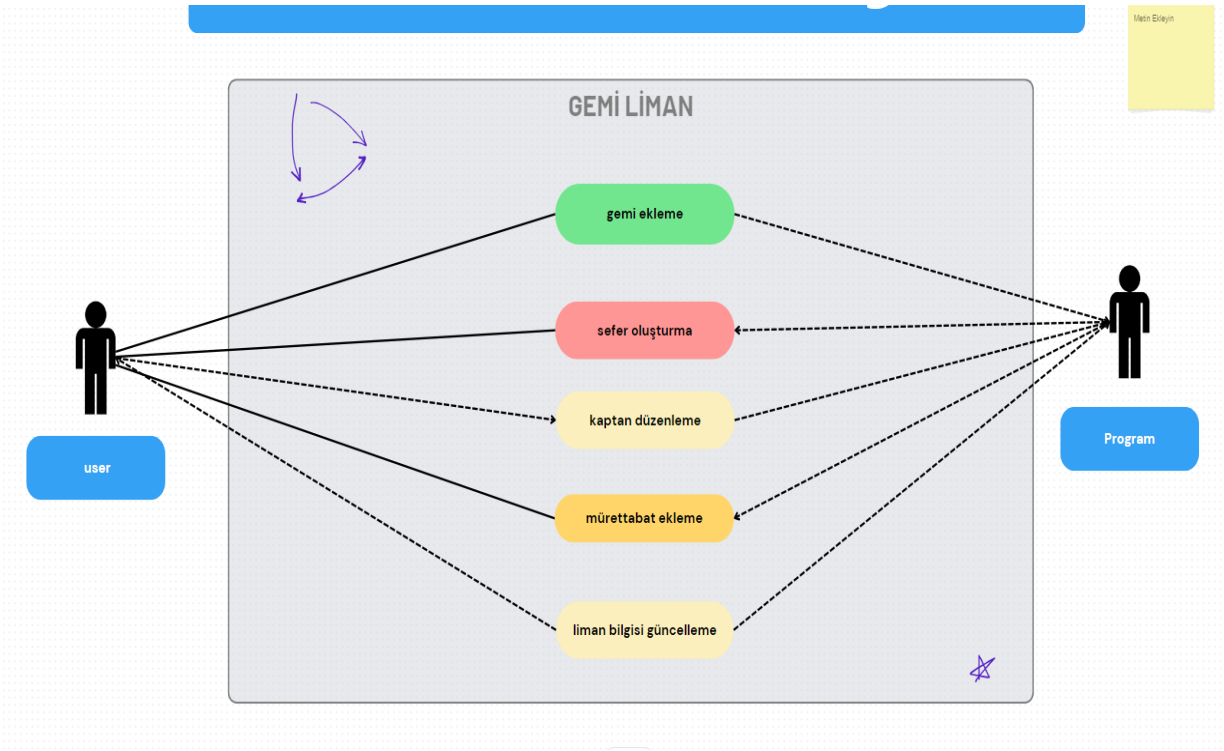
- **Ana Pencere (MainWindow):**
- Kullanıcıların veritabanındaki tabloları seçebilmesi için bir ComboBox sağlanmaktadır
- Seçilen tablonun içeriğini göstermek için bir QTextEdit alanı bulunmaktadır.
- Kullanıcıların tablodaki verilere erişim sağlaması için ekleme, silme ve güncelleme gibi işlemler için butonlar bulunmaktadır.
- **Veri Ekleme (AddDataDialog):**
- Kullanıcının bir tabloya yeni veri ekleyebilmesi için gerekli alanlar sağlanıyor.
- Kullanıcının hangi tabloya veri ekleyeceğini seçebilmesi için bir ComboBox bulunuyor.
- Kullanıcının eklemek istediği veriyi girmesi için gerekli alanlar sağlanıyor.
- Veri eklenmesi işlemi başarılı veya başarısız olduğunda kullanıcıya bilgi mesajları gösterilmektedir.
- **Veri Silme (DeleteDataDialog):**
- Kullanıcının bir tablodan veri silebilmesi için gerekli alanlar sağlanıyor.
- Hangi tablodan veri silineceğini ve silinecek olan verinin kimliğini girmesi için gerekli alanlar bulunmaktadır.
- Veri silme işlemi başarılı veya başarısız olduğunda kullanıcıya bilgi mesajları gösterilmektedir.
- **Veri Güncelleme (UpgradeDataDialog):**
- Kullanıcının bir tablodaki veriyi güncelleyebilmesi için gerekli

alanlar sağlanmaktadır.
- Hangi tablodaki verinin güncelleneceğini, güncellenecek sütunun adını ve yeni değeri girmesi için gerekli alanlar bulunmaktadır.
- Veri güncelleme işlemi başarılı veya başarısız olduğunda kullanıcıya bilgi mesajları gösterilmektedir.

2.2 Fonksiyonel Gereksinimler:

- **Veritabanı Bağlantısı:**
- Uygulamanın başlatılmasıyla birlikte, SQLite veritabanına bağlanılıyor.
- Bağlantı başarısız olduğunda kullanıcıya hata mesajı gösterilmektedir.
- **Tabloların Oluşturulması:**
- Gerekli tabloların (Gemi, Liman, Gemi Seferi, vb.) SQLite veri tabanında oluşturuluyor.
- **Veri Gösterimi:**
- Kullanıcının seçtiği tablonun içeriğinin doğru bir şekilde gösteriliyor.
- Her bir tablonun sütun başlıklarının ve verilerinin düzgün bir şekilde ekranda gösteriliyor.
- **Veri Ekleme, Silme ve Güncelleme:**
- Kullanıcıların veri ekleyeme, silme ve güncelleme yapabiliyor.
- Bu işlemlerin her biri doğru ve güvenli bir şekilde gerçekleştirilebiliyor.
- Kullanıcıya işlem sonucuyla ilgili uygun geri bildirimler sağlanmaktadır.
- **Güvenlik ve Veri Bütünlüğü:**
- Kullanıcıların veritabanında yalnızca yetkilendirildikleri işlemleri gerçekleştirebilir.
- Veritabanındaki verilerin bütünlüğü sağlanılıyor ve hatalı veri girişlerine karşı koruma önlemleri alınıyor.

2.2 Use-Case diyagramı



3 TASARIM

3.1 Mimari tasarım

- **Ana Sınıflar ve İlişkiler:**
- **ShipTable:** Bu sınıf, gemi tablosunu oluşturmak ve gemi verilerini saklamak için kullanılmıştır. Diğer gemi türlerinin atasıdır.
- **CruiseShip, OilShip, ContainerShip:** Bu sınıflar, farklı gemi tipleri için (turistik, petrol, konteyner)özel tabloları oluşturur ve ilgili gemi özelliklerini saklar. ShipTable sınıfından türetilmiştir.
- **VoyageTable:** Gemi seferlerini ve ilgili bilgileri saklamak için kullanılan bir sınıftır. Gemilerle ilişkili bir dış anahtar içerir.
- **Harbors:** Liman bilgilerini saklamak için kullanılan bir sınıftır.
- **VisitedHarbour:** Ziyaret edilen limanları ve bu limanlara ait gemileri saklamak için kullanılan bir sınıftır. Harbors ve ShipTable sınıflarıyla ilişkilidir.
- **Captains, CrewMembers:** Personel bilgilerini saklamak için kullanılan sınıflardır. Captains sınıfı gemi kaptanlarını, CrewMembers sınıfı ise mürettebat üyelerini temsil eder.

- **Arayüz Sınıfları:**

- **MainWindow:** Ana pencere sınıfıdır. Veritabanındaki tabloları gösterir ve kullanıcıların bu tablolarda işlem yapmasını sağlar.
- **AddDataDialog:** Kullanıcının yeni veri eklemesine olanak tanır. Hangi tabloya veri ekleyeceğini seçmesine ve gerekli bilgileri girmesine izin verir.
- **DeleteDataDialog:** Kullanıcının veri silmesine olanak tanır. Hangi tablodan veri sileceğini ve silinecek verinin kimliğini girmesine izin verir.
- **UpgradeDataDialog:** Kullanıcının mevcut veriyi güncellemesine olanak tanır. Hangi tablodaki veriyi güncelleyeceğini, güncellenecek sütunun adını ve yeni değeri girmesine izin verir.

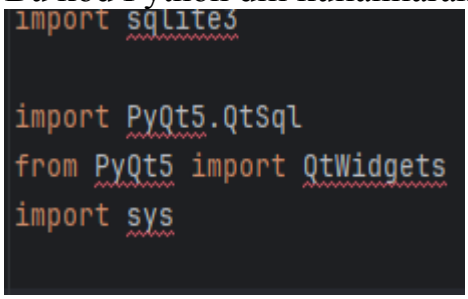
- **Mimarî Kararlar ve Özellikler:**

- **Kalıtım Kullanımı:** Farklı gemi tipleri için ortak özellikleri saklamak için kalıtım kullanılmıştır. Bu sayede kod tekrarı önlenmiş ve sürdürülebilirlik artırılmıştır.
- **SQLite Veritabanı:** Küçük ölçekli projeler için hafif ve kullanımı kolay olan SQLite veritabanı tercih edilmiştir.
- **PyQt5 Arayüz Kütüphanesi:** Kullanıcı dostu bir arayüz oluşturmak için PyQt5 kütüphanesi kullanılmıştır. Bu kütüphane, çeşitli arayüz bileşenlerini sağlar ve Python ile uyumlu bir şekilde çalışır.
- **Fonksiyonel ve Arayüz Gereksinimlerin Karşılanması:** Projenin

mimarisi, kullanıcıların veritabanındaki verilere kolayca erişim sağlayabileceği ve işlem yapabileceği şekilde tasarlanmıştır.

3.2 Kullanılacak teknolojiler

Bu kod Python dili kullanılarak yazılmıştır



sqlite3:
Python'da SQLite veritabanı ile etkileşim sağlamak için standart kütüphanedir. Veritabanı bağlantısı oluşturmayı, sorgu çalıştırmayı ve sonuçları almayı sağlar.
PyQt5:
Python için popüler bir GUI (Grafik Kullanıcı Arayüzü) kütüphanesidir. Qt5'in Python sürümüdür ve zengin bir arayüz oluşturma imkanı sunar.
QtWidgets:
PyQt5 kütüphanesinin bir alt modülüdür. PyQt5 ile grafik kullanıcı arayüzü oluştururken sıkça kullanılan arayüz bileşenlerini içerir.
sys:
Python'un standart kütüphanelerinden biridir. Sistemle ilgili işlemler yapmayı sağlar. Örneğin, programı sonlandırmak için **sys.exit()** gibi fonksiyonlar sağlar.
PyQt5.QtSql:
PyQt5 içinde bulunan bir modüldür ve veritabanı işlemleri için gerekli araçları sağlar. SQLite gibi veritabanlarına bağlanmayı, sorguları çalıştırmayı ve sonuçları almayı kolaylaştırır.
QtGui:
PyQt5 kütüphanesinin bir alt modülüdür ve grafik arayüz oluştururken kullanılan çeşitli grafiksel öğeleri içerir

3.3 Veri tabanı tasarımı

SHIPS Tablosu:

- Gemilere ait genel bilgileri depolar.
- SÜTUNLAR:
  - SERIAL_NO: Gemilerin seri numarasını temsil eder (Anahtar).
  - NAME: Geminin adını temsil eder.
  - WEIGHT: Geminin ağırlığını temsil eder.
  - PRODUCT_YEAR: Geminin üretim yılını temsil eder.
  - SHIP_TYPE: Geminin türünü temsil eder.

CRUISE_SHIP Tablosu:

- Yolcu gemilerine ait özel bilgileri depolar.
- SÜTUNLAR:
  - SHIP_SERIAL_NO: Gemilerin seri numarasını temsil eder (Anahtar ve SHIPS tablosu ile ilişkilendirilmiştir).
  - SHIP_CAPACITY: Yolcu gemilerinin kapasitesini temsil eder.

OIL_SHIP Tablosu:

- Petrol tankerlerine ait özel bilgileri depolar.
- SÜTUNLAR:
  - SHIP_SERIAL_NO: Gemilerin seri numarasını temsil eder (Anahtar ve SHIPS tablosu ile ilişkilendirilmiştir).
  - SHIP_OIL_CAPACITY: Petrol tankerlerinin taşıma kapasitesini temsil eder.

CONTAINER_SHIP Tablosu:

- Konteyner gemilerine ait özel bilgileri depolar.
- SÜTUNLAR:
  - SHIP_SERIAL_NO: Gemilerin seri numarasını temsil eder (Anahtar ve SHIPS tablosu ile ilişkilendirilmiştir).
  - SHIP_CONTAINER_AMOUNT: Konteyner gemilerinin taşıma kapasitesini temsil eder.
  - SHIP_MAX_CAPACITY: Konteyner gemilerinin maksimum taşıma kapasitesini temsil eder.

VOYAGES Tablosu:

- Gemi seferlerine ait bilgileri depolar.
- SÜTUNLAR:
  - ID: Seferin benzersiz kimliğini temsil eder (Anahtar).
  - TAKEOFF_DATE: Seferin kalkış tarihini temsil eder.
  - RETURN_DATE: Seferin dönüş tarihini temsil eder.
  - TAKEOFF_PLACE: Seferin kalkış noktasını temsil eder.
  - SERIAL_NO: Gemilerin seri numarasını temsil eder (SHIPS tablosu ile ilişkilendirilmiştir).
  - CAP: Yolcu kapasitesini temsil eder.

- CREW: Mürettebat sayısını temsil eder.

HARBORS Tablosu:

- Liman bilgilerini depolar.
- SÜTUNLAR:
- HARBOUR_NAME: Limanın adını temsil eder (Anahtar).
- COUNTRY: Limanın bulunduğu ülkeyi temsil eder.
- POPULATION: Liman şehrinin nüfusunu temsil eder.
- PASSPORT: Geçerli pasaport sayısını temsil eder.
- BILLCOST: Fatura maliyetini temsil eder.

VISITED_HARBOURS Tablosu:

- Ziyaret edilen limanları depolar.
- SÜTUNLAR:
- HARBOUR_NAME: Limanın adını temsil eder (Anahtar ve HARBORS tablosu ile ilişkilendirilmiştir).
- HARBOUR_COUNTRY: Limanın bulunduğu ülkeyi temsil eder (Anahtar ve HARBORS tablosu ile ilişkilendirilmiştir).
- SERIAL_NO: Gemilerin seri numarasını temsil eder (SHIPS tablosu ile ilişkilendirilmiştir).

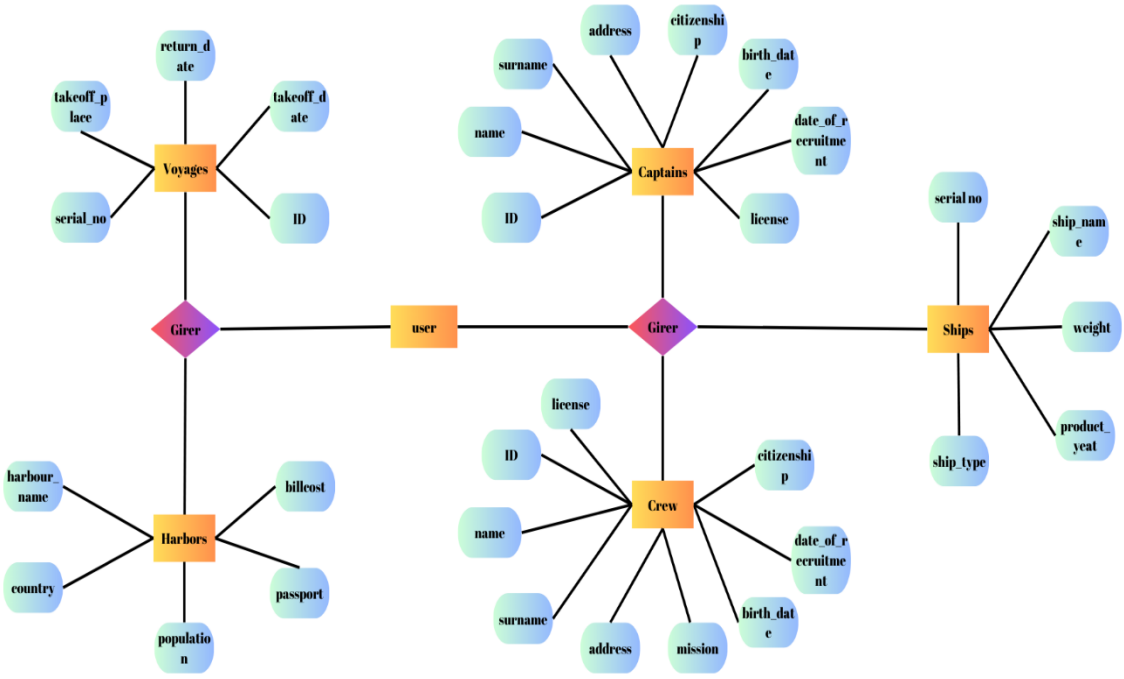
CAPTAINS Tablosu:

- Kaptanların bilgilerini depolar.
- SÜTUNLAR:
- ID: Kaptanın benzersiz kimliğini temsil eder (Anahtar).
- NAME: Kaptanın adını temsil eder.
- SURNAME: Kaptanın soyadını temsil eder.
- ADDRESS: Kaptanın adresini temsil eder.
- CITIZENSHIP: Kaptanın vatandaşlık bilgisini temsil eder.
- BIRTH_DATE: Kaptanın doğum tarihini temsil eder.
- DATE_OF_RECRUITMENT: Kaptanın işe alınma tarihini temsil eder.
- LICENSE: Kaptanın lisans bilgisini temsil eder.

CREW_MEMBERS Tablosu:

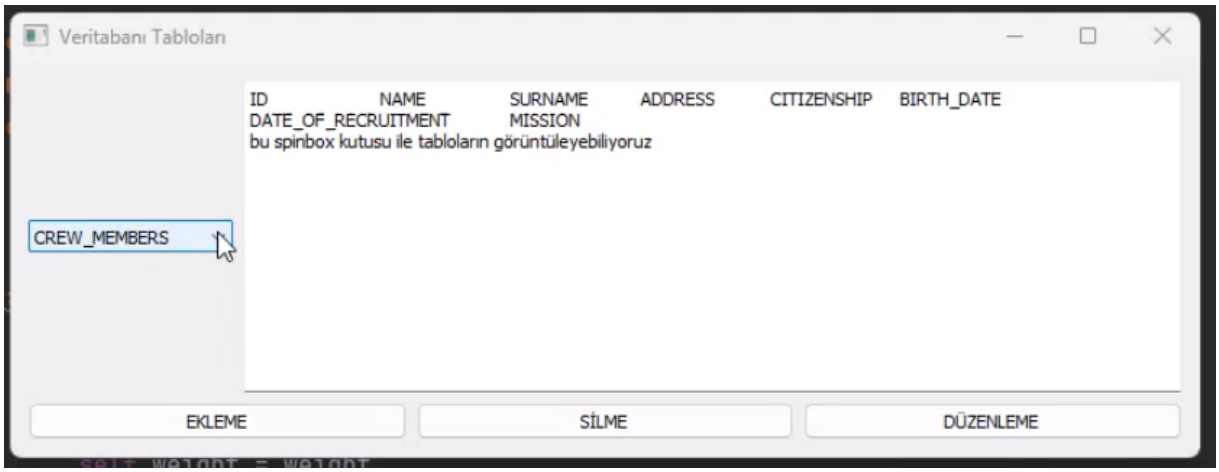
- Mürettebat üyelerinin bilgilerini depolar.
- SÜTUNLAR:
- ID: Mürettebat üyesinin benzersiz kimliğini temsil eder (Anahtar).
- NAME: Mürettebat üyesinin adını temsil eder.
- SURNAME: Mürettebat üyesinin soyadını temsil eder.
- ADDRESS: Mürettebat üyesinin adresini temsil eder.
- CITIZENSHIP: Mürettebat üyesinin vatandaşlık bilgisini temsil eder.
- BIRTH_DATE: Mürettebat üyesinin doğum tarihini temsil eder.
- DATE_OF_RECRUITMENT: Mürettebat üyesinin işe alınma tarihini temsil eder.
- MISSION: Mürettebat üyesinin görev bilgisini temsil eder.

ER Diyagramı:

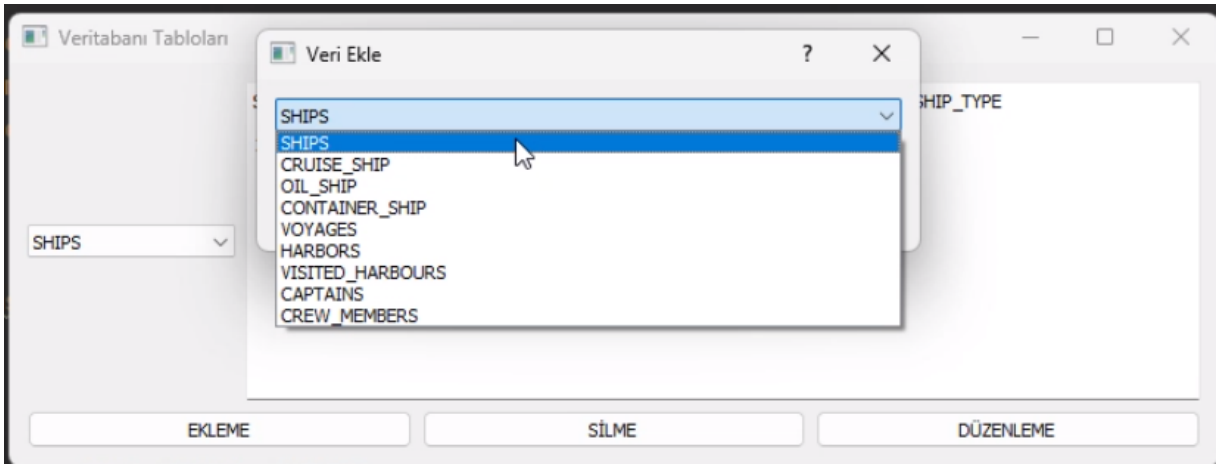


3.4 Kullanıcı arayüzü tasarımı

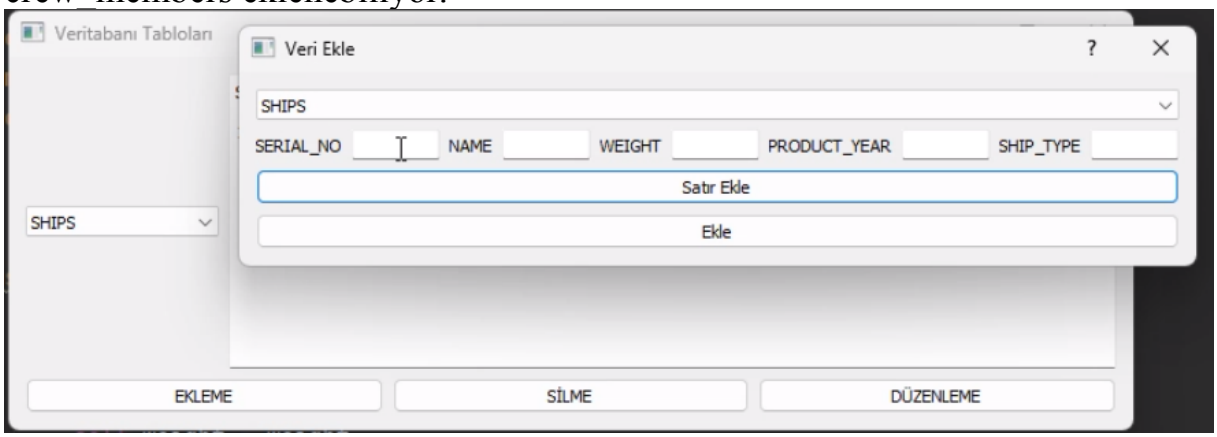
İlk olarak kodu çalıştırıyoruz kod çalıştıktan sonra şu şekil bir çıktı geliyor.



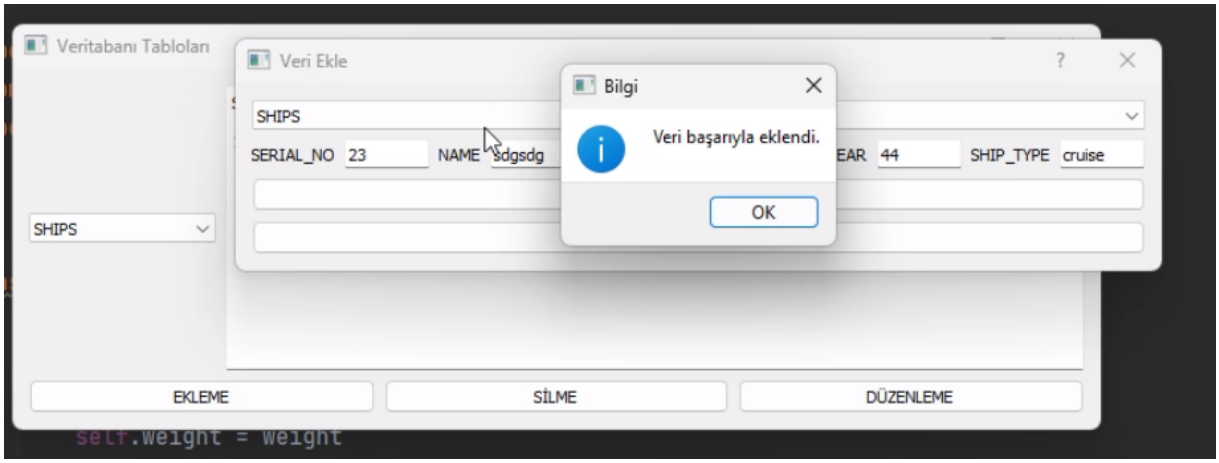
Bu şekilde bir pencere açılıyor. Pencerede kayıt ekle, kayıt sil, kayıt düzenleme butonları bulunuyor. Bu butonlardan herhangi birine tıklayınca başka bir pencere açılıyor



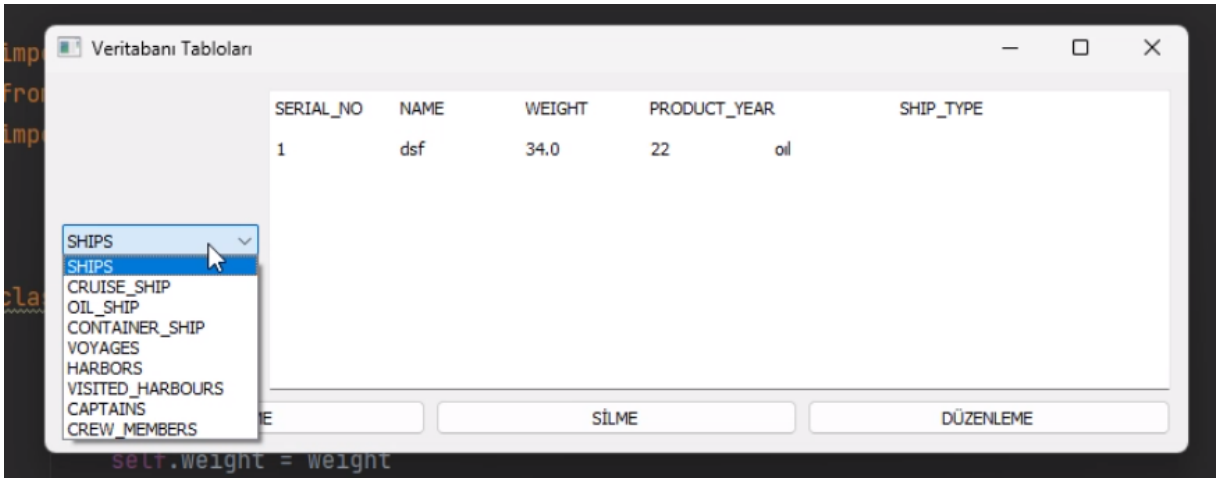
Bu şekilde bir pencere açılıyor. Eklenecek olan cruise ships , oıl_shıps, contaıner, shıps, voyages, harbors, vısıted_harbours, captaıns, crew_members eklenebiliyor.



Doldurulması gereken değerleri giriyoruz serıal_no, name, weıght, product_year, shıp_type ekle butonuna basıyoruz



Veri başarıyla eklendi mesajı karşımıza çıkıyor.



Veriler veritabanamıza ekleniyor. Satır ekrana geliyor

4 UYGULAMA
4.1 Kodlanan bileşenlerin açıklamaları

```python
class ShipTable:
    def __init__(self, serial_no, name, weight, product_year, ship_type):
        '''Alınacak bilgilerin nesnelerini oluşturma'''
        self.serial_no = serial_no
        self.name = name
        self.weight = weight
        self.product_year = product_year
        self.ship_type = ship_type

        self.create_connect()

    2 usages
    def create_connect(self):
        '''Tablonun sql kodu ile oluşturulması'''
        self.connect = sqlite3.connect("Odev2.db")     # veri tabanına bağlanma
        self.cursor = self.connect.cursor()

        table = "CREATE TABLE IF NOT EXISTS SHIPS(" \
                "SERIAL_NO INTEGER PRIMARY KEY UNIQUE, " \
                "NAME TEXT NOT NULL, " \
                "WEIGHT FLOAT NOT NULL, " \
                "PRODUCT_YEAR INTEGER NOT NULL," \
                "SHIP_TYPE TEXT NOT NULL)"

        self.cursor.execute(table)
        self.connect.commit()                          # veri tabanındaki değişikleri kaydetme
```

__init__ Metodu:
Bu metod, bir **ShipTable** nesnesi oluşturulduğunda otomatik olarak çağrılır.
serial_no, **name**, **weight**, **product_year**, ve **ship_type** gibi gemi özelliklerini alır ve bunları sınıfın örnek değişkenlerine atar.
Ardından **create_connect** metodu çağrılarak veritabanı bağlantısı oluşturulur.

create_connect Metodu:
Bu metod, SQLite veritabanına bağlanmak için kullanılır.
sqlite3.connect fonksiyonu kullanılarak "Odev2.db" adlı SQLite veritabanına bağlantı oluşturulur.
Bağlantı üzerinden bir imleç (**cursor**) oluşturulur.
Daha sonra, **CREATE TABLE IF NOT EXISTS** SQL ifadesi kullanılarak **SHIPS** adında bir tablo oluşturulur.
Tablo sütunları **SERIAL_NO**, **NAME**, **WEIGHT**, **PRODUCT_YEAR**, ve **SHIP_TYPE** olarak tanımlanır.
SERIAL_NO sütunu birincil anahtar (**PRIMARY KEY**) ve eşsiz (**UNIQUE**) olarak belirtilir.
Herhangi bir değişiklik yapıldıktan sonra, **commit** metodu kullanılarak veritabanındaki değişiklikler kaydedilir.

class SHIP:

```
class SHIP:
    def __init__(self, serialNum, name, grossTonnage, madeIn):
        self.serialNum = serialNum
        self.name = name
        self.grossTonnage = grossTonnage
        self.madeIn = madeIn
```

Bu bölüm, bir gemi nesnesinin temel özelliklerini tanımlayan bir sınıf olan SHIP sınıfını tanımlar. Bu sınıf, geminin seri numarası (serialNum), adı (name), brüt tonajı (grossTonnage) ve üretim yeri (madeIn) gibi özelliklerini içerir.

__init__ metodu: Bu metot, bir SHIP nesnesinin başlatılması sırasında çağrılır. Parametre olarak seri numarası, adı, brüt tonajı ve üretim yeri alır. Bu parametreler, sınıfın örneğinin özelliklerini temsil eder. Metot, bu parametreleri kullanarak örneğin özelliklerini başlatır ve atanmış değerlere sahip olmasını sağlar.

class CRUISE_SHIP(SHIP):

```
class CruiseShip(ShipTable):
    def __init__(self, serial_no, name, weight, product_year, ship_type, ship_capacity):
        super().__init__(serial_no, name, weight, product_year, ship_type)
        '''Kalıtım ile alınmayanların alınacak bilgilerin, nesnelerini oluşturulma'''
        self.ship_capacity = ship_capacity

        self.create_connect()

    2 usages
    def create_connect(self):
        '''Tablonun sql kodu ile oluşturulması'''
        self.connect = sqlite3.connect("Odev2.db")     # veri tabanına bağlanma
        self.cursor = self.connect.cursor()

        table = "CREATE TABLE IF NOT EXISTS CRUISE_SHIP(" \
                "SHIP_SERIAL_NO INTEGER PRIMARY KEY UNIQUE," \
                "SHIP_CAPACITY INTEGER NOT NULL, " \
                "FOREIGN KEY (SHIP_SERIAL_NO) REFERENCES SHIPS(SERIAL_NO))"

        self.cursor.execute(table)
        self.connect.commit()                          # veri tabanındaki değişikleri kaydetme
```

Bu bölüm, SHIP sınıfından türetilmiş bir alt sınıf olan CRUISE_SHIP sınıfını tanımlar. CRUISE_SHIP sınıfı, SHIP sınıfının özelliklerine ek

olarak bir kapasite özelliğini içerir.

__init__ metodu: Bu metot, bir CRUISE_SHIP nesnesinin başlatılması sırasında çağrılır. Parametre olarak seri numarası, adı, brüt tonajı, üretim yeri ve kapasite alır. super().__init__(serialNum, name, grossTonnage, madeIn) ifadesi, CRUISE_SHIP sınıfının üst sınıfı olan SHIP sınıfının __init__ metodunu çağırarak, geminin temel özelliklerini başlatır.
Daha sonra self.capacity = capacity ifadesi ile CRUISE_SHIP sınıfının özgün özelliği olan kapasiteyi tanımlarız. self.type = 'CRUISE' ifadesi ise geminin türünü belirtir ve bu durumda bir yolcu gemisi olduğunu gösterir.

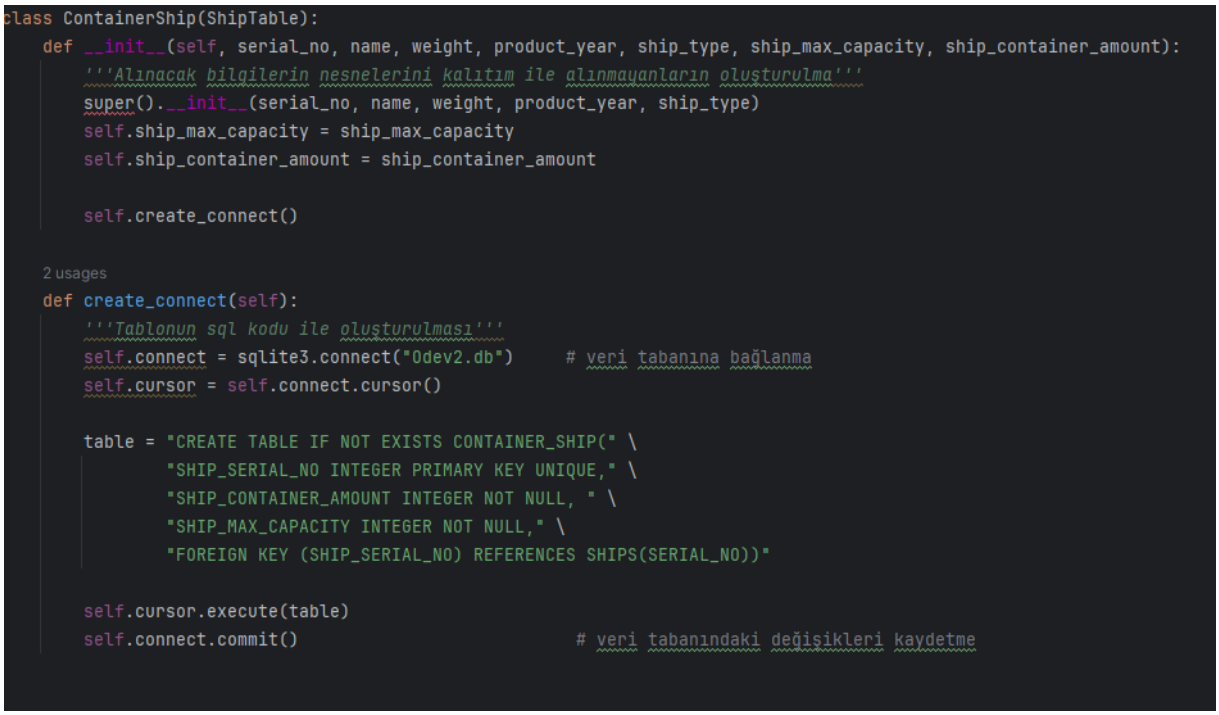
class OIL_SHIP(SHIP):

```
class OilShip(ShipTable):
    def __init__(self, serial_no, name, weight, product_year, ship_type, ship_oil_capacity):
        '''Kalıtım ile alınmayanların alınacak bilgilerin, nesnelerini oluşturulma'''
        super().__init__(serial_no, name, weight, product_year, ship_type)
        self.ship_oil_capacity = ship_oil_capacity

        self.create_connect()

    2 usages
    def create_connect(self):
        '''Tablonun sql kodu ile oluşturulması'''
        self.connect = sqlite3.connect("Odev2.db")     # veri tabanına bağlanma
        self.cursor = self.connect.cursor()

        table = "CREATE TABLE IF NOT EXISTS OIL_SHIP(" \
                "SHIP_SERIAL_NO INTEGER PRIMARY KEY UNIQUE," \
                "SHIP_OIL_CAPACITY INTEGER NOT NULL, " \
                "FOREIGN KEY (SHIP_SERIAL_NO) REFERENCES SHIPS(SERIAL_NO))"

        self.cursor.execute(table)
        self.connect.commit()                          # veri tabanındaki değişikleri kaydetme
```

Bu bölüm, SHIP sınıfından türetilmiş bir alt sınıf olan OIL_SHIP sınıfını tanımlar. OIL_SHIP sınıfı, SHIP sınıfının özelliklerine ek olarak bir petrol kapasitesi özelliğini içerir.

__init__ metodu: Bu metot, bir OIL_SHIP nesnesinin başlatılması sırasında çağrılır. Parametre olarak seri numarası, adı, brüt tonajı, üretim yeri ve petrol kapasitesi alır. super().__init__(serialNum, name, grossTonnage, madeIn) ifadesi, OIL_SHIP sınıfının üst sınıfı olan SHIP sınıfının __init__ metodunu çağırarak, geminin temel özelliklerini başlatır.
Daha sonra self.oilCapacity = oilCapacity ifadesi ile OIL_SHIP sınıfının özgün özelliği olan petrol kapasitesini tanımlarız. self.type = 'OIL' ifadesi ise geminin türünü belirtir ve bu durumda bir petrol gemisi olduğunu gösterir.

class CONTAINER_SHIP(SHIP):



```
class ContainerShip(ShipTable):
    def __init__(self, serial_no, name, weight, product_year, ship_type, ship_max_capacity, ship_container_amount):
        '''Alınacak bilgilerin nesnelerini kalıtım ile alınmayanların oluşturulma'''
        super().__init__(serial_no, name, weight, product_year, ship_type)
        self.ship_max_capacity = ship_max_capacity
        self.ship_container_amount = ship_container_amount

        self.create_connect()

    2 usages
    def create_connect(self):
        '''Tablonun sql kodu ile oluşturulması'''
        self.connect = sqlite3.connect("0dev2.db")     # veri tabanına bağlanma
        self.cursor = self.connect.cursor()

        table = "CREATE TABLE IF NOT EXISTS CONTAINER_SHIP(" \
                "SHIP_SERIAL_NO INTEGER PRIMARY KEY UNIQUE," \
                "SHIP_CONTAINER_AMOUNT INTEGER NOT NULL, " \
                "SHIP_MAX_CAPACITY INTEGER NOT NULL," \
                "FOREIGN KEY (SHIP_SERIAL_NO) REFERENCES SHIPS(SERIAL_NO))"

        self.cursor.execute(table)
        self.connect.commit()                          # veri tabanındaki değişikleri kaydetme
```

Bu bölüm, SHIP sınıfından türetilmiş bir alt sınıf olan CONTAINER_SHIP sınıfını tanımlar. CONTAINER_SHIP sınıfı, SHIP sınıfının özelliklerine ek olarak bir konteyner kapasitesi ve maksimum kapasite özelliklerini içerir.

__init__ metodu: Bu metot, bir CONTAINER_SHIP nesnesinin başlatılması sırasında çağrılır. Parametre olarak seri numarası, adı, brüt tonajı, üretim yeri, konteyner kapasitesi ve maksimum kapasite alır. super().__init__(serialNum, name, grossTonnage, madeIn) ifadesi, CONTAINER_SHIP sınıfının üst sınıfı olan SHIP sınıfının __init__ metodunu çağırarak, geminin temel özelliklerini başlatır.
Daha sonra self.containerCapacity = containerCapacity ifadesi ile CONTAINER_SHIP sınıfının özgün özelliği olan konteyner kapasitesini tanımlarız. self.maxCapacity = maxCapacity ifadesi ile de maksimum kapasiteyi tanımlarız. self.type = 'CONTAINER' ifadesi ise geminin türünü belirtir ve bu durumda bir konteyner gemisi olduğunu gösterir.

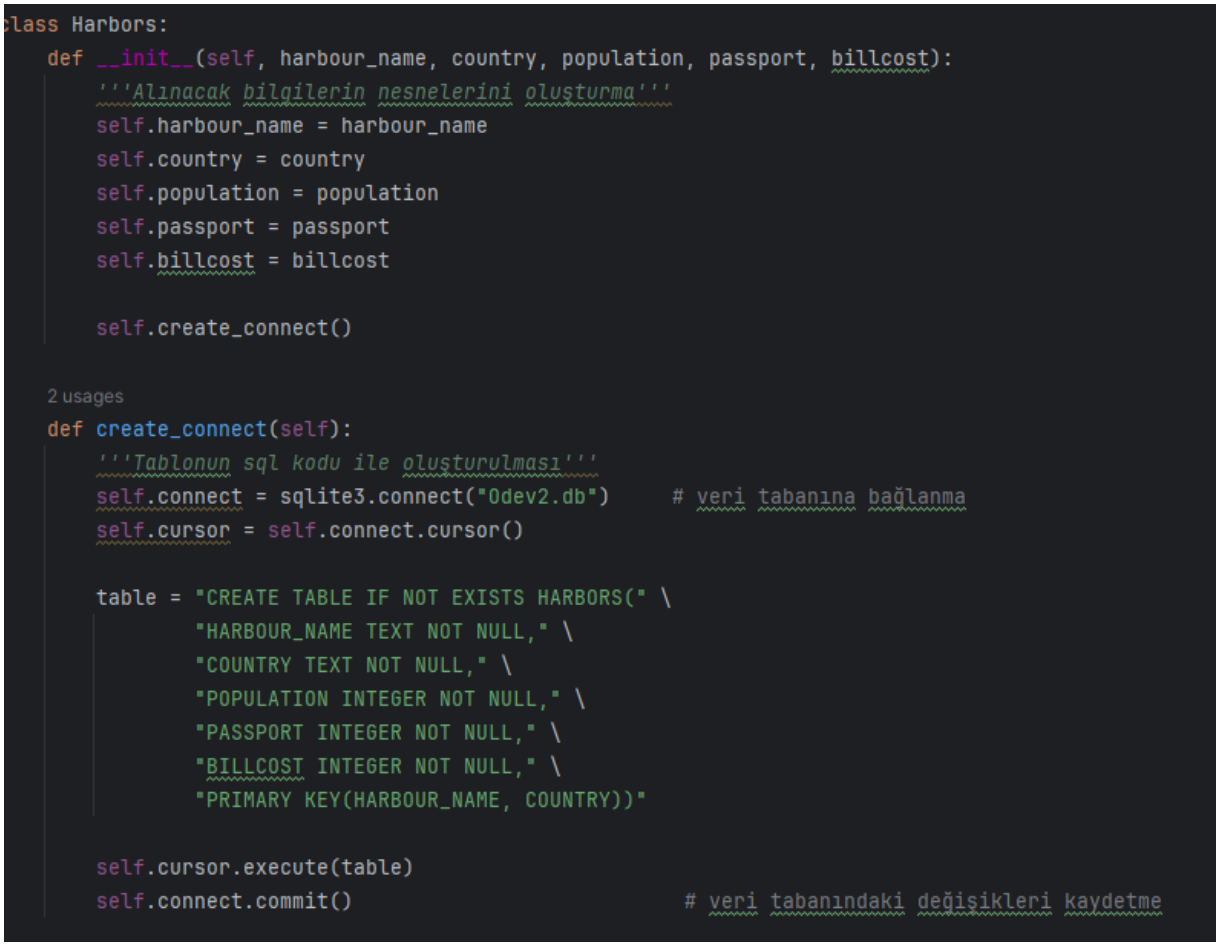
class VOYAGETABLE:

```
class VoyageTable:
    def __init__(self, ID, takeoff_date, return_date, takeoff_place, serial_no):
        '''Alınacak bilgilerin nesnelerini oluşturma'''
        self.ID = ID
        self.takeoff_date = takeoff_date
        self.return_date = return_date
        self.takeoff_place = takeoff_place
        self.serial_no = serial_no

        self.create_connect()

    2 usages
    def create_connect(self):
        '''Tablonun sql kodu ile oluşturulması'''
        self.connect = sqlite3.connect("Odev2.db")     # veri tabanına bağlanma
        self.cursor = self.connect.cursor()

        table = "CREATE TABLE IF NOT EXISTS VOYAGES(" \
                "ID INTEGER UNIQUE," \
                "TAKEOFF_DATE TEXT NOT NULL," \
                "RETURN_DATE TEXT NOT NULL," \
                "TAKEOFF_PLACE TEXT NOT NULL," \
                "SERIAL_NO INTEGER NOT NULL," \
                "CAP INTEGER NOT NULL," \
                "CREW INTEGER NOT NULL," \
                "FOREIGN KEY(SERIAL_NO) REFERENCES SHIPS(SERIAL_NO)," \
                "CONSTRAINT DATE_CHECK UNIQUE(SERIAL_NO, TAKEOFF_DATE, TAKEOFF_PLACE)," \
                "CONSTRAINT COUNT_CHECK CHECK(CAP >= 2), CHECK(CREW >= 1)," \
                "PRIMARY KEY (ID))"

        self.cursor.execute(table)
        self.connect.commit()                          # veri tabanındaki değişikleri kaydetme
```

- **__init__** metodu: Bu metod, **VoyageTable** sınıfının örneklerinin oluşturulmasını sağlar. Gerekli bilgileri (**ID**, **takeoff_date**, **return_date**, **takeoff_place**, **serial_no**) alır ve bu bilgileri ilgili örnek değişkenlerine atar. Ayrıca, **create_connect()** metodunu çağırarak veritabanına bağlantı oluşturur.
- **create_connect** metodu: Bu metod, SQLite veritabanına bağlanmak için kullanılır. İlk olarak, **sqlite3.connect()** fonksiyonuyla "Odev2.db" adlı veritabanına bir bağlantı oluşturulur. Sonra, bağlantı üzerinde bir imleç (cursor) oluşturulur. Daha sonra, **VOYAGES** adında bir tablo oluşturulur ya da varsa tablo mevcut haliyle korunur (**CREATE TABLE IF NOT EXISTS** ifadesi ile). Tablo şu sütunlardan oluşur: **ID**, **TAKEOFF_DATE**, **RETURN_DATE**, **TAKEOFF_PLACE**, **SERIAL_NO**, **CAP**, **CREW**. Ayrıca, **SERIAL_NO** sütunu, **SHIPS** tablosundaki **SERIAL_NO** sütunuyla ilişkilendirilmiş bir dış anahtar (foreign key) olarak tanımlanır.

DATE_CHECK adında bir benzersizlik kısıtlaması belirtilir. **CAP** sütunu için minimum değer kontrolü yapılır (**CHECK** ifadesi ile). Son olarak, tablonun ID sütunu birincil anahtar (primary key) olarak belirlenir (**PRIMARY KEY (ID)** ifadesi ile).

class HARBOR:



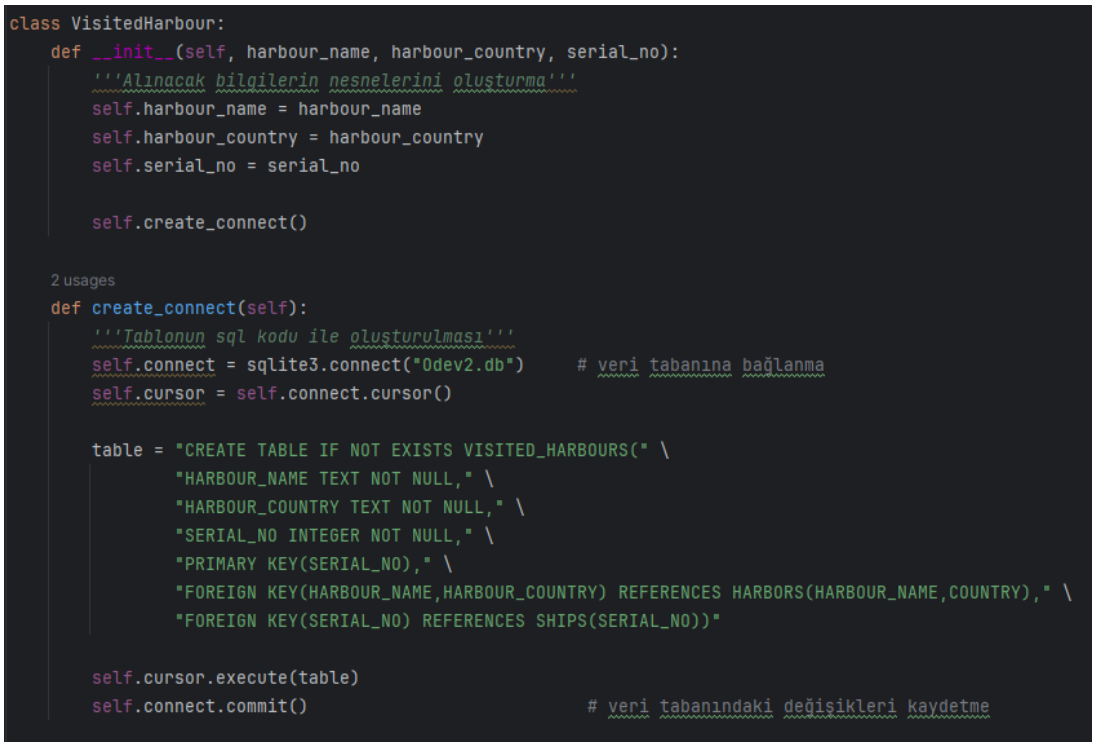
```
class Harbors:
    def __init__(self, harbour_name, country, population, passport, billcost):
        '''Alınacak bilgilerin nesnelerini oluşturma'''
        self.harbour_name = harbour_name
        self.country = country
        self.population = population
        self.passport = passport
        self.billcost = billcost

        self.create_connect()

    2 usages
    def create_connect(self):
        '''Tablonun sql kodu ile oluşturulması'''
        self.connect = sqlite3.connect("Odev2.db")     # veri tabanına bağlanma
        self.cursor = self.connect.cursor()

        table = "CREATE TABLE IF NOT EXISTS HARBORS(" \
                "HARBOUR_NAME TEXT NOT NULL," \
                "COUNTRY TEXT NOT NULL," \
                "POPULATION INTEGER NOT NULL," \
                "PASSPORT INTEGER NOT NULL," \
                "BILLCOST INTEGER NOT NULL," \
                "PRIMARY KEY(HARBOUR_NAME, COUNTRY))"

        self.cursor.execute(table)
        self.connect.commit()                          # veri tabanındaki değişikleri kaydetme
```

Bu bölüm, limanların özelliklerini tanımlayan bir sınıf olan HARBOR sınıfını tanımlar.

_ Harbors sınıfı: Bu sınıf, limanların özelliklerini tanımlar. harbour_name, country, population, passport, ve billcost gibi liman özelliklerini içerir.
__init__ metodu: Bu metod, bir Harbors nesnesi oluşturulduğunda çağrılır. Liman özelliklerini (adı, ülke adı, nüfusu, pasaport gereksinimi ve fatura maliyeti) alır ve ilgili sınıf özelliklerine atar. Daha sonra create_connect() metodunu çağırarak veritabanına bağlanır.
create_connect metodu: Bu metod, SQLite veritabanına bağlanmayı sağlar. İlk olarak, sqlite3.connect() fonksiyonuyla "Odev2.db" adlı veritabanına bir bağlantı oluşturulur. Daha sonra, bağlantı üzerinde bir imleç (cursor) oluşturulur. Ardından, HARBORS adında bir tablo oluşturulur ya da varsa

tablo mevcut haliyle korunur (CREATE TABLE IF NOT EXISTS ifadesi ile). Tablo şu sütunlardan oluşur: HARBOUR_NAME, COUNTRY, POPULATION, PASSPORT, ve BILLCOST. HARBOUR_NAME ve COUNTRY sütunları birincil anahtar (primary key) olarak belirlenir (PRIMARY KEY(HARBOUR_NAME, COUNTRY) ifadesi ile). Bu, bir limanın adı ve ülkesi kombinasyonunun benzersiz olmasını sağlar.

class VISITED_HARBORS:



```
class VisitedHarbour:
    def __init__(self, harbour_name, harbour_country, serial_no):
        '''Alınacak bilgilerin nesnelerini oluşturma'''
        self.harbour_name = harbour_name
        self.harbour_country = harbour_country
        self.serial_no = serial_no

        self.create_connect()

    2 usages
    def create_connect(self):
        '''Tablonun sql kodu ile oluşturulması'''
        self.connect = sqlite3.connect("Odev2.db")     # veri tabanına bağlanma
        self.cursor = self.connect.cursor()

        table = "CREATE TABLE IF NOT EXISTS VISITED_HARBOURS(" \
                "HARBOUR_NAME TEXT NOT NULL," \
                "HARBOUR_COUNTRY TEXT NOT NULL," \
                "SERIAL_NO INTEGER NOT NULL," \
                "PRIMARY KEY(SERIAL_NO)," \
                "FOREIGN KEY(HARBOUR_NAME,HARBOUR_COUNTRY) REFERENCES HARBORS(HARBOUR_NAME,COUNTRY)," \
                "FOREIGN KEY(SERIAL_NO) REFERENCES SHIPS(SERIAL_NO))"

        self.cursor.execute(table)
        self.connect.commit()                          # veri tabanındaki değişikleri kaydetme
```

Bu bölüm, ziyaret edilen limanların özelliklerini tanımlayan bir sınıf olan VISITED_HARBORS sınıfını tanımlar.

VisitedHarbour sınıfı: Bu sınıf, gemilerin ziyaret ettiği limanların bilgilerini saklar. harbour_name, harbour_country, ve serial_no gibi özellikleri içerir.
__init__ metodu: Bu metod, bir VisitedHarbour nesnesi oluşturulduğunda çağrılır. Liman özelliklerini (adı, ülke adı ve seri numarası) alır ve ilgili sınıf özelliklerine atar. Daha sonra create_connect() metodunu çağırarak veritabanına bağlanır.
create_connect metodu: Bu metod, SQLite veritabanına bağlanmayı sağlar. İlk olarak, sqlite3.connect() fonksiyonuyla "Odev2.db" adlı veritabanına bir bağlantı oluşturulur. Daha sonra, bağlantı üzerinde bir imleç (cursor) oluşturulur. Ardından, VISITED_HARBOURS adında bir tablo oluşturulur ya da varsa tablo mevcut haliyle korunur (CREATE TABLE IF NOT EXISTS ifadesi ile). Tablo şu sütunlardan oluşur: HARBOUR_NAME,

HARBOUR_COUNTRY, ve SERIAL_NO. SERIAL_NO sütunu birincil anahtar (primary key) olarak belirlenir (PRIMARY KEY(SERIAL_NO) ifadesi ile). Ayrıca, HARBOUR_NAME ve HARBOUR_COUNTRY sütunları, HARBORS tablosundaki HARBOUR_NAME ve COUNTRY sütunlarına dış anahtar (foreign key) olarak atıfta bulunur (FOREIGN KEY(HARBOUR_NAME,HARBOUR_COUNTRY) REFERENCES HARBORS(HARBOUR_NAME,COUNTRY) ifadesi ile). SERIAL_NO sütunu da SHIPS tablosundaki SERIAL_NO sütununa dış anahtar olarak atıfta bulunur (FOREIGN KEY(SERIAL_NO) REFERENCES SHIPS(SERIAL_NO) ifadesi ile).

class CAPTAINS:

```
class Captains:
    def __init__(self, ID, name, surname, address, citizenship, birth_date, date_of_recruitment, license):
        '''Alınacak bilgilerin nesnelerini oluşturma'''
        self.ID = ID
        self.name = name
        self.surname = surname
        self.address = address
        self.citizenship = citizenship
        self.birth_date = birth_date
        self.date_of_recruitment = date_of_recruitment
        self.license = license

        self.create_connect()

    2 usages
    def create_connect(self):
        '''Tablonun sql kodu ile oluşturulması'''
        self.connect = sqlite3.connect("Odev2.db")     # veri tabanına bağlanma
        self.cursor = self.connect.cursor()

        table = "CREATE TABLE IF NOT EXISTS CAPTAINS(" \
                "ID INTEGER PRIMARY KEY," \
                "NAME TEXT NULL," \
                "SURNAME TEXT NULL," \
                "ADDRESS TEXT NULL," \
                "CITIZENSHIP TEXT NULL," \
                "BIRTH_DATE TEXT NULL," \
                "DATE_OF_RECRUITMENT TEXT NULL," \
                "LICENSE TEXT)"

        self.cursor.execute(table)
        self.connect.commit()                          # veri tabanındaki değişikleri kaydetme
```

Bu bölüm, gemi kaptanlarını temsil eden CAPTAINS sınıfını tanımlar.

- **Captains** sınıfı: Bu sınıf, kaptanların özelliklerini tanımlar. **ID**, **name**, **surname**, **address**, **citizenship**, **birth_date**, **date_of_recruitment** ve **license** gibi kaptan özelliklerini içerir.
- **__init__** metodu: Bu metod, bir **Captains** nesnesi oluşturulduğunda çağrılır. Kaptan özelliklerini (kimlik numarası, adı, soyadı, adresi,

vatandaşlığı, doğum tarihi, işe alım tarihi ve lisansı) alır ve ilgili sınıf özelliklerine atar. Daha sonra **create_connect()** metodunu çağırarak veritabanına bağlanır.

- **create_connect** metodu: Bu metod, SQLite veritabanına bağlanmayı sağlar. İlk olarak, **sqlite3.connect()** fonksiyonuyla "Odev2.db" adlı veritabanına bir bağlantı oluşturulur. Daha sonra, bağlantı üzerinde bir imleç (cursor) oluşturulur. Ardından, **CAPTAINS** adında bir tablo oluşturulur ya da varsa tablo mevcut haliyle korunur (**CREATE TABLE IF NOT EXISTS** ifadesi ile). Tablo şu sütunlardan oluşur: **ID**, **NAME**, **SURNAME**, **ADDRESS**, **CITIZENSHIP**, **BIRTH_DATE**, **DATE_OF_RECRUITMENT**, ve **LICENSE**. **ID** sütunu birincil anahtar (primary key) olarak belirlenir (**PRIMARY KEY(ID)** ifadesi ile).

class CREW(CAPTAINS):

```
class CrewMembers(Captains):
    def __init__(self, ID, name, surname, address, citizenship, birth_date, date_of_recruitment, mission, license):
        '''Alınacak bilgilerin nesnelerini oluşturma'''
        super().__init__(ID, name, surname, address, citizenship, birth_date, date_of_recruitment,license)
        self.mission = mission

        self.create_connect()

    2 usages
    def create_connect(self):
        '''Tablonun sql kodu ile oluşturulması'''
        self.connect = sqlite3.connect("Odev2.db")     # veri tabanına bağlanma
        self.cursor = self.connect.cursor()

        table = "CREATE TABLE IF NOT EXISTS CREW_MEMBERS(" \
                "ID INTEGER PRIMARY KEY," \
                "NAME TEXT NULL," \
                "SURNAME TEXT NULL," \
                "ADDRESS TEXT NULL," \
                "CITIZENSHIP TEXT NULL," \
                "BIRTH_DATE TEXT NULL," \
                "DATE_OF_RECRUITMENT TEXT NULL," \
                "MISSION TEXT)"

        self.cursor.execute(table)
        self.connect.commit()                          # veri tabanındaki değişikleri kaydetme
```

Bu bölüm, gemi mürettebatını temsil eden CREW sınıfını tanımlar. CREW sınıfı, CAPTAINS sınıfından miras alır ve çalışanların özelliklerine ek olarak mürettebatın özelliklerini tanımlar.

- **CrewMembers** sınıfı: Bu sınıf, mürettebat üyelerinin özelliklerini tanımlar. **ID**, **name**, **surname**, **address**, **citizenship**, **birth_date**, **date_of_recruitment**, **mission**, ve **license** gibi mürettebat üyesi özelliklerini içerir. **Captains** sınıfından kalıtım almış gibi görünüyor, ancak **Captains** sınıfının tanımı burada verilmemiş.
- **__init__** metodu: Bu metod, bir **CrewMembers** nesnesi

oluşturulduğunda çağrılır. Gerekli bilgileri alır ve ilgili sınıf özelliklerine atar. **super().__init__()** çağrısıyla üst sınıfın yapıcı metodunu çağırarak, **Captains** sınıfının yapıcı metodunun çalışmasını sağlar. Daha sonra **create_connect()** metodunu çağırarak veritabanına bağlanır.

- **create_connect** metodu: Bu metod, SQLite veritabanına bağlanmayı sağlar. İlk olarak, **sqlite3.connect()** fonksiyonuyla "Odev2.db" adlı veritabanına bir bağlantı oluşturulur. Daha sonra, bağlantı üzerinde bir imleç (cursor) oluşturulur. Ardından, **CREW_MEMBERS** adında bir tablo oluşturulur ya da varsa tablo mevcut haliyle korunur (**CREATE TABLE IF NOT EXISTS** ifadesi ile). Tablo şu sütunlardan oluşur: **ID**, **NAME**, **SURNAME**, **ADDRESS**, **CITIZENSHIP**, **BIRTH_DATE**, **DATE_OF_RECRUITMENT**, ve **MISSION**. **ID** sütunu birincil anahtar (primary key) olarak belirlenir (**PRIMARY KEY(ID)** ifadesi ile).

4.2 Görev dağılımı

- Projenin planlaması birlikte yapılmış olup kod yazma aşamasında SQL kodlarını Muhammed Yusuf Kaya, arayüzü Talha Tuna yazmıştır. Geriye kalan fonksiyon ve sınıflar birlikte yazılmıştır. Rapor aşamasında Giriş, Gereksinim Analizi, Test ve Doğrulama başlıklarını Muhammed Yusuf Kaya hazırlamıştır. Tasarım, Uygulama başlıklarını da Talha Tuna hazırlamıştır.

4.3 Karşılaşılan zorluklar ve çözüm yöntemleri

Tablolar Arasındaki İlişkilerin Kurulması:

- Örneğin, gemilerin seferlerine ait bilgileri depolayan "VOYAGES" tablosu ile gemi bilgilerini depolayan "SHIPS" tablosu arasında ilişki kurulurken zorluklar yaşanmıştır. Bu ilişki, "VOYAGES" tablosundaki "SERIAL_NO" sütunu ile "SHIPS" tablosundaki "SERIAL_NO" sütunu arasında tanımlanmaktadır. Ancak, uyumsuz veri tipleri veya eksik veri bütünlüğü kısıtlamaları gibi durumlar yaşandı.

Fonksiyonların ve Sınıfların Arayüze Eklenmesi:

- Veri tabanı işlemlerini gerçekleştiren fonksiyonlar veya sınıfların arayüze eklenmesi sırasında uyumsuzluklar veya hata mesajlarıyla karşılaşıldı. Özellikle, veri işleme ve güvenlik kontrolleri gibi karmaşık işlevlerin doğru bir şekilde arayüze entegre edilmesi zaman aldı.

Arayüz Tasarımı Değişiklikleri:

- Optimum kullanıcı odaklı bir arayüz hazırlama sürecinde, kullanıcı geri bildirimleri veya gereksinimlerindeki değişiklikler nedeniyle arayüz tasarımında birkaç kez değişiklik yapılması gerekti. Bu durum, geliştirme sürecini uzattı.

4.4 Proje isterlerine göre eksik yönler

- Bizim açımızdan projenin eksik yönü varsa farkedilmemiştir.

5 TEST VE DOĞRULAMA

5.1 Yazılımın test süreci

Arayüz hataları giderildi arayüzlü bir program olduğu için test kodu yerine kullanıcı girdileri ile test edilmiştir.

5.2 Yazılımın doğrulanması

Proje istenilen şekilde yapılmaya çalışılmıştır son durumda herhangi bir hataya rastlanmamıştır

Github Link

https://github.com/yusufky56

https://github.com/TalhaTuna2