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

VERİ TABANI YÖNETİM SİSTEMLERİ PROJESİ

Yavuz Selim Gürsoy
220501003

Emrah Şahin
220502025

DERS SORUMLUSU:
PROF. DR. NEVCİHAN DURU

TARİH:
05/05/2024

1 GİRİŞ

1.1 Projenin amacı

- Bu ödevin amacı, bir gemi şirketinin seferlerini ve gemi operasyonlarını yönetmek için bir yazılım geliştirmektir. Yazılım, gemiler, seferler, limanlar, kaptanlar ve mürettebat gibi çeşitli veri türlerini yönetmek için bir veri tabanı ile entegre olmalıdır. Şirketin gemi envanterini, sefer kayıtlarını, liman bilgilerini, kaptan ve mürettebat bilgilerini takip etmek ve bu verilere erişmek için kullanılabilir bir arayüz sağlamalıdır. Ayrıca, veri tabanındaki verilerin eklenmesi, silinmesi, düzenlenmesi gibi temel işlemleri gerçekleştirebilmelidir. Bu sayede şirket, gemilerin sefere çıkması, limanlara uğraması ve personelin yönetimi gibi işlemleri daha etkin bir şekilde gerçekleştirebilir.

2 GEREKSİNİM ANALİZİ

2.1 Arayüz gereksinimleri

Gemi Bilgileri Ekranı:

- Yeni gemi eklemek için form.
- Var olan gemileri listeleyen bir görünüm.
- Gemi detaylarını gösteren bir görünüm, burada gemi bilgileri ve özellikleri bulunmalıdır.

Sefer Bilgileri Ekranı:

- Yeni bir sefer oluşturmak için form.
- Var olan seferleri listeleyen bir görünüm.
- Sefer detaylarını gösteren bir görünüm, burada seferin tarihleri, liman bilgileri ve gemi detayları yer almalıdır.

Kaptan ve Mürettebat Yönetimi Ekranı:

- Yeni kaptan ve mürettebat eklemek için form.
- Var olan kaptan ve mürettebatı listeleyen bir görünüm.
- Kaptan ve mürettebat detaylarını gösteren bir görünüm, burada kişisel bilgiler ve görevleri bulunmalıdır.

Liman Bilgileri Ekranı:

- Yeni liman eklemek için form.
- Var olan limanları listeleyen bir görünüm.
- Liman detaylarını gösteren bir görünüm, burada limanın ülkesi, nüfusu, pasaport gereksinimi ve demirleme ücreti yer almalıdır.

Veri Yönetimi İşlemleri:

- Veri eklemek, düzenlemek ve silmek için butonlar veya menüler.
- Veri tabanı işlemlerini gerçekleştiren arka planda çalışan fonksiyonlar.

İşlem Bildirimleri ve Onayları:

- Kullanıcıya veri eklenmesi, düzenlenmesi veya silinmesi gibi işlemlerin başarılı veya başarısız olduğuna dair bildirimler veya onaylar.

- Varsa donanım arayüzü gereksinimlerinin maddeler halinde açıklanması

2.2 Fonksiyonel gereksinimler

Gemi Yönetimi:

- Yolcu gemileri, petrol tankerleri ve konteyner gemileri olmak üzere farklı tiplerde gemilerin eklenmesi, silinmesi, düzenlenmesi.
- Her gemi için seri numarası, adı, ağırlığı, yapım yılı gibi bilgilerin tutulması.
- Yolcu gemileri için yolcu kapasitesi, petrol tankerleri için petrol kapasitesi (litre), konteyner gemileri için konteyner sayısı kapasitesi ve maksimum ağırlık gibi ek bilgilerin saklanması.

Sefer Yönetimi:

- Her sefer için bir geminin atanması ve bu gemi için en az 2 kaptan ve 1 mürettebatın atanması.
- Sefer kayıtları için ID, yola çıkış tarihi, dönüş tarihi, yola çıkış limanı gibi bilgilerin saklanması.
- Geçmişte yapılmış, gelecekte planlanan ve olası seferlerin bilgilerinin tutulması.

Liman Yönetimi:

- Limanların eklenmesi, silinmesi, düzenlenmesi.
- Her liman için liman adı, ülkesi, nüfusu, pasaport isteyip istemediği, demirleme ücreti gibi bilgilerin saklanması.
- Limanın adı ve ülkesinin benzersiz olması gerekliliği.
- Uğranmamış fakat uğranma ihtimali olan limanların kayıtlarının yapılması.

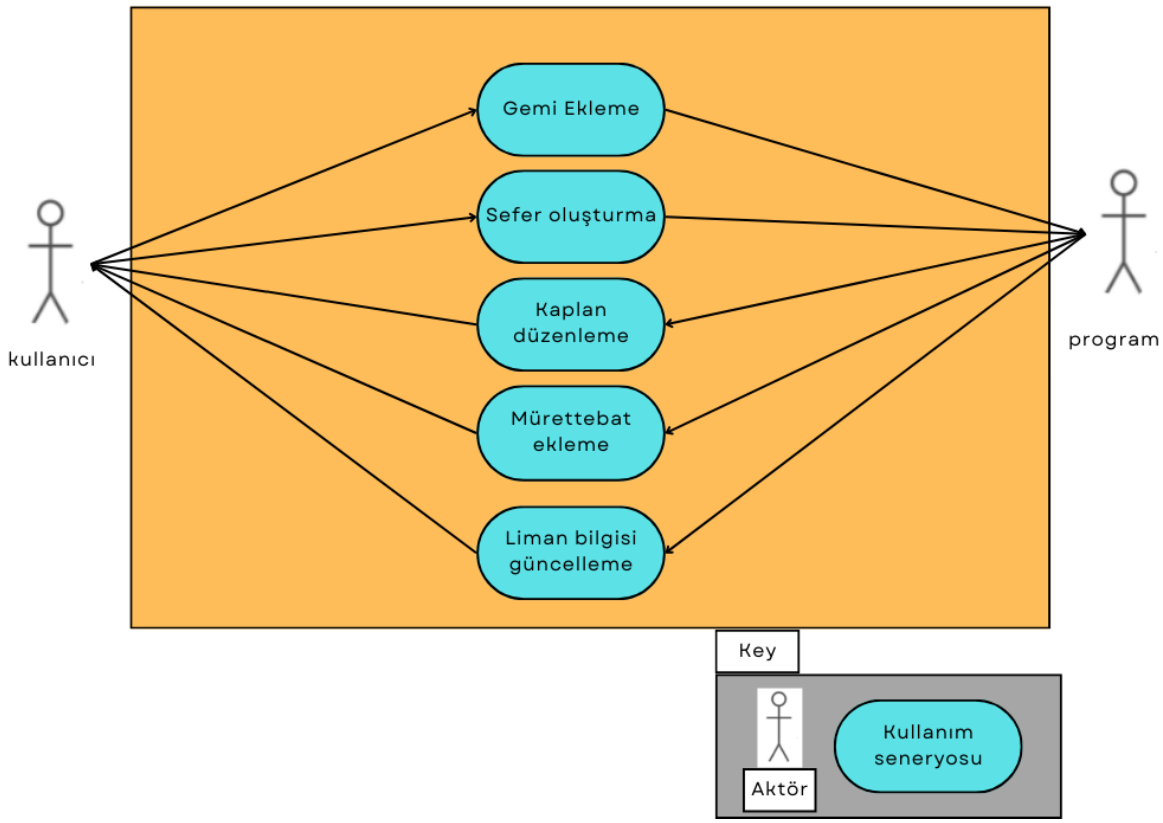
Personel Yönetimi:

- Kaptan ve mürettebat bilgilerinin eklenmesi, silinmesi, düzenlenmesi.
- Her personel için ID, ad, soyad, adres, vatandaşlık, doğum tarihi, işe giriş tarihi gibi bilgilerin tutulması.
- Kaptanların lisansları ve mürettebatın görevleri gibi ek bilgilerin saklanması.
- Kaptan ve mürettebatın aynı anda sadece bir seferde bulunabilmesi kuralının uygulanması.

Veri Yönetimi:

- Veri tabanında gerekli olan her bir varlık için tabloların oluşturulması.
- Verilerin doğru bir şekilde tutulması ve güncellenmesi.
- Veri tabanı ile ilgili tüm işlemlerin yazılım tarafından gerçekleştirilmesi, nesneler aracılığıyla yönetilmesi.

2.3 Use-Case diyagramı



3 TASARIM

3.1 Mimari tasarım

Veri Tabanı Tasarımı:

- Şirketin ihtiyacı olan verilerin tutulması için uygun bir veri tabanı oluşturulmalıdır. Bu veri tabanı SQL Server gibi ilişkisel bir veri tabanı yönetim sistemi kullanılarak oluşturulabilir.
- Veri tabanı tabloları, gemiler, seferler, limanlar, kaptanlar, ve mürettebat gibi ana varlık türlerine karşılık gelmelidir.
- Her tablo, ilgili varlık türünün özelliklerini içermelidir. Örneğin, gemi tablosu gemi seri numarası, adı, ağırlığı gibi bilgileri içermelidir.

Sınıf Tasarımı:

- Her varlık türü için bir sınıf oluşturulmalıdır. Bu sınıflar, ilgili varlığın özelliklerini ve o varlıkla ilgili işlemleri içermelidir.
- Örneğin, Gemi sınıfı geminin özelliklerini ve gemi ile ilgili işlemleri içermelidir, Sefer sınıfı seferin özelliklerini ve sefer ile ilgili işlemleri içermelidir.

Nesne Yönetimi:

- Veri tabanından alınan bilgiler, ilgili sınıfların nesnelerine dönüştürülmelidir. Bu, veri tabanı işlemlerinin ve sınıfların bir araya getirilmesini sağlar.
- Örneğin, bir gemi veri tabanından alındığında, bu bilgiler Gemi sınıfının bir nesnesine dönüştürülmelidir.

Form Ekranları Tasarımı:

- Kullanıcıların verileri ekleyebileceği, silebileceği, düzenleyebileceği ve görüntüleyebileceği form ekranları oluşturulmalıdır.
- Her varlık türü için uygun bir form ekranı oluşturulmalıdır. Örneğin, gemi ekranı gemilerle ilgili verilerin görüntülenmesini ve düzenlenmesini sağlamalıdır.

Veri Tabanı ve Form Ekranı Arasındaki Yönetim:

- Veri tabanından alınan veya veri tabanına yazılan bilgiler, form ekranları aracılığıyla yönetilmelidir.
- Kullanıcılar form ekranları üzerinden veri girişi yapmalı ve bu veriler veri tabanına aktarılmalıdır. Benzer şekilde, veri tabanındaki veriler form ekranlarına aktarılmalı ve kullanıcılara görüntülenmelidir.

İş Mantığı ve Kural Kontrolleri:

- İş mantığı ve kural kontrolleri, gerekli doğrulamaların yapılmasını sağlamalıdır. Örneğin, bir sefer oluşturulurken en az 2 kaptan ve 1 mürettebatın atanmış olması gerekmektedir. Bu tür kuralların uygulanması gereklidir.

Güvenlik:

- Veri tabanı erişimi ve form ekranlarına erişim yetkilendirilmelidir. Kullanıcıların yalnızca belirli işlemleri gerçekleştirebilecekleri yetkilendirilmelidir. Örneğin, sadece yönetici rolleri sefer oluşturma veya silme işlemlerini gerçekleştirebilir.

3.2 Kullanılacak teknolojiler

Bu kod Python dili kullanılarak yazılmıştır

```
import tkinter as tk
import sqlite3 as sql
import random as rnd
import runpy
import os
```

tkinter: Tkinter, Python'un standart kitaplıklarından biridir ve GUI (Graphical User Interface - Grafiksel Kullanıcı Arayüzü) uygulamaları oluşturmak için kullanılır. Tkinter, Tk GUI toolkit'in Python sürümüdür ve kullanımı oldukça yaygındır. Tkinter ile pencere oluşturabilir, düğmeler, metin kutuları, menüler gibi bileşenler ekleyebilir ve kullanıcı arayüzü etkileşimlerini yönetebilirsiniz.

sqlite3: SQLite, küçükten orta ölçekli uygulamalarda yerel veri tabanı olarak kullanılan bir hafif SQL veri tabanı motorudur. Python'da SQLite veri tabanı işlemleri için kullanılan standart kütüphanedir. SQLite, bir sunucu gerektirmeden yerel bir dosya üzerinde SQL sorguları çalıştırarak veri tabanı işlemlerini gerçekleştirmenizi sağlar.

random: Random modülü, rastgele sayı üretmek ve diğer rastgele seçim işlemlerini gerçekleştirmek için kullanılır. Örneğin, belirli bir aralıkta rastgele bir sayı seçmek veya bir liste öğesini rastgele seçmek gibi işlemler yapabilirsiniz.

runpy: Runpy modülü, Python betiklerini çalıştırmak için kullanılır. Bu, bir Python betiğini bir modül olarak yüklemek ve çalıştırmak için kullanışlı bir araçtır.

os: Os modülü, işletim sistemi işlevleriyle etkileşimde bulunmak için kullanılır. Bu modül, dosya sistemi işlemleri yapmak, ortam değişkenlerine erişmek, dizinleri değiştirmek, dosya adlarını değiştirmek gibi işlemleri

gerçekleştirmenizi sağlar.

3.3 Veri tabanı tasarımı

Gemi Sınıfları:
SHIP, CRUISE_SHIP, OIL_SHIP, CONTAINER_SHIP gibi gemi türlerini temsil eder. Her gemi sınıfı, gemiye ait özellikleri depolar.

Yolculuk Sınıfı: EXPEDITIONS, bir geminin belirli bir süre için belirli bir limana yapacağı yolculuğu temsil eder. Bu sınıf, yolculukla ilgili bilgileri depolar.

Liman Sınıfı: HARBOR, limanların özelliklerini depolar.
Ziyaret Edilen Limanlar Sınıfı: VISITED_HARBORS, bir geminin ziyaret ettiği limanları ve ziyaret durumunu depolar.

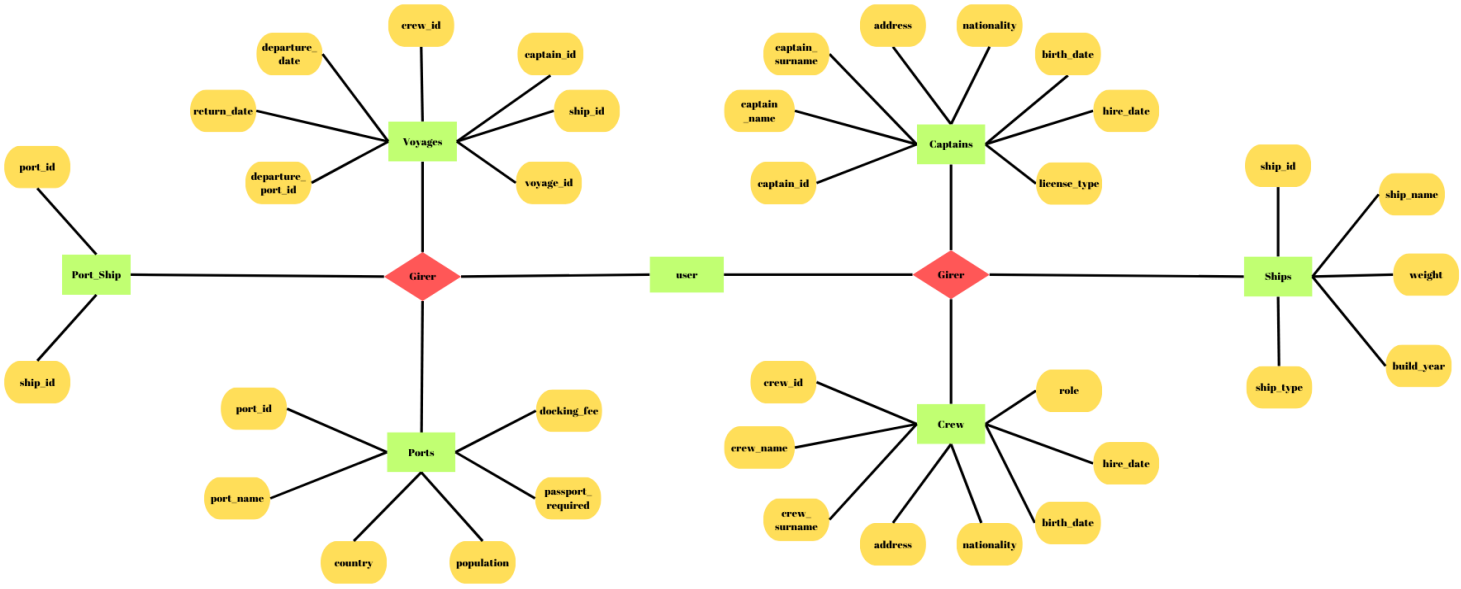
Çalışan Sınıfları: EMPLOYEE, CAPTAIN, CREW gibi gemi personelini temsil eder. Her bir sınıf, personelin özelliklerini depolar.

Ana Menü ve İkincil Menü Oluşturma: _construct_main_menu() ve construct_menu() fonksiyonları, kullanıcıya ana menü ve ilgili ikincil menüleri gösterir. Kullanıcı bu menüler aracılığıyla veri tabanına erişebilir ve işlemler yapabilir.

Veri tabanı İşlemleri: insert_sample(), delete_sample(), update_sample(), display_table() gibi fonksiyonlar, veri tabanında veri eklemek, silmek, güncellemek ve görüntülemek için kullanılır. Örneğin, bir gemi eklemek için insert_sample() fonksiyonu kullanılabilir.

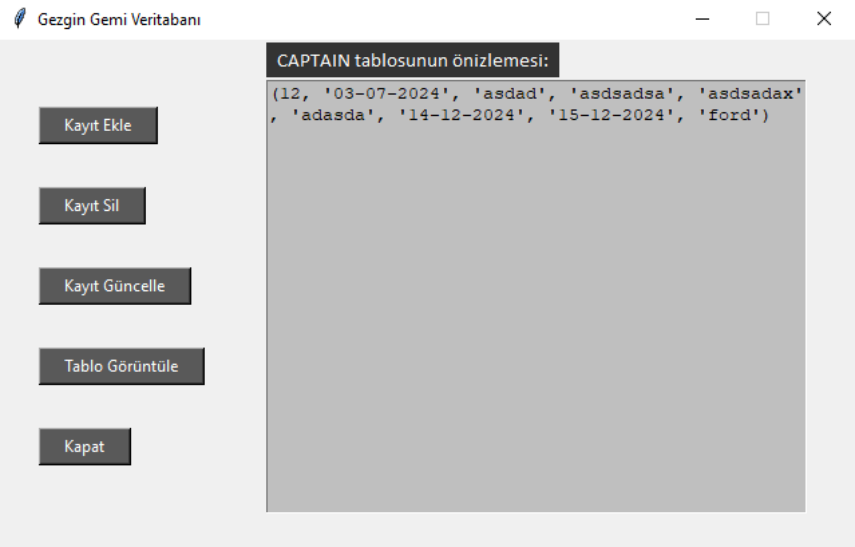
Veri Tabanı Bağlantısı: sqlite3 modülü kullanılarak gezginGemi.db adında bir SQLite veri tabanına bağlanır.

ER Diyagramı:

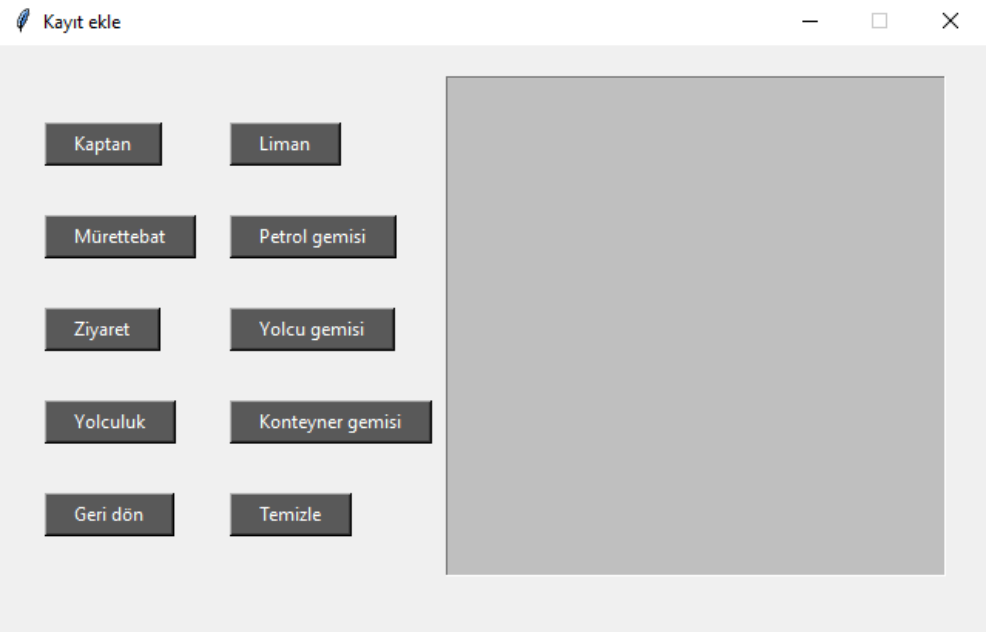


3.4 Kullanıcı arayüzü tasarımı

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

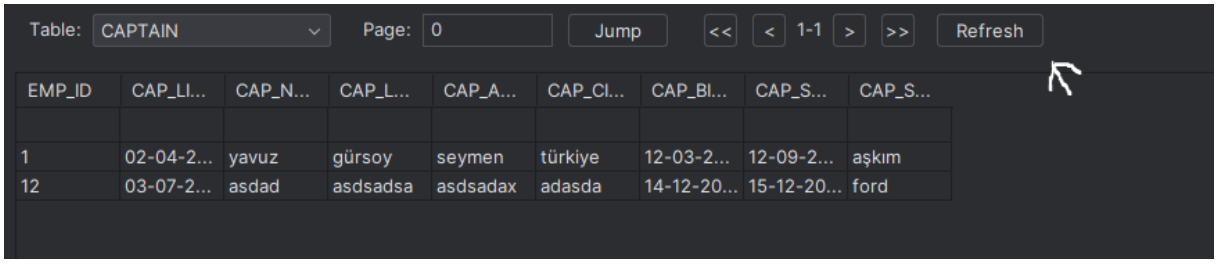


Bu şekilde bir pencere açılıyor. Pencerede kayıt ekle, kayıt sil, kayıt güncelle, tablo görüntüle, kapat butonları bulunuyor. Bu butonlardan herhangi birine tıklayınca başka bir pencere açılıyor. (kapat hariç)



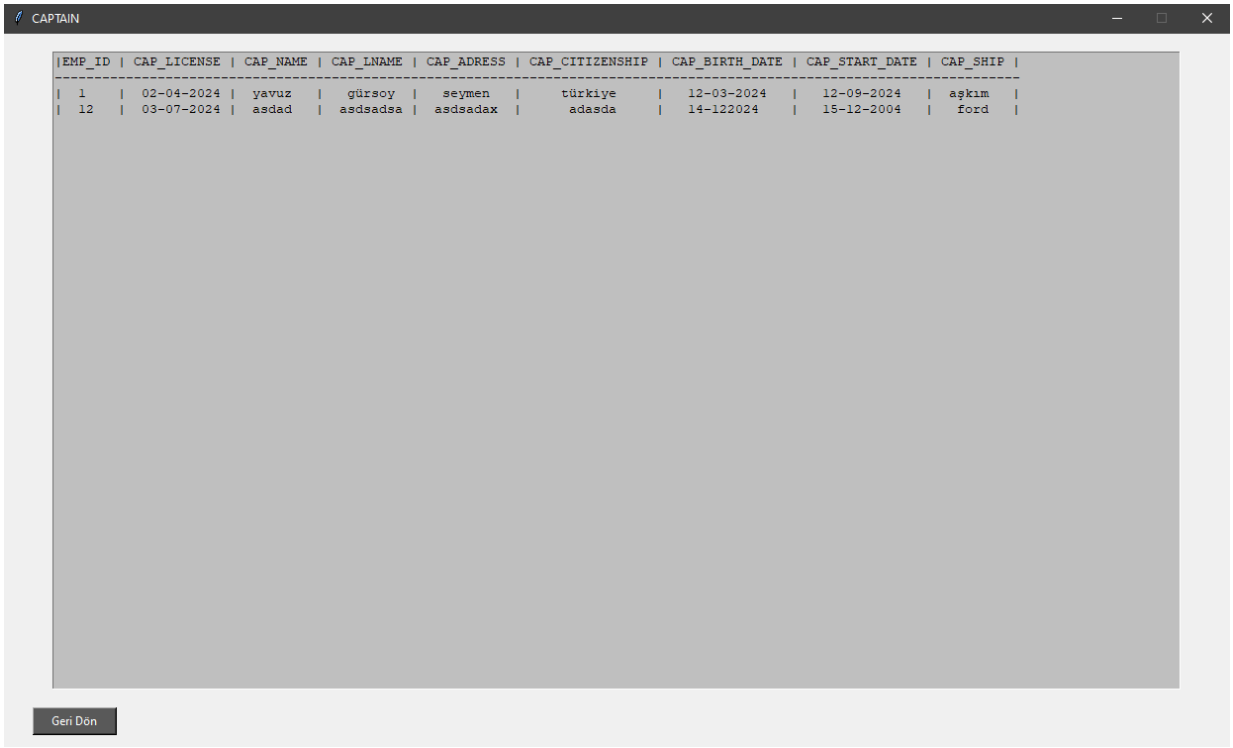
Bu şekilde bir pencere açılıyor. Yandaki koyu gri kutucuğa eklemek istediklerimizi yazıyoruz. Örneğin kaptan eklemek istiyorum. İlk önce EMP_ID'sini giriyorum sonra CAP_LICENSE sonra CAP_NAME sonra CAP_LNAME,CAP_ADRESS,CAP_CITIZENSHİP,CAP_BIRTH_DATE ,CAP_START_DATE,CAP_SHIP'İ sırasıyla giriyoruz. Eklemenin yapılabilmesi için kullanıcının sütun isimlerini bildiğini varsayıyoruz.

Değerleri girdikten sonra kaptan butonuna tıklıyoruz. Verilerin tutulduğu tabloda görmek için ekledikten sonra refresh butonuna basıyoruz ve eklediğimiz kişi geliyor.



Aynı şekilde diğer butonları kullanarak ekleme yapılabiliriz.

Burada veri tabanını görüntülemek için uygulamanın içinde olan bir yöntem daha geliştirdik. Tablo Görüntüle butonu da buradaki sonucun aynısını veriyor.



4 UYGULAMA

4.1 Kodlanan bileşenlerin açıklamaları

```
if "gezginGemi.db" not in os.listdir(os.getcwd()):
    runpy.run_path("query.py")
```

Eğer veri tabanı zaten varsa, bağlantı kur. Yoksa, "query.py" dosyasındaki sorguyu çalıştırarak bir veri tabanı oluştur.

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 CRUISE_SHIP(SHIP):
    def __init__(self, serialNum, name, grossTonnage, madeIn, capacity):
        super().__init__(serialNum, name, grossTonnage, madeIn)
        self.capacity = capacity
        self.type = 'CRUISE'
```

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.

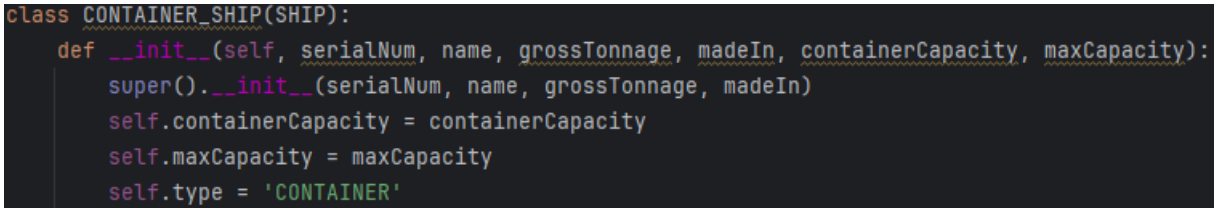
CLASS OIL SHIP(SHIP):

```
class OIL_SHIP(SHIP):
    def __init__(self, serialNum, name, grossTonnage, madeIn, oilCapacity):
        super().__init__(serialNum, name, grossTonnage, madeIn)
        self.oilCapacity = oilCapacity
        self.type = 'OIL'
```

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):

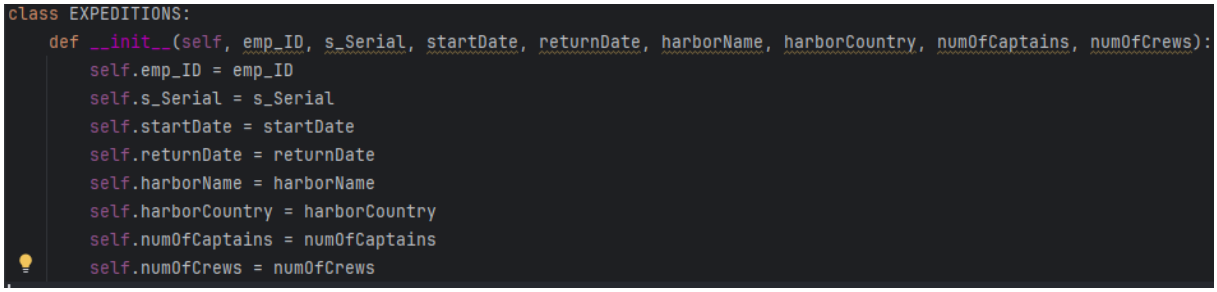


```
class CONTAINER_SHIP(SHIP):
    def __init__(self, serialNum, name, grossTonnage, madeIn, containerCapacity, maxCapacity):
        super().__init__(serialNum, name, grossTonnage, madeIn)
        self.containerCapacity = containerCapacity
        self.maxCapacity = maxCapacity
        self.type = 'CONTAINER'
```

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 EXPEDITIONS:



```
class EXPEDITIONS:
    def __init__(self, emp_ID, s_Serial, startDate, returnDate, harborName, harborCountry, numOfCaptains, numOfCrews):
        self.emp_ID = emp_ID
        self.s_Serial = s_Serial
        self.startDate = startDate
        self.returnDate = returnDate
        self.harborName = harborName
        self.harborCountry = harborCountry
        self.numOfCaptains = numOfCaptains
        self.numOfCrews = numOfCrews
```

Bu bölüm, seferlerin özelliklerini tanımlayan bir sınıf olan EXPEDITIONS sınıfını tanımlar.
__init__ metodu: Bu metot, bir EXPEDITIONS nesnesinin başlatılması sırasında çağrılır. Parametre olarak çalışan kimliği (emp_ID), gemi seri numarası (s_Serial), başlangıç tarihi (startDate), dönüş tarihi (returnDate), liman adı (harborName), liman ülkesi (harborCountry), kaptan sayısı (numOfCaptains) ve mürettebat sayısı (numOfCrews) alır. Bu parametreler, sınıfın örneğinin özelliklerini temsil eder.
Bu özellikler, bir geminin bir seferi sırasında ihtiyaç duyulan bilgileri içerir. Örneğin, bir geminin hangi limandan kalktığı, hangi limana

gideceği, hangi tarihlerde yola çıkıp döneceği, kaç kaptan ve mürettebatın gemide olduğu gibi bilgileri içerir.

CLASS HARBOR:

```
class HARBOR:
    def __init__(self, name, country, population, passportNeeded, fee):
        self.name = name
        self.country = country
        self.population = int(population)
        self.passportNeeded = passportNeeded
        self.fee = int(fee)
```

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

__init__ metodu: Bu metot, bir HARBOR nesnesinin başlatılması sırasında çağrılır. Parametre olarak limanın adı (name), bulunduğu ülke (country), nüfusu (population), pasaport gereksinimi (passportNeeded) ve ücreti (fee) alır. Bu parametreler, sınıfın örneğinin özelliklerini temsil eder.
Bu özellikler, bir limanın sahip olduğu temel bilgileri içerir. Örneğin, bir limanın adı, bulunduğu ülke, nüfusu, pasaport gereksinimi ve giriş ücreti gibi bilgileri içerir.

CLASS VISITED_HARBORS:

```
class VISITED_HARBORS:
    def __init__(self, name, country, s_Serial, visitState):
        self.name = name
        self.country = country
        self.s_Serial = s_Serial
        self.visitState = visitState
```

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

__init__ metodu: Bu metot, bir VISITED_HARBORS nesnesinin başlatılması sırasında çağrılır. Parametre olarak limanın adı (name), bulunduğu ülke (country), ziyaret edilen geminin seri numarası (s_Serial) ve ziyaret durumu (visitState) alır. Bu parametreler, sınıfın örneğinin özelliklerini temsil eder.
Bu özellikler, bir geminin seyir sırasında ziyaret ettiği limanların bilgilerini içerir. Örneğin, ziyaret edilen limanın adı, bulunduğu ülke, ziyaret edilen geminin seri numarası ve ziyaret durumu gibi bilgileri içerir.

CLASS EMPLOYEE:

```
class EMPLOYEE:
    def __init__(self, ID, job):
        self.ID = ID
        self.job = job
```

Bu bölüm, çalışanların özelliklerini tanımlayan bir sınıf olan EMPLOYEE sınıfını tanımlar.

__init__ metodu: Bu metot, bir EMPLOYEE nesnesinin başlatılması sırasında çağrılır. Parametre olarak çalışanın kimliği (ID) ve işi (job) alır. Bu parametreler, sınıfın örneğinin özelliklerini temsil eder.
Bu özellikler, bir çalışanın temel bilgilerini içerir. Örneğin, çalışanın kimliği ve hangi pozisyonda çalıştığı gibi bilgileri içerir.

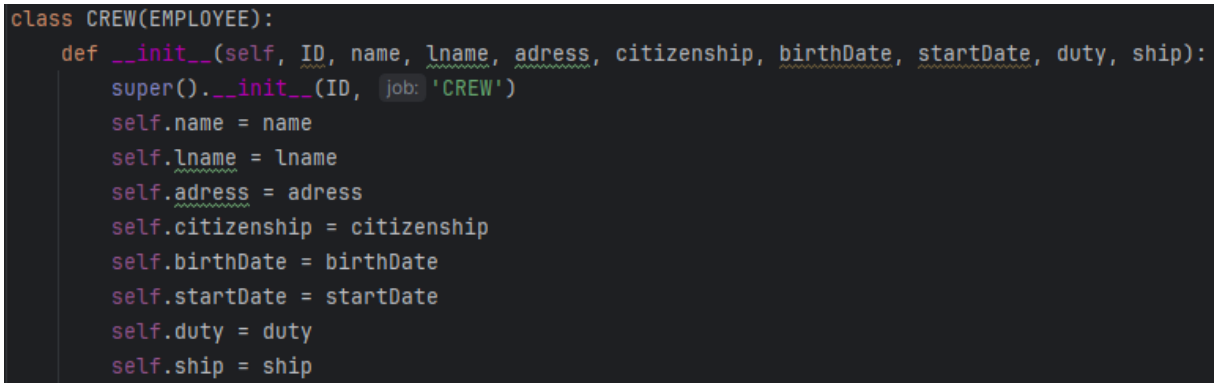
CLASS CAPTAIN(EMPLOYEE):

```
class CAPTAIN(EMPLOYEE):
    def __init__(self, ID, license, name, lname, adress, citizenship, birthDate, startDate, ship):
        super().__init__(ID, job: 'CAPTAIN')
        self.license = license
        self.name = name
        self.lname = lname
        self.adress = adress
        self.citizenship = citizenship
        self.birthDate = birthDate
        self.startDate = startDate
        self.ship = ship
```

Bu bölüm, gemi kaptanlarını temsil eden CAPTAIN sınıfını tanımlar. CAPTAIN sınıfı, EMPLOYEE sınıfından miras alır ve çalışanların özelliklerine ek olarak kaptanların özelliklerini tanımlar.

__init__ metodu: Bu metot, bir CAPTAIN nesnesinin başlatılması sırasında çağrılır. Parametre olarak kaptanın kimliği (ID), lisansı (license), adı (name), soyadı (lname), adresi (adress), vatandaşlığı (citizenship), doğum tarihi (birthDate), işe başlama tarihi (startDate) ve görev yapacağı gemi (ship) alır. super().__init__(ID, 'CAPTAIN') çağrısı, EMPLOYEE sınıfının __init__ metodunu çağırarak çalışanın kimliğini ve işini belirler.
Bu özellikler, bir kaptanın temel bilgilerini ve bir kaptanın özellikle dikkate alınması gereken özel bilgilerini içerir. Örneğin, bir kaptanın lisansı, adı, soyadı, adresi, doğum tarihi, işe başlama tarihi ve görev yapacağı gemi gibi bilgileri içerir.

CLASS CREW(EMPLOYEE):

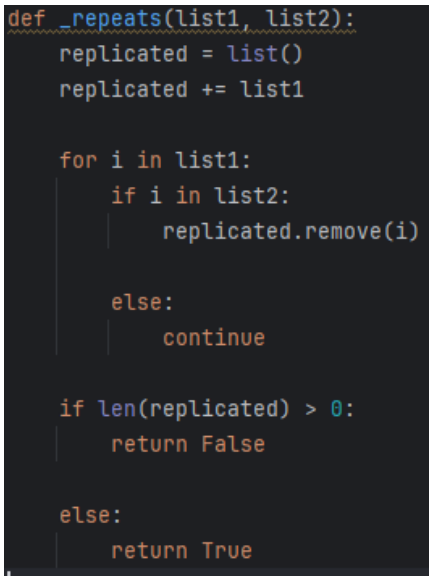


```
class CREW(EMPLOYEE):
    def __init__(self, ID, name, lname, adress, citizenship, birthDate, startDate, duty, ship):
        super().__init__(ID, job: 'CREW')
        self.name = name
        self.lname = lname
        self.adress = adress
        self.citizenship = citizenship
        self.birthDate = birthDate
        self.startDate = startDate
        self.duty = duty
        self.ship = ship
```

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

__init__ metodu: Bu metot, bir CREW nesnesinin başlatılması sırasında çağrılır. Parametre olarak mürettebatın kimliği (ID), adı (name), soyadı (lname), adresi (adress), vatandaşlığı (citizenship), doğum tarihi (birthDate), işe başlama tarihi (startDate), görevi (duty) ve görev yapacağı gemi (ship) alır. super().__init__(ID, 'CREW') çağrısı, EMPLOYEE sınıfının __init__ metodunu çağırarak çalışanın kimliğini ve işini belirler. Bu özellikler, bir mürettebat üyesinin temel bilgilerini ve bir mürettebat üyesinin özellikle dikkate alınması gereken özel bilgilerini içerir. Örneğin, bir mürettebat üyesinin adı, soyadı, adresi, vatandaşlığı, doğum tarihi, işe başlama tarihi, görevi ve görev yapacağı gemi gibi bilgileri içerir.

def _repeats(list1, list2):



```
def _repeats(list1, list2):
    replicated = list()
    replicated += list1

    for i in list1:
        if i in list2:
            replicated.remove(i)

        else:
            continue

    if len(replicated) > 0:
        return False

    else:
        return True
```

Bu _repeats fonksiyonu, iki liste alır ve birinci listedeki öğelerin ikinci listede tekrar edip etmediğini kontrol eder. Eğer birinci listedeki herhangi bir öğe ikinci listede tekrar ediyorsa, fonksiyon False değerini döndürür. Eğer birinci listedeki öğeler ikinci listede tekrar etmiyorsa, fonksiyon True değerini döndürür.

İşleyiş şu şekildedir:

replicated adında bir boş liste oluşturulur.
list1 listesinin tüm öğeleri replicated listesine kopyalanır.
Bir döngü başlatılır ve list1'deki her öğe için kontrol yapılır.
Eğer öğe list2 içinde ise, replicated listesinden bu öğe çıkarılır.
Eğer replicated listesinde hala öğe bulunuyorsa (yani, iki liste arasında tekrar eden bir öğe varsa), fonksiyon False döndürülür.
Eğer replicated listesi boş ise (yani, iki liste arasında tekrar eden bir öğe yoksa), fonksiyon True döndürülür.

def _construct_main_menu():

```
def _construct_main_menu():
    button = tk.Button(text="Kapat", bg="grey35", fg="white", command=root.destroy, padx=15, pady=2)
    button.place(x=30, y=290)

    button = tk.Button(text="Tablo Görüntüle", bg="grey35", fg="white", command=lambda: _construct_menu(3), padx=15,
                       pady=2)
    button.place(x=30, y=230)

    button = tk.Button(text="Kayıt Güncelle", bg="grey35", fg="white", command=lambda: _construct_menu(2), padx=15,
                       pady=2)
    button.place(x=30, y=170)

    button = tk.Button(text="Kayıt Sil", bg="grey35", fg="white", command=lambda: _construct_menu(1), padx=15, pady=2)
    button.place(x=30, y=110)

    button = tk.Button(text="Kayıt Ekle", bg="grey35", fg="white", command=lambda: _construct_menu(0), padx=15, pady=2)
    button.place(x=30, y=50)

    text = tk.Text(root, height=20, width=50, bg="grey75", fg="black")
    text.place(x=200, y=30)

    msg = tk.Message(root, bg="grey20", fg="white", text=f"{choice} tablosunun önizlemesi:",
                     width=500, font=("Calibri", 11))
    msg.place(x=200, y=2)

    outputList = c.fetchall()
    for i in outputList:
        text.insert(tk.END, str(i) + "\n")
    text.config(state=tk.DISABLED)
```

Bu construct_main_menu() fonksiyonu, ana menünün oluşturulmasından sorumludur. Bu menüde kullanıcıya çeşitli işlevler sunulur ve mevcut tabloların önizlemesi sağlanır.

İşlevleri şu şekildedir:

Ana menünün düğmeleri oluşturulur. Her bir düğme, farklı bir işlevi gerçekleştirecek şekilde yapılandırılır. Örneğin, "Kayıt Ekle" düğmesine tıklandığında _construct_menu(0) fonksiyonu çağrılır.
Tablo önizleme bölümü oluşturulur. Bu bölümde, c.fetchall() ile alınan tablo verileri, bir metin kutusuna eklenir ve kullanıcıya görüntülenir.
Mesaj kutusu (Message widget) oluşturulur. Bu kutuda, kullanıcının görmekte olduğu tablonun adı belirtilir.
Metin kutusu, önizleme verilerini görüntüledikten sonra değiştirilemez hale getirilir.

def _handle_input(textBox, sampleTable, action, classType, c):

```python
def _handle_input(textBox, sampleTable, action, classType, c):
    inputText = textBox.get(1.0, tk.END)
    textBox.delete(1.0, tk.END)
    # Menüdeki text box kısmını temizle.

    outputStr = str()
    splittedOnce = list()
    sampleAttrs = list()

    for i in inputText:
        if i == '\n':
            continue
        else:
            outputStr += i
        splittedOnce = outputStr.split(';')

    for x in splittedOnce:
        sampleAttrs += x.split(',')
    # Girilen değerleri virgülle ayır.

    actionCaseList = [insert_sample, delete_sample, update_sample, display_table]

    for index, case in enumerate(actionCaseList):
        if index == action and (index == 0 or index == 1):
            return case(sampleTable, sampleAttrs, classType, c)

        elif index == action and index == 2:
            return case(sampleTable, sampleAttrs, c)

        elif index == action and index == 3:
            return case(sampleTable)
```

Bu _handle_input() fonksiyonu, kullanıcının girdiği metni işlemek ve uygun işlevi çağırmak için kullanılır. Ayrıca, bu işlevin parametrelerini açıklayalım:

textBox: Kullanıcının girdiğini aldığımız metin kutusu.
sampleTable: İşlenecek olan örnek tablo adı.
action: Kullanıcının talep ettiği işlemi belirten bir sayı. Bu, "kayıt ekle", "kayıt sil", "kayıt güncelle" veya "tablo görüntüle" gibi işlemlere karşılık gelir.
classType: İşlenecek örnek sınıfının türü.
c: SQLite veritabanı bağlantısını temsil eden bir nesne.
İşlevin çalışma mantığı şu adımları izler:
Metin kutusundaki metni (inputText) alır.
Metni satır sonu karakterlerine (\n) göre böler ve her satırı bir öğe olarak alır.
Her satırı virgülle (',') böler ve her özelliği ayrı bir öğe olarak sampleAttrs

listesine ekler.
Belirlenen action sayısına göre, uygun işlevi çağırır. Eğer ekleme veya silme işlemi ise, ilgili işlevin yanı sıra sampleTable, sampleAttrs, classType ve c parametrelerini de gönderir.

def _construct_menu(action):

```
def _construct_menu(action):
    actionList = ["Kayıt ekle", "Kayıt sil", "Kayıt güncelle", "Tablo görüntüle"]
    enumerate(actionList)

    newWindow = tk.Toplevel(root)
    newWindow.title(f'{actionList[action]}')
    newWindow.resizable( width: False,  height: False)
    newWindow.grab_set()
    newWindow.geometry('640x380')

    text = tk.Text(newWindow, height=20, width=40, bg="grey75", fg="black")
    text.place(x=290, y=20)

    button = tk.Button(newWindow, text="Kaptan", bg="grey35", fg="white",
                       command=lambda: _handle_input(text,  sampleTable: "CAPTAIN", action, CAPTAIN, c), padx=15, pady=2)
    button.place(x=30, y=50)

    button = tk.Button(newWindow, text="Mürettebat", bg="grey35", fg="white",
                       command=lambda: _handle_input(text,  sampleTable: "CREW", action, CREW, c), padx=15, pady=2)
    button.place(x=30, y=110)

    button = tk.Button(newWindow, text="Ziyaret", bg="grey35", fg="white",
                       command=lambda: _handle_input(text,  sampleTable: "VISITED_HARBORS", action, VISITED_HARBORS, c), padx=15,
                       pady=2)
    button.place(x=30, y=170)

    button = tk.Button(newWindow, text="Yolculuk", bg="grey35", fg="white",
                       command=lambda: _handle_input(text,  sampleTable: "EXPEDITIONS", action, EXPEDITIONS, c), padx=15, pady=2)
    button.place(x=30, y=230)

    button = tk.Button(newWindow, text="Geri dön", bg="grey35", fg="white", command=newWindow.destroy, padx=15, pady=2)
    button.place(x=30, y=290)

    button = tk.Button(newWindow, text="Liman", bg="grey35", fg="white",
                       command=lambda: _handle_input(text,  sampleTable: "HARBOR", action, HARBOR, c), padx=15, pady=2)
    button.place(x=150, y=50)

    button = tk.Button(newWindow, text="Petrol gemisi", bg="grey35", fg="white",
                       command=lambda: _handle_input(text,  sampleTable: "OIL_SHIP", action, OIL_SHIP, c), padx=15, pady=2)
    button.place(x=150, y=110)
button = tk.Button(newWindow, text="Yolcu gemisi", bg="grey35", fg="white",
                   command=lambda: _handle_input(text,  sampleTable: "CRUISE_SHIP", action, CRUISE_SHIP, c), padx=15, pady=2)
button.place(x=150, y=170)

button = tk.Button(newWindow, text="Konteyner gemisi", bg="grey35", fg="white",
                   command=lambda: _handle_input(text,  sampleTable: "CONTAINER_SHIP", action, CONTAINER_SHIP, c), padx=15,
                   pady=2)
button.place(x=150, y=230)

button = tk.Button(newWindow, text="Temizle", bg="grey35", fg="white", command=lambda: text.delete( index1: 1.0, tk.END),
                   padx=15, pady=2)
button.place(x=150, y=290)
```

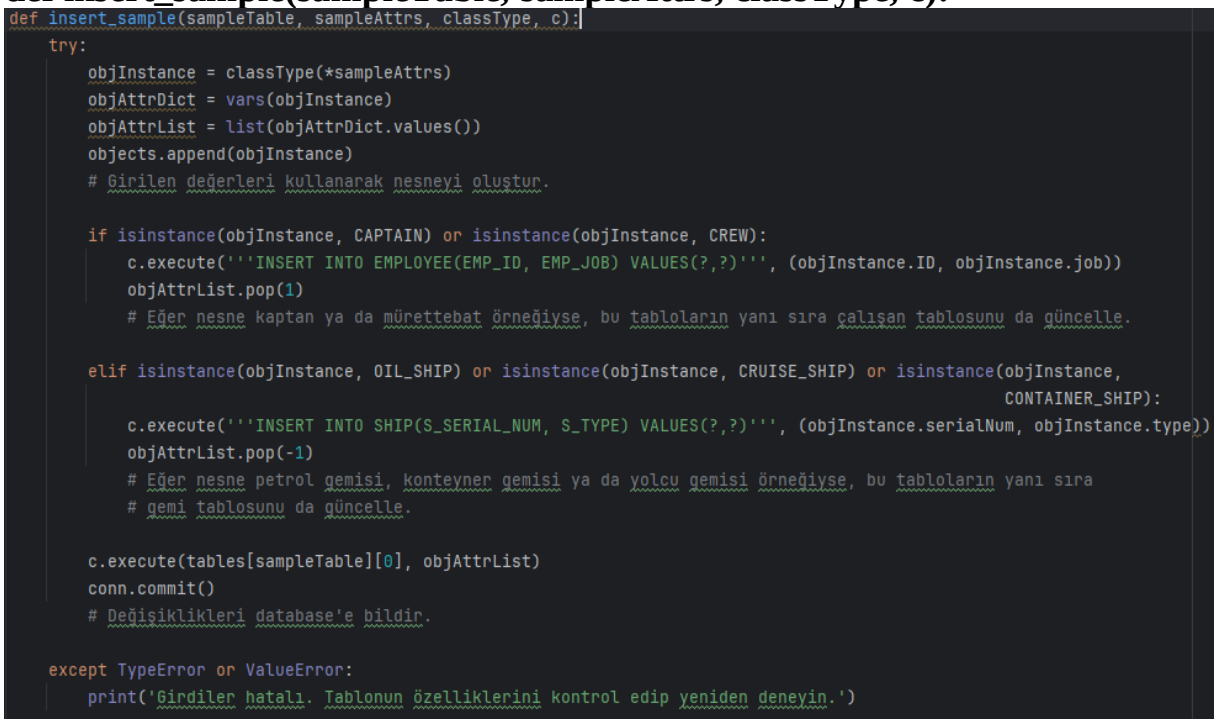
Bu _construct_menu() fonksiyonu, kullanıcıya farklı veritabanı işlemleri için bir menü sunar. Bu menü, "Kayıt Ekle", "Kayıt Sil", "Kayıt Güncelle" ve "Tablo Görüntüle" gibi seçenekleri içerir.

Fonksiyonun parametreleri şunlardır:

action: Kullanıcının talep ettiği işlemi belirten bir sayı. Bu sayı, kullanıcının hangi işlemi yapmak istediğini belirler.
Fonksiyonun işleyişi şu adımları izler:

action parametresine göre, doğru başlıkla bir pencere oluşturulur.
Bu pencerede farklı işlem seçenekleri için düğmeler oluşturulur.
Her düğmenin üzerine tıklandığında _handle_input() işlevi çağrılır. Bu işlev, kullanıcının girdiği metni işler ve ilgili veri tabanı işlemini gerçekleştirir.
Yaratılan pencere kapatılmadan önce, kullanıcıya "Geri Dön" seçeneği sunulur.
Daha fazla kullanıcı girdisi alınabileceği için, bir metin kutusu da pencereye eklenir ve işlemlerin sonuçları burada gösterilir.
Bu fonksiyon, kullanıcı arayüzünü oluşturur ve kullanıcının veri tabanı işlemlerini kolayca yapmasını sağlar.

def insert_sample(sampleTable, sampleAttrs, classType, c):



```
def insert_sample(sampleTable, sampleAttrs, classType, c):
    try:
        objInstance = classType(*sampleAttrs)
        objAttrDict = vars(objInstance)
        objAttrList = list(objAttrDict.values())
        objects.append(objInstance)
        # Girilen değerleri kullanarak nesneyi oluştur.

        if isinstance(objInstance, CAPTAIN) or isinstance(objInstance, CREW):
            c.execute('''INSERT INTO EMPLOYEE(EMP_ID, EMP_JOB) VALUES(?,?)''', (objInstance.ID, objInstance.job))
            objAttrList.pop(1)
            # Eğer nesne kaptan ya da mürettebat örneğiyse, bu tabloların yanı sıra çalışan tablosunu da güncelle.

        elif isinstance(objInstance, OIL_SHIP) or isinstance(objInstance, CRUISE_SHIP) or isinstance(objInstance,
                                                                                                     CONTAINER_SHIP):
            c.execute('''INSERT INTO SHIP(S_SERIAL_NUM, S_TYPE) VALUES(?,?)''', (objInstance.serialNum, objInstance.type))
            objAttrList.pop(-1)
            # Eğer nesne petrol gemisi, konteyner gemisi ya da yolcu gemisi örneğiyse, bu tabloların yanı sıra
            # gemi tablosunu da güncelle.

        c.execute(tables[sampleTable][0], objAttrList)
        conn.commit()
        # Değişiklikleri database'e bildir.

    except TypeError or ValueError:
        print('Girdiler hatalı. Tablonun özelliklerini kontrol edip yeniden deneyin.')
```

Bu insert_sample() fonksiyonu, kullanıcının veritabanına yeni örnekler eklemesini sağlar. Bu örnekler, farklı sınıflara ait nesneler olabilir.

Fonksiyonun parametreleri şunlardır:

sampleTable: Kullanıcının eklemek istediği örneğin tablosunu belirtir.
sampleAttrs: Eklenmek istenen örneğin özelliklerini içeren bir liste.
classType: Eklenmek istenen örneğin sınıfını belirtir.
c: Veri tabanı işlemlerini gerçekleştirmek için bir bağlantı nesnesi.
Bu fonksiyonun işlevi şu adımlarla gerçekleştirilir:

sampleAttrs listesi kullanılarak yeni bir nesne (objInstance) oluşturulur.
Oluşturulan nesnenin özellikleri (objAttrDict) bir sözlük olarak alınır ve bu sözlükten değerler (objAttrList) bir liste olarak alınır.
Oluşturulan nesne objects listesine eklenir.
Oluşturulan nesnenin tipine (classType) bağlı olarak işlem yapılır:
Eğer nesne bir kaptan veya mürettebat üyesi ise, ilgili veri tabanı tablosuna (EMPLOYEE) eklenir ve ilgili özellikler çıkarılır.
Eğer nesne bir petrol gemisi, konteyner gemisi veya yolcu gemisi ise, ilgili gemi tablosuna (SHIP) eklenir ve ilgili özellik çıkarılır.
sampleTable parametresinde belirtilen tabloya, eklenmek istenen

özelliklerin listesi (objAttrList) eklenir.
Değişiklikler veri tabanına kaydedilir.
Eğer herhangi bir hata oluşursa (TypeError veya ValueError), kullanıcıya bir hata mesajı gösterilir.
def delete_sample(sampleTable, sampleAttrs, classType, c):

```
def delete_sample(sampleTable, sampleAttrs, classType, c):
    sqlStr = f"{tables[sampleTable][1]}"
    c.execute(sqlStr, sampleAttrs)
    conn.commit()
    # tables sözlüğündeki 2. girdiyi (DELETE) çalıştır ve değişiklikleri databese'e bildir.

    for i in objects:
        if isinstance(i, classType) and _repeats(sampleAttrs, list(vars(i).values())):
            objects.remove(i)
            del i
            # Eğer eklenen örnek bu oturumda eklendiyse silinmeden önce geri döndürülebilir.
            # Ancak kullanıcı örneği silmek istediğine eminse, bu örneği objects listesinden kaldır.
```

Bu delete_sample() fonksiyonu, kullanıcının belirli örnekleri veri tabanından silmesini sağlar.

Fonksiyonun parametreleri şunlardır:

sampleTable: Kullanıcının silmek istediği örneğin tablosunu belirtir.
sampleAttrs: Silinmek istenen örneğin özelliklerini içeren bir liste.
classType: Silinmek istenen örneğin sınıfını belirtir.
c: Veri tabanı işlemlerini gerçekleştirmek için bir bağlantı nesnesi.
Bu fonksiyonun işlevi şu adımlarla gerçekleştirilir:

sampleTable parametresindeki silme sorgusunu (DELETE) tables sözlüğünden alınan SQL ifadesiyle (tables[sampleTable][1]) birleştirerek çalıştırır.
Değişiklikleri veri tabanına kaydederek (conn.commit()), belirtilen örneği veri tabanından siler.
objects listesinde dolaşarak, silinen örneği bulur ve bu örneği listeden kaldırır.
Bu işlemler sonunda belirtilen örnekler veri tabanından silinir ve eğer o oturumda eklenmişse, objects listesinden de kaldırılır.

def update_sample(sampleTable, sampleAttrs, c):

```python
def update_sample(sampleTable, sampleAttrs, c):
    sqlStr = f"UPDATE {sampleTable} SET {sampleAttrs[-2]} = {sampleAttrs[-1]} WHERE "
    # SQL'i çalıştırmak için bir string hazırla.

    for i in range(len(tables[sampleTable][2])):
        sqlStr += f"{tables[sampleTable][2][i]} = {sampleAttrs[i]}"
        sqlStr += f" AND "
    # İlk indisler birincil anahtar değerleri (sütun adları değil değerleri),
    # Sondan önceki indis değiştirilecek sütunun ismi,
    # Son indis ise yeni değer.

    sqlStr = sqlStr[:-5]
    # For döngüsünden sonra metnin sonunda bir " AND " ifadesi kalacak.
    # Onu sil.

    c.execute(sqlStr)
    conn.commit()
    # Değişiklikleri database'e bildir.
```

Bu update_sample() fonksiyonu, kullanıcının belirli bir örneğin özelliklerini güncellemesini sağlar.

Fonksiyonun parametreleri şunlardır:

sampleTable: Güncellenmek istenen örneğin tablosunu belirtir.
sampleAttrs: Güncellenmek istenen örneğin özelliklerini içeren bir liste.
c: Veri tabanı işlemlerini gerçekleştirmek için bir bağlantı nesnesi.
Bu fonksiyonun işlevi şu adımlarla gerçekleştirilir:

Güncelleme sorgusunu oluşturmak için sampleTable ve sampleAttrs kullanılarak bir SQL stringi hazırlanır.
Güncellenecek sütun ismi ve yeni değeri bu stringe eklenir.
Sütunları belirlemek için tables sözlüğünden gelen bilgiler ve sampleAttrs kullanılır.
Hazırlanan SQL stringi c.execute() ile çalıştırılarak veritabanında güncelleme yapılır.
Son olarak, değişiklikler veri tabanına kaydedilir (conn.commit()).
Bu işlemler sonunda belirtilen örneğin özellikleri veri tabanında güncellenir.

def display_table(sampleTable):

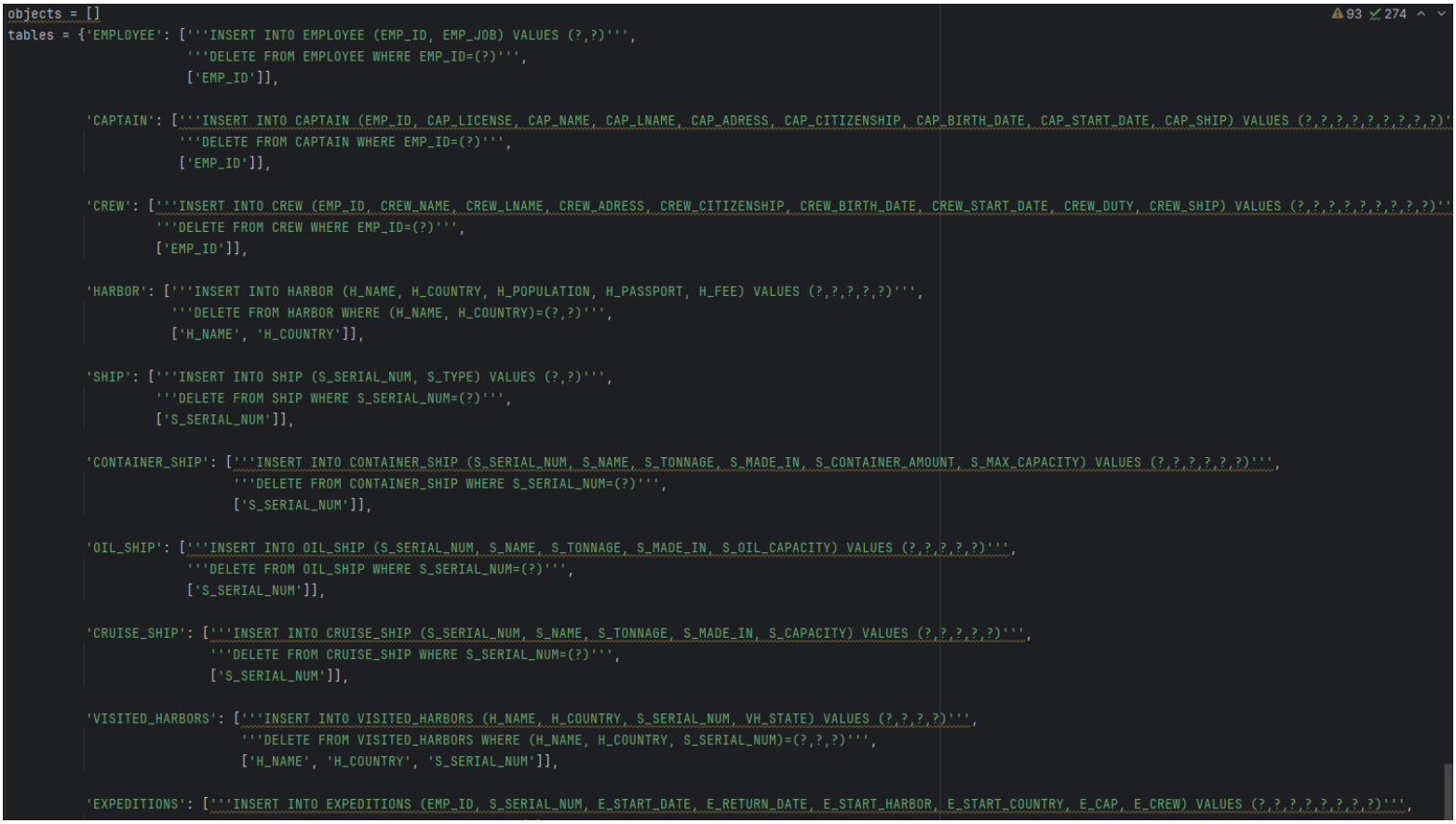
```
def display_table(sampleTable):
    newWindow = tk.Toplevel(root)
    newWindow.title(f'{sampleTable}')
    newWindow.resizable( width: False,  height: False)
    newWindow.grab_set()
    newWindow.geometry('1240x720')
    # İkincil menünün üstüne boyutu değiştirilemeyen bir sekme aç.

    cursor = conn.cursor()
    cursor.execute(f"SELECT * FROM {sampleTable}")
    records = cursor.fetchall()
    # Tablo verilerini al.

    text = tk.Text(newWindow, height=40, width=142, bg="grey75", fg="black")
    text.place(x=50, y=18)

    button = tk.Button(newWindow, text="Geri Dön", bg="grey35", fg="white", command=newWindow.destroy, padx=15, pady=2)
    button.place(x=30, y=680)
    # Butonu ve metin kutusunu oluştur.

    try:

        cursor.execute(f"PRAGMA table_info({sampleTable})")
        columns = [column[1] for column in cursor.fetchall()]
        # Tablo başlıklarını al.

        columnKeyWidths = [len(str(column)) for column in columns]
        columnValueWidths = [max(len(str(record[i])) for record in records) for i in range(len(columns))]
        columnWidths = []
        # Her sütun için en uzun girdiyi bul.

        for i in range(len(columnKeyWidths)):
            if columnKeyWidths[i] >= columnValueWidths[i]:
                columnWidths.append(columnKeyWidths[i])
            else:
                columnWidths.append(columnValueWidths[i])
            # Başlık ve örnek değerlerinden hangisi daha büyükse onu columnWidths listesine koy.

        text.insert(tk.END,  chars: "|")
        for record in records:
            text.insert(tk.END,  chars: "|")
            for i, field in enumerate(record):
                text.insert(tk.END,  chars: f"{str(field).center(columnWidths[i])} | ")
            text.insert(tk.END,  chars: "\n")
        # Verileri ekrana yazdır.

        text.config(state=tk.DISABLED)
        # Metin kutusunun durumunu değiştirilemez olarak değiştir.

    except ValueError:
        text.insert(tk.END,  chars: "")
```

Bu display_table() fonksiyonu, belirtilen bir tablonun içeriğini bir pencerede göstermek için kullanılır.

Fonksiyonun parametresi:

sampleTable: Gösterilmek istenen tablonun adını belirtir.
Bu fonksiyonun işlevi şu adımlarla gerçekleştirilir:

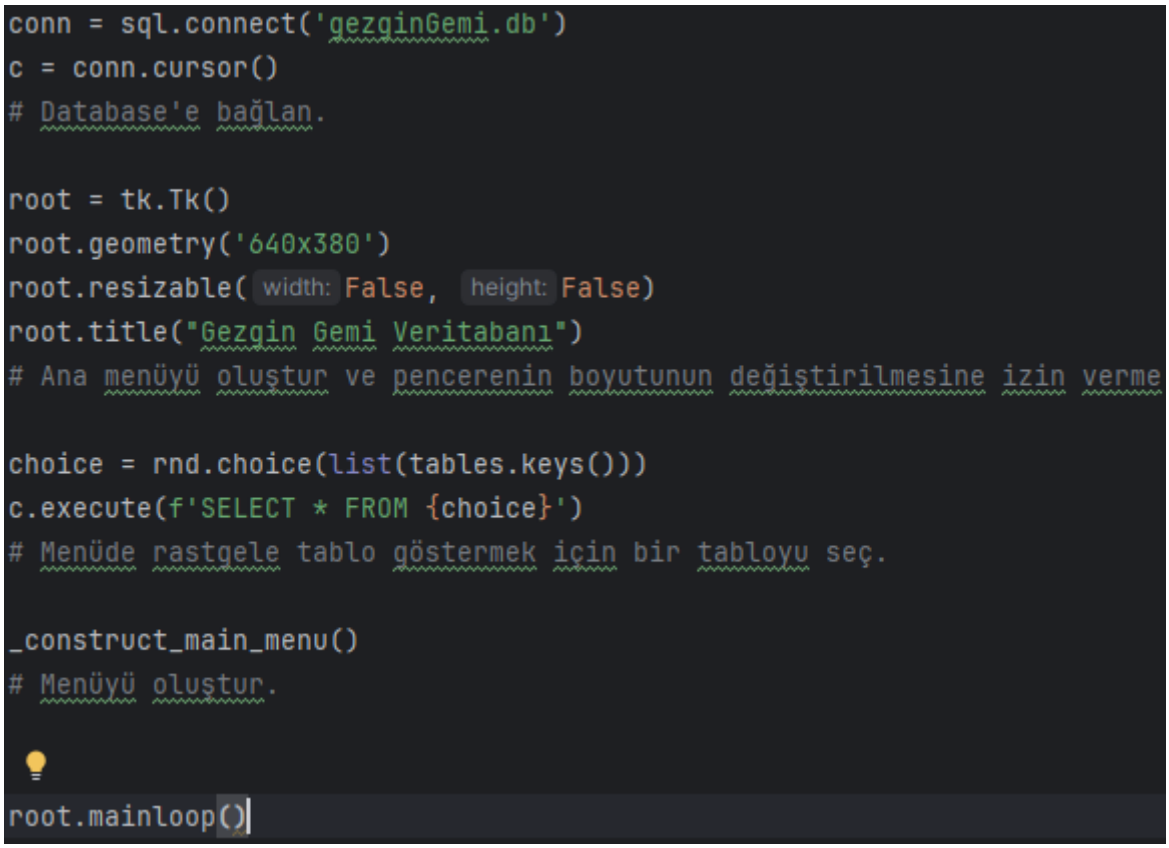
Yeni bir pencere oluşturulur ve pencerenin başlığı belirtilen tablonun adıyla ayarlanır.
Veri tabanı bağlantısından bir imleç (cursor) oluşturulur ve tablodaki tüm verileri seçmek için SQL sorgusu çalıştırılır.
Alınan kayıtlar bir değişkende saklanır.
Pencerede bir metin kutusu oluşturulur ve boyutu ayarlanır.
Geri dön butonu oluşturulur ve pencereye yerleştirilir.
Tablonun başlıkları ve verileri metin kutusuna eklenir.
Metin kutusunun düzenlenemez (read-only) hale getirilir.
Bu işlemler sonunda belirtilen tablonun başlıkları ve verileri pencerede gösterilir. Eğer tablonun içeriği boş ise, metin kutusuna hiçbir şey eklenmez.



```
objects = []
tables = {'EMPLOYEE': ['''INSERT INTO EMPLOYEE (EMP_ID, EMP_JOB) VALUES (?,?)''',
                       '''DELETE FROM EMPLOYEE WHERE EMP_ID=(?)''',
                       ['EMP_ID']],

          'CAPTAIN': ['''INSERT INTO CAPTAIN (EMP_ID, CAP_LICENSE, CAP_NAME, CAP_LNAME, CAP_ADRESS, CAP_CITIZENSHIP, CAP_BIRTH_DATE, CAP_START_DATE, CAP_SHIP) VALUES (?,?,?,?,?,?,?,?,?)''
                      '''DELETE FROM CAPTAIN WHERE EMP_ID=(?)''',
                      ['EMP_ID']],

          'CREW': ['''INSERT INTO CREW (EMP_ID, CREW_NAME, CREW_LNAME, CREW_ADRESS, CREW_CITIZENSHIP, CREW_BIRTH_DATE, CREW_START_DATE, CREW_DUTY, CREW_SHIP) VALUES (?,?,?,?,?,?,?,?,?)''
                   '''DELETE FROM CREW WHERE EMP_ID=(?)''',
                   ['EMP_ID']],

          'HARBOR': ['''INSERT INTO HARBOR (H_NAME, H_COUNTRY, H_POPULATION, H_PASSPORT, H_FEE) VALUES (?,?,?,?,?)''',
                     '''DELETE FROM HARBOR WHERE (H_NAME, H_COUNTRY)=(?,?)''',
                     ['H_NAME', 'H_COUNTRY']],

          'SHIP': ['''INSERT INTO SHIP (S_SERIAL_NUM, S_TYPE) VALUES (?,?)''',
                   '''DELETE FROM SHIP WHERE S_SERIAL_NUM=(?)''',
                   ['S_SERIAL_NUM']],

          'CONTAINER_SHIP': ['''INSERT INTO CONTAINER_SHIP (S_SERIAL_NUM, S_NAME, S_TONNAGE, S_MADE_IN, S_CONTAINER_AMOUNT, S_MAX_CAPACITY) VALUES (?,?,?,?,?,?)''',
                              '''DELETE FROM CONTAINER_SHIP WHERE S_SERIAL_NUM=(?)''',
                              ['S_SERIAL_NUM']],

          'OIL_SHIP': ['''INSERT INTO OIL_SHIP (S_SERIAL_NUM, S_NAME, S_TONNAGE, S_MADE_IN, S_OIL_CAPACITY) VALUES (?,?,?,?,?)''',
                       '''DELETE FROM OIL_SHIP WHERE S_SERIAL_NUM=(?)''',
                       ['S_SERIAL_NUM']],

          'CRUISE_SHIP': ['''INSERT INTO CRUISE_SHIP (S_SERIAL_NUM, S_NAME, S_TONNAGE, S_MADE_IN, S_CAPACITY) VALUES (?,?,?,?,?)''',
                          '''DELETE FROM CRUISE_SHIP WHERE S_SERIAL_NUM=(?)''',
                          ['S_SERIAL_NUM']],

          'VISITED_HARBORS': ['''INSERT INTO VISITED_HARBORS (H_NAME, H_COUNTRY, S_SERIAL_NUM, VH_STATE) VALUES (?,?,?,?)''',
                              '''DELETE FROM VISITED_HARBORS WHERE (H_NAME, H_COUNTRY, S_SERIAL_NUM)=(?,?,?)''',
                              ['H_NAME', 'H_COUNTRY', 'S_SERIAL_NUM']],

          'EXPEDITIONS': ['''INSERT INTO EXPEDITIONS (EMP_ID, S_SERIAL_NUM, E_START_DATE, E_RETURN_DATE, E_START_HARBOR, E_START_COUNTRY, E_CAP, E_CREW) VALUES (?,?,?,?,?,?,?,?)''',
```

Bu kod, objects ve tables adlı iki sözlük oluşturur.

objects sözlüğü, programın çalışması sırasında oluşturulan nesneleri saklamak için kullanılır. Bu nesneler, veri tabanına ekleme, silme veya güncelleme işlemleri sırasında yaratılır.
tables sözlüğü, farklı tablolar için SQL sorgularını, silme işlemlerini gerçekleştirmek için gereken bilgileri ve anahtar sütunları içerir. Her tablo adı, ilgili tabloya özgü bir alt liste içinde saklanır. Alt listeler, o tabloya özgü bir dizi SQL sorgusu içerir:
İlk öğe: Tabloya veri eklemek için kullanılan SQL INSERT sorgusu.
İkinci öğe: Tablodan veri silmek için kullanılan SQL DELETE sorgusu.
Üçüncü öğe: Tablonun birincil anahtar sütunlarını içeren bir liste.
Örneğin, 'EMPLOYEE' tablosuna erişmek için, tables['EMPLOYEE'] kullanılır ve bu, 'EMPLOYEE' tablosuna özgü SQL sorgularını içeren bir alt liste döndürür.



Bu kısım, bir Tkinter uygulaması oluşturur ve ana pencereyi başlatır.

İlk olarak, SQLite veri tabanına (gezginGemi.db) bağlantı kurulur. Bağlantıyı temsil etmek için conn adında bir değişken oluşturulur ve cursor oluşturulur.
Ardından, Tkinter'da bir ana pencere oluşturulur. Pencerenin boyutu 640x380 piksel olarak ayarlanır ve boyutunun değiştirilmesine izin verilmez. Pencerenin başlığı "Gezgin Gemi Veritabanı" olarak ayarlanır.
Rastgele bir tablo seçmek için choice adında bir değişken oluşturulur. Bu, tables sözlüğünden rastgele bir tabloyu seçer ve seçilen tabloyu SELECT * sorgusu ile okur.
_construct_main_menu() fonksiyonu çağrılır. Bu fonksiyon, ana menüyü oluşturur. Menü, çeşitli işlemleri gerçekleştirmek için düğmeler içerir: Kayıt ekleme, silme, güncelleme, tabloyu görüntüleme ve pencereyi kapatma.
root.mainloop() çağrısı, Tkinter uygulamasının ana döngüsünü başlatır. Bu döngü, kullanıcı etkileşimini bekler ve pencerenin kapatılmasına kadar devam eder.

4.2 Görev dağılımı

- Projenin planlaması birlikte yapılmış olup kod yazma aşamasında SQL kodlarını Emrah Şahin, arayüzü Yavuz Selim Gürsoy yazmıştır. Geriye kalan fonksiyon ve sınıflar birlikte yazılmıştır.
- Rapor aşamasında Giriş, Gereksinim Analizi, Test ve Doğrulama başlıklarını Emrah Şahin hazırlamıştır. Tasarım, Uygulama başlıklarını da Yavuz Selim Gürsoy hazırlamıştır.

4.3 Karşılaşılan zorluklar ve çözüm yöntemleri

- SQL kodları hazırlanışı periyodunda hangi tabloları hangi anahtar ile bağlama aşamasında zorluklar yaşanmıştır.
- Yazılan fonksiyonlar ve sınıfları arayüze ekleme aşamasında zorluklar yaşanmıştır.
- Arayüzün tasarımı birkaç kez değişip olabilecek en optimum kullanıcı odaklı arayüz hazırlama aşamasında bir takım zorluklar yaşanmıştır.
- ON UPDATE CASCADE komutunun kullanılmasında zorluklar yaşanmıştır. Bu komut kütüphane içerisinde kullanıldığında etki göstermemektedir. Bunun için komutun yaptığı iş elle yazılmak zorunda kalınmıştır.

4.4 Proje isterlerine göre eksik yönler

- Fark edilen eksik yönler veya hatalar yoktur.

5 TEST VE DOĞRULAMA

5.1 Yazılımın test süreci

- Yazılım için bir test süreci hazırlanmıştır ama projede arayüz olduğu için test kodu kullanıcı girişi olmadan çalışmamaktadır. Bu yüzden test kodu kullanılmamıştır.

```python
import unittest
import tkinter as tk
import sqlite3 as sql
import random as rnd

class TestMainApplication(unittest.TestCase):

    def test_ship_classes(self):
        serial_
        name =
        tonnage = 1000
        made_in = "Test Country"

        # Test Ship Objects
        test_cruise_ship = CRUISE_SHIP(serial_num, name, tonnage, made_in, 500)
        test_oil_ship = OIL_SHIP(serial_num, name, tonnage, made_in, 2000)
        test_container_ship = CONTAINER_SHIP(serial_num, name, tonnage, made_in, 1000, 2000)

        self.assertEqual(test_cruise_ship.type, 'CRUISE')
        self.assertEqual(test_oil_ship.type, 'OIL')
        self.assertEqual(test_container_ship.type, 'CONTAINER')

    def test_secondary_menus(self):
        # Test Inserting Sample
        test_input = "1234,John,Doe,Address,USA,1990-01-01,2024-01-01,Captain,TestShip"
        sample_attrs = test_input.split(',')
        _handle_input(text, "CAPTAIN", 0, CAPTAIN, c)
        c.execute("SELECT * FROM CAPTAIN WHERE EMP_ID = 1234")
        result = c.fetchall()
        self.assertEqual(len(result), 1)
```

TestMainApplication
def test_ship_classes(self) -> None

```python
        # Test Deleting Sample
        _handle_input(text, "CAPTAIN", 1, CAPTAIN, c)
        c.execute("SELECT * FROM CAPTAIN WHERE EMP_ID = 1234")
        result = c.fetchall()
        self.assertEqual(len(result), 0)

        # Test Updating Sample
        test_input = "1234,EMP_JOB,CAPTAIN"
        sample_attrs = test_input.split(',')
        _handle_input(text, "CAPTAIN", 2, CAPTAIN, c)
        c.execute("SELECT * FROM CAPTAIN WHERE EMP_ID = 1234")
        result = c.fetchall()
        self.assertEqual(result[0][1], 'CAPTAIN')

        # Test Displaying Table
        _handle_input(text, "CAPTAIN", 3, CAPTAIN, c)
        # Check if a new window is opened with table displayed

if __name__ == '__main__':
    unittest.main()
```

5.2 Yazılımın doğrulanması

- Programcıların yaptığı bu projede herhangi bir hata ile karşılaşılmamış olup öğretim görevlilerin programcılardan isteği şekilde kod hazırlanmıştır.

6 GitHub Adresleri

https://github.com/emrahsahn/EmrahSahin
https://github.com/Yavuz-Selim-Gursoy