26.12.2020

İstanbul Sabahattin Zaim Üniversitesi Bilgisayar 3.Sınıf Veri Tabanı Projesi

STUDENT MANAGEMENT SYSTEM

ERDEM DEMIR 030118060
TAYYİB BAYRAM 030118106

Github: https://github.com/TaBayram/StudentManagement/

Server'a bağlanabilmek için: Tutorial

Login Şifresi: 1234

Uygulamamızın Jar dosyayı vardır açmak için tıklayın.

Jar Konumu: ..\StudentManagement\out\artifacts\StudentManagement_jar

Abstract

Bu raporumuz Veri Tabanı dersinin projesi üzerine oluşturulmuştur. Derste işlediğimiz tüm konuları yer alan bir sistem uygulaması istenmiştir.

Bu amaç doğrultusunda Java kullanarak Student Management System'i yaptık.

Introduction

İlk başta öğrenci database'ini oluşturacağız bu veritabanı ile iletişim kuran front-end sistemi kuracağız. Bu front-end de öğrenci, öğretmen, fakülte ve bölümleri ,gösterme ekleme silme vs. gibi işlemleri yapacağız.

Methodology

Öncelikle yaptığımız programı hem Windows hem de MAC'te kullanılabileceğinden ötürü kullanacağımız programlama dilinin Java olacağına karar verdik. Javanın yazılım platformu olan JavaFX'i arayüzümüz olarak kullandık. Öğrencilerini verilerinin tutulmasını sağlayan veritabanı programlama dilini ise Microsoft Structured Query Language Server (MSSQL)'i seçtik. Öğrenci isimleri ,numaraları ,okudukları bölümler gibi veritabanının tüm işlemlerini yapabilecek ve ayrıca öğrenci gruplandırma ,sınıflandırma ve çeşitle veritabanı tekniklerini kullanarak okuma kolaylığı sağlıyacağız.

Database

1.Student Database Tables

- **Department Table**

```
CREATE TABLE [dbo].[DepartmentTable](
	[ID] [int] IDENTITY(1000,1) NOT NULL,
	[Name] [nvarchar](50) NULL,
	[Language] [nvarchar](50) NULL,
	[DepartmentChair] [int] NULL,
	[FacultyID] [int] NULL,
	[CourseCount] [int] NULL,
 CONSTRAINT [PK_DepartmentTable] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]) ON [PRIMARY]
```

- **Faculty Table**

```
CREATE TABLE [dbo].[FacultyTable](
	[ID] [int] IDENTITY(100,1) NOT NULL,
	[Name] [nvarchar](50) NULL,
	[FacultyChair] [int] NULL,
 CONSTRAINT [PK_FacultyTable] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
```

- **Left Teacher Table**

```
CREATE TABLE [dbo].[LeftTeacherTable](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[info] [nvarchar](100) NOT NULL,
 CONSTRAINT [PK_LeftTeacherTable] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
```

➢ Student Table

```
CREATE TABLE [dbo].[StudentTable](
	[ID] [int] IDENTITY(10000,2) NOT NULL,
	[Name] [nvarchar](50) NOT NULL,
	[Surname] [nvarchar](50) NOT NULL,
	[Password] [nvarchar](50) NOT NULL,
	[Email] [nvarchar](50) NOT NULL,
	[DepartmentID] [int] NULL,
	[RegisteredDate] [datetime] NULL,
	[Semester] [int] NULL,
	[AdvisorID] [int] NULL,
	[GPA] [float] NULL,
 CONSTRAINT [PK_StudentTable] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
```

➢ Teacher Table

```
CREATE TABLE [dbo].[TeacherTable](
	[ID] [int] IDENTITY(1000,1) NOT NULL,
	[Name] [nvarchar](50) NULL,
	[Surname] [nvarchar](50) NULL,
	[Email] [nvarchar](50) NULL,
	[Title] [nvarchar](50) NULL,
	[Password] [nvarchar](50) NULL,
	[DepartmentID] [int] NULL,
	[RegisteredDate] [datetime] NULL,
 CONSTRAINT [PK_Teacher] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
```

2.Foreign Keys

- **FK_DepartmentTable_FacultyTable :**

```
ALTER TABLE [dbo].[DepartmentTable]  WITH NOCHECK ADD  CONSTRAINT
[FK_DepartmentTable_FacultyTable] FOREIGN KEY([FacultyID])
REFERENCES [dbo].[FacultyTable] ([ID])
ON DELETE CASCADE

ALTER TABLE [dbo].[DepartmentTable] CHECK CONSTRAINT [FK_DepartmentTable_FacultyTable]
```

- **FK_DepartmentTable_Teacher :**

```
ALTER TABLE [dbo].[DepartmentTable]  WITH NOCHECK ADD  CONSTRAINT
[FK_DepartmentTable_Teacher] FOREIGN KEY([DepartmentChair])
REFERENCES [dbo].[TeacherTable] ([ID])
ON DELETE SET NULL

ALTER TABLE [dbo].[DepartmentTable] NOCHECK CONSTRAINT [FK_DepartmentTable_Teacher]
```

- **FK_StudentTable_DepartmentTable :**

```
ALTER TABLE [dbo].[StudentTable]  WITH NOCHECK ADD  CONSTRAINT
[FK_StudentTable_DepartmentTable] FOREIGN KEY([DepartmentID])
REFERENCES [dbo].[DepartmentTable] ([ID])
ON DELETE SET NULL

ALTER TABLE [dbo].[StudentTable] NOCHECK CONSTRAINT [FK_StudentTable_DepartmentTable]
```

- **FK_StudentTable_Teacher:**

```
ALTER TABLE [dbo].[StudentTable]  WITH CHECK ADD  CONSTRAINT [FK_StudentTable_Teacher]
FOREIGN KEY([AdvisorID])
REFERENCES [dbo].[TeacherTable] ([ID])
ON DELETE SET NULL

ALTER TABLE [dbo].[StudentTable] CHECK CONSTRAINT [FK_StudentTable_Teacher]
```

- **FK_Teacher_DepartmentTable:**

```
ALTER TABLE [dbo].[TeacherTable]  WITH CHECK ADD  CONSTRAINT
[FK_Teacher_DepartmentTable] FOREIGN KEY([DepartmentID])
REFERENCES [dbo].[DepartmentTable] ([ID])
 ON DELETE SET NULL

 ALTER TABLE [dbo].[TeacherTable] CHECK CONSTRAINT [FK_Teacher_DepartmentTable]
```

3.Constraint

- **StudentTable**

```
ALTER TABLE [dbo].[StudentTable]  WITH CHECK ADD  CONSTRAINT [CK_StudentTable_GPA]
CHECK  (([GPA]>=(0) AND [GPA]<=(4)))

ALTER TABLE [dbo].[StudentTable] CHECK CONSTRAINT [CK_StudentTable_GPA]
```

4.Default

- **StudentTable**

```
ALTER TABLE [dbo].[StudentTable] ADD  CONSTRAINT [DF_StudentTable_Semester]  DEFAULT
((0)) FOR [Semester]
```

4.Procedures

- **spABS :** Verilen numaranın mutlak değerini döndürür

```
Create procedure [dbo].[spABS]
@number float,
@result float OUTPUT
as
Begin
set @result = ABS(@number)
End
```

- **spAddDepartment :** Verilen özelliklerle yeni Departman ekler.

```
Create Procedure [dbo].[spAddDepartment]
@Name nvarchar(50),
@Language nvarchar(50),
@DepartmentChair int,
@FacultyID int
@CourseCount int
as
Begin
insert into
DepartmentTable(Name,Language,DepartmentChair,FacultyID,CourseCount) OUTPUT
INSERTED.ID
values(LTRIM(RTRIM(@Name)),LTRIM(RTRIM(@Language)),@DepartmentChair,@FacultyID,
@CourseCount)
End
```

- **spAddFaculty :** Verilen özelliklerle yeni Fakülte ekler

```
Create Procedure [dbo].[spAddFaculty]
@Name nvarchar(50),
@FacultyChair int
as
Begin
insert into FacultyTable (Name,FacultyChair) OUTPUT INSERTED.ID
values(LTRIM(RTRIM(@Name)),@FacultyChair)
End
```

- **spAddStudent :** Verilen özelliklerle yeni Öğrenci ekler.

```
Create Procedure [dbo].[spAddStudent]
@Name nvarchar(50),
@Surname nvarchar(50),
@Password nvarchar(50),
@Email nvarchar(50),
@DepartmentID int,
@AdvisorID int
as
Begin
insert into StudentTable (Name,Surname,Password,Email,DepartmentID,AdvisorID)
values(LTRIM(RTRIM(@Name)), UPPER(LTRIM(RTRIM(@Surname))),
LTRIM(RTRIM(@Password)), LTRIM(RTRIM(@Email)), @DepartmentID,@AdvisorID)
SELECT SCOPE_IDENTITY()
End
```

- **spAddTeacher :** Verilen özelliklerle yeni Öğretmen ekler.

```
Create Procedure [dbo].[spAddTeacher]
@Name nvarchar(50),
@Surname nvarchar(50),
@Password nvarchar(50),
@Email nvarchar(50),
@Title nvarchar(50),
@DepartmentID int
as
Begin
insert into
TeacherTable(Name,Surname,Password,Email,Title,DepartmentID,RegisteredDate)
OUTPUT INSERTED.ID
values(LTRIM(RTRIM(@Name)), UPPER(LTRIM(RTRIM(@Surname))),
LTRIM(RTRIM(@Password)),LTRIM(RTRIM(@Email)),LTRIM(RTRIM(@Title)),@DepartmentID
,GETDATE())
End
```

- **spAllStudent :** Bütün öğrencileri döndürür

```
Create Procedure [dbo].[spAllStudent]
as
Begin
Select * from vAllStudent
End
```

- **spGetStudentCountByDepartment :** Verilen departmana göre öğrenci listesi döndürür.

```
Create procedure [dbo].[spGetStudentCountByDepartment]
@departmentID int,
@result int OUTPUT
as
Begin
Select @result = COUNT(ID) from StudentTable where DepartmentID = @departmentID
End
```

- **spUpdateDepartment :** Belirtilen departmanı günceller.

```
Create Procedure [dbo].[spUpdateDepartment]
@ID int,
@Name nvarchar(50),
@Language nvarchar(50),
@DepartmentChair nvarchar(50),
@FacultyID nvarchar(50)
@CourseCount int
as
Begin
Update DepartmentTable
SET Name = @Name, Language = @Language,DepartmentChair= @DepartmentChair
,FacultyID = @FacultyID, CourseCount = @CourseCount
WHERE ID = @ID
End
```

- **spUpdateFaculty :** Belirtilen fakülteyi günceller.

```
Create Procedure [dbo].[spUpdateFaculty]
@ID int,
@Name nvarchar(50),
@FacultyChair int
as
Begin
Update FacultyTable
SET Name = @Name, FacultyChair = ISNULL(@FacultyChair,0)
WHERE ID = @ID
End
```

- **spUpdateStudent :** Belirtilen öğrenciyi günceller

```
Create Procedure [dbo].[spUpdateStudent]
@ID int,
@Name nvarchar(50),
@Surname nvarchar(50),
@Password nvarchar(50),
@Email nvarchar(50),
@DepartmentID int
as
Begin
Update StudentTable
SET Name = @Name, Surname = @Surname,Password= @Password,Email =
@Email,DepartmentID= @DepartmentID
WHERE ID = @ID
End
```

- **spUpdateTeacher :** Belirtilen öğretmeni günceller.

```
Create Procedure [dbo].[spUpdateTeacher]
@ID int,
@Name nvarchar(50),
@Surname nvarchar(50),
@Password nvarchar(50),
@Email nvarchar(50),
@Title nvarchar(50),
@DepartmentID int
as
Begin
Update TeacherTable
SET Name = @Name, Surname = @Surname,Password= @Password,Email = @Email,Title =
@Title, DepartmentID= @DepartmentID
WHERE ID = @ID
End
```

5.Views

- **vAllStudent** : Bütün öğrencileri döndürür

```
CREATE VIEW [dbo].[vAllStudent]
AS
SELECT ID, Name, Surname, Password, Email, DepartmentID, RegisteredDate,
Semester, AdvisorID, COALESCE (GPA, 0) AS GPA, CONVERT(nvarchar,
RegisteredDate, 105) AS RegisteredDateFormatted FROM dbo.StudentTable
```

6.Functions

- **Semester:**Verilen öğrenci kayıt tarihi ile bugünü alıp kaçıncı dönemde olduğunu döndürür.

```
Create FUNCTION [dbo].[Semester](@registeredDate date)
RETURNS INT
AS
BEGIN
DECLARE @Semester INT
set @Semester = DATEDIFF(MONTH, @registeredDate, GETDATE())
set @Semester /= 6
RETURN @Semester
END
```

7.Triggers

- **dtTeacherTableLeftTeachers :** Teacher tablosundan silinen öğretmenleri burdan back-up yapabiliriz .

```
ALTER TRIGGER [dbo].[dtTeacherTableLeftTeachers]
ON [dbo].[TeacherTable]
FOR DELETE
AS
BEGIN
Declare @ID int
Declare @Name nvarchar(50)
Select @ID = ID from deleted
Select @Name = Name + SPACE(1) + Surname from deleted
insert into LeftTeacherTable (info)
values('An existing teacher, ID  = ' + Cast(@Id as nvarchar(10)) + ', ' + @Name
+ ' is deleted at ' + Cast(Getdate() as nvarchar(20)))
END
```

- **dtStudentTableRegisteredDate** :Öğrenci tablosuna yeni bir öğrenci eklendiğinde otomatik olarak öğrencinin tarihini update eder.

```
ALTER TRIGGER [dbo].[dtStudentTableRegisteredDate]
ON [dbo].[StudentTable]
FOR INSERT
AS
BEGIN
Update StudentTable
Set RegisteredDate = GETDATE()
WHERE ID = (SELECT ID from inserted)
END
```

8.Clustered & non-Clustereded

- FacultyTable

```
CREATE UNIQUE NONCLUSTERED INDEX [NonClusteredIndex-20201215-182812] ON
[dbo].[FacultyTable]
(
	[Name] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
```

- StudentTable

```
CREATE NONCLUSTERED INDEX [NonClusteredIndex-20201215-182434] ON [dbo].[StudentTable]
(
	[Name] ASC,
	[Surname] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
```

- TeacherTable

```
CREATE NONCLUSTERED INDEX [IX_TeacherTable_Name] ON [dbo].[TeacherTable]
(
	[Name] ASC ) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
```

JAVA

Classes

- Student
    - String DepartmentName;
    - int ID;
    - String Name;
    - String Surname;
    - String Password;
    - String Email;
    - String DepartmentID;
    - Date RegisteredDate;
    - String RegisteredDateFormatted;
    - Integer Semester;
    - Integer AdvisorID;
    - Float GPA;
- Teacher
    - int ID;
    - String Name;
    - String Surname;
    - String Email;
    - String Title;
    - String Password;
    - String HiddenPassword;
    - String DepartmentID;
    - Date RegisteredDate;
    - String DepartmentName;
- Department
    - int ID;
    - String Name;
    - String language;
    - String DepartmentChair;
    - Button button;
    - String DepartmentFacultyID;
    - Int CourseCount;

- Faculty
    - int ID;
    - String Name;
    - String FacultyChair;
    - String FacultyChairName;
    - int FacultyCourseCount;
- Person
    - int ID;
    - String Name;
    - String Surname;
    - String Password;
    - String Email;
    - String DepartmentID;
    - String DepartmentName;
    - String RegisteredDateFormatted;

Controller

❖ public void initialize()

Uygulama başlarken bir takım kontrol elemanlarını ayarlar.

❖ public void ShowFaculty(ActionEvent actionEvent)

Fakülte tablosunu gösterip liste için DatabaseTaskCreate'i çağır.

❖ public void FacultyTableChangeCommit(TableColumn.CellEditEvent event)

Fakülte tablosunda herhangi bir değişikliğinde değişikliği kontrol edip işlem yapar.

❖ public void ShowFacultyChairName(ActionEvent actionEvent)

Fakültenin dekanın ID'sini ya da adını tabloya yazdırır.

❖ public void FacultyAdd(ActionEvent actionEvent)

Fakülte ekle butonuna bastığında yeni satır oluşturur.

❖ public void FacultyRemove(ActionEvent actionEvent)

Fakülte sil butonuna bastığında fakülte silme işlemini çağırır.

❖ private void SetFacultyTableViewProperties(){

Fakülte Tablosunu ayarlar.

- **public void ShowAllStudent(ActionEvent actionEvent) throws IOException**

Öğrenci tablosunu gösterip liste için DatabaseTaskCreate'i çağırır.

- **public void ShowStudentsByDepartmentID(int ID)**

Belirtilen departmana göre öğrenci listesi için DatabaseTaskCreate'i çağırır.

- **public void StudentAdd(ActionEvent actionEvent)**

Öğrenci ekle butonuna bastığında yeni satır oluşturur.

- **public void StudentRemove(ActionEvent actionEvent)**

Öğrenci sil butonuna bastığında öğrenci silme işlemini çağırır.

- **public void StudentTableChangeCommit(TableColumn.CellEditEvent event)**

Öğrenci tablosunda herhangi bir değişikliğinde değişikliği kontrol edip işlem yapar.

- **public void ShowStudentsDepartmentName(ActionEvent actionEvent)**

Öğrenci bölümün ID'sini ya da adını tabloya yazdırır.

- **private void SetStudentTableViewProperties()**

Öğrenci Tablosunu ayarlar.

- **public void ShowDepartments(ActionEvent actionEvent) throws SQLException, ClassNotFoundException**

Departman tablosunu gösterip liste için DatabaseTaskCreate'i çağırır.

- **public void DepartmentTableChangeCommit(TableColumn.CellEditEvent event)**

Departman tablosunda herhangi bir değişikliğinde değişikliği kontrol edip işlem yapar.

- **public void DepartmentAdd(ActionEvent actionEvent)**

Departman ekle butonuna bastığında yeni satır oluşturur.

- **public void DepartmentRemove(ActionEvent actionEvent)**

Departman sil butonuna bastığında departman silme işlemini çağırır.

- **private void SetDepartmentTableViewProperties()**

Departman Tablosunu ayarlar.

- **public void ShowAllTeacher(ActionEvent actionEvent)**

Öğretmen tablosunu gösterip liste için DatabaseTaskCreate'i çağırır.

- **public void ShowTeachersDepartmentName(ActionEvent actionEvent)**

Öğretmen bölümün ID'sini ya da adını tabloya yazdırır.

- **public void TeacherAdd(ActionEvent actionEvent)**

Öğretmen ekle butonuna bastığında yeni satır oluşturur.

- **public void TeacherRemove(ActionEvent actionEvent)**

Öğretmen sil butonuna bastığında öğretmen silme işlemini çağırır.

- **public void TeacherTableChangeCommit(TableColumn.CellEditEvent event)**

Öğretmen tablosunda herhangi bir değişikliğinde değişikliği kontrol edip işlem yapar.

- **private void SetTeacherTableViewProperties()**

Öğretmen Tablosunu ayarlar.

- **public void ShowAllPeople(ActionEvent actionEvent)**

Kişi tabolsunu gösterip liste için DatabaseTaskCreate'i çağırır.

- **private void SetPersonTableViewProperties()**

Öğrenci Tablosunu ayarlar.

- **public void SetSelectedDepartmentID(int ID, String name)**

Departmana özel öğrenci gösterildiğinde departmanı kaydeder.

- **public void ShowAllStudentCountByDepartment()**

Departmana özel öğrenci listesinin öğrenci sayısını yazdırır.

- **public void MainPaneHideOthersExceptThis(Node control)**

Gönderilen tablo controlu gösterip diğerlerini gizler.

- **private void ButtonPaneHideOthersExceptThis(Node control)**

Gönderilen buton ebeveyni gösterip diğerlerini gizler.

- **private void InfoPaneVisible(boolean wantItShown)**

İsim ve liste sayısının ebeveynini gösterir/gizler.

- **public void DoMath(KeyEvent keyEvent)**

Seçilen matematik işlemini control edip işlem için fonksiyon çağırır.

- **public void PasswordColumnCheckBox(ActionEvent actionEvent)**

Öğretmen tablosunun şifrelerini şifreler/deşifreler.

- **public void KeyPressPassword(KeyEvent keyEvent)**

Log-in ekranın şifresini kontrol eder.

- **public void ShowLeftTeachers(ActionEvent actionEvent)**

Çıkan öğretmen tablosunu gösterip, listeni getirir.

- **public static class ErrorAlert**

Herhangi bir yanlış giriş için uyarı ekranı gösteren class.

- **private boolean HasSpecialCharacters(String name, String text)**

Verilen string'de herhangi bir özel karakter olup olmadığını kontrol eder.

- **private boolean HasStudentCorrectEmailFormat(Student student, boolean change)**

Verilen öğrencinin emailinin doğru formatta olup olmadığını kontrol eder.

- **private boolean HasTeacherCorrectEmailFormat(Teacher teacher, boolean change)**

Verilen öğretmenin emailinin doğru formatta olup olmadığını kontrol eder.

- **private boolean HasCorrectPasswordFormat(String password)**

Verilen şifrenin doğru formatta olup olmadığını kontrol eder.

- **private boolean HasOnlyNumbers(String name, String text)**

Verilen stringed sadece numara olup olmadığını kontrol eder.

- **private void VeilIndicator(Task task)**

Task çalışırken uygulamayı loading durumuna sokar.

- **public void DatabaseTaskCreate(String getAllType)**

Verilen türe göre yeni bir taskli thread oluşturup threadı çalıştırır.

- **public void DatabaseTaskCreate(int DepartmentID)**

Verilen departmana göre yeni bir taskli thread oluşturup threadi çalıştırır.

- **private Task TaskGetAllStudent()**

Bütün öğrencilerin listesini db fonksiyonundan getirir.

- **private Task TaskGetAllTeacher()**

Bütün öğretmenlerin listesini db fonksiyonundan getirir.

- **private Task TaskGetAllFaculty()**

Bütün fakültelerin listesini db fonksiyonundan getirir.

- **private Task TaskGetAllDepartment()**

Bütün departmanların listesini db fonksiyonundan getirir.

- **private Task TaskGetAllPerson()**

Bütün kişilerin listesini db fonksiyonundan getirir.

- **private Task TaskGetStudentsByDepartment(int ID)**

Departmana ait bütün öğrencilerin listesini db fonksiyonundan getirir.

- **public void TaskCreateDatabase()**

Veri Tabanı kontrolu için yeni bir Thread oluşturur.

Database

- **public static ObservableList<Faculty> GetAllFaculties()**

Veri Tabanından fakülte tablosunundaki fakültelerin listesini alır.

- **public static boolean GetFacultyChairNames(ObservableList<Faculty> faculties)**

Veri Tabanından fakülte tablosunundaki fakültelerin dekanının ismini alır.

- **public static int AddFaculty(Faculty faculty)**

Veri Tabanının fakülte tablosununa yeni fakülte ekler.

- **public static boolean AlterFaculty(Faculty faculty)**

Veri Tabanının fakülte tablosununun verilen fakültesini günceller.

- **public static boolean RemoveFaculty(Faculty faculty)**

Veri Tabanının fakülte tablosununun verilen fakültesini siler.

- **public static boolean DoesFacultyExist(int ID)**

Veri Tabanından fakülte tabolsunda belirtilen ID ile bir fakülte olup olmadığına bakar.

- **public static ObservableList<Student> GetAllStudents()**

Veri Tabanından öğrenci tablosunundaki öğrenci lerinlistesini alır.

- **public static boolean GetStudentDepartmentName(ObservableList<Student> students)**

Veri Tabanından öğrenci tablosunundaki departmanların isimlerini alır.

- **public static ObservableList<Student> GetStudentListByDepartmentID(int ID)**

Veri Tabanından belirtilen departmana ait öğrenci listesini alır.

- **public static int AddStudent(Student student)**

Veri Tabanının öğrenci tablosununa yeni öğrenci ekler.

- **public static boolean AlterStudent(Student student)**

Veri Tabanının öğrenci tablosununun verilen öğrenciyi günceller

- **public static boolean RemoveStudent(Student student)**

Veri Tabanının öğrenci tablosununun verilen öğrenciyi siler.

- **public static ObservableList<Department> GetAllDepartments()**

Veri Tabanından departman tablosunundaki departmanların listesini alır

- **public static ArrayList<String> GetDepartmentNames()**

Veri Tabanından departman tablosununun isimlerini alır.

- **public static boolean DoesDepartmentExist(int ID)**

Veri Tabanından departman tabolsunda belirtilen ID ile bir departman olup olmadığına bakar.

- **public static int AddDepartment(Department department)**

Veri Tabanının departman tablosununa yeni departman ekler.

- **public static boolean AlterDepartment(Department department)**

Veri Tabanının departman tablosununun verilen departmanı günceller.

- **public static boolean RemoveDepartment(Department department)**

Veri Tabanının departman tablosununun verilen departmanı siler

- **public static ObservableList<Teacher> GetAllTeachers()**

Veri Tabanından öğretmen tablosunundaki öğretmenlerin listesini alır.

- **public static boolean GetTeacherDepartmentName(ObservableList<Teacher> teachers)**

Veri Tabanından öğretmen tablosunundaki departmanların ismini alır.

- **public static int AddTeacher(Teacher teacher)**

Veri Tabanının öğretmen tablosununa yeni öğretmen ekler.

- **public static boolean AlterTeacher(Teacher teacher)**

Veri Tabanının öğretmen tablosununun verilen öğretmeni günceller.

- **public static boolean RemoveTeacher(Teacher teacher)**

Veri Tabanının öğretmen tablosununun verilen öğretmeni siler.

- **public static boolean DoesTeacherExist(int ID)**

Veri Tabanından öğretmen tablosunda belirtilen ID ile bir öğretmen olup olmadığına bakar.

- **public static ObservableList<Person> GetAllPeople()**

Veri Tabanından öğretmen ve öğrencilerin birleşmiş tablosununun listesini alır.

- **public static int GetStudentCountByDepartmentID(int ID)**

Veri Tabanından belirtilen departmandaki öğrenci sayısını alır.

- **public static int GetCount(String table)**

Veri Tabanından belirtilen tablodaki satır sayısını alır.

- **public static double GetMathResult(double number, int operation)**

Veri Tabanından verilen işlem ve sayıya göre işlem yapar.

- **public static ObservableList<String> GetLeftTeachers()**

Veri Tabanından ayrılan öğretmen tablosunundaki bilgi listesini alır.

- **public static Task CreateDatabaseFromScratch()**

Bu uygulamanın veri tabanının olup olmadığını kontrol eder. Yoksa sıfırdan veri tabanını oluşturur.

- **private static class DatabaseError**

Database hataları için uyarı çıkaran class.

Conclusion

Proje sonunda ön-raporumuzun dediği ders manipülasyonunu eklemedik çünkü ders konularımızın hepsini kullandık ve ders manipülasyonu eklemenin konu bakımından zenginlik katmayacağını anladık. Birbirine tam entegre olmaması dolayısıyla Java ile Microsoft SQL Server bağlanmamız bizi zorladı. Buna rağmen Java'nın diğer konularda bize rahatlık sağladı. JavaFX sayesinde uygulamamız tasarlayabildik , güzel bir tema oluşturduk.

Veritabanındaki özellikler çok geniş olup bu kadarlık bir kısmının bile böyle bir uygulama yapabileceğini gördük. Veri Tabanı dersi sayesinde bu konuları gerçek hayatta nasıl işlendiğini öğrendik.