SAKARYA
ÜNİVERSİTESİ

Ad : Younes Rahebi
Numara : B221210588
Grub : 1B

CINEMA VERITABANI

VERİTABANI YÖNETİM SİSTEMLERİ

Dr.Öğr.Üyesi İSMAİL ÖZTEL

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
BİLGİSAYAR MÜHENDİSLİĞİ PR.

IŞ KURALLARI

- Bir filmin birçok türü olabilir.
- Bir tür birçok filmde olabilir.

- Bir aktör birçok filmde olabilir.
- Bir filmde birçok aktör olabilir.

- Bir bilet bir filme aittir.
- Bir filmin birçok bileti olabilir.

- Bir filmin birçok salonu olabilir
- Bir salonun bir filmi olabilir.

- Bir salonda birçok koltuk olabilir.
- Bir koltuk bir salonda olabilir.

- Bir ödeme birçok bilet içerebilir.
- Bir bilet bir ödemeye aittir.

- Bir filmin bir yönetmeni olabilir.
- Bir yönetmen birçok film yönete bilir.

Ilişkisel Şema

Film (Film_ID:int, Film_Ad: nvarchar(MAX), Değerlendirme:int, Yayın_Yıl:int, Yönetmen_ID:int, UstundeOrtalama:bool)

Film_Aktör (Film_ID:int, Aktör_ID:int)

Aktör (Aktör_ID:int, Aktör_Ad: nvarchar(MAX))

Film_Tür (Film_ID:int, Tür_ID:int)

Tür (Tür_ID:int , Tür_Ad: nvarchar(MAX))

Yönetmen (Yönetmen_ID:int , Yönetmen_Ad: nvarchar(MAX))

Salon (Salon_ID:int, Film_ID:int , Sinema_Ad: nvarchar(MAX), Kapasite:int)

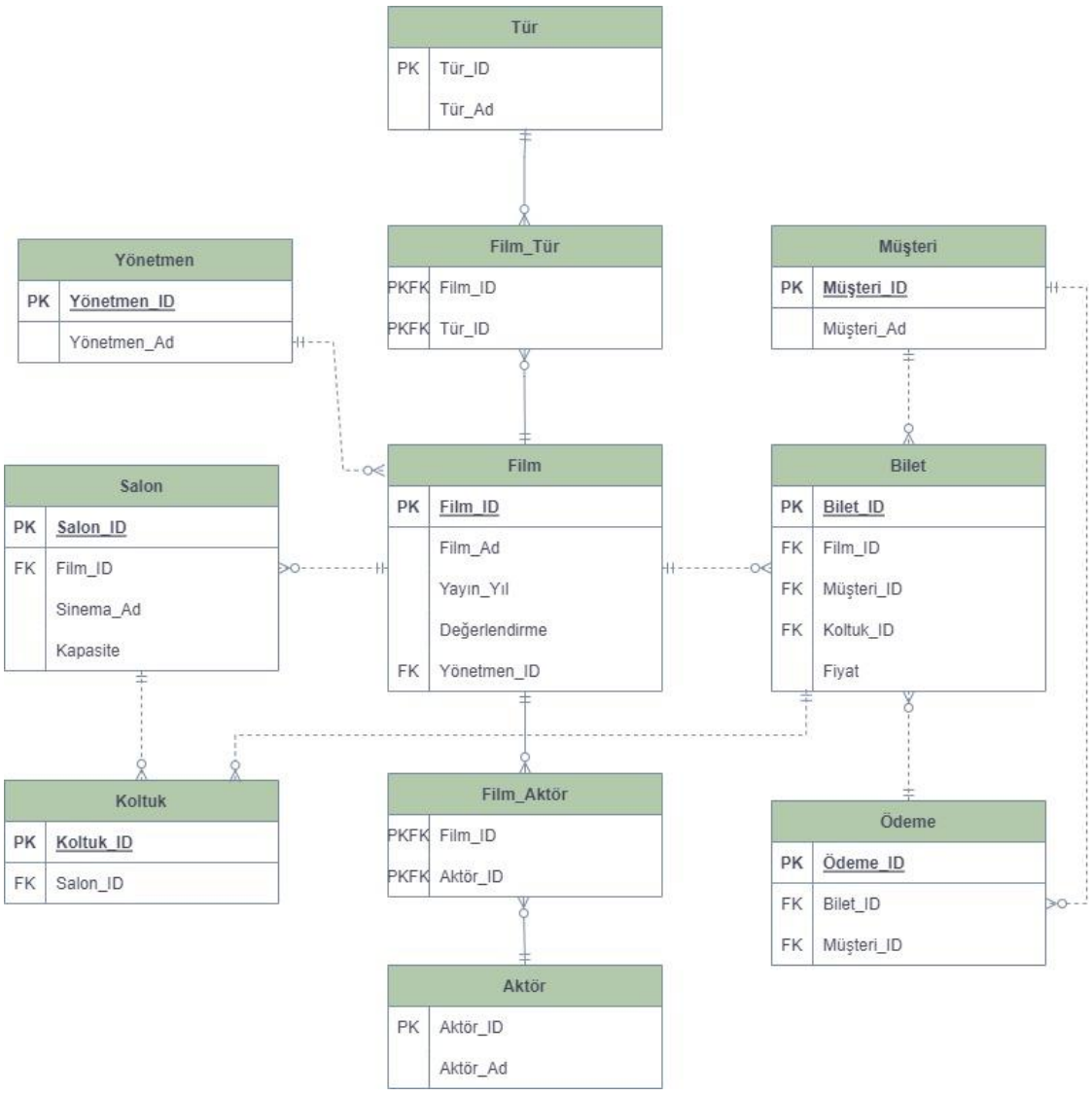
Koltuk (Koltuk_ID:int, Salon_ID:int)

Bilet (Bilet_ID:int, Film_ID:int, Müşteri_ID:int, Koltuk_ID:int, Fiyat:int)

Ödeme (Ödeme_ID:int, Bilet_ID:int, Müşteri_ID:int)

Müşteri (Müşteri_ID:int, Müşteri_Ad: nvarchar(MAX))

Varlik Baginti Diyagrami



SQL İFADELERİ

```
-- PostgreSQL database dump
-- Dumped from database version 14.1
-- Dumped by pg_dump version 14.0

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

SET default_tablespace = '';

SET default_table_access_method = heap;


--
-- Name: Film; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Film" (
    "Film_ID" SERIAL PRIMARY KEY,
    "Film_Ad" character varying,
    "Degerlendirme" DOUBLE PRECISION,
    "Yayin_Yil" integer,
    "Yönetmen_ID" integer,
    "UstundeOrtalama" BOOLEAN
);

ALTER TABLE public."Film" OWNER TO postgres;

--
-- Name: Film_Aktör; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Film_Aktör" (
```

```
    "Film_ID" integer NOT NULL,
    "Aktör_ID" integer NOT NULL
);

ALTER TABLE public."Film_Aktör" OWNER TO postgres;

--
-- Name: Aktör; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Aktör" (
    "Aktör_ID" SERIAL PRIMARY KEY,
    "Aktör_Ad" character varying
);

ALTER TABLE public."Aktör" OWNER TO postgres;

--
-- Name: Film_Tür; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Film_Tür" (
    "Film_ID" integer NOT NULL,
    "Tür_ID" integer NOT NULL
);

ALTER TABLE public."Film_Tür" OWNER TO postgres;

--
-- Name: Tür; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Tür" (
    "Tür_ID" SERIAL PRIMARY KEY,
    "Tür_Ad" character varying
);

ALTER TABLE public."Tür" OWNER TO postgres;

--
-- Name: Yönetmen; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Yönetmen" (
    "Yönetmen_ID" SERIAL PRIMARY KEY,
```

```
    "Yönetmen_Ad" character varying
);

ALTER TABLE public."Yönetmen" OWNER TO postgres;

--
-- Name: Salon; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Salon" (
    "Salon_ID" SERIAL PRIMARY KEY,
    "Film_ID" integer,
    "Sinema_Ad" character varying,
    "Kapasite" integer
);

ALTER TABLE public."Salon" OWNER TO postgres;

--
-- Name: Koltuk; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Koltuk" (
    "Koltuk_ID" SERIAL PRIMARY KEY,
    "Salon_ID" integer NOT NULL
);

ALTER TABLE public."Koltuk" OWNER TO postgres;

--
-- Name: Bilet; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Bilet" (
    "Bilet_ID" SERIAL PRIMARY KEY,
    "Film_ID" integer NOT NULL,
    "Müşteri_ID" integer NOT NULL,
    "Koltuk_ID" integer NOT NULL,
    "Fiyat" integer
);

ALTER TABLE public."Bilet" OWNER TO postgres;

--
-- Name: Ödeme; Type: TABLE; Schema: public; Owner: postgres
```

```
--

CREATE TABLE public."Ödeme" (
    "Ödeme_ID" SERIAL PRIMARY KEY,
    "Bilet_ID" integer NOT NULL,
    "Müşteri_ID" integer NOT NULL
);

ALTER TABLE public."Ödeme" OWNER TO postgres;

--
-- Name: Müşteri; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Müşteri" (
    "Müşteri_ID" SERIAL PRIMARY KEY,
    "Müşteri_Ad" character varying
);

ALTER TABLE public."Müşteri" OWNER TO postgres;




-- Foreign Key Constraints



--
-- Name: Film Film_Yönetmen_fkey; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE public."Film" ADD CONSTRAINT "Film_Yönetmen_fkey" FOREIGN KEY
("Yönetmen_ID") REFERENCES public."Yönetmen"("Yönetmen_ID");

--
-- Name: Salon Salon_Film_fkey; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE public."Salon" ADD CONSTRAINT "Salon_Film_fkey" FOREIGN KEY
("Film_ID") REFERENCES public."Film"("Film_ID");

--
```

```
-- Name: Koltuk Koltuk_Salon_fkey; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE public."Koltuk" ADD CONSTRAINT "Koltuk_Salon_fkey" FOREIGN KEY
("Salon_ID") REFERENCES public."Salon"("Salon_ID");



-- Composite Keys



--
-- Name: Film_Aktör Film_Aktör_pkey; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE public."Film_Aktör" ADD CONSTRAINT "Film_Aktör_pkey" PRIMARY KEY
("Film_ID", "Aktör_ID");

--
-- Name: Film_Aktör FilmAktör_Film_fkey; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE public."Film_Aktör" ADD CONSTRAINT "FilmAktör_Film_fkey" FOREIGN KEY
("Film_ID") REFERENCES public."Film"("Film_ID");

--
-- Name: Film_Aktör FilmAktör_Aktor_fkey; Type: CONSTRAINT; Schema: public;
Owner: postgres
--

ALTER TABLE public."Film_Aktör" ADD CONSTRAINT "FilmAktör_Aktör_fkey" FOREIGN KEY
("Aktör_ID") REFERENCES public."Aktör"("Aktör_ID");

--
-- Name: Film_Tür Film_Tür_pkey; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE public."Film_Tür" ADD CONSTRAINT "Film_Tür_pkey" PRIMARY KEY
("Film_ID", "Tür_ID");
```

```
--
-- Name: Film_Tür FilmTür_Film_fkey; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE public."Film_Tür" ADD CONSTRAINT "FilmTür_Film_fkey" FOREIGN KEY
("Film_ID") REFERENCES public."Film"("Film_ID");

--
-- Name: Film_Tür FilmTür_Tür_fkey; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE public."Film_Tür" ADD CONSTRAINT "FilmTür_Tür_fkey" FOREIGN KEY
("Tür_ID") REFERENCES public."Tür"("Tür_ID");



CREATE OR REPLACE VIEW public."FilmDetails"
AS
SELECT
    f."Film_ID",
    f."Film_Ad",
    f."Degerlendirme",
    f."Yayin_Yil",
    f."UstundeOrtalama",
    y."Yönetmen_Ad",
    string_agg(DISTINCT t."Tür_Ad", ', ') "Tür_Ad",
    string_agg(DISTINCT a."Aktör_Ad", ', ') "Aktör_Ad"
FROM
    public."Film" f
LEFT JOIN
    public."Yönetmen" y ON f."Yönetmen_ID" = y."Yönetmen_ID"
LEFT JOIN
    public."Film_Tür" ft ON f."Film_ID" = ft."Film_ID"
LEFT JOIN
    public."Tür" t ON ft."Tür_ID" = t."Tür_ID"
LEFT JOIN
    public."Film_Aktör" fa ON f."Film_ID" = fa."Film_ID"
LEFT JOIN
    public."Aktör" a ON fa."Aktör_ID" = a."Aktör_ID"
GROUP BY
    f."Film_ID", f."Film_Ad", f."Degerlendirme", f."Yayin_Yil", y."Yönetmen_Ad";
```

```sql
CREATE OR REPLACE VIEW public."TicketDetails"
AS
SELECT
    m."Müşteri_Ad",
    s."Sinema_Ad",
    f."Film_Ad",
    b."Fiyat",
    CASE
        WHEN f."UstundeOrtalama" == TRUE THEN 'Üstünde'
        ELSE 'Altında'
    END AS "DegerlendirmeDurumu"
FROM
    public."Müşteri" m
JOIN
    public."Bilet" b ON m."Müşteri_ID" = b."Müşteri_ID"
JOIN
    public."Film" f ON b."Film_ID" = f."Film_ID"
JOIN
    public."Salon" s ON f."Film_ID" = s."Film_ID";
```

FONKSİYONLAR VE TRİGGERLER

```sql
--
--SET search_path TO public;
--



-- Bilet fiyatı hesaplayan fonksiyonun oluşturulması
CREATE OR REPLACE FUNCTION biletFiyat(filmId INTEGER)
RETURNS double precision
LANGUAGE plpgsql
AS $function$
DECLARE
```

```
    degerlendirme double precision; -- Filmin değerlendirme puanını tutacak
değişken
    bilet_fiyat double precision; -- Hesaplanacak bilet fiyatını tutacak değişken
BEGIN
    -- Girilen film ID'sine göre filmin değerlendirme puanını çekme
    SELECT "Degerlendirme" INTO degerlendirme
    FROM "Film"
    WHERE "Film_ID" = filmId;

    -- Bilet fiyatını, değerlendirme puanına göre hesaplama
    -- Örneğin, değerlendirme puanı 10 ise bilet fiyatı 10 * 0.5 + 10 = 15 olacak
    bilet_fiyat := degerlendirme * 0.5 + degerlendirme;

    -- Hesaplanan bilet fiyatını döndürme
    RETURN bilet_fiyat;
END;
$function$;




-- Tetikleyici fonksiyonun tanımı, yeni bilet eklendiğinde fiyatı hesaplar
CREATE OR REPLACE FUNCTION biletFiyatGuncelle()
RETURNS TRIGGER AS $$
BEGIN
  -- biletFiyat fonksiyonunu kullanarak fiyatı hesapla ve Bilet tablosunda ilgili
kaydı güncelle
  NEW."Fiyat" := biletFiyat(NEW."Film_ID");

  -- Güncellenmiş Bilet nesnesini döndür
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Tetikleyicinin oluşturulması
CREATE TRIGGER trg_biletFiyatGuncelle
BEFORE INSERT ON public."Bilet" FOR EACH ROW
EXECUTE FUNCTION biletFiyatGuncelle() ;




-- Yeni salona ait koltukları oluşturacak tetikleyici fonksiyonun tanımı
CREATE OR REPLACE FUNCTION koltukOlustur()
RETURNS TRIGGER AS $$
```

```
BEGIN
  -- Salon kapasitesi kadar koltuk oluştur
  FOR i IN 1..NEW."Kapasite" LOOP
    INSERT INTO public."Koltuk" ("Salon_ID") VALUES (NEW."Salon_ID");
  END LOOP;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Tetikleyicinin oluşturulması
CREATE TRIGGER trg_koltukOlustur
AFTER INSERT ON public."Salon" -- Yeni bir salon kaydı oluşturulduktan sonra
tetiklenecek
FOR EACH ROW EXECUTE FUNCTION koltukOlustur(); -- Her kayıt için
create_seats_for_salon fonksiyonunu çağırır.




-- Tetikleyici fonksiyonun tanımı
CREATE OR REPLACE FUNCTION koltukSil()
RETURNS TRIGGER AS $$
BEGIN
  -- 'Salon' silindiğinde aynı 'Salon_ID'ye sahip 'Koltuk'ları sil
  DELETE FROM public."Koltuk" WHERE "Salon_ID" = OLD."Salon_ID";

  -- Eski salon bilgisini dön
  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Tetikleyicinin oluşturulması
CREATE TRIGGER trg_koltukSil
BEFORE DELETE ON public."Salon" -- 'Salon' üzerinde bir silme işlemi yapıldığında
tetiklenecek
FOR EACH ROW EXECUTE FUNCTION koltukSil(); -- Silinen her satır için
'delete_seats_with_salon' fonksiyonunu çalıştır



CREATE OR REPLACE FUNCTION degerlendirmeDurumu()
RETURNS TRIGGER AS $$
DECLARE
    ortalamaDegerlendirme DOUBLE PRECISION;
```

```
BEGIN
    SELECT AVG("Degerlendirme") INTO ortalamaDegerlendirme FROM public."Film";

    IF NEW."Degerlendirme" > ortalamaDegerlendirme THEN
        -- Yeni değerlendirme ortalamanın üstündeyse, UstundeOrtalama TRUE olarak
güncelle
        NEW."UstundeOrtalama" := TRUE;
    ELSE
        -- Değilse FALSE olarak güncelle
        NEW."UstundeOrtalama" := FALSE;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_degerlendirmeDurumu
BEFORE INSERT OR UPDATE OF "Degerlendirme"
ON public."Film"
FOR EACH ROW
EXECUTE FUNCTION degerlendirmeDurumu();



-- PostgreSQL database dump complete
```