



ATILIM UNIVERSITY

2025-2026 FALL SEMESTER

CMPE341-Database Design and Management

FILM-TICKET RESERVATION

Group Members:

Hülya Ceren Lüleci – 25243510021

Deniz Doğan – 23243510019

Ali Çağrı Sepet – 21243510096

Abdijabar Abdalla Haji Ali – 20243510096

Supervisor: Halil Mert Ceylan

Table of Contents

1.	Introduction.....	E
	rror! Bookmark not defined.	
	1.1 Problem Definition and Project Objectives.....	Error! Bookmark not defined.
	1.2 Formation of the Project Team.....	1
	1.3 Definition and Analysis of the Problem.....	1
2.	Requirements.....	E
	rror! Bookmark not defined.	
	2.1 Functional Requirements.....	Error! Bookmark not defined.
	2.2 Non- Functional Requirements.....	3
	2.3 Data Storage and Operations.....	4
3.	ER/EER Diagram of The System.....	5
4.	Logical Design.....	6
5.	Physical Design	10
6.	Interface	16
	References	21

Table of Figures

Figure 1: Conceptual Schema Design.....	5
Figure 2: Logical Data Model.....	6
Figure 3: Checking with Oracle	15
Figure 4: User Interface (UI) Homepage.....	16
Figure 5: Administrator Panel	16
Figure 6: Insert Customer Operation.....	16
Figure 7: Remove Customer Operation	17
Figure 8: Create New Customer Code.....	18
Figure 9: Folder Structure.....	20

1. Introduction

1.1 Problem Definition and Project Objectives

The increasing adoption of online platforms has accelerated the push for digital modernization across many industries, including the entertainment field. Film theaters represent a major segment of this industry, which brings forward the significant requirement for modern film ticket reservation systems. Today, most reservations are still managed manually or through external applications, highlighting the need for a dedicated and efficient system. These methods are lacking in terms of performance effectiveness, informational openness, and user convenience for both customers and system administrators [1].

Our project, titled “Film-Ticket Reservation System,” is designed to address this deficiency by developing an easily accessible database-based system designed to simplify the booking process while offering administrative control features for cinema management. The system will support features such as viewing current film listings, presenting session schedules, submitting reservations, and handling reservations. Additionally, administrators will be able to edit showtime information, update the seating plan, and manage ticket and concession pricing.

1.2 Formation of the Project Team

This project was carried out by four students. Each member participated in all tasks, and progress was maintained through organized discussions, peer review, and an iterative feedback cycle. At the beginning stage, the issue was determined, and an Entity Relationship Diagram (ERD) [2] was developed. Each member first designed a separate version of the ER diagram, and all drafts were analyzed as a group to reveal mismatches and improve the diagram structure. Following a series of revisions, the completed ER structure was produced. Each stage of the project was checked and approved by all members, ensuring its sequential implementation.

1.3 Definition and Analysis of the Problem

Manual and non-automated systems have several shortcomings that negatively affect user experience, such as limited accessibility for customers, seat allocation and booking errors, and ineffective handling of film schedules and showtimes [3]. To resolve these limitations, our proposed system employs a database-supported system that enhances performance and combines essential functions. Administrators will be able to insert, modify, and remove database entries through the platform, while customers can view film schedules, showtimes, and ticket information in a more structured and user-friendly manner.

2. Requirements

At this part by analyzing the projects requirements; users, missions, data and system requirements determined. The requirements are both considered customers interface and the administrator interface. With that collection process:

- Cinema systems that are available
- Communication among team members
- By determining the different users' different usages scenarios

Those are some methods. By the end of that process, we can determine the functions that the system is going to perform, and the technical properties are now consistent.

2.1 Functional Requirements

Customer-Side Requirements

- Users need to see the present film list.
- For all showtime hours and the cinema hall attributes need to be seen.
- User needs to see the available seats for the ones that the user selected.
- System should allow users to select seats and make reservations.
- Users can see and cancel the reservations.
- When the reservation is completed, the system needs to be assigned the related seat as full.

Admin-Side Requirements

- Administrator can add films and new film sessions.
- Administrator can update the film information (film name, type, time, explanation, URL ext.)
- Administrator can delete the present film from the system.
- Administrators can make plans for the ticket prices and promotions.
- Administrators can make cinema hall arrangements or seating plan by updating.
- Administrators can display the reservations list.

System Requirements

- System can handle more than one reservation demand by not any collisions.
- Every reservation calculation needs to be logged in database.
- The seat placements and changes can be seen by the users in real time.

2.2 Non- Functional Requirements

Performance Requirements

- Film and showtime list needs to be installed sometime.
- Database demands need to be fast and optimized.

Usability Requirements

- Interface needs to be developed in a user-friendly manner, soft and not to cause any conflicts.
- It enables access through mobile and desktop devices.

Security Requirements

- Administrator panel only accessible by the allowed ones.
- Reservation information needs to be stored with data security in the database.
- Any unrequired reservations or data manipulations need to be cancelled.

Reliability Requirements

- In case the system faces any errors or warnings, it should be explained explicitly.
- If the provider or the database is closed for any reason the data hierarchy and the organizations must be secured a stored.

Scalability Requirements

- The system must handle the possible increase in cinema halls, films, and consumer amounts.
- The system needs to be designed according to possible film sessions and segment increments.

2.3 Data Storage and Operations

The data that will be stored in that system, basic operations and database requirements can be shown like that [4]:

Stored Data:

- **Film Details:** Film name, Film type, Film explanations, Film duration, Film id
- **Cinema Halls:** Hall id, Name, Capacity
- **Ticket Details:** Ticket price, Showtime details, Film id, Hall id, Ticket id, Seat id
- **Showtime Details:** Start time, Showtime date, Showtime id, Film id, Hall id
- **Customer Details:** Name, Surname, Phone number, Email, Customer id
- **Seats:** Seat number, Hall id, Seat id
- **Consumables:** Consumable name, Price

Once the data is stored, customers can easily watch films scheduled to be shown on a specific date, track the release date, or reserve one or more seats not taken by other customers.

Core Operations:

- Seat occupancy needs to be updated in real time.
- Constraints which protects the data consistency and integrity.
- Backup and logging.

3. ER/EER Diagram of The System

- A Film can optionally have Showtimes, but every Showtime must be connected to one Film (1: N).
- Showtimes can exist independently of Tickets, while every Ticket must belong to a Showtime (1: N).
- Each Hall must include Seats, and every Seat belongs to one Hall (1: N).
- Every Ticket is assigned to one Seat, while a Seat may be unassigned (1:1).
- A Hall may have Tickets, but each Ticket must be related to a Hall (1: N).
- Customers can optionally make purchases; but, all Tickets and Consumables must be purchased by a Customer (1: N).
- Every Showtime occurs in one Hall, while a Hall may exist without a Showtime (N:1).

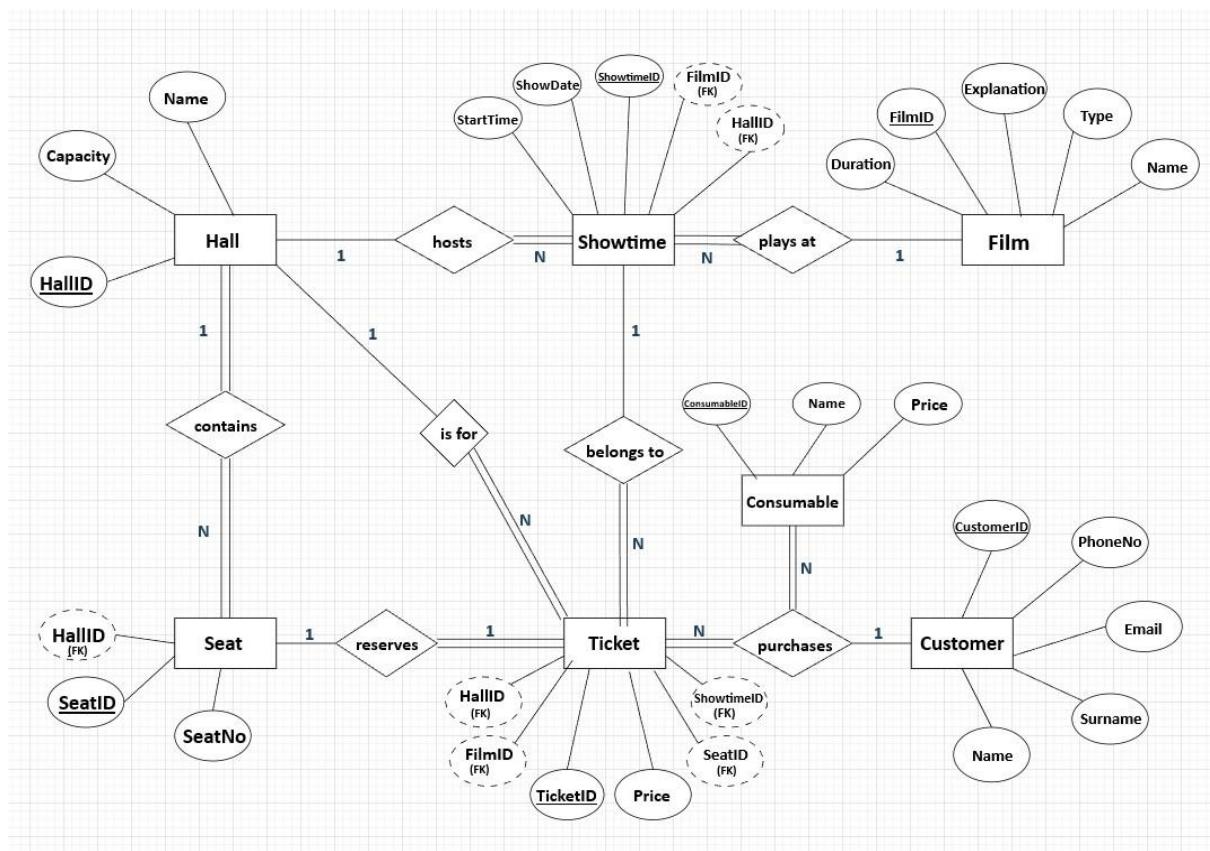


Figure 1: Conceptual Schema Design

4. Logical Design

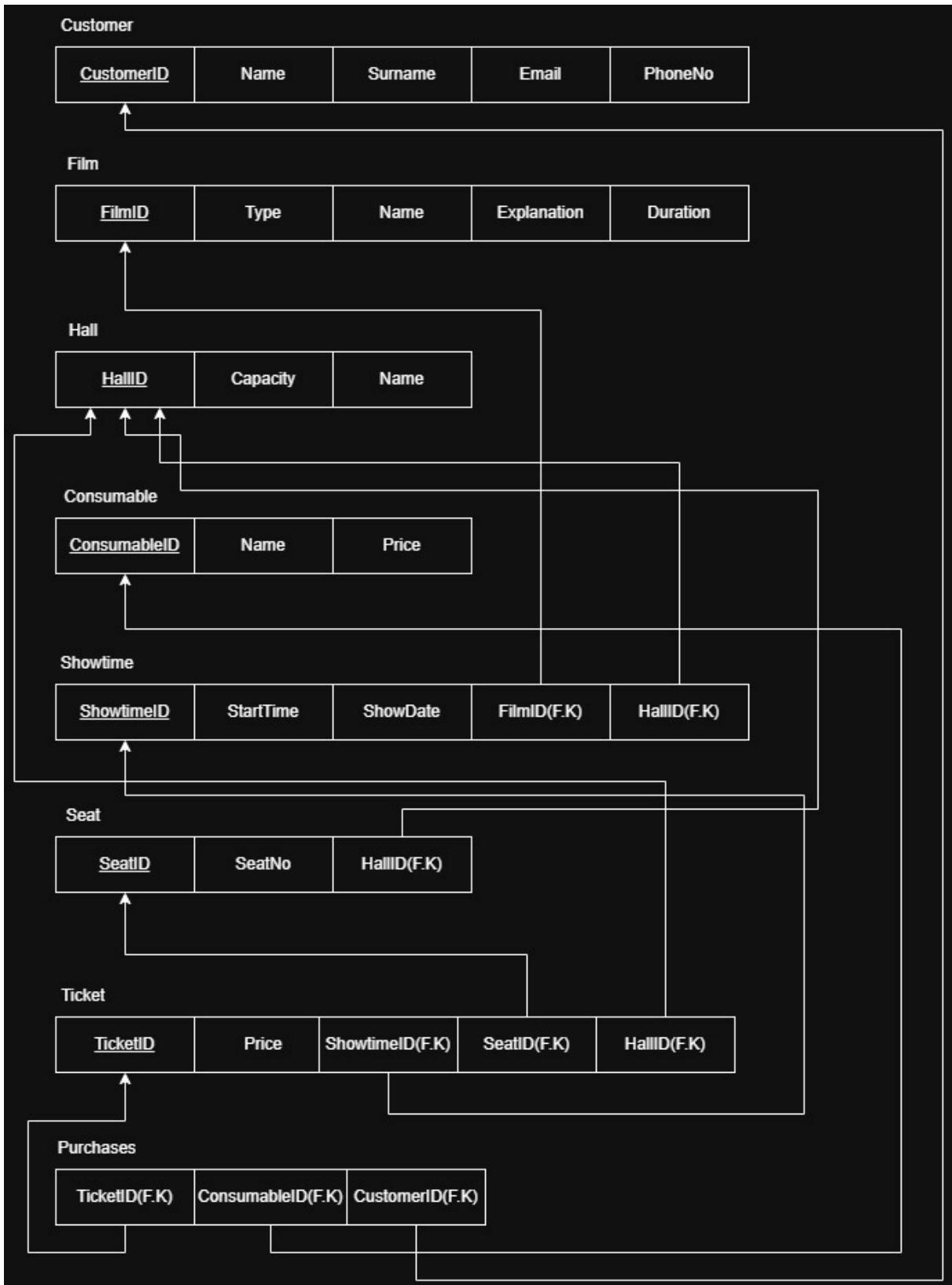


Figure 2: Logical Data Model

In this part, the conceptual ER/EER diagram of the Film-Ticket Reservation System is transformed into a relational database schema. Each entity in the diagram is mapped to a relationship; primary and foreign keys are assigned to demonstrate data integrity and relationship constraints.

Customer

The Customer table stores personal info of customers who purchase tickets or consumables.

Customer

(CustomerID (PK), Name, Surname, Email, PhoneNo);

Film

The Film table contains info about films that are shown in the cinema.

Film

(FilmID (PK), Type, Name, Explanation, Duration);

Hall

The Hall table represents the cinema halls where movies are shown.

Hall

(HallID(PK), Capacity, Name)

Showtime

The Showtime table represents the screening schedule of movies in specific halls.

Showtime

(ShowtimeID (PK), StartTime, ShowDate, FilmID(F.K), HallID(F.K),

FilmID F.K -> Film (FilmID),

HallID F.K -> Hall (HallID));

Seat

The Seat table stores the seat numbers and hall information.

Seat

(SeatID(PK), SeatNo, HallID F.K),

HallID F.K -> Hall (HallID

Ticket

The Ticket table stores ticket info for seat reservations made for a specific showtime.

Ticket

(TicketID (PK), Price,

ShowtimeID F.K -> Showtime (ShowtimeID),

SeatID F.K -> Seat (SeatID),

HallID F.K -> Hall (HallID));

The combination of ShowtimeID and SeatID is used to prevent double booking of the same seat for the same showtimes.

Consumable

The Consumable table contains food and beverage items.

Consumable

(ConsumableID (PK), Name, Price);

Purchases

The Purchases table records customer purchases. A purchase may include a ticket, a consumable, or both.

Purchases

(TicketID F.K -> Ticket (TicketID),

ConsumableID F.K -> Consumable (ConsumableID),

CustomerID F.K -> Customer (CustomerID));

Logical Design Summary

- One Film can have multiple Showtimes, but each showtime is related to exactly one movie.
- Each Hall contains multiple Seats and may host multiple Showtimes.
- Each Ticket is associated with one Showtime, one Seat and one Hall.
- Customers can purchase multiple Tickets and Consumables.
- The Purchases table resolves the relationships between customers, tickets and consumables.

5. Physical Design

```
create table Customer (
    CustomerID number primary key,
    Name varchar (20) not null,
    Surname varchar (20) not null,
    Email varchar (50) unique,
    PhoneNo number (11) unique -- PhoneNo stored as numeric values for simplicity.
);
```

```
create table Film (
    FilmID number primary key,
    Type varchar (20) not null,
    Name varchar (30),
    Explanation varchar (20),
    Duration number
);
```

```
create table Consumable (
    ConsumableID number primary key,
    Name varchar (15) not null,
    Price decimal (5,2)
);
```

```
create table Hall (
    HallID number primary key,
    Capacity number,
    Name varchar (12) not null unique
);
```

```
create table Showtime (
    ShowtimeID number primary key,
    StartTime varchar (6) not null,
    ShowDate date not null,
    FilmID number,
    HallID number,
    foreign key (FilmID) references Film (FilmID),
    foreign key (HallID) references Hall (HallID)
);
```

```
create table Seat (
    SeatID number primary key,
    SeatNo varchar (5) not null,
    HallID number,
    foreign key (HallID) references Hall (HallID)
);
```

```
create table Ticket (
    TicketID number primary key,
    Price decimal (8,2) not null,
    ShowtimeID number,
    SeatID number,
    HallID number,
    foreign key (ShowtimeID) references Showtime (ShowtimeID),
    foreign key (SeatID) references Seat (SeatID),
    foreign key (HallID) references Hall (HallID)
);
```

```
create table Purchases (
    TicketID number,
    ConsumableID number,
    CustomerID number,
    foreign key (TicketID) references Ticket (TicketID),
    foreign key (ConsumableID) references Consumable (ConsumableID),
    foreign key (CustomerID) references Customer (CustomerID)
);
```

CUSTOMER

```
INSERT INTO Customer VALUES (1, 'Hülya Ceren', 'Lüleci', 'ceren.luleci@atilim.edu.tr',  
501111111111);
```

```
INSERT INTO Customer VALUES (2, 'Deniz', 'Doğan', 'deniz.dogan@atilim.edu.tr',  
501222222222);
```

```
INSERT INTO Customer VALUES (3, 'Ali Çağrı', 'Sepet', 'ali.sepet@atilim.edu.tr',  
501333333333);
```

```
INSERT INTO Customer VALUES (4, 'Abdijabar', 'Haji Ali', 'abdijabar.ali@atilim.edu.tr',  
501444444444);
```

FILM

```
INSERT INTO Film VALUES (1, 'Sci-Fi', 'Inception', 'Dream', 148);
```

```
INSERT INTO Film VALUES (2, 'Sci-Fi', 'Interstellar', 'Space', 169);
```

```
INSERT INTO Film VALUES (3, 'Comedy', 'The Mask', 'Fun', 101);
```

HALL

```
INSERT INTO Hall VALUES (1, 100, 'H1');
```

```
INSERT INTO Hall VALUES (2, 80, 'H2');
```

```
INSERT INTO Hall VALUES (3, 120, 'H3');
```

SHOWTIME

```
INSERT INTO Showtime VALUES (1, '18:00', DATE '2025-12-20', 1, 1);
```

```
INSERT INTO Showtime VALUES (2, '21:00', DATE '2025-12-20', 2, 2);
```

```
INSERT INTO Showtime VALUES (3, '20:00', DATE '2025-12-21', 3, 3);
```

SEAT

```
INSERT INTO Seat VALUES (1, 'A1', 1);
```

```
INSERT INTO Seat VALUES (2, 'A2', 1);
```

```
INSERT INTO Seat VALUES (3, 'B1', 2);
```

TICKET

```
INSERT INTO Ticket VALUES (1, 120.00, 1, 1, 1);
```

```
INSERT INTO Ticket VALUES (2, 120.00, 1, 2, 1);
```

```
INSERT INTO Ticket VALUES (3, 100.00, 2, 3, 2);
```

CONSUMABLE

```
INSERT INTO Consumable VALUES (1, 'Popcorn', 50.00);
```

```
INSERT INTO Consumable VALUES (2, 'Cola', 30.00);
```

```
INSERT INTO Consumable VALUES (3, 'Nachos', 60.00);
```

PURCHASES

```
INSERT INTO Purchases VALUES (1, NULL, 1);
```

```
INSERT INTO Purchases VALUES (2, 2, 2);
```

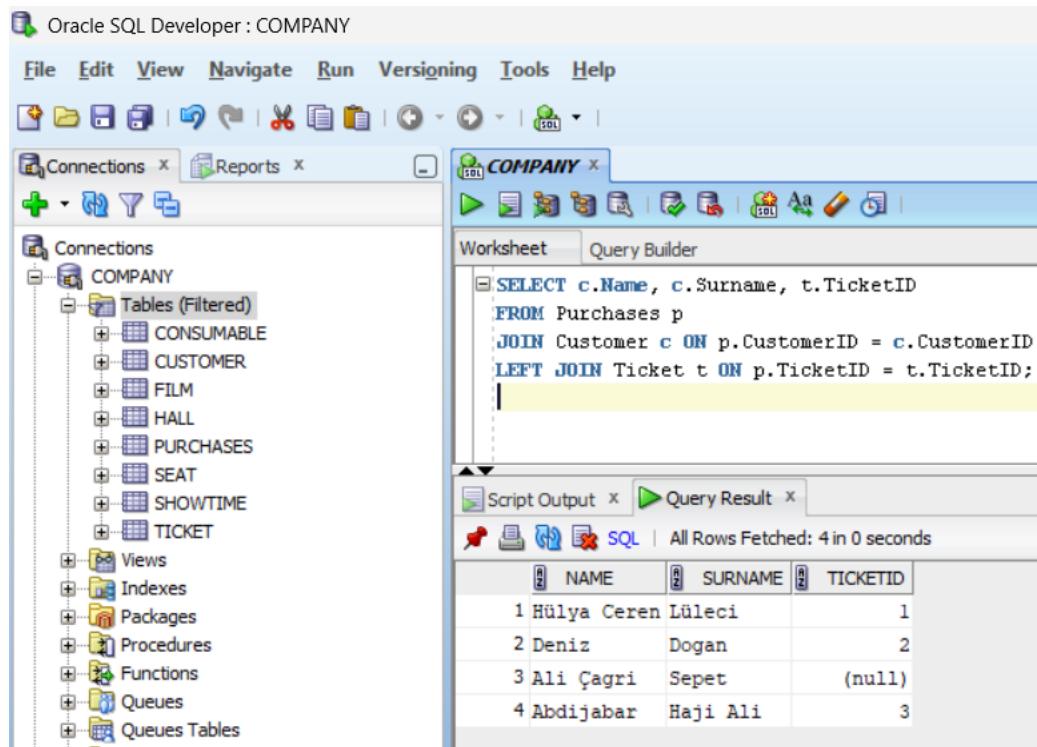
```
INSERT INTO Purchases VALUES (NULL, 1, 3);
```

```
INSERT INTO Purchases VALUES (3, NULL, 4);
```

Check:

```
SELECT c.Name, c.Surname, t.TicketID  
FROM Purchases p  
JOIN Customer c ON p.CustomerID = c.CustomerID  
LEFT JOIN Ticket t ON p.TicketID = t.TicketID;
```

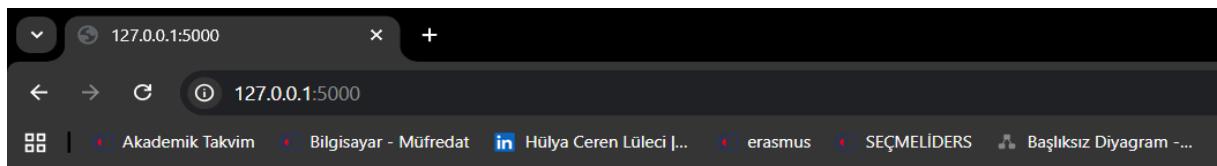
Figure 3: Checking with Oracle



6. Interface

- The user interface of the project is implemented as a web-based interface using HTML and Flask.
- The interface allows the user to interact with the Oracle database through simple forms.
- Users can add new customers and remove existing customers from the database.
- The interface dynamically displays the current state of the database.
- The homepage of the Film-Ticket Reservation System provides a simple entry point for administrators to access customer and ticket management operations.

URL: <http://127.0.0.1:5000/>



Welcome to Film-Ticket Reservation System

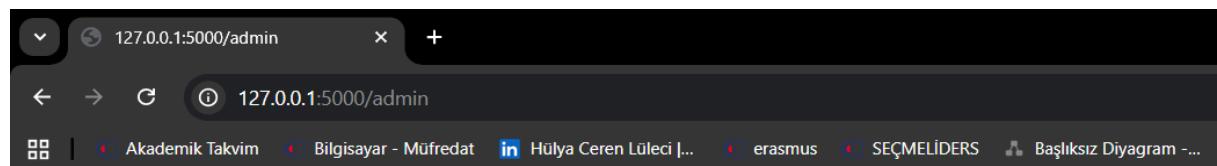
This web-based interface allows administrators to manage customer data and interact with the Oracle database in real time.

[Go to Admin Panel](#)

Figure 4: User Interface (UI) Homepage

- The administrator panel enables authorized users to manage customer-related operations in the Film-Ticket Reservation System. Through this interface, administrators can insert new customers, delete existing customers, and navigate back to the homepage.

URL: <http://127.0.0.1:5000/admin>



Administrator Panel

This panel allows the administrator to manage customer-related operations within the Film-Ticket Reservation System.

- [Add New Customer](#)
- [Remove Existing Customer](#)

[Return to Homepage](#)

Figure 5: Administrator Panel

Figure 6: Insert Customer Operation

127.0.0.1:5000/add_customer

Add New Customer

This page allows the administrator to insert a new customer into the Film-Ticket Reservation System database.

Name: Halil Mert

Surname: Ceylan

Email: mert.ceylan@atilim.edu.tr

Phone: 5051234568

Existing Customers

The table below displays all customers currently stored in the database.

ID	Name	Surname	Email	Phone
1	Hülya Ceren	Lüleci	ceren.luleci@atilim.edu.tr	501111111111
2	Deniz	Dogan	deniz.dogan@atilim.edu.tr	501222222222
3	Ali Çagri	Sepet	ali.sepет@atilim.edu.tr	501333333333
4	Abdijabar	Haji Ali	abdijabar.ali@atilim.edu.tr	501444444444

[Return to Admin Panel](#)

127.0.0.1:5000/add_customer

Add New Customer

This page allows the administrator to insert a new customer into the Film-Ticket Reservation System database.

Name:

Surname:

Email:

Phone:

Existing Customers

The table below displays all customers currently stored in the database.

ID	Name	Surname	Email	Phone
1	Hülya Ceren	Lüleci	ceren.luleci@atilim.edu.tr	501111111111
2	Deniz	Dogan	deniz.dogan@atilim.edu.tr	501222222222
3	Ali Çagri	Sepet	ali.sepет@atilim.edu.tr	501333333333
4	Abdijabar	Haji Ali	abdijabar.ali@atilim.edu.tr	501444444444
5	Halil Mert	Ceylan	mert.ceylan@atilim.edu.tr	5051234568

[Return to Admin Panel](#)

Figure 7: Remove Customer Operation

Remove Customer Operation with Integrity Constraint:

Remove Existing Customer

This page allows the administrator to remove an existing customer from the Film-Ticket Reservation System database by using the customer ID.

Customer ID:

Existing Customers

The table below displays the list of customers currently stored in the database, which helps the administrator identify the correct customer ID.

ID	Name	Surname
1	Hülya Ceren	Lüleci
2	Deniz	Dogan
3	Ali Çagri	Sepet
4	Abdijabar	Haji Ali
5	Halil Mert	Ceylan

[Return to Admin Panel](#)

oracledb.exceptions.IntegrityError

oracledb.exceptions.IntegrityError: ORA-02292: integrity constraint (HR.SYS_C004362) violated - child record found

Traceback (most recent call last)

```

File "C:\Users\pcdell\AppData\Local\Programs\Python\Python313\Lib\site-packages\flask\app.py", line 1536, in __call__
    return self.wsgi_app(environ, start_response)
       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

File "C:\Users\pcdell\AppData\Local\Programs\Python\Python313\Lib\site-packages\flask\app.py", line 1514, in wsgi_app
    response = self.handle_exception(e)
       ^^^^^^^^^^^^^^^^^^^^^^^^^^

File "C:\Users\pcdell\AppData\Local\Programs\Python\Python313\Lib\site-packages\flask\app.py", line 1511, in wsgi_app
    response = self.full_dispatch_request()
       ^^^^^^^^^^^^^^^^^^^^^^^^^^

File "C:\Users\pcdell\AppData\Local\Programs\Python\Python313\Lib\site-packages\flask\app.py", line 919, in full_dispatch_request
    rv = self.handle_user_exception(e)
       ^^^^^^^^^^^^^^^^^^^^^^

File "C:\Users\pcdell\AppData\Local\Programs\Python\Python313\Lib\site-packages\flask\app.py", line 917, in full_dispatch_request
    rv = self.dispatch_request()
       ^^^^^^^^^^^^^^^^^^

File "C:\Users\pcdell\AppData\Local\Programs\Python\Python313\Lib\site-packages\flask\app.py", line 902, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args) # type: ignore[no-any-return]
       ^^^^^^^^^^^^^^^^^^

File "C:\Users\pcdell\Desktop\cmpe341_interface\app.py", line 122, in remove_customer
    cur.execute("DELETE FROM Customer WHERE CustomerID = :1", [cid])
       ^^^^^^

File "C:\Users\pcdell\AppData\Local\Programs\Python\Python313\Lib\site-packages\oracledb\cursor.py", line 844, in execute
    impl.execute(self)

```

This figure demonstrates the system's integrity control mechanism. When an administrator attempts to remove a customer who has related purchase records, the database prevents the deletion operation by enforcing foreign key constraints. This ensures data consistency.

127.0.0.1:5000/remove_customer

Customer ID: Remove Customer

Remove Existing Customer

This page allows the administrator to remove an existing customer from the Film-Ticket Reservation System database by using the customer ID.

Customer ID: Remove Customer

Existing Customers

The table below displays the list of customers currently stored in the database, which helps the administrator identify the correct customer ID.

ID	Name	Surname
1	Hülya Ceren	Lüleci
2	Deniz	Dogan
3	Ali Çagri	Sepet
4	Abdijabar	Haji Ali
5	Halil Mert	Ceylan

[Return to Admin Panel](#)

Removed Customer ID: 5

127.0.0.1:5000/remove_customer

Remove Existing Customer

This page allows the administrator to remove an existing customer from the Film-Ticket Reservation System database by using the customer ID.

Customer ID: Remove Customer

Existing Customers

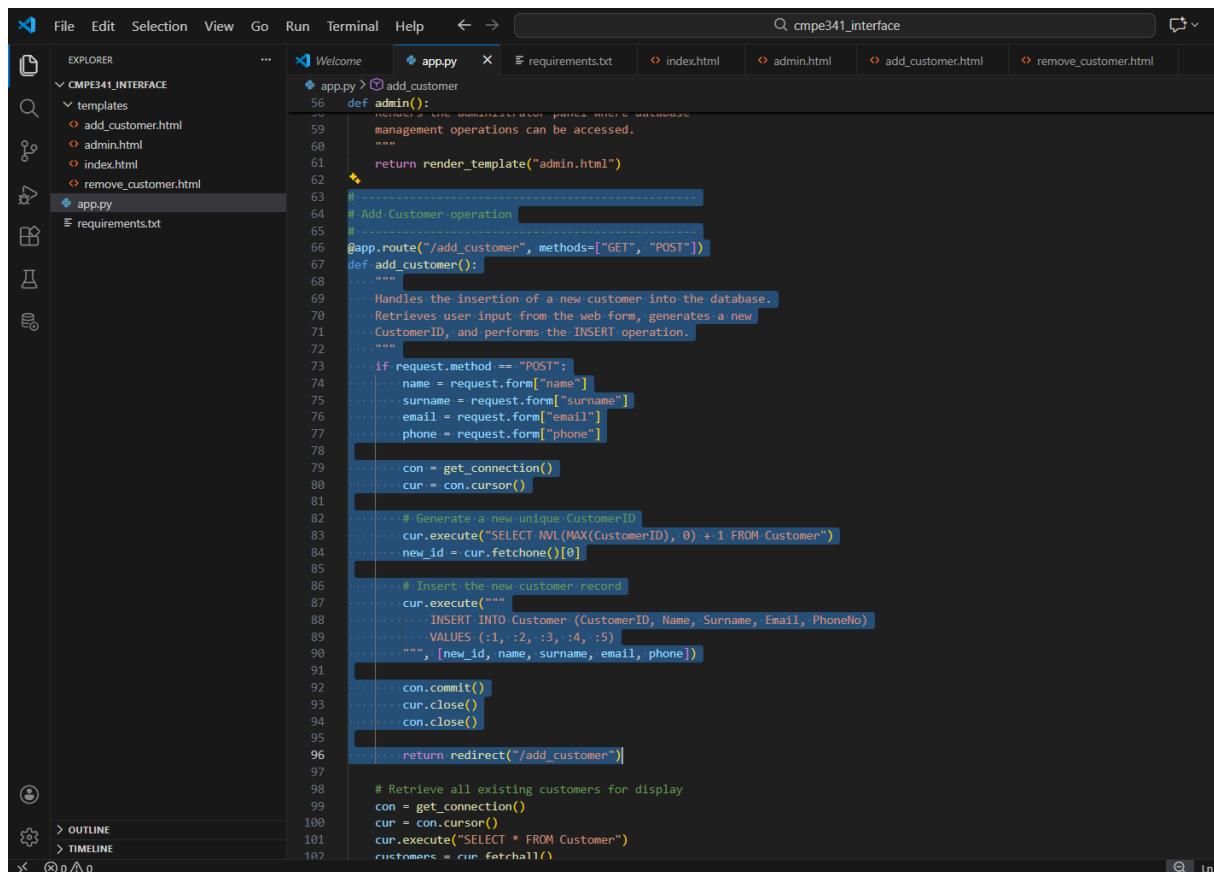
The table below displays the list of customers currently stored in the database, which helps the administrator identify the correct customer ID.

ID	Name	Surname
1	Hülya Ceren	Lüleci
2	Deniz	Dogan
3	Ali Çagri	Sepet
4	Abdijabar	Haji Ali

[Return to Admin Panel](#)

Figure 8: Create New Customer Code

Figure 8 illustrates the backend implementation of the “Add Customer” operation. The Flask route retrieves user input from the web form, generates a new customer ID, and inserts the customer data into the Oracle database using SQL commands. This operation ensures dynamic interaction between the user interface and the database.



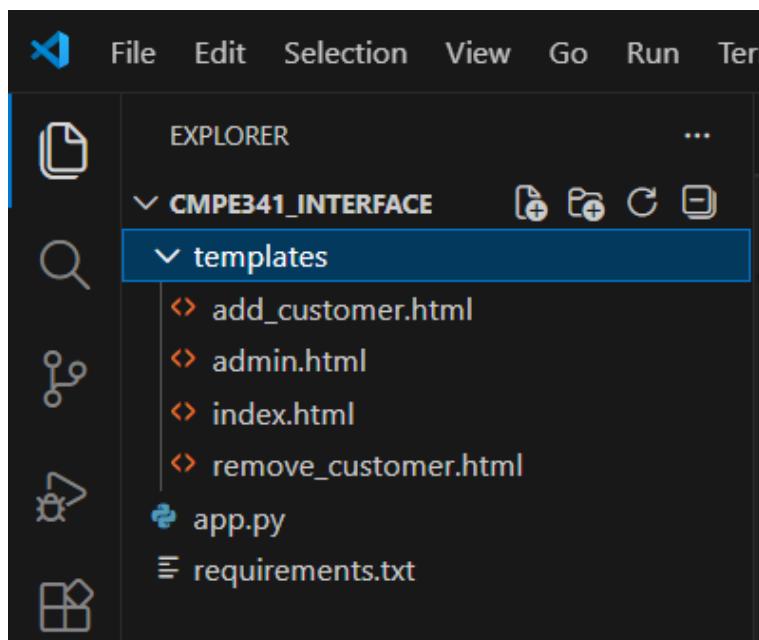
```
File Edit Selection View Go Run Terminal Help ← → cmpe341_interface
EXPLORER CMPE341_INTERFACE ...
templates
  add_customer.html
  admin.html
  index.html
  remove_customer.html
app.py
requirements.txt

Welcome app.py add_customer requirements.txt index.html admin.html add_customer.html remove_customer.html

56 def admin():
57     """
58     management operations can be accessed.
59     """
60     return render_template("admin.html")
61
62 # -----
63 # Add Customer operation
64 # -----
65
66 @app.route("/add_customer", methods=["GET", "POST"])
67 def add_customer():
68     """
69     Handles the insertion of a new customer into the database.
70     Retrieves user input from the web form, generates a new
71     CustomerID, and performs the INSERT operation.
72     """
73     if request.method == "POST":
74         name = request.form["name"]
75         surname = request.form["surname"]
76         email = request.form["email"]
77         phone = request.form["phone"]
78
79         con = get_connection()
80         cur = con.cursor()
81
82         # Generate a new unique CustomerID
83         cur.execute("SELECT NVL(MAX(CustomerID), 0) + 1 FROM Customer")
84         new_id = cur.fetchone()[0]
85
86         # Insert the new customer record
87         cur.execute("""
88             INSERT INTO Customer (CustomerID, Name, Surname, Email, PhoneNo)
89             VALUES (:1, :2, :3, :4, :5)
90             """, [new_id, name, surname, email, phone])
91
92         con.commit()
93         cur.close()
94         con.close()
95
96     return redirect("/add_customer")
97
98     # Retrieve all existing customers for display
99     con = get_connection()
100    cur = con.cursor()
101    cur.execute("SELECT * FROM Customer")
102    customers = cur.fetchall()

102
```

Figure 9: Folder Structure



References

- [1] M. S. Ali and N. Ahmed, “Design of Database Schema for Online Ticket Booking Systems,” *International Journal of Information Technology*, vol. 12, no. 2, pp. 389–395, 2020.
- [2] T. S. Nguyen, “A lightweight database design for online ticketing platforms,” *Asian Journal of Computing and Information Systems*, vol. 4, no. 3, pp. 101–106, 2019.
- [3] M. Al-Hassan and S. Qureshi, “Database-driven ticket reservation system for entertainment venues,” *Journal of Applied Information Technology*, vol. 5, no. 2, pp. 41–47, 2018.
- [4] R. D. Silva and M. P. Fernando, “Database structure optimization for online ticket reservation systems,” *Journal of Software Systems and Applications*, vol. 5, no. 4, pp. 72–78, 2020.