

ISTANBUL TECHNICAL UNIVERSITY

BLG317E

Database Systems

ITUCSDB2204 Project Documentation

150190915 : Yasmeeen Abdelghany

150180802 : Mehmet Furkan Uyanık

Fall 2022

Contents

| | | |
|----------|--|-----------|
| 1 | User Guide | 1 |
| 1.1 | Parts Implemented by Yasmeeen Abdelghany | 2 |
| 1.1.1 | Sign up / Log in page | 2 |
| 1.1.2 | Film Page | 4 |
| 1.1.3 | How to add a new film | 4 |
| 1.1.4 | How to update a film | 6 |
| 1.1.5 | How to delete a film | 7 |
| 1.2 | Parts Implemented by Mehmet Furkan Uyanık | 9 |
| 1.2.1 | Staff Page | 9 |
| 1.2.2 | Add New Staff | 9 |
| 1.2.3 | Deleting A Staff | 12 |
| 1.2.4 | Updating A Staff | 13 |
| 2 | Developer Guide | 14 |
| 2.1 | Parts Implemented by Yasmeeen Abdelghany | 14 |
| 2.1.1 | Users table | 14 |
| 2.1.2 | Film table | 14 |
| 2.1.3 | Film.category table | 15 |
| 2.1.4 | Category table | 15 |
| 2.1.5 | Language table | 16 |
| 2.1.6 | Sign up | 16 |
| 2.1.7 | Log in | 18 |
| 2.1.8 | Display table codes | 19 |
| 2.1.9 | Insert operation codes | 21 |
| 2.1.10 | Update operation codes | 28 |
| 2.1.11 | Delete operation codes | 33 |
| 2.2 | Parts Implemented by Mehmet Furkan Uyanık | 33 |
| 2.2.1 | Staff Page | 33 |
| 2.2.2 | Reading Operations for Staff Page | 34 |
| 2.2.3 | Dynamic Selection of City Store and Home Addresses | 35 |
| 2.2.4 | Adding New Staff | 39 |
| 2.2.5 | Deleting a Staff | 43 |
| 2.2.6 | Updating Staff | 44 |

1 User Guide

The Sakila database is a nicely normalized schema modeling a DVD rental store, featuring things like films, actors, film-actor relationships, and a central inventory table that connects films, stores, and rentals.

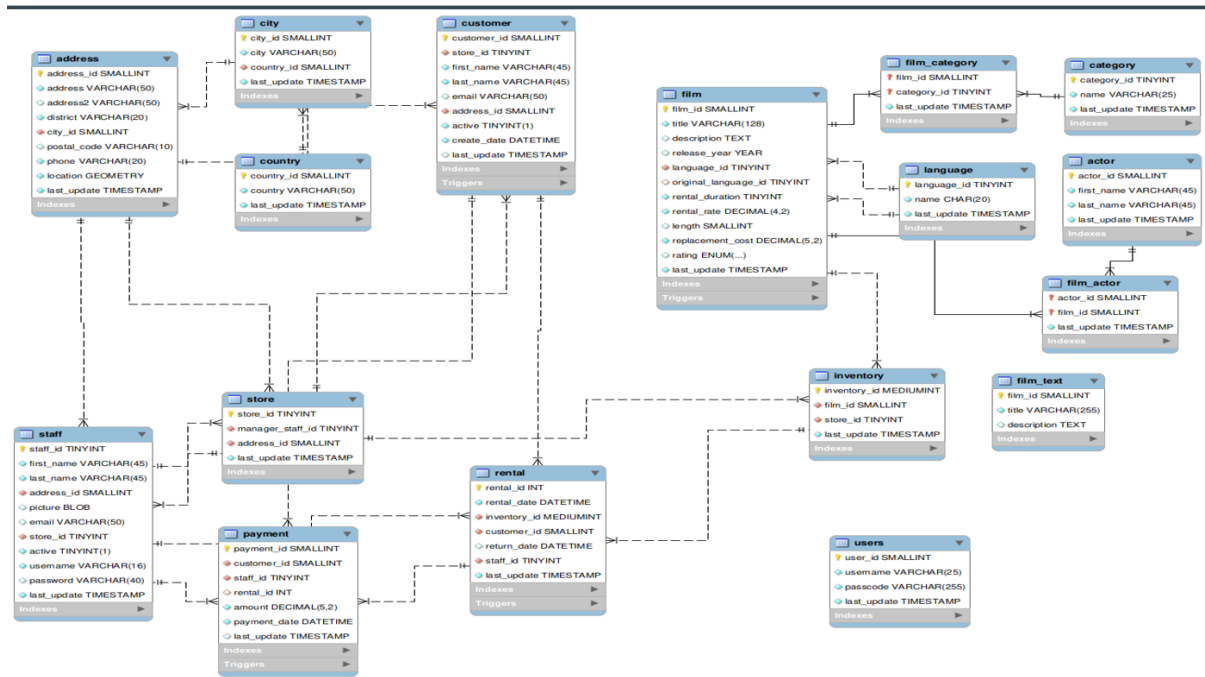


Figure 1: The ER Diagram

1.1 Parts Implemented by Yasmeen Abdelghany

1.1.1 Sign up / Log in page

The user must first sign up or log in if he/she is already a member of the store to be able to access the web pages. If the user is signing up for the first time, a new membership will be created or the user can log in using his/her own username and password instead.

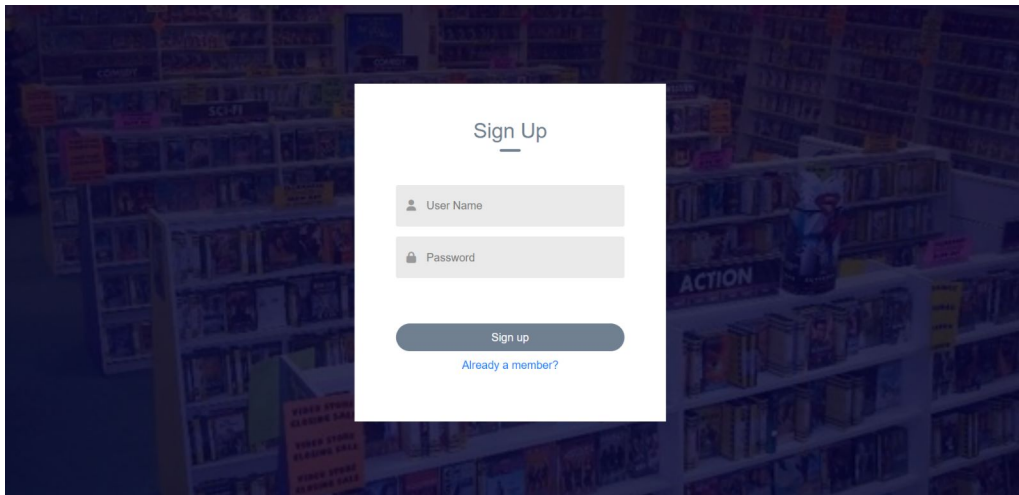


Figure 2: Sign Up page of the application

In the signup page shown above, the new user must add a valid username and password of at least 7 characters each and hit the submit button. If the username is not taken, the user will be redirected to the home page shown in Figure 4 below. The user can't access anything on the website unless he/she signs up or logs in first. Otherwise, the user will be alerted that he/she did not enter a username or a password that is at least 7 characters, or that the username already exists.

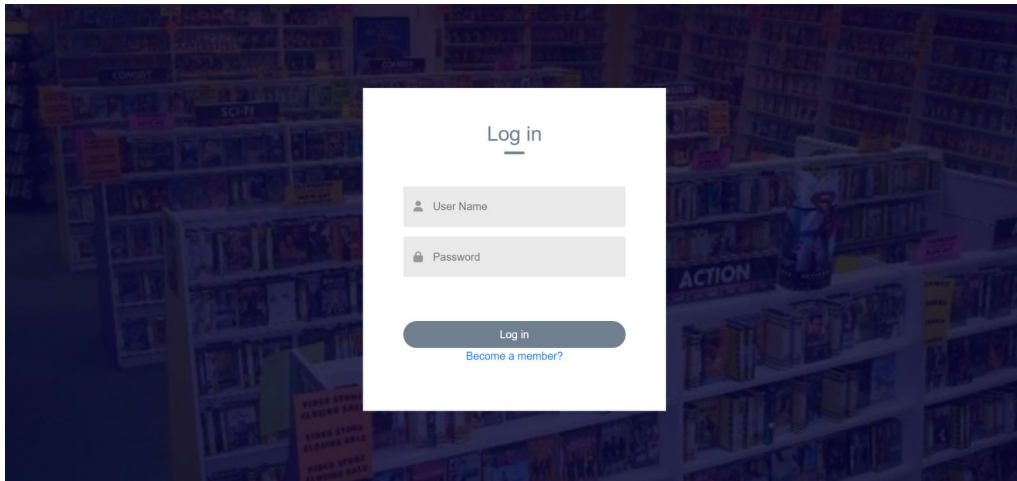


Figure 3: log in page of the application

In the login page shown above, the user must enter his/her username and password that was created previously and then hit the submit button. If the username actually exists and the password matches, the user will be redirected to the home page shown in Figure 4 below, and he/she will be able to access everything on the website. Otherwise, the user will be alerted that he/she did not enter the right username or that the password is wrong.

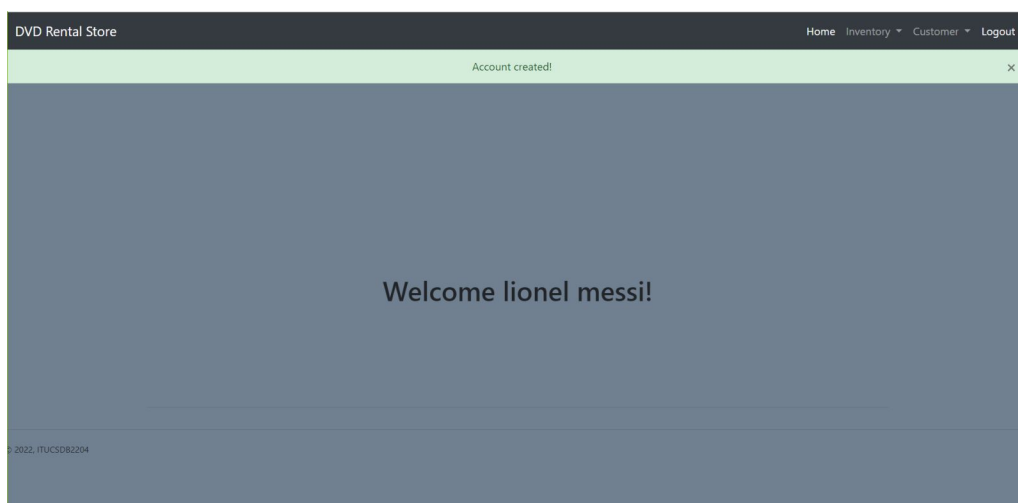
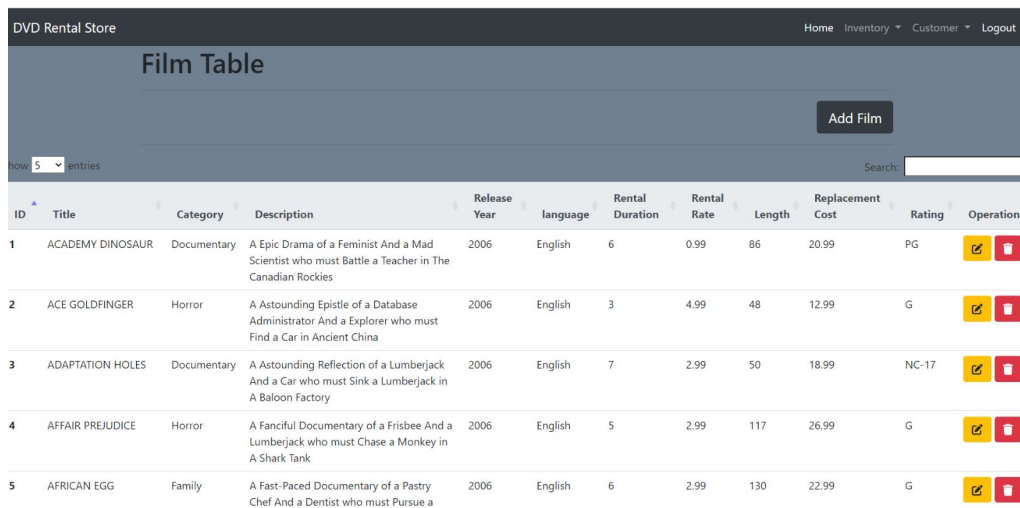


Figure 4: Home page of the application

Once the user enters, a flash message appears and a welcome text with the user's name appears. To log out from the website, the user can click on the logout button found on the top right side of the webpage, and he/she will be redirected to the login page again.

1.1.2 Film Page

In the inventory button found on the navigation bar of the home page, the film page can be accessed. The film page combines the film, film_category, category, and language tables of our database. As shown in the figure below, A table of all the records taken from all 4 database tables combined is displayed. New films can be added by clicking on the Add new button found above the table. Additionally, each film that already exists can be edited or deleted using the edit or delete button found under the operations column.













| ID | Title | Category | Description | Release Year | language | Rental Duration | Rental Rate | Length | Replacement Cost | Rating | Operation |
|----|------------------|-------------|--|--------------|----------|-----------------|-------------|--------|------------------|--------|---|
| 1 | ACADEMY DINOSAUR | Documentary | A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies | 2006 | English | 6 | 0.99 | 86 | 20.99 | PG |   |
| 2 | ACE GOLDFINGER | Horror | A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China | 2006 | English | 3 | 4.99 | 48 | 12.99 | G |   |
| 3 | ADAPTATION HOLES | Documentary | A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory | 2006 | English | 7 | 2.99 | 50 | 18.99 | NC-17 |   |
| 4 | AFFAIR PREJUDICE | Horror | A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank | 2006 | English | 5 | 2.99 | 117 | 26.99 | G |   |
| 5 | AFRICAN EGG | Family | A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a | 2006 | English | 6 | 2.99 | 130 | 22.99 | G |   |

Figure 5: Adding a new film

1.1.3 How to add a new film

When the Add New button is clicked, a pop-up window or a modal will appear as in the figure below. In this window, input text area, numbers, and selective options are available for the user to enter the information needed to add an additional film to the store.

| ID | Title | Category |
|----|------------------|-------------|
| 1 | ACADEMY DINOSAUR | Documentary |
| 2 | ACE GOLDFINGER | Horror |
| 3 | ADAPTATION HOLES | Documentary |
| 4 | AFFAIR PREJUDICE | Horror |
| 5 | AFRICAN EGG | Family |

Figure 6: Adding a new film, language, and category

If the user wants to add a different category or language, he/she must select the --Other-- option, and a new input text area will appear as shown in the figure below. Writing the new option in the text box is enough and it will be permanently added to the options.

| ID | Title | Category |
|----|------------------|-------------|
| 1 | ACADEMY DINOSAUR | Documentary |
| 2 | ACE GOLDFINGER | Horror |
| 3 | ADAPTATION HOLES | Documentary |
| 4 | AFFAIR PREJUDICE | Horror |
| 5 | AFRICAN EGG | Family |

Figure 7: Operation is successful!

The user can't submit the form unless he/she fills in all the necessary fields. Once the Add button is clicked, the user will be alerted if the insertion operation in the film, film_category, category, or language tables was successful or not as shown in the figure below. The newly added record can be seen at the very end of the table. By clicking on the order option of the ID column, the IDs will be ordered in a descending way and the last newly added record should be seen.

The screenshot shows the 'DVD Rental Store' home page. At the top, there are navigation links: Home, Inventory, Customer, and Logout. Below the navigation bar, there are three green success messages: 'New category added successfully', 'Data inserted into film table successfully!', and 'Data inserted into film_category table successfully!'. The main section is titled 'Film Table' and contains an 'Add Film' button. Below the button is a table with the following columns: ID, Title, Category, Description, Release Year, language, Rental Duration, Rental Rate, Length, Replacement Cost, Rating, and Operation. The table contains three rows of film data.

| ID | Title | Category | Description | Release Year | language | Rental Duration | Rental Rate | Length | Replacement Cost | Rating | Operation |
|----|------------------|-------------|--|--------------|----------|-----------------|-------------|--------|------------------|--------|-----------------|
| 1 | ACADEMY DINOSAUR | HEHEHE | A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies | 2006 | Latino | 6 | 0.99 | 86 | 20.99 | NC-17 | [Edit] [Delete] |
| 2 | ACE GOLDFINGER | Horror | A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China | 2006 | English | 3 | 4.99 | 48 | 12.99 | G | [Edit] [Delete] |
| 3 | ADAPTATION HOLES | Documentary | A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Balloon Factory | 2006 | English | 7 | 2.99 | 50 | 18.99 | NC-17 | [Edit] [Delete] |

Figure 8: Home page of the application

1.1.4 How to update a film

To edit the information of a film, the edit button of the corresponding film should be clicked. A pop-up window similar to the one in the previous section appears, but, this time, it is filled up with the values from the film to be edited.

The screenshot shows the 'Update Film' pop-up window. The window has a title bar 'Update Film' and a close button. It contains the following fields: Title (ACADEMY DINOSAUR), Description (A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies), Release Year (2006), Length (86), Rental Duration (6), Rental Rate (0.99), Replacement Cost (20.99), Category (Action), and Language (English). There are also radio buttons for Rating: G, PG, PG-13, R, NC-17. At the bottom, there are 'Cancel' and 'Save Edits' buttons. The background shows the 'Film Table' with the same data as in Figure 8.

Figure 9: Updating film_ID1

Again, the user can edit the category and language of the film by selecting one of the options or adding a new one by selecting the other option and typing it in the input text box as shown below.

Update Film

Title: ACADEMY DINOSAUR

Description: A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies

Release Year: 2006 Length: 86 Rental Duration: 6

Rental Rate: 0.99 Replacement Cost: 20.99

Category: --Other-- Language: --Other--

Rating: ☐ G ☐ PG ☐ PG-13 ☐ R ☐ NC-17

Cancel Save Edits

Figure 10: Updating the film with a new language and category

The user can't submit the form unless he/she fills in all the necessary fields. Once the Save Edits button is clicked, the user will be alerted if the update operation in the film and film_category tables was successful or not as shown in the figure below. If a new language or category was added as well, a flash message will appear.

Film Table

Add Film

| ID | Title | Category | Description | Release Year | language | Rental Duration | Rental Rate | Length | Replacement Cost | Rating | Operation |
|----|------------------|-------------|--|--------------|----------|-----------------|-------------|--------|------------------|--------|-----------------|
| 1 | ACADEMY DINOSAUR | HEHEHE | A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies | 2006 | Latino | 6 | 0.99 | 86 | 20.99 | NC-17 | [Edit] [Delete] |
| 2 | ACE GOLDFINGER | Horror | A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China | 2006 | English | 3 | 4.99 | 48 | 12.99 | G | [Edit] [Delete] |
| 3 | ADAPTATION HOLES | Documentary | A Saturation Reflection of a Lumberjack And a Canadian must Sink a | 2006 | English | 7 | 2.99 | 60 | 18.99 | NC-17 | [Edit] [Delete] |

Figure 11: Operation is successful!

1.1.5 How to delete a film

To delete a film, the delete button of the corresponding row should be clicked and a pop-up alert message will appear to ask the user for confirmation before deleting as shown in the figure below.

DVD Rental Store

localhost:8080 web sitesinin mesajı

Are you sure you want to delete?

Tayama İptal

Home Inventory Customer Logout

Film Table

Show 5 entries

Add Film

| ID | Title | Category | Description | Release Year | language | Rental Duration | Rental Rate | Length | Replacement Cost | Rating | Operation |
|----|------------------|-------------|---|--------------|----------|-----------------|-------------|--------|------------------|--------|-----------|
| 3 | ADAPTATION HOLES | Documentary | A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Balloon Factory | 2006 | English | 7 | 2.99 | 50 | 18.99 | NC-17 | |
| 4 | AFFAIR PREJUDICE | Horror | A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank | 2006 | English | 5 | 2.99 | 117 | 26.99 | G | |
| 5 | AFRICAN EGG | Family | A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in The Gulf of Mexico | 2006 | English | 6 | 2.99 | 130 | 22.99 | G | |
| 6 | AGENT TRUMAN | Foreign | A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China | 2006 | English | 3 | 2.99 | 169 | 17.99 | PG | |
| 7 | AIRPLANE SIERRA | Comedy | A Touching Saga of a Hunter And a Butler | 2006 | English | 6 | 4.99 | 62 | 28.99 | PG-13 | |

Figure 12: Are you sure you want to delete this film window

Once confirmed, the film will be deleted from this table and a flash message will also appear to show if the operation was successful or not.

DVD Rental Store

Home Inventory Customer Logout

The film was deleted successfully!

Film Table

Show 5 entries

Add Film

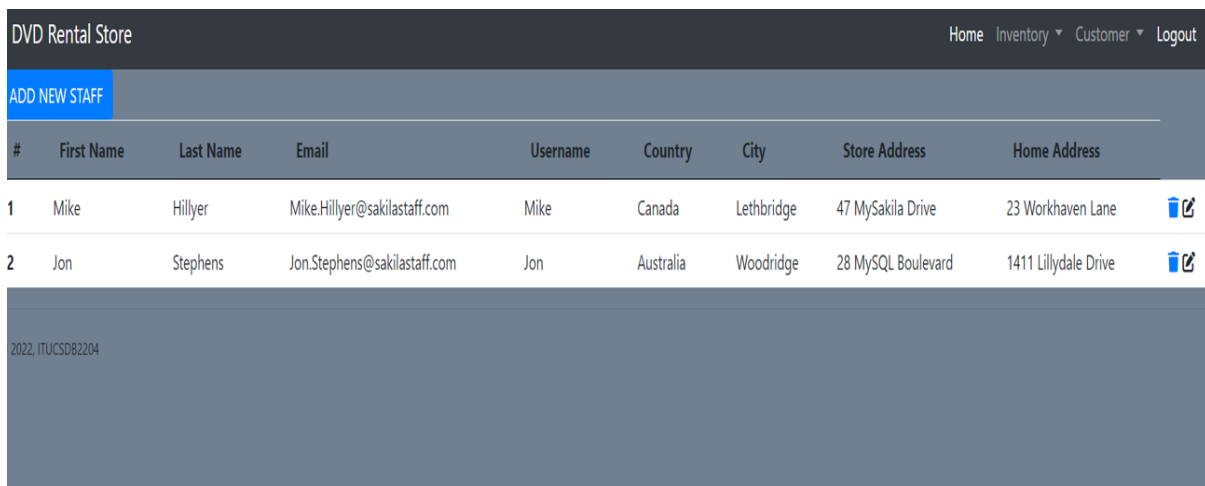
| ID | Title | Category | Description | Release Year | language | Rental Duration | Rental Rate | Length | Replacement Cost | Rating | Operation |
|----|------------------|-------------|---|--------------|----------|-----------------|-------------|--------|------------------|--------|-----------|
| 3 | ADAPTATION HOLES | Documentary | A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Balloon Factory | 2006 | English | 7 | 2.99 | 50 | 18.99 | NC-17 | |
| 4 | AFFAIR PREJUDICE | Horror | A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank | 2006 | English | 5 | 2.99 | 117 | 26.99 | G | |
| 5 | AFRICAN EGG | Family | A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in The Gulf of Mexico | 2006 | English | 6 | 2.99 | 130 | 22.99 | G | |
| 6 | AGENT TRUMAN | Foreign | A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in | 2006 | English | 3 | 2.99 | 169 | 17.99 | PG | |





Figure 13: Operation is successful!

1.2 Parts Implemented by Mehmet Furkan Uyanık

1.2.1 Staff Page

This page shows staff members, including information for email address, country, city, store address and home address informations. There four select options for country, city, store and home address. The country is from country table, city is from city table, store is from store table and home address is from address table. User can add new staffs to existing stores. If a store does not exist, user can not be add new staff to it. The user can also delete and update the staff.



| # | First Name | Last Name | Email | Username | Country | City | Store Address | Home Address | |
|---|------------|-----------|------------------------------|----------|-----------|------------|--------------------|----------------------|---|
| 1 | Mike | Hillyer | Mike.Hillyer@sakilastaff.com | Mike | Canada | Lethbridge | 47 MySakila Drive | 23 Workhaven Lane |   |
| 2 | Jon | Stephens | Jon.Stephens@sakilastaff.com | Jon | Australia | Woodridge | 28 MySQL Boulevard | 1411 Lillydale Drive |   |

2022, ITUCSD82204

Figure 14: The Staff Page

1.2.2 Add New Staff

This modal contains input areas for all information for a staff. Staff First Name, Last Name, email, password. The country must be the country the stores exists in, so the countries of existing stores are shown to the user. So is city and store address.



First Name
Your First Name

Last Name
Your Last Name

Email address
Enter email

Username
Username

Password
Password

Country:
Australia

City: Woodridge Store Address: 28 MySQL Boulevard Home Address: 28 MySQL Boulevard

Submit

Figure 15: Add new Staff Modal

City,store and home addresses are dynamically changing by country. For example, the country in the select part is Australia, so the cities and addresses,where the stores exists in, of Australia is shown.If the user wants to change it to Canada, so Canada's stores address, and cities are shown.



First Name
Your First Name

Last Name
Your Last Name

Email address
Enter email

Username
Username

Password
Password

Country:
Canada

City: Lethbridge Store Address: 47 MySakila Drive Home Address: 47 MySakila Drive

Submit

Figure 16: Dynamic City Selection

However, if the user wants to add new home address, he/she must be choose the 'other' option in Home Address select. Then a input text section appears, and the user is able to add new home address. This new home address is also inserted to the address table.

First Name

Last Name

Email address

Username

Password

Country:

City: Store Address: Home Address:

Figure 17: Adding New Home Address

Once the user hits the submit button, the new staff is created and inserted to the appropriate tables.

Data inserted into staff table successfully! ×

ADD NEW STAFF

| # | First Name | Last Name | Email | Username | Country | City | Store Address | Home Address | |
|---|---------------|-----------|------------------------------|----------|-----------|------------|--------------------|----------------------|--|
| 1 | Mike | Hillyer | Mike.Hillyer@sakilastaff.com | Mike | Canada | Lethbridge | 47 MySakila Drive | 23 Workhaven Lane | |
| 2 | Jon | Stephens | Jon.Stephens@sakilastaff.com | Jon | Australia | Woodridge | 28 MySQL Boulevard | 1411 Lillydale Drive | |
| 8 | Mehmet Furkan | Uyanik | mehmet@gmail.com | Frkn07 | Canada | Lethbridge | 47 MySakila Drive | Kadikoy | |

© 2022, ITUCSD82204

Figure 18: Inserting to just Staff Table

| Data inserted into address table successfully! | | | | | | | | |
|--|---------------|-----------|------------------------------|----------|-----------|------------|--------------------|----------------------|
| Data inserted into staff table successfully! | | | | | | | | |
| ADD NEW STAFF | | | | | | | | |
| # | First Name | Last Name | Email | Username | Country | City | Store Address | Home Address |
| 1 | Mike | Hillyer | Mike.Hillyer@sakilastaff.com | Mike | Canada | Lethbridge | 47 MySakila Drive | 23 Workhaven Lane |
| 2 | Jon | Stephens | Jon.Stephens@sakilastaff.com | Jon | Australia | Woodridge | 28 MySQL Boulevard | 1411 Lillydale Drive |
| 8 | Mehmet Furkan | Uyanik | mehmet@gmail.com | Frkn07 | Canada | Lethbridge | 47 MySakila Drive | Kadikoy |
| 9 | Berk | Uyanik | brk@gmail.com | brk | Australia | Woodridge | 28 MySQL Boulevard | Barcelona |
| © 2022, ITUCSDB2204 | | | | | | | | |

Figure 19: Inserting to Staff Table and Address Table

1.2.3 Deleting A Staff

First, to delete a staff, the user must click to the trash icon. If the user deletes the manager, the whole stores that has the manager wanted to be deleted will be deleted because of the ON DELETE CASCADE. Thus, my scenario does not allow the user to delete the manager of stores. So, the user can see an error message like "The Managers Can Not Be Deleted".

| MANAGERS CAN'T BE DELETED | | | | | | | | |
|---------------------------|---------------|-----------|------------------------------|----------|-----------|------------|--------------------|----------------------|
| ADD NEW STAFF | | | | | | | | |
| # | First Name | Last Name | Email | Username | Country | City | Store Address | Home Address |
| 1 | Mike | Hillyer | Mike.Hillyer@sakilastaff.com | Mike | Canada | Lethbridge | 47 MySakila Drive | 23 Workhaven Lane |
| 2 | Jon | Stephens | Jon.Stephens@sakilastaff.com | Jon | Australia | Woodridge | 28 MySQL Boulevard | 1411 Lillydale Drive |
| 8 | Mehmet Furkan | Uyanik | mehmet@gmail.com | Frkn07 | Canada | Lethbridge | 47 MySakila Drive | Kadikoy |
| © 2022, ITUCSDB2204 | | | | | | | | |

Figure 20: Managers Can Not Be Deleted

However, the user can delete the staff that is not a manager of a store.

| Data deleted from staff table successfully! | | | | | | | | | |
|---|---------------|-----------|------------------------------|----------|-----------|------------|--------------------|----------------------|---|
| ADD NEW STAFF | | | | | | | | | |
| # | First Name | Last Name | Email | Username | Country | City | Store Address | Home Address | |
| 1 | Mike | Hillyer | Mike.Hillyer@sakilastaff.com | Mike | Canada | Lethbridge | 47 MySakila Drive | 23 Workhaven Lane |   |
| 2 | Jon | Stephens | Jon.Stephens@sakilastaff.com | Jon | Australia | Woodridge | 28 MySQL Boulevard | 1411 Lillydale Drive |   |
| 8 | Mehmet Furkan | Uyanik | mehmet@gmail.com | Frkn07 | Canada | Lethbridge | 47 MySakila Drive | Kadikoy |   |
| © 2022, ITUCSD82204 | | | | | | | | | |

Figure 21: Deleting a staff that is not a manager of a store

1.2.4 Updating A Staff

The user can do editing with the edit icon. So, an edit modal pops up. The edit modal consists of the ID of staff which is not writable, name, email, username, country, city, store, and home address selections. If the user edits the store of the non-manager staff, so the store of the staff will be changed.








| Staff table updated successfully! | | | | | | | | | |
|-----------------------------------|---------------|-----------|------------------------------|----------|-----------|------------|--------------------|----------------------|---|
| ADD NEW STAFF | | | | | | | | | |
| # | First Name | Last Name | Email | Username | Country | City | Store Address | Home Address | |
| 1 | Mike | Hillyer | Mike.Hillyer@sakilastaff.com | Mike | Canada | Lethbridge | 47 MySakila Drive | 23 Workhaven Lane |   |
| 2 | Jon | Stephens | Jon.Stephens@sakilastaff.com | Jon | Australia | Woodridge | 28 MySQL Boulevard | 1411 Lillydale Drive |   |
| 8 | Mehmet Furkan | Uyanik | mehmet@gmail.com | Frkn07 | Canada | Lethbridge | 47 MySakila Drive | Kadikoy |   |
| 9 | Berk | Uyanik | brk@gmail.com | brkkk | Canada | Lethbridge | 47 MySakila Drive | Kadikoy |   |
| © 2022, ITUCSD82204 | | | | | | | | | |

Figure 22: Editing Non Managers

However, the user can not edit the staff who is the manager of a store. The page can not let them edit the staff who is the manager.

| MANAGERS CAN'T BE EDITED | | | | | | | | |
|--------------------------|---------------|-----------|------------------------------|----------|-----------|------------|--------------------|----------------------|
| ADD NEW STAFF | | | | | | | | |
| # | First Name | Last Name | Email | Username | Country | City | Store Address | Home Address |
| 1 | Mike | Hillyer | Mike.Hillyer@sakilastaff.com | Mike | Canada | Lethbridge | 47 MySakila Drive | 23 Workhaven Lane |
| 2 | Jon | Stephens | Jon.Stephens@sakilastaff.com | Jon | Australia | Woodridge | 28 MySQL Boulevard | 1411 Lillydale Drive |
| 8 | Mehmet Furkan | Uyanik | mehmet@gmail.com | Frkn07 | Canada | Lethbridge | 47 MySakila Drive | Kadikoy |
| 9 | Berk | Uyanik | brk@gmail.com | brkkk | Canada | Lethbridge | 47 MySakila Drive | Kadikoy |

© 2022, ITUCSDB2204

Figure 23: Managers Can not Be Edited

2 Developer Guide

2.1 Parts Implemented by Yasmeen Abdelghany

2.1.1 Users table

The Users table stores users' information like names and passwords. It has 3 non-key columns.

- username is of VARCHAR(25) data type
- passcode is of VARCHAR(25) data type
- last_update is of TIMESTAMP data type

2.1.2 Film table

The film table is a list of all films potentially in stock in the stores. The actual in-stock copies of each film are represented in the inventory table.

The film table refers to the language table and is referred to by the film_category, film_actor, and inventory tables.

It consists of 9 non-columns.

- film_id: A surrogate primary key is used to uniquely identify each film in the table.
- title: The title of the film.
- description: A short description or plot summary of the film.
- release_year: The year in which the movie was released.

- `language_id`: A foreign key pointing at the language table; identifies the language of the film.
- `original_language_id`: A foreign key pointing at the language table; identifies the original language of the film. Used when a film has been dubbed into a new language.
- `rental_duration`: The length of the rental period, in days.
- `rental_rate`: The cost to rent the film for the period specified in the `rental_duration` column.
- `length`: The duration of the film, in minutes.
- `replacement_cost`: The amount charged to the customer if the film is not returned or is returned in a damaged state.
- `rating`: The rating assigned to the film. Can be one of G, PG, PG-13, R, or NC-17.
- `last_update`: When the row was created or most recently updated.

2.1.3 Film_category table

The `film_category` table is used to support a many-to-many relationship between films and categories. For each category applied to a film, there will be one row in the `film_category` table listing the category and film.

The `film_category` table refers to the `film` and `category` tables using foreign keys. It has 2 key columns and 1 non-key column:

- `film_id`: A foreign key identifying the film.
- `category_id`: A foreign key identifying the category.
- `last_update`: When the row was created or most recently updated.

2.1.4 Category table

The `category` table lists the categories that can be assigned to a film.

The `category` table is joined to the `film` table by means of the `film_category` table. It has 2 non-key columns:

- `category_id`: A surrogate primary key is used to uniquely identify each category in the table.
- `name`: The name of the category.
- `last_update`: When the row was created or most recently updated.

2.1.5 Language table

The language table is a lookup table listing the possible languages that films can have for their language and original language values.

The language table is referred to by the film table.

It has 2 non-key columns:

- `language_id`: A surrogate primary key is used to uniquely identify each language. : The English name of the language.
- `last_update`: When the row was created or most recently updated.

2.1.6 Sign up

When the user signs up by filling in the required fields and hitting the sign-up button, the entered username is first checked using `check_user` function. If the user actually exists, the user is redirected to the signup page. If the user entered a username or a password with less than 7 characters, he/she will be alerted. Otherwise, the password gets hashed, a new user is added in the `USer` class, and the user record is added to the `Users` table. Also, a new session is instantiated, and the user is redirected to the home page.

```

1 <form action="{{ url_for('sign') }}" method="POST">
2   <div class="input-group1">
3     <div class="input-field1">
4       <i class="fa-solid fa-user"></i>
5       <input type="text" class="form-control2" id="name" name="name"
placeholder="User Name"
6         required="1">
7     </div>
8     <div class="input-field1">
9       <i class="fa-solid fa-lock"></i>
10      <input type="password" class="form-control2" id="password" name
="password">

```

```

11         placeholder="Password" required="1">
12     </div>
13 </div>
14 <div class="btn-field1">
15     <button type="submit" class="button" id="signupBtn">Sign up</button>
16 ><br><br>
17 </div>
18 </form>

```

Listing 1: Sign Up HTML form

```

1 @app.route('/sign', methods=["POST", "GET"])
2 def sign():
3     if request.method == "POST":
4         name = request.form.get('name')
5         password = request.form.get('password')
6         user = views.check_user(name)
7         if user:
8             flash("User already exists! Log in instead", category="error")
9         elif len(name) < 7:
10             flash("Username should be longer than 7 characters!", category=
11 = "error")
12         elif len(password) < 7:
13             flash("Password should be longer than 7 characters!", category=
14 "error")
15         else:
16             new_user = auth.User(username=name, password=
17 generate_password_hash(password, method='sha256'))
18             views.insert_user(new_user.username, new_user.password)
19             user = views.check_user(new_user.username)
20             session['name'] = user['username']
21             return redirect(url_for('home_page'))
22
23     return render_template("sign.html")

```

Listing 2: Python Code for POST method

```

1 def insert_user(username, password):
2     try:
3         cur = mydb.cursor()
4         statement = """ insert into users
5             (username, passcode)
6             values (%s, %s)

```

```

7         """
8         cur.execute(statement, (username, password))
9         mydb.commit()
10        flash("Account created!", category="success")
11    except:
12        flash("Data isn't inserted into user table! Please try again!!",
category="error")

```

Listing 3: Python-Insert User Query

2.1.7 Log in

When the user logs in by filling the required fields, the username is checked using the `check_user` function. If the user already has an account but entered the wrong password, he will be alerted. Otherwise, a new session is instantiated, and the user is redirected to the home page.

```

1 <form action="{{ url_for('sign') }}" method="POST">
2     <div class="input-group1">
3         <div class="input-field1">
4             <i class="fa-solid fa-user"></i>
5             <input type="text" class="form-control2" id="name" name="name"
placeholder="User Name"
6                 required="1">
7         </div>
8         <div class="input-field1">
9             <i class="fa-solid fa-lock"></i>
10            <input type="password" class="form-control2" id="password" name
="password"
11                placeholder="Password" required="1">
12            </div>
13        </div>
14        <div class="btn-field1">
15            <button type="submit" class="button" id="signupBtn">Sign up</button
><br><br>
16        </div>
17
18 </form>

```

Listing 4: Log In HTML form

```

1 @app.route('/login', methods=["POST", "GET"])
2 def login():
3     if request.method == "POST":

```

```

4     name = request.form['name']
5     password = request.form['password']
6     user = views.check_user(name)
7     if len(user) > 0:
8         if check_password_hash(user['passcode'], password):
9             session['name'] = user['username']
10            flash("Logged in successfully!", category='success')
11            return redirect(url_for('home_page'))
12        else:
13            flash("Incorrect password!", category="error")
14
15    return render_template("login.html", boolean=True)

```

Listing 5: Python Code for POST method

```

1 def check_user(username):
2     try:
3         cur = mydb.cursor()
4         statement = """ select * from users
5                        where username = %s
6                        """
7         cur.execute(statement, (username, ))
8         user = cur.fetchone()
9         cur.close()
10        return user
11    except:
12        flash("Username not found! Sign up or retype your username.",
13              category="error")

```

Listing 6: Python-Check User availability Query

2.1.8 Display table codes

Once the film page is opened, the film table is displayed using Select with join queries as shown below. The table has the film, film_category, category, and language tables combined on one page.

```

1 <table class="table table-responsive-sm table-sm table-hover" id="
   film_table">
2     <thead class="thead-light">
3         <tr>
4             <th scope="col">ID</th>
5             <th scope="col">Title</th>
6             <th scope="col">Category</th>

```

```

7         <th scope="col">Description</th>
8         <th scope="col">Release Year</th>
9         <th scope="col">language</th>
10        <th scope="col">Rental Duration</th>
11        <th scope="col">Rental Rate</th>
12        <th scope="col">Length</th>
13        <th scope="col">Replacement Cost</th>
14        <th scope="col">Rating</th>
15        <th scope="col">Operation</th>
16    </tr>
17 </thead>
18 <tbody>
19     {% for records in film_data %}
20     <tr>
21         <th scope="row">{{records.film_id}}</th>
22         <td>{{records.title}}</td>
23         <td>{{records.name}}</td>
24         <td>{{records.description}}</td>
25         <td>{{records.release_year}}</td>
26         <td>{{records.language}}</td>
27         <td>{{records.rental_duration}}</td>
28         <td>{{records.rental_rate}}</td>
29         <td>{{records.length}}</td>
30         <td>{{records.replacement_cost}}</td>
31         <td>{{records.rating}}</td>
32         <td><a href="/update/{{records.film_id}}" class="btn btn-
warning btn-xs" data-toggle="modal"
33             data-target="#modaledit{{records.film_id}}"><i class="
fa-sharp fa-solid fa-pen-to-square"></i></a>
34         <a href="/delete/{{records.film_id}}" class="btn btn-danger
btn-xs"
35             onclick="return confirm('Are you sure you want to
delete?')"><i class="fa-solid fa-trash"></i></a>
36         </td>
37     </tr>
38     {% endfor %}
39 </tbody>
40 </table>

```

Listing 7: Film HTML table

```

1 @app.route('/film')
2 def film():
3     film_data = views.fetch_film()

```

```

4     lang_data = views.fetch_language()
5     cat_data = views.fetch_category()
6     return render_template("film.html", film_data=film_data, lang_data=
lang_data, cat_data=cat_data)

```

Listing 8: Python code

_category

```

1 ##### INNER JOINING FILM,FILM_CATEGORY AND LANGUAGE TABLES #####
2 def fetch_film():
3     try:
4         cursor = mydb.cursor()
5         cursor.execute("""SELECT
6
7             f.film_id, f.title, c.name, f.description,
8             f.release_year,
9             l.name as language, f.rental_duration, f.
10            rental_rate, f.length,
11            f.replacement_cost, f.rating
12            FROM
13            film f
14            JOIN
15            language l ON f.language_id = l.language_id
16            JOIN
17            film_category fc on f.film_id = fc.film_id
18            JOIN
19            category c ON fc.category_id = c.
20            category_id
21            order by f.film_id;""")
22
23         records = cursor.fetchall()
24         return records
25     except:
26         flash("Couldn't fetch data from database!", category="error")

```

Listing 9: Python-Selection Queries of Film

2.1.9 Insert operation codes

When the add new film button is clicked and the modal appears, the user can enter all the information in the corresponding fields. If the other option is selected, a new language or category is added to the corresponding tables of the database. The select options are actually fetched from the database tables as well as shown in the codes below. After hitting the add button, all the data gets inserted into their corresponding tables in the database.

```

1 <div id="myModal" class="modal fade" role="dialog" aria-hidden="true">
2
3     <div class="modal-dialog modal-lg">
4
5         <div class="modal-content">
6
7             <div class="modal-header">
8                 <h5 class="modal-title" id="addFilm">Add a new film</h5>
9                 <button type="button" class="close" data-dismiss="modal"
aria-label="Close">
10                     <span aria-hidden="true">&times;
11                 </button>
12             </div>
13
14             <div class="modal-body">
15
16                 <form action="{{ url_for('get_film') }}" method="POST">
17
18                     <div class="form-group">
19                         <label for="title" class="col-form-label">Title</
label>
20                         <input type="text" class="form-control" name="
title" placeholder="Type title" required="1">
21
22                         <label for="Description-text" class="col-form-label
">Description</label><br>
23                         <textarea id="description" name="description" rows=
"3" cols="50" required="1"
24                             placeholder=" Type a short description or plot
summary of the film"
25                             class="textarea"></textarea>
26                         <br><br>
27
28                         <label>Release Year</label>
29                         <input type="text" class="datepicker" placeholder="
select year" name="year_datepicker"
30                             required="1" />
31
32                         <label>Length</label>
33                         <input type="number" class="form-control" id="
replyNumber" min="0" step="1"
34                             data-bind="value:replyNumber" name="length"
required="1" />

```



```

35
36         <label>Rental Duration</label>
37         <input type="number" class="form-control2" id="
replyNumber" min="0" step="1"
38             data-bind="value:replyNumber" required="1" name
="rent_dur" /><br>
39
40         <label>Rental Rate</label>
41         <input type="number" min="0" step="0.01" max="99.99
" class="form-control2" required="1"
42             name="rent_rate" />
43
44         <label>Replacement Cost</label>
45         <input type="number" min="0" step="0.01" max="
999.99" class="form-control2" required="1"
46             name="cost" /> <br />
47
48         <label for="category">Category</label>
49         <select name="cat_menu" id="cat_menu" required="1"
50             onchange="if (this.value=='other'){this.form['
other'].style.visibility='visible'}else {this.form['other'].style.
visibility='hidden'};">
51             {% for cat in cat_data %}
52             <option name="cat_menu" value="{{cat}}">{{cat}}
</option>
53             {% endfor %}
54             <option value="other">--Other--</option>
55         </select>
56         <input type="text" class="other_text" name="
other" style="visibility:hidden;" />
57
58         <label for="languages">Language</label>
59         <select name="lang_menu" id="lang_menu" required="1
"
60             onchange="if (this.value=='Other'){this.form['
Other'].style.visibility='visible'}else {this.form['Other'].style.
visibility='hidden'};">
61             {% for lang in lang_data %}
62             <option name="lang_menu" value="{{lang}}">{{
lang}}</option>
63             {% endfor %}
64             <option value="Other">--Other--</option>
65         </select>

```

```

66         <input type="textbox" class="other_text" name="
Other" style="visibility:hidden;" /><br />
67
68         <label>Rating: </label>
69         <input type="radio" name="ratings" value="G" id="
rating1" aria-selected="true" required="1">
70         <label for="rating1">G</label>
71         <input type="radio" name="ratings" value="PG" id="
rating2">
72         <label for="rating2">PG</label>
73         <input type="radio" name="ratings" value="PG-13" id=
="rating3">
74         <label for="rating3">PG-13</label>
75         <input type="radio" name="ratings" value="R" id="
rating4">
76         <label for="rating4">R</label>
77         <input type="radio" name="ratings" value="NC-17" id
="rating5">
78         <label for="rating5">NC-17</label>
79     </div>
80     <div class="modal-footer">
81         <button type="button" class="btn btn-secondary"
data-dismiss="modal">Cancel</button>
82         <button id="submit" type="submit" class="btn btn-
primary">Add</button>
83     </div>
84 </form>
85 </div>
86 </div>
87 </div>
88 </div>

```

Listing 10: Add new Film HTML form

```

1 ##### ADDING FILM TABLE #####
2 @app.route('/get_film', methods=['POST'])
3 def get_film():
4     if request.method == "POST":
5         title = request.form['title']
6         description = request.form['description']
7         year = request.form['year_datepicker']
8         category = request.form.get('cat_menu')
9         if category == 'other':
10             category = request.form['other']

```

```

11         views.insert_category(category)
12         language = request.form.get('lang_menu')
13         if language == 'Other':
14             language = request.form['Other']
15             views.insert_language(language)
16         rating = request.form.get('ratings')
17         rent_dur = request.form['rent_dur']
18         rent_rate = request.form['rent_rate']
19         length = request.form['length']
20         cost = request.form['cost']
21         cat_id = views.fetch_category().index(category) + 1
22         lang_id = views.fetch_language().index(language) + 1
23         views.insert_film(title, description, year, lang_id, rent_dur,
24                           rent_rate, length, cost, rating)
25         views.fetch_id(title, cat_id)
26
27         return redirect(url_for('film'))
28
29     return render_template("film.html")

```

Listing 11: Python Code

_category Queries

```

1
2 ##### INSERTING INTO FILM TABLE #####
3 def insert_film(title, description, release_year, language_id,
4                 rental_duration,
5                 rental_rate, length, replacement_cost, rating):
6     try:
7         cur = mydb.cursor()
8         statement = """ insert into film
9                        (title, description, release_year, language_id,
10                        rental_duration,
11                        rental_rate, length, replacement_cost, rating)
12                        values (%s, %s, %s, %s, %s, %s, %s, %s, %s)
13                        """
14         cur.execute(statement, (title, description, release_year,
15                                language_id, rental_duration,
16                                rental_rate, length, replacement_cost,
17                                rating))
18         mydb.commit()
19         flash("Data inserted into film table successfully!", category="
20 success")
21     except:

```

```

17         flash("Data isn't inserted into film table! Please try again!!",
18               category="error")
19
20 ##### FETCHING LANGUAGE #####
21 def fetch_language():
22     try:
23         cursor = mydb.cursor()
24         cursor.execute("""SELECT
25                             l.name
26                             FROM
27                             language l;""")
28         records = cursor.fetchall()
29         languageArr=[]
30         for row in records:
31             languageArr.append(row['name'])
32
33         return languageArr
34     except:
35         flash("Couldn't fetch data from database!", category="error")
36
37 ##### FETCHING CATEGORY #####
38 def fetch_category():
39     try:
40         cursor = mydb.cursor()
41         cursor.execute("""SELECT
42                             name
43                             FROM
44                             category;""")
45         records = cursor.fetchall()
46         catArr=[]
47         for row in records:
48             catArr.append(row['name'])
49
50         return catArr
51     except:
52         flash("Couldn't fetch data from database!", category="error")
53
54 ##### INSERTING INTO FILM_CATEGORY TABLE #####
55 def insert_film_cat(film_id, cat_id):
56     try:
57         cur = mydb.cursor()
58

```

```

59     statement = """ insert into film_category (film_id, category_id)
values (%s, %s) """
60     cur.execute(statement, (film_id['film_id'], cat_id))
61     mydb.commit()
62     flash("Data inserted into film_category table successfully!",
category="success")
63     except:
64         flash("Data isn't inserted into film_category table! Please try
again!!", category="error")
65
66 ##### FETCHING FILM ID #####
67 def fetch_id(title, cat_id):
68     try:
69         cur = mydb.cursor()
70         cur.execute(
71             """SELECT film_id FROM film WHERE title = %s """ , (title,))
72         film_id = cur.fetchone()
73         insert_film_cat(film_id, cat_id)
74     except:
75         flash("No such film exists. Can't add to film_category table",
category="error")
76 ##### INSERT INTO LANGUAGE TABLE #####
77 def insert_language(language):
78     try:
79         cur = mydb.cursor()
80         statement = """ insert into language (name) values (%s) """
81         cur.execute(statement, (language, ))
82         mydb.commit()
83         flash("New language added successfully", category="success")
84     except:
85         flash("Couldn't add a new language! Please try agian.", category="
error")
86
87 ##### INSERT INTO CATEGORY TABLE #####
88 def insert_category(category):
89     if request.method == "POST":
90         try:
91             cur = mydb.cursor()
92             statement = """ insert into category (name) values (%s) """
93             cur.execute(statement, (category, ))
94             mydb.commit()
95             flash("New category added successfully", category="success")
96         except:

```

```

97         flash("Couldn't add a new category! Please try again.",
               category="error")

```

Listing 12: Python-Insert into film

2.1.10 Update operation codes

When the edit button is selected, the user can change any of the fields and the update queries will be executed. Again, a new category or language can be inserted in the database tables, if the user chose to add new ones instead as shown in the codes below.

```

1  {% for records in film_data %}
2  <div id="modaledit{{records.film_id}}" class="modal fade" role="dialog"
   aria-hidden="true">
3
4      <div class="modal-dialog modal-lg">
5
6          <div class="modal-content">
7
8              <div class="modal-header">
9                  <h5 class="modal-title" id="addFilm">Update Film</h5>
10                 <button type="button" class="close" data-dismiss="modal"
   aria-label="Close">
11                     <span aria-hidden="true">&times;</span>
12                 </button>
13             </div>
14
15             <div class="modal-body">
16
17                 <form action="{{ url_for('update') }}" method="POST">
18
19                     <div class="form-group">
20                         <label for="title" class="col-form-label">Title</
   label>
21
22                         <input type="hidden" name="id" value="{{records.
   film_id}}">
23
24                         <input type="text" class="form-control" name="
   title" value="{{records.title}}"
25                             placeholder="Type title" required="1">
26
27                         <label for="Description-text" class="col-form-label
   ">Description</label><br>

```

```

26         <textarea id="description" name="description" rows=
"3" cols="50" class="textarea"
27         placeholder=" Type a short description or plot
summary of the film"
28         required="1">{{records.description}}</textarea>
29         <br><br>
30
31         <label>Release Year</label>
32         <input type="text" class="datepicker" name="
year_datepicker" value="{{records.release_year}}"
33         placeholder="select year" required="1" />
34
35         <label>Length</label>
36         <input type="number" class="form-control2" id="
replyNumber" min="0" step="1"
37         data-bind="value:replyNumber" name="length"
required="1" value="{{records.length}}" />
38
39         <label>Rental Duration</label>
40         <input type="number" class="form-control2" id="
replyNumber" min="0" step="1"
41         data-bind="value:replyNumber" required="1" name
="rent_dur" value="{{records.rental_duration}}" /><br>
42
43         <label>Rental Rate </label>
44         <input type="number" min="0" step="0.01" max="99.99
" class="form-control2" required="1"
45         name="rent_rate" value="{{records.rental_rate}}
" />
46
47         <label>Replacement Cost</label>
48         <input type="number" min="0" step="0.01" max="
999.99" class="form-control2" required="1"
49         name="cost" value="{{records.replacement_cost}}
" />
50         <br />
51
52         <label for="category">Category</label>
53         <select name="cat_menu" id="cat_menu" value="{{
records.name}}" required="1"
54         onchange="if (this.value=='other'){this.form['
other'].style.visibility='visible'}else {this.form['other'].style.
visibility='hidden'};">

```

```

55         {% for cat in cat_data %}
56         <option name="cat_menu" value="{{cat}}">{{cat}}
</option>
57         {% endfor %}
58         <option value="other">--Other--</option>
59     </select>
60     <input type="text" class="other_text" name="
other" style="visibility:hidden;"/>
61
62     <label for="languages">Language</label>
63     <select name="lang_menu" id="lang_menu" value="{{
records.language}}" required="1"
64         onchange="if (this.value=='Other'){this.form['
Other'].style.visibility='visible'}else {this.form['Other'].style.
visibility='hidden'};">
65         {% for lang in lang_data %}
66         <option name="lang_menu" value="{{lang}}">{{
lang}}</option>
67         {% endfor %}
68         <option value="Other">--Other--</option>
69     </select>
70     <input type="text" class="other_text" name="
Other" style="visibility:hidden;" /><br />
71     <label>Rating: </label>
72     <input type="radio" name="ratings" value="G" id="
rating1" aria-selected="true" required="1">
73     <label for="rating1">G</label>
74     <input type="radio" name="ratings" value="PG" id="
rating2">
75     <label for="rating2">PG</label>
76     <input type="radio" name="ratings" value="PG-13" id
="rating3">
77     <label for="rating3">PG-13</label>
78     <input type="radio" name="ratings" value="R" id="
rating4">
79     <label for="rating4">R</label>
80     <input type="radio" name="ratings" value="NC-17" id
="rating5">
81     <label for="rating5">NC-17</label>
82 </div>
83 <div class="modal-footer">
84     <button type="button" class="btn btn-secondary"
data-dismiss="modal">Cancel</button>

```



```

85         <button type="submit" class="btn btn-primary">Save
Edits</button>
86     </div>
87 </form>
88 </div>
89
90
91 </div>
92 </div>
93 </div>
94 {% endfor %}

```

Listing 13: Edit film HTML form

```

1 ##### UPDATING FILM TABLE #####
2 @app.route('/update', methods=['POST', 'GET'])
3 def update():
4     if request.method == "POST":
5         film_id = request.form['id']
6         title = request.form['title']
7         description = request.form['description']
8         year = request.form['year_datepicker']
9         category = request.form.get('cat_menu')
10        if category == 'other':
11            category = request.form['other']
12            views.insert_category(category)
13        language = request.form.get('lang_menu')
14        if language == 'Other':
15            language = request.form['Other']
16            views.insert_language(language)
17        rating = request.form.get('ratings')
18        rent_dur = request.form['rent_dur']
19        rent_rate = request.form['rent_rate']
20        length = request.form['length']
21        cost = request.form['cost']
22        feature = ",".join(request.form.getlist('features'))
23        cat_id = views.fetch_category().index(category) + 1
24        lang_id = views.fetch_language().index(language) + 1
25
26        views.update_film(title, description, year, lang_id,
27                           rent_dur, rent_rate, length, cost, rating, film_id)
28
29        views.update_film_category(cat_id, film_id)
30

```

```

31         return redirect(url_for('film'))
32
33     return render_template("film.html")

```

Listing 14: Python Code

```

1 ##### UPDATING FILM TABLE #####
2 def update_film(title, description, year, lang_id,
3                 rent_dur, rent_rate, length, cost, rating, film_id):
4     try:
5         cur = mydb.cursor()
6         cur.execute(
7             """UPDATE film
8                 SET title = %s, description = %s, release_year = %s,
9                 language_id = %s,
10                rental_duration = %s, rental_rate = %s, length = %s,
11                replacement_cost = %s,
12                rating = %s
13                WHERE film_id = %s """ , (title, description, year, lang_id,
14                                         rent_dur, rent_rate, length, cost,
15                                         rating, film_id))
16         mydb.commit()
17         flash("Data updated in film table successfully!", category="success")
18     except:
19         flash("Data couldn't be updated in film table! Please try again.",
20              category="error")
21
22 ##### UPDATING FILM_CATEGORY TABLE #####
23 def update_film_category(cat_id, film_id):
24     try:
25         cur = mydb.cursor()
26         cur.execute(
27             """UPDATE film_category
28                 SET category_id = %s
29                 WHERE film_id = %s """ , (cat_id, film_id))
30         mydb.commit()
31         flash("Data updated in film_category table successfully!", category
32              ="success")
33     except:
34         flash("Data couldn't be updated in film_category table! Please try
35              again.", category="error")

```

Listing 15: Python-Update film and film_category Queries

2.1.11 Delete operation codes

When the deleted button is clicked and confirmed, the delete queries get executed. Since the foreign key constraints are set to Cascade, the corresponding film ids from all the dependent tables get deleted as well.

```
1 ##### DELETING FROM TABLE #####
2 @app.route('/delete/<string:film_id>', methods=['GET'])
3 def delete(film_id):
4     views.delete_film(film_id)
```

Listing 16: Python Code

```
1 ##### DELETE FROM FILM TABLE #####
2 def delete_film(film_id):
3     try:
4         cur = mydb.cursor()
5         cur.execute(
6             """DELETE FROM film
7             WHERE film_id = %s """ , (film_id, ))
8         mydb.commit()
9         flash("The film was deleted successfully!", category="success")
10    except:
11        flash("The film couldn't be deleted! PLease try again.", category="
error")
```

Listing 17: Python-Delete Query

2.2 Parts Implemented by Mehmet Furkan Uyanık

2.2.1 Staff Page

The staff table lists all staff members, including information for email address, login information, and picture.

The staff table refers to the store and address tables using foreign keys, and is referred to by the rental, payment, and store tables.

Columns of Staff Table:

1. staff-id: A surrogate primary key that uniquely identifies the staff member.
2. first-name: The first name of the staff member.
3. last-name: The last name of the staff member.

4. address-id: A foreign key to the staff member address in the address table.
5. picture: A BLOB containing a photograph of the employee.
6. email: The staff member email address.
7. store-id: The staff member “home store.” The employee can work at other stores but is generally assigned to the store listed.
8. active: Whether this is an active employee. If employees leave, their rows are not deleted from this table; instead, this column is set to FALSE.
9. username: The user name used by the staff member to access the rental system.
10. password: The password used by the staff member to access the rental system. The password should be stored as a hash using the SHA2() function.
11. last-update: When the row was created or most recently updated.

However, not all of the columns of the staff table is shown on the Staff Page. Only The columns staff-id,first and last name,email,username,country,city,store and home address is shown on the staff page.

2.2.2 Reading Operations for Staff Page

```

1  <table class="table table-responsive-sm table-sm table-hover" id="
    staff_table">
2  <thead>
3  <tr>
4      <th scope="col">#</th>
5      <th scope="col">First Name</th>
6      <th scope="col">Last Name</th>
7      <th scope="col">Email</th>
8      <th scope="col">Username</th>
9      <th scope="col">Country</th>
10     <th scope="col">City</th>
11     <th scope="col">Store Address</th>
12     <th scope="col">Home Address</th>
13 </tr>
14 </thead>
15 <tbody>
16 <tr>
17     {% for tuple in tuples %}

```

```

18     <th scope="row">{{tuple.staff_id}}</th>
19     <td>{{tuple.first_name}}</td>
20     <td>{{tuple.last_name}}</td>
21     <td>{{tuple.email}}</td>
22     <td>{{tuple.username}}</td>
23     <td>{{tuple.country}}</td>
24     <td>{{tuple.city}}</td>
25     <td>{{tuple.Store_Address}}</td>
26     <td>{{tuple.Home_Address}}</td>
27     <td class="operation">
28         <a class="delete" href="/delete_from_staff/{{tuple.staff_id}}">
29         <i class="fa-solid fa-trash"></i></a>
30         <a id="{{tuple.staff_id}}" class="updateRow"><i class="fa-sharp
31         fa-solid fa-pen-to-square"></i></a>
32     </td>
33 </tr>
34 {% endfor %}
35 </tbody>
36 </table>

```

Listing 18: Staff Table HTML Table

```

1 @app.route('/staff/')
2 def staff():
3     datas = views.fetch_staff()
4     countries = views.fetch_country()
5     return render_template('staff.html', tuples=datas, country=
countries)

```

Listing 19: server.py

Datas are the staff informations coming from staff table and they are the tuples of HTML page. Countries are for dynamic show of city, country, store and home address.

2.2.3 Dynamic Selection of City Store and Home Addresses

```

1 def fetch_staff():
2     try:
3         cur = mydb.cursor()
4         cur.execute("""Select s.staff_id,s.first_name,s.last_name,
5                         s.username,s.email,co.country,ci.city,a.address
6                         as Store_Address,a2.address as Home_Address
7                     from sakila.staff s
8                     join sakila.city ci on s.city_id=ci.city_id
9                     join sakila.country co on s.country_id=co.country_id
10                    join sakila.address a on s.address_id=a.address_id
11                    join sakila.address a2 on s.address_id=a2.address_id
12                    """)
13         datas = cur.fetchall()
14     except:
15         pass
16     return datas

```

```

7             INNER JOIN sakila.store as st ON st.store_id =
s.store_id
8             INNER JOIN sakila.address as a ON st.address_id
= a.address_id
9             INNER JOIN sakila.city as ci ON a.city_id = ci.
city_id
10            INNER JOIN sakila.country as co ON co.
country_id = ci.country_id
11            INNER JOIN sakila.address as a2 ON a2.
address_id = s.address_id order by s.staff_id"")
12        control = cur.fetchall()
13    except:
14        flash("Couldn't fetch data from database!", category="error")
15    return control

```

Listing 20: views.py

In order to show country,city,store address and home address of a staff, staff table is joined with store table,address table,city table and country table with appropriate keys. Also, when an error occurs, it shows a flash message.

Dynamic showing of city,staff and home address are done by Javascript.

```

1 for(let i=0;i<country_select.length;i++){
2     country_select[i].onchange = function(){
3         country = country_select[i].value;
4         fetch('/staff_city/'+Number(country)).then(function(response){
5             response.json().then(function(data){
6                 city_option_html = '';
7                 for(let city of data.staff_citycountry){
8                     city_option_html+=`<option value=${city.city_id}> ${city.city}</
option>`;
9                 }
10                city_select[i].innerHTML = city_option_html;
11            });
12        }
13    }

```

Listing 21: Dynamic Showing of City according to id of country

Here whenever the user changes country, the cities will be changed according to country. In staff.js file, the fetching operation is done by like above code. It fetches cities ,which is jsonify object, from python and adds to city select tag.

Below, you can see the python code of fetching. City id and city name columns are fetched according to id of the selected country from HTML.

```

1 def fetch_city(id):
2     try:
3         cur = mydb.cursor()
4         result = cur.execute(
5             """Select ci.city_id,ci.city FROM sakila.city ci
6             INNER JOIN sakila.address as a on a.city_id = ci.city_id
7             INNER JOIN sakila.store as st on st.address_id = a.address_id
8             where ci.country_id = %s""" % id)
9         city = cur.fetchall()
10    except:
11        flash("Couldn't fetch data from city table! Please try again.",
12            category="error")
13    return city

```

Listing 22: Fetching Cities for dynamic city show for views.py

```

1 @app.route('/staff_city/<id>')
2 def staff_city(id):
3     city = views.fetch_city(id)
4     cityArray = []
5     for row in city:
6         cityObj = {
7             'city_id': row['city_id'],
8             'city': row['city']}
9         cityArray.append(cityObj)
10    return jsonify({'staff_citycountry': cityArray})

```

Listing 23: Fetching Cities for dynamic city show for server.py

In above code, the function takes cities from fetchcity function from views and converting them to jsonify object for javascript file.

```

1 let store_address_option_html = '';
2     city = city_select[i].value;
3     fetch('/staff_store_address/'+Number(city)).then(function(response)
4     {
5         response.json().then(function(data) {
6             store_address_option_html = '';
7             for(let address of data.staff_store_address){
8                 store_address_option_html+=`<option value=${address.store_id
9             }> ${address.address}</option>`;
10            }
11            st_address_select[i].innerHTML = store_address_option_html;

```

```
10         });
```

Listing 24: Fetching Stores addresses for dynamic store address show according to id of city for server.py

Above code fetches store address of stores according to selected city

```
1 def fetch_address(id):
2     try:
3         cur = mydb.cursor()
4         result = cur.execute(
5             """Select st.store_id,a.address from sakila.address a
6             INNER JOIN sakila.store as st on st.address_id = a.address_id
7             WHERE a.city_id =%s""" % id)
8         store_Address = cur.fetchall()
9     except:
10         flash("Couldn't fetch data from address table! Please try again.",
11             category="error")
12     return store_Address
```

Listing 25: This is views.py. Fetching store address according to id of city for server.py

Above code sends store address to staff-store-address function in server.py

```
1 @app.route('/staff_store_address/<id>')
2 def staff_store_address(id):
3     store_Address = views.fetch_address(id)
4     store_Address_Arr = []
5     for row in store_Address:
6         cityObj = {
7             'store_id': row['store_id'],
8             'address': row['address']}
9         store_Address_Arr.append(cityObj)
10    return jsonify({'staff_store_address': store_Address_Arr})
```

Listing 26: This is servers.py. Returning jsonify object to the javascript file to show stores according to city id

```
1 let home_address_option_html = '';
2     city = city_select[i].value;
3     fetch('/staff_home_address/'+Number(city)).then(function(response) {
4         response.json().then(function(data) {
5
6             home_address_option_html = '';
7             for(let address of data.staff_home_address){
8                 home_address_option_html+=`<option value=${address.address_id
9             }> ${address.address}</option>`;
```



```

9         }
10        home_address_select[i].innerHTML = home_address_option_html;
11        if(i==0)
12            home_address_select[i].innerHTML += '<option value="other">
Other</option> `';
13    });

```

Listing 27: This is the javascript code of dynamic home address show according to city id. It fetches data from staff-home-address function in server.py

```

1 def fetch_Home_Address(id):
2     try:
3         cur = mydb.cursor()
4         result = cur.execute("Select a.address_id,a.address from sakila.
address a INNER JOIN sakila.city as ci on ci.city_id = a.city_id where
ci.city_id=%s" % id)
5         home_Address = cur.fetchall()
6         return home_Address
7     except:
8         flash("Couldn't fetch data from address and city tables! Please try
again.", category="error")

```

Listing 28: This is the views.py. This function fetches data for dynamic show in server.py

```

1 @app.route('/staff_home_address/<id>')
2 def staff_home_address(id):
3     home_Address = views.fetch_Home_Address(id)
4     home_Address_Arr=[]
5     for row in home_Address:
6         cityObj={
7             'address_id':row['address_id'],
8             'address':row['address']}
9         home_Address_Arr.append(cityObj)
10    return jsonify({'staff_home_address':home_Address_Arr})

```

Listing 29: This is the server.py. This function converts addresses

2.2.4 Adding New Staff

When the user wants to add new staff, add new staff button opens an add modal page. HTML code of the add modal page:

```

1 <div id="modal" class="modal">
2 <div id="add_to_staff" class="add_to_staff">

```

```

3     <form action="{{ url_for('add_new_staff') }}" method="POST">
4         <div class="form-group">
5             <label for="firstName">First Name</label>
6             <input type="text" name = "fname" class="form-control" id="
firstName" placeholder="Your First Name" required="1">
7         </div>
8         <div class="form-group">
9             <label for="lastName">Last Name</label>
10            <input type="text" name="lname" class="form-control" id="lastName"
placeholder="Your Last Name" required="1">
11        </div>
12        <div class="form-group">
13            <label for="exampleInputEmail1">Email address</label>
14            <input name="email" type="email" class="form-control" id="email"
placeholder="Enter email" required="1">
15        </div>
16        <div class="form-group">
17            <label for="exampleInputEmail1">Username</label>
18            <input name="username" type="text" class="form-control" id="
username" placeholder="Username" required="1">
19        </div>
20        <div class="form-group">
21            <label for="exampleInputPassword1">Password</label>
22            <input name="password" type="password" class="form-control" id="
exampleInputPassword1" placeholder="Password" required="1">
23        </div>
24        <div class="form-group">
25            <label for="email">Country:</label>
26            <select class="form-control country" id="country" name="country"
required="1">
27                {% for row in country %}
28                <option value="{{row.country_id}}">{{row.country}}</option>
29                {% endfor %}
30            </select>
31        </div>
32        <div class="form-group">
33            <label for="city">City:</label>
34            <select class="form-select city" id="city" name="city" required="
1"></select>
35        </div>
36        <div class="form-group">
37            <label for="st_Address">Store Address:</label>

```

```

38         <select class="form-select st_Address" id="st_Address" name="
st_Address" required="1"></select>
39     </div>
40     <div class="form-group">
41         <label for="h_Address">Home Address: </label>
42         <select class="form-select h_Address" id="h_Address" name="
h_Address" required="1">
43
44             </select>
45
46     </div>
47     <div class="form-group">
48         <input type="text" name="adding_other_Home_Address" class="
other_Home_Address" id="add_New_Home_Adres">
49     </div>
50
51     <button id="add" type="submit" class="btn btn-primary">Submit</
button>
52
53 </form>
54 </div>
55
56 </div>

```

Listing 30: Add New Staff Modal

```

1 @app.route('/add_new_staff', methods=['GET', 'POST'])
2 def add_new_staff():
3     first_name = request.form['fname']
4     last_name = request.form['lname']
5     email = request.form['email']
6     username = request.form['username']
7     password = request.form['password']
8     storeId = request.form['st_Address']
9     cityId = request.form['city']
10    addressId = request.form['h_Address']
11    if addressId == "other":
12        address = request.form['adding_other_Home_Address']
13        addressId = views.insert_address(cityId, address)
14        addressId = addressId['address_id']
15    views.insert_staff(first_name, last_name, addressId, email, storeId,
username, password)

```

```
16 return redirect('/staff/')
```

Listing 31: Add New Staff server.py

Above code is the function getting all the inputs from user and sending them to insert-staff in views.py. If the user wants to new home address, it is checked in the if statement, and send it to insert address function in views.py

```
1 ##### INSERTING INTO STAFF TABLE #####
2 def insert_staff(first_name, last_name, addressId, email, storeId, username
  , password):
3     try:
4         cur = mydb.cursor()
5         cur.execute("""INSERT INTO sakila.staff
6             (first_name,last_name,address_id,email,store_id,active,
7             username,password)
8             values (%s,%s,%s,%s,%s,1,%s,%s)""",
9             (first_name, last_name, addressId, email, storeId,
10             username, password))
11         mydb.commit()
12         cur.close()
13         flash("Data inserted into staff table successfully!", category=
14         "success")
15     except:
16         flash("Data not inserted into staff table successfully!",
17         category="wrong")
18 ##### INSERTING NEW ADDRESS #####
19 def insert_address(id,name):
20     try:
21         cur = mydb.cursor()
22         cur.execute("Insert into sakila.address (address,city_id,district,
23         phone) values (%s,%s,%s,%s)", (name,id, "", ""))
24         mydb.commit()
25         cur.close()
26         cur2 = mydb.cursor()
27         cur2.execute("Select address_id from sakila.address where address =
28         %s" , (name,))
29         address = cur2.fetchone()
30         flash("Data inserted into address table successfully!", category="
31         success")
32         return address
33     except:
34         flash("Data is not inserted into address table successfully!",
```

```
category="error")
```

Listing 32: Add New Staff and new address(if necessary) in views.py

2.2.5 Deleting a Staff

Delete function takes the id of the staff wanting to be deleted and controls if the staff is manager. And if the staff is manager, the user will not be deleted, otherwise he/she can be deleted.

```
1 <a class="delete" href="/delete_from_staff/{{tuple.staff_id}}"><i class="fa  
-solid fa-trash"></i></a>
```

Listing 33: Delete icon

```
1 @app.route('/delete_from_staff/<int:id>')  
2 def delete_from_staff(id):  
3     views.delete_staff(id)  
4     return redirect('/staff/')
```

Listing 34: Delete function in server.py

```
1 ##### DELETE FROM STAFF TABLE #####  
2 def delete_staff(id):  
3     try:  
4         if manager_control(id):  
5             flash("MANAGERS CAN'T BE DELETED")  
6         else:  
7             cur = mydb.cursor()  
8             cur.execute("DELETE FROM sakila.staff where staff_id = %s" % id  
9             )  
10             mydb.commit()  
11             cur.close()  
12             flash("Data deleted from staff table successfully!", category="success")  
13         except:  
14             flash("Couldn't delete from staff! Please try again", category="error")  
15 ##### CONTROLLING WHETHER THE STAFF IS MANAGER #####  
16 def manager_control(id):  
17     cur=mydb.cursor()  
18     cur.execute("Select st.manager_staff_id from sakila.staff st where  
manager_staff_id = %s" % id)
```

```

19     isManager = cur.fetchone()
20     if type(isManager) == dict:
21
22         return True
23     else:
24         return False

```

Listing 35: Delete and manager control function in views.py

2.2.6 Updating Staff

When edit icon is clicked, it opens a update staff modal.

```

1 <div id="updateModal" class="modal">
2   <div id="updateStaff" class="add_to_staff">
3     <form action="/update_staff/" method="POST">
4       <div class="form-group">
5         <label for="id">ID:</label>
6         <input type="text" name="id" id="id" class="form-control" readonly>
7       </div>
8       <div class="form-group">
9         <label for="f_name">First Name:</label>
10        <input type="text" name="f_name" id="f_name" class="form-control"
11        required="1">
12      </div>
13      <div class="form-group">
14        <label for="f_name">Last Name:</label>
15        <input type="text" name="l_name" id="l_name" class="form-control"
16        required="1">
17      </div>
18      <div class="form-group">
19        <label for="f_name">Email:</label>
20        <input type="text" name="em" id="em" class="form-control" required=
21        "1">
22      </div>
23      <div class="form-group">
24        <label for="exampleInputEmail1">Username</label>
25        <input name="usn" type="text" class="form-control" id="usn"
26        placeholder="Username" required="1">
27      </div>
28      <div class="form-group">
29        <label for="email">Country:</label>
30        <select class="form-control country" id="u_country" name="u_country"
31        required="1">

```

```

27         {% for row in country %}
28         <option value="{{row.country_id}}">{{row.country}}</option>
29         {% endfor %}
30     </select>
31 </div>
32 <div class="form-group">
33     <label for="city">City:</label>
34     <select class="form-select city" id="u_city" name="u_city" required="
35 1"></select>
36 </div>
37 <div class="form-group">
38     <label for="st_Address">Store Address:</label>
39     <select class="form-select st_Address" id="storeAddress" name="
40 storeAddress" required="1"></select>
41 </div>
42 <div class="form-group">
43     <label for="h_Address">Home Address: </label>
44     <select class="form-select h_Address" id="homeAddress" name="
45 homeAddress" required="1">
46 </select>
47 </div>
48     <button id="upd" type="submit" class="btn btn-primary">Update</button>
49 </div>

```

Listing 36: Update Staff Modal

```

1
2 @app.route('/update_staff/', methods=['GET', 'POST'])
3 def update_staff():
4     id = request.form['id']
5     int_id = int(id)
6     fname = request.form['f_name']
7     lname = request.form['l_name']
8     email = request.form['em']
9     username = request.form['usn']
10    stAdd = request.form['storeAddress']
11    hAdd = request.form['homeAddress']
12    views.update_staff(fname, lname, email, stAdd, username, hAdd, int_id)
13

```

```
14 return redirect('/staff/')
```

Listing 37: Update Function in server.py

Above code takes the input values and send them to the update-staff function in views.py

```
1
2 ##### UPDATING STAFF TABLE #####
3 def update_staff(fname, lname, email, stAdd, username, hAdd, int_id):
4     try:
5         if manager_control(int_id):
6             flash("MANAGERS CAN'T BE EDITED")
7         else:
8             cur = mydb.cursor()
9             cur.execute("Update sakila.staff SET first_name = %s ,
10 last_name = %s,email=%s,store_id=%s,username=%s,address_id=%s where
11 staff_id = %s",
12 (fname, lname, email, stAdd, username, hAdd, int_id))
13 mydb.commit()
14 cur.close()
15 flash("Staff table updated successfully!", category="success")
16 except:
17     flash("Couldn't update staff table! Please try again", category="
18 error")
19
20 ##### CONTROLLING WHETHER THE STAFF IS MANAGER #####
21 def manager_control(id):
22     cur=mydb.cursor()
23     cur.execute("Select st.manager_staff_id from sakila.store st where
24 manager_staff_id = %s" % id)
25 isManager = cur.fetchone()
26 if type(isManager) == dict:
27
28     return True
29 else:
30     return False
```

Listing 38: Update Function and Manager Control in views.py

To control manager, function fetches manager id from store table. If the staff is a manager, it is a dict type object. Like it has storeid,addressid,managerid etc. However there is no returning from cur.execute so it is non type object. The function checkes type of the isManager variable which has the data of execution. If it returns a value it is dict type, if it does not it is non type. According to this, the program determines the staff is manager or not