Birla Institute of Technology and Science

CS F212

Database Systems

Mess Management System

<u>Submitted To</u>

Ramakant Upadhyay

<u>Submitted By</u>

Group 4

Sanchit Kabra (2020A7PS0010P)

Aman Rameshchandra Agrawal (2022A7PS0036P)

Sarthak Bhagwan Ingle (2022A7PS0037P)

Kshitiz Gupta (2022A7PS0057P)

Rudresh Joshi (2022A7PS0159P)

Index

1.1 Problem Statement

Traditional campus mess systems often struggle with several inefficiencies that hinder both operational effectiveness and student satisfaction. These systems rely heavily on manual processes, leading to errors and delays in meal booking, leave applications, and feedback collection. This not only creates frustration among students but also contributes to food wastage due to inaccurate meal planning and preparation. Moreover, the lack of flexibility in accommodating students' dietary preferences and varying schedules further exacerbates the problem.

These inefficiencies underscore the need for a modernized approach to mess management that can streamline operations, reduce waste, and enhance the overall dining experience for students. By addressing these challenges, the Mess Management System aims to revolutionize campus dining by providing a comprehensive solution that automates key processes, promotes sustainability, and meets the evolving needs of the student community.

1.2 Requirements/Features

The Mess Management System is designed to encompass a wide array of features that cater to the diverse needs of students while ensuring efficient resource allocation and utilization. Central to its functionality is the user registration system, which provides students with seamless access to the platform using their student ID or email. This serves as the gateway for students to utilize various features such as meal booking, where they can specify their dining plans for each day and receive incentives for providing advance notice. Additionally, the system offers mess leave applications to allow students to communicate their absence effectively, thereby minimizing unnecessary food preparation and waste

Recognizing the importance of flexibility and convenience, the system also includes features such as add-on meal preparation, which offers students additional meal options like khichdi, and food packing services for those who are traveling or have busy schedules. To promote sustainability and resource efficiency, the system implements leftover tracking capabilities to monitor and manage surplus food effectively. Furthermore, by allowing mess-wise management, the system ensures localized control and accountability, enabling administrators to tailor services to the specific needs of each mess.

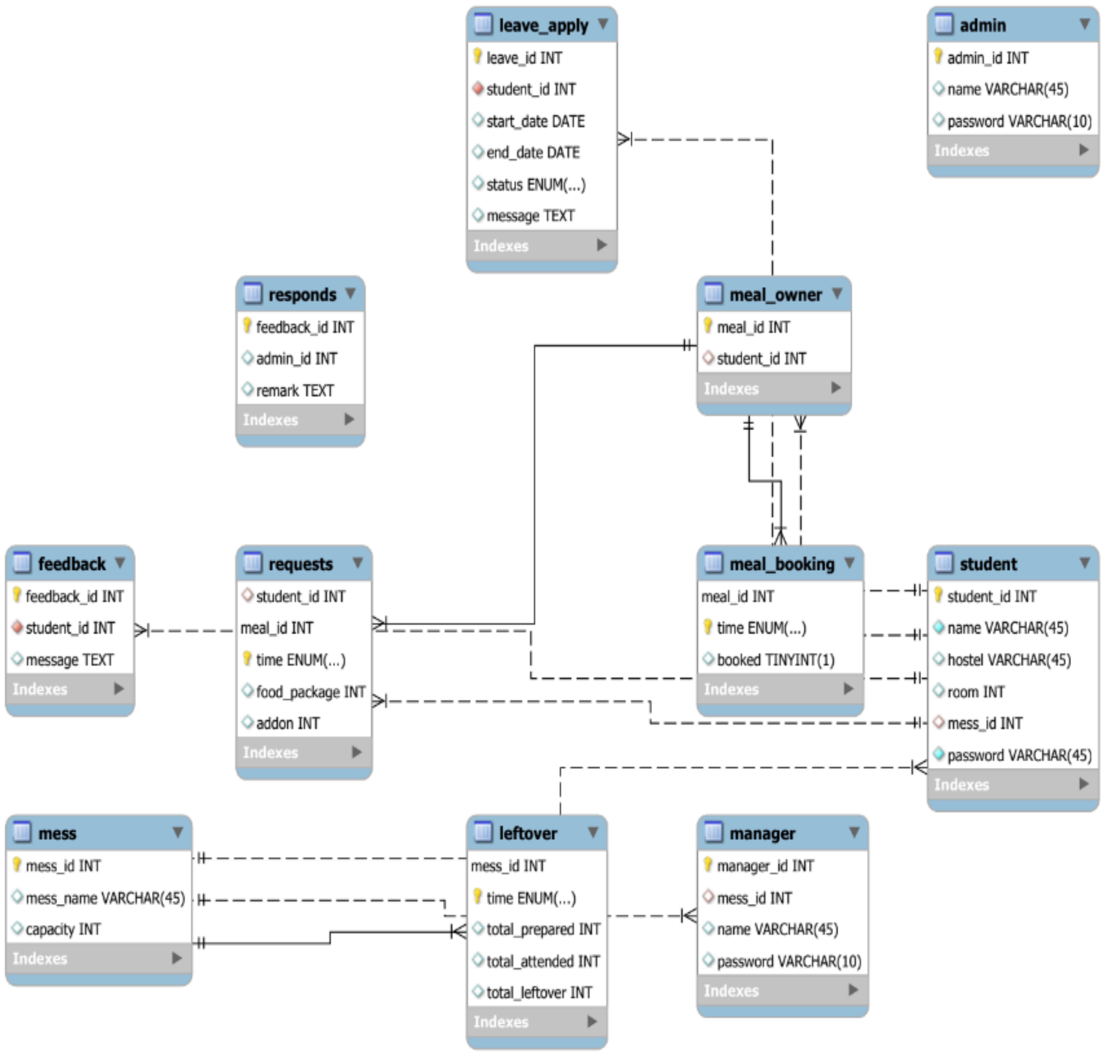
A robust feedback system empowers students to provide valuable input on the quality of mess services, facilitating continuous improvement. Meanwhile, a notification system keeps students informed about meal timings, menu changes, and other relevant updates in real-time. Lastly, an intuitive admin panel provides administrators with the tools they need to oversee operations, manage users, and monitor feedback effectively, thereby ensuring the smooth functioning of the system and enhancing user satisfaction.

1.3 Normalization Explained

After deriving the attribute closure and functional dependencies of original Relational Schema, we came to know that most of the relations (Tables) are redundancy free and are in 3NF/BCNF.

Although, the relational (Table) ‘Meal’ was in 1NF and not in 2NF due to Partial dependencies. In order to remove partial dependency, we performed a decomposition which resulted in 2 new relations. The two relations are - Meal_Booking (Meal_ID, Time, Booked) and Meal_Owner (Meal_ID, Student_ID). The decomposition is lossless but not dependency preserving. Though a functional dependency is lost, it does not affect the overall efficiency of our Database System.

1.4 Update Entity-Relation Model



1.5 Updated Relational Schema

1. Student

Attribute	Type	Key
Student_ID	INT	Primary
Name	VARCHAR	-
Hostel	VARCHAR	-
Room	INT	-
Mess_ID	INT	Foreign (References Mess)

2. Meal_Booking

Attribute	Type	Key
Meal_ID	INT	Primary
Time	ENUM(‘Breakfast’, ‘Lunch’, ‘Dinner’)	Primary
Booked	BOOLEAN	-

3. Meal_Owner

Attribute	Type	Key
Meal_ID	INT	Primary
Student_ID	INT	Foreign Key (References Student)

4. Mess

Attribute	**Type**	**Key**
Mess_ID	INT	Primary
Mess_Name	VARCHAR	-
Capacity	INT	-
Attended	INT	-

5. Admin

Attribute	**Type**	**Key**
Admin_ID	INT	Primary
Name	VARCHAR	-

6. Feedback

Attribute	**Type**	**Key**
Feedback_ID	INT	Primary
Student_ID	INT	Foreign (References Student)
Message	TEXT	-

7. Leave

Attribute	**Type**	**Key**
Leave_ID	INT	Primary
Student_ID	INT	Foreign (References Student)
Start_Date	DATE	-
End_Date	DATE	-
Status	ENUM(‘Pending’, ‘Approved’, ‘Rejected’)	-

8. Manager

Attribute	Type	Key
Manager_ID	INT	Primary
Mess_ID	INT	Foreign (References Mess)
Name	VARCHAR	-

9. Leftover

Attribute	Type	Key
Mess_ID	INT	Primary, Foreign (References Mess)
Total_Prepared	INT	-
Total_Leftover	INT	-
Time	ENUM(‘Breakfast’, ‘Lunch’, ‘Dinner’)	Primary

10. Requests

Attribute	Type	Key
Student_ID	INT	Primary, Foreign (References Student)
Manager_ID	INT	Foreign (References Manager)
Addon	BOOLEAN	-
Food_Package	BOOLEAN	-

11. Responds

Attribute	Type	Key
Feedback_ID	INT	Primary Key, Foreign (References Feedback)
Admin_ID	INT	Foreign (References Admin)
Remark	TEXT	-

1.6 Technical Details

Tech Stack Used -

- **Frontend - JavaFx**
- **Backend - MySQL**

MySQL Workbench - Mess Management System.mwb

MySQL Database Schema - mmsSchema.sql

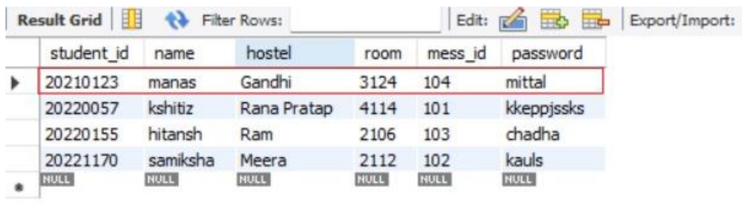
List of Queries Used - List of Queries.sql

Features Implemented -

1. **User Registration -** A student can register to the Portal by the SignUp option provided on the LoginSignUp Page.

   Snapshots -

   (Student Data Inserted in student table)

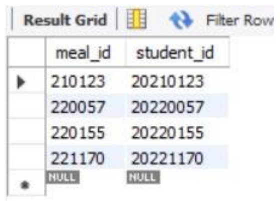


student_id	name	hostel	room	mess_id	password
20210123	manas	Gandhi	3124	104	mittal
20220057	kshitiz	Rana Pratap	4114	101	kkeppjssks
20220155	hitansh	Ram	2106	103	chadha
20221170	samiksha	Meera	2112	102	kauls
NULL	NULL	NULL	NULL	NULL	NULL

   Query Used -

```
-- Insert new student record query for signup
INSERT INTO student (student_id, name, hostel, room, mess_id, password) VALUES (?, ?, ?, ?, ?, ?);
-- This query inserts a new student record into the student table when a user signs up.
-- Sample query:
-- INSERT INTO student (student_id, name, hostel, room, mess_id, password) VALUES (20220057, 'kshitiz', 'RP', 4114, 101, '1234');
```

   (Meal ID Generated)

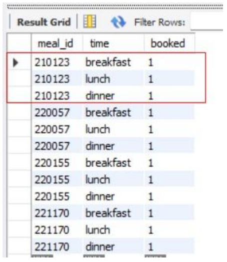


meal_id	student_id
210123	20210123
220057	20220057
220155	20220155
221170	20221170
NULL	NULL

   Query Used -

```
-- Insert new record into meal_owner table for signup
INSERT INTO meal_owner (meal_id, student_id) VALUES (?, ?);
-- This query inserts a new record into the meal_owner table, mapping the meal ID to the student ID.
-- Sample query:
-- INSERT INTO meal_owner (meal_id, student_id) VALUES (220057, 20220057);
-- The meal_id is generated automatically based on the last six digits of the student_id.
```

(Initializing Meal Booking Table)

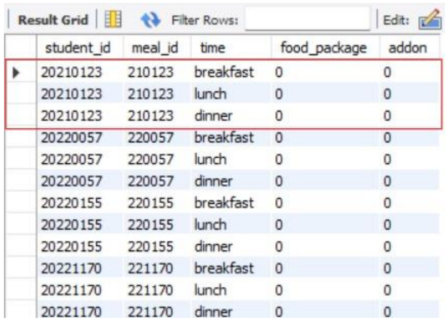


meal_id	time	booked
210123	breakfast	1
210123	lunch	1
210123	dinner	1
220057	breakfast	1
220057	lunch	1
220057	dinner	1
220155	breakfast	1
220155	lunch	1
220155	dinner	1
221170	breakfast	1
221170	lunch	1
221170	dinner	1

Query Used -

```
-- Insert initial meal bookings query for signup
INSERT INTO meal_booking (meal_id, time, booked) VALUES (?, ?, 1);
-- This query inserts initial meal bookings for the newly signed-up student for all three meal times (breakfast, lunch, dinner).
-- Sample query:
-- INSERT INTO meal_booking (meal_id, time, booked) VALUES (220057, 'breakfast', 1);
```

(Insert initial meal requests query for signup)

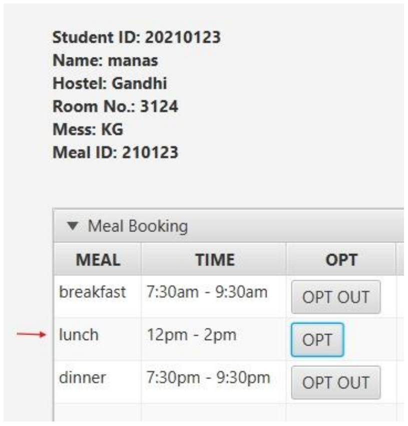


student_id	meal_id	time	food_package	addon
20210123	210123	breakfast	0	0
20210123	210123	lunch	0	0
20210123	210123	dinner	0	0
20220057	220057	breakfast	0	0
20220057	220057	lunch	0	0
20220057	220057	dinner	0	0
20220155	220155	breakfast	0	0
20220155	220155	lunch	0	0
20220155	220155	dinner	0	0
20221170	221170	breakfast	0	0
20221170	221170	lunch	0	0
20221170	221170	dinner	0	0

Query Used -

```
-- Insert initial meal requests query for signup
INSERT INTO requests (student_id, meal_id, time) VALUES (?, ?, ?);
-- This query inserts initial meal requests for the newly signed-up student for all three meal times (breakfast, lunch, dinner).
-- Sample query:
-- INSERT INTO requests (student_id, meal_id, time) VALUES (20220057, 220057, 'breakfast');
```

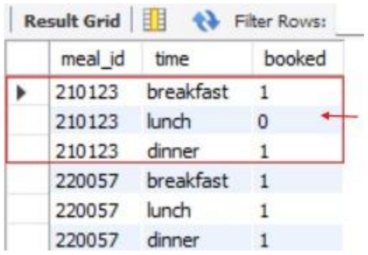
2. **Meal Booking -** Enables students to specify if they will eat in the mess at a particular time (Breakfast, Lunch, Dinner).
   Snapshots - (User Clicks on OPT OUT button)


Student ID: 20210123
Name: manas
Hostel: Gandhi
Room No.: 3124
Mess: KG
Meal ID: 210123

Meal Booking

MEAL	TIME	OPT
breakfast	7:30am - 9:30am	OPT OUT
lunch	12pm - 2pm	OPT
dinner	7:30pm - 9:30pm	OPT OUT

(Booked Status is marked '0' for lunch, as he pressed the OPTED OUT button for lunch)

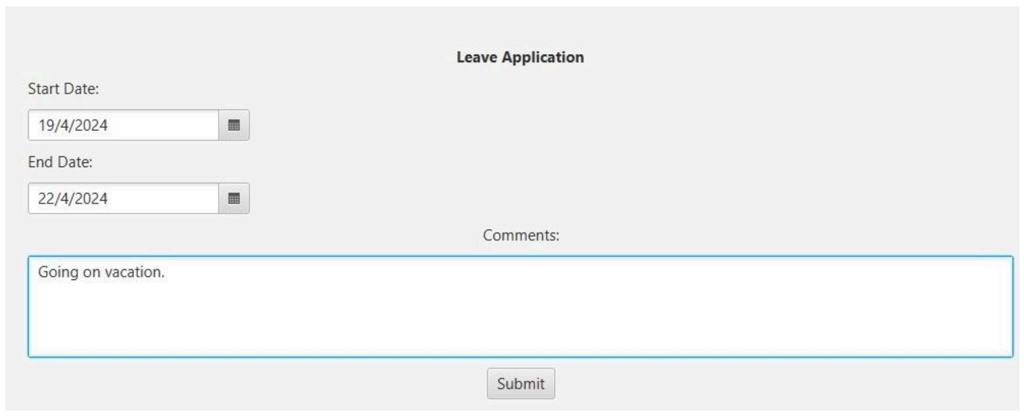


meal_id	time	booked
210123	breakfast	1
210123	lunch	0
210123	dinner	1
220057	breakfast	1
220057	lunch	1
220057	dinner	1

Query Used -

```
-- 5. Update statement to toggle the booked status in the meal_booking table based on meal ID and time
UPDATE meal_booking
SET booked = CASE
              WHEN meal_id = ? AND time = ? THEN ?
              ELSE booked
             END
WHERE meal_id = ? AND time = ?;

-- Example: Toggle meal booking status for meal ID 210123 and time 'lunch' to 1
-- UPDATE meal_booking
-- SET booked = CASE
--              WHEN meal_id = 210123 AND time = 'lunch' THEN 1
--              ELSE booked
--             END
-- WHERE meal_id = 210123 AND time = 'lunch'
```

3. **Mess Leave Application -** Provide an option for students to apply for mess leave to minimize food waste.
   Snapshots - (Student Enters the details for Leave on Portal)

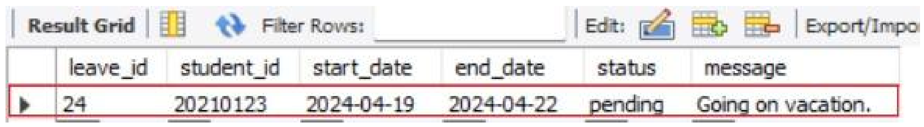


Start Date: 19/4/2024

End Date: 22/4/2024

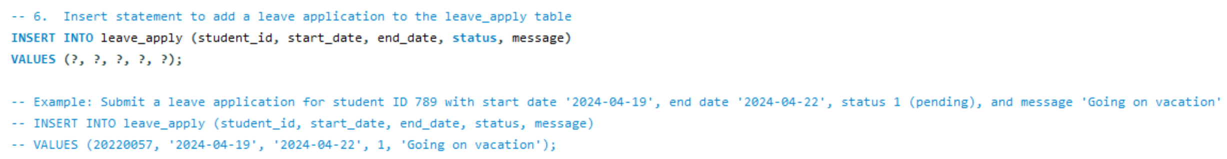
Comments: Going on vacation.

(Details are added to the leave_apply table)


Result Grid
Filter Rows:
Edit:
Export/Impo

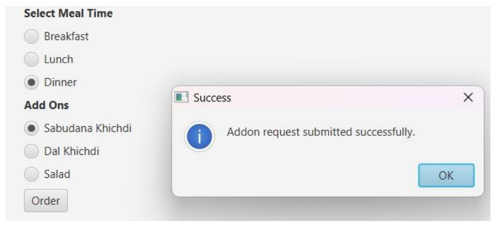
leave_id	student_id	start_date	end_date	status	message
24	20210123	2024-04-19	2024-04-22	pending	Going on vacation.

Query Used -



```
-- 6.  Insert statement to add a leave application to the leave_apply table
INSERT INTO leave_apply (student_id, start_date, end_date, status, message)
VALUES (?, ?, ?, ?, ?);

-- Example: Submit a leave application for student ID 789 with start date '2024-04-19', end date '2024-04-22', status 1 (pending), and message 'Going on vacation'
-- INSERT INTO leave_apply (student_id, start_date, end_date, status, message)
-- VALUES (20220057, '2024-04-19', '2024-04-22', 1, 'Going on vacation');
```

4. **Addon Preparation -** Include an option to prepare addons like khichdi for students.
   Snapshot (Student Selects Sabudana Khichdi as Addon in Dinner on Portal)



Select Meal Time

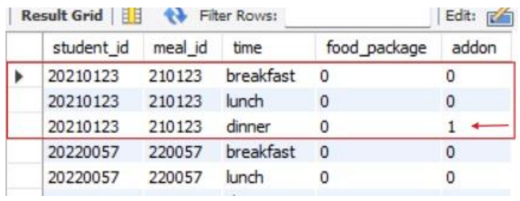
( ) Breakfast
( ) Lunch
(●) Dinner

Add Ons

(●) Sabudana Khichdi
( ) Dal Khichdi
( ) Salad

Addon request submitted successfully.

(Details are updated in ‘requests’ table in database)

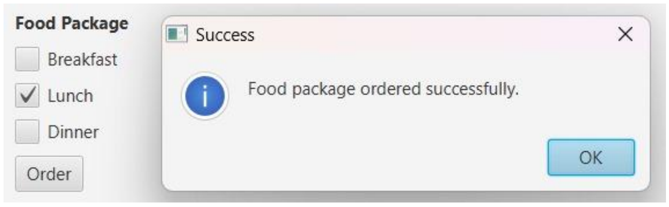

Result Grid
Filter Rows:
Edit:

student_id	meal_id	time	food_package	addon
20210123	210123	breakfast	0	0
20210123	210123	lunch	0	0
20210123	210123	dinner	0	1
20220057	220057	breakfast	0	0
20220057	220057	lunch	0	0

Query Used -

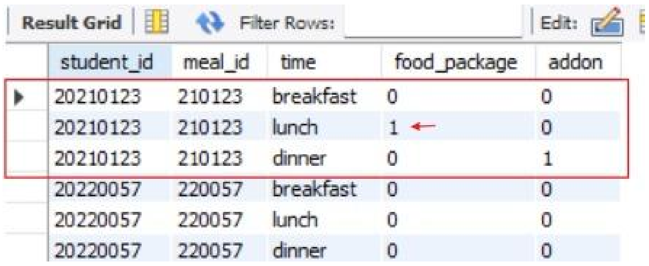
```sql
-- 8.  Update statement to set the addon in the requests table based on student ID, meal ID, and time
UPDATE requests
SET addon = ?
WHERE student_id = ? AND meal_id = ? AND time = ?;

-- Example: Order addon for student ID 20210123, meal ID 210123, and time 'dinner'
-- UPDATE requests
-- SET addon = 1
-- WHERE student_id = 20210123 AND meal_id = 210123 AND time = 'dinner';
```

5. **Food Packing -**
   Keep a feature of packing food for students who are going home from campus for travel or for those who are busy in lectures and skipped their meal.
   Snapshot (Student chooses to pack his lunch on Portal)



Food package ordered successfully.

(Food Package status is updated in requests table)

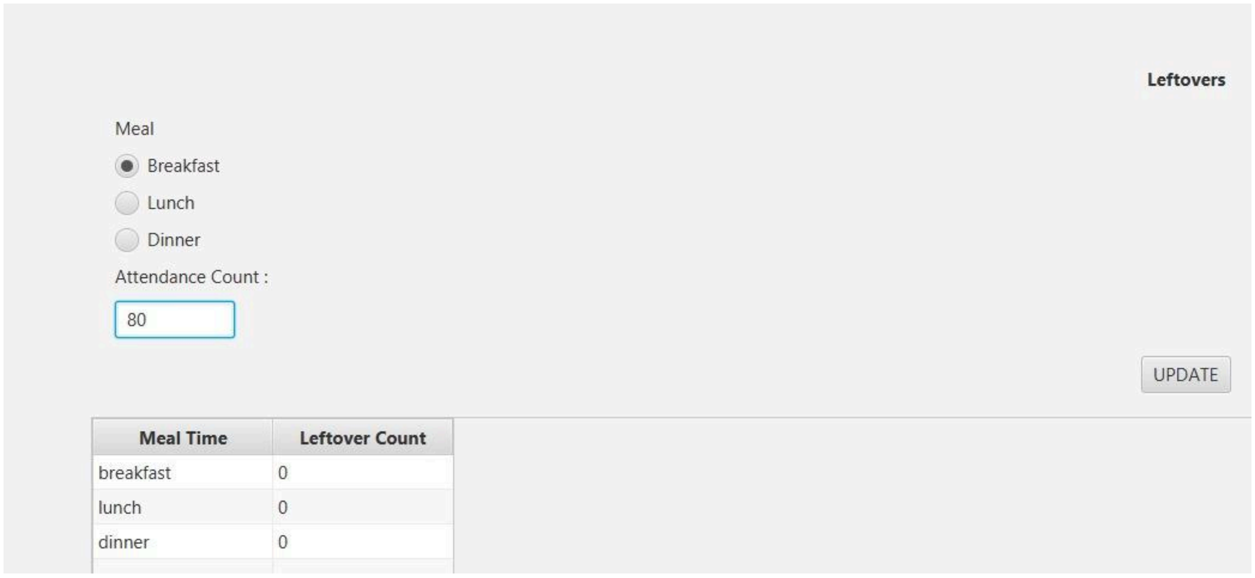

Result Grid
Filter Rows:
Edit:

student_id	meal_id	time	food_package	addon
20210123	210123	breakfast	0	0
20210123	210123	lunch	1	0
20210123	210123	dinner	0	1
20220057	220057	breakfast	0	0
20220057	220057	lunch	0	0
20220057	220057	dinner	0	0

Query Used -

```sql
-- 9.  Update statement to set the food_package in the requests table based on student ID, meal ID, and time
UPDATE requests SET food_package = CASE
    WHEN student_id = ? AND meal_id = ? AND time = ? THEN 1
    WHEN student_id = ? AND meal_id = ? AND time = ? THEN 1
    WHEN student_id = ? AND meal_id = ? AND time = ? THEN 1
    ELSE food_package END
-- The query structure varies based on which checkBox the User Selects
```

6. **Leftover Tracking -** Keep track of leftover foods from messes in the campus.
   Snapshots - (Manager of a particular Mess enters the Attendance of student in mess)


Leftovers

Meal

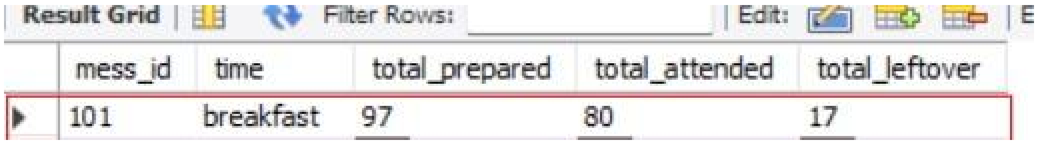
[✓ Breakfast] Lunch Dinner

Attendance Count : 80

UPDATE

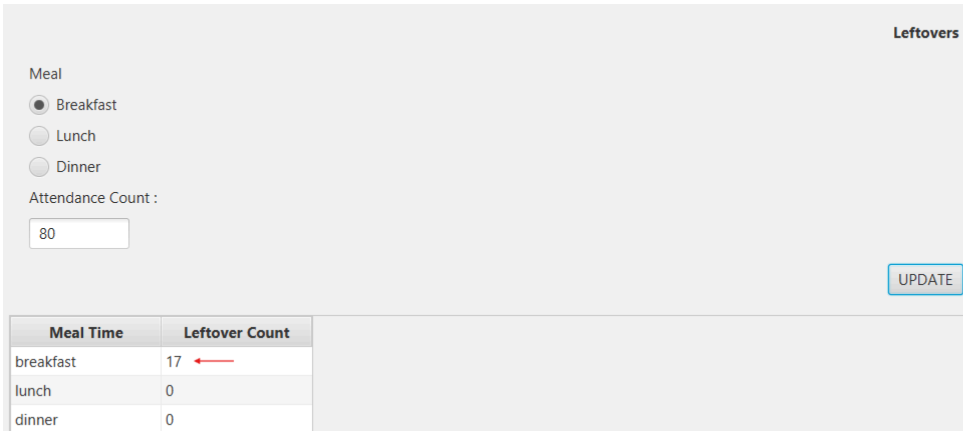
Meal Time	Leftover Count
breakfast	0
lunch	0
dinner	0

(These fields are initialized based on a calculation) -
total_leftover = total_prepared - total_attended
total_prepared = capacity(of that mess) - count of OPTED OUT student eating in that mess at that time from meal_booking table


Result Grid | Filter Rows: | Edit:

mess_id	time	total_prepared	total_attended	total_leftover
101	breakfast	97	80	17

(Leftover count gets updated on panel)


Leftovers

Meal

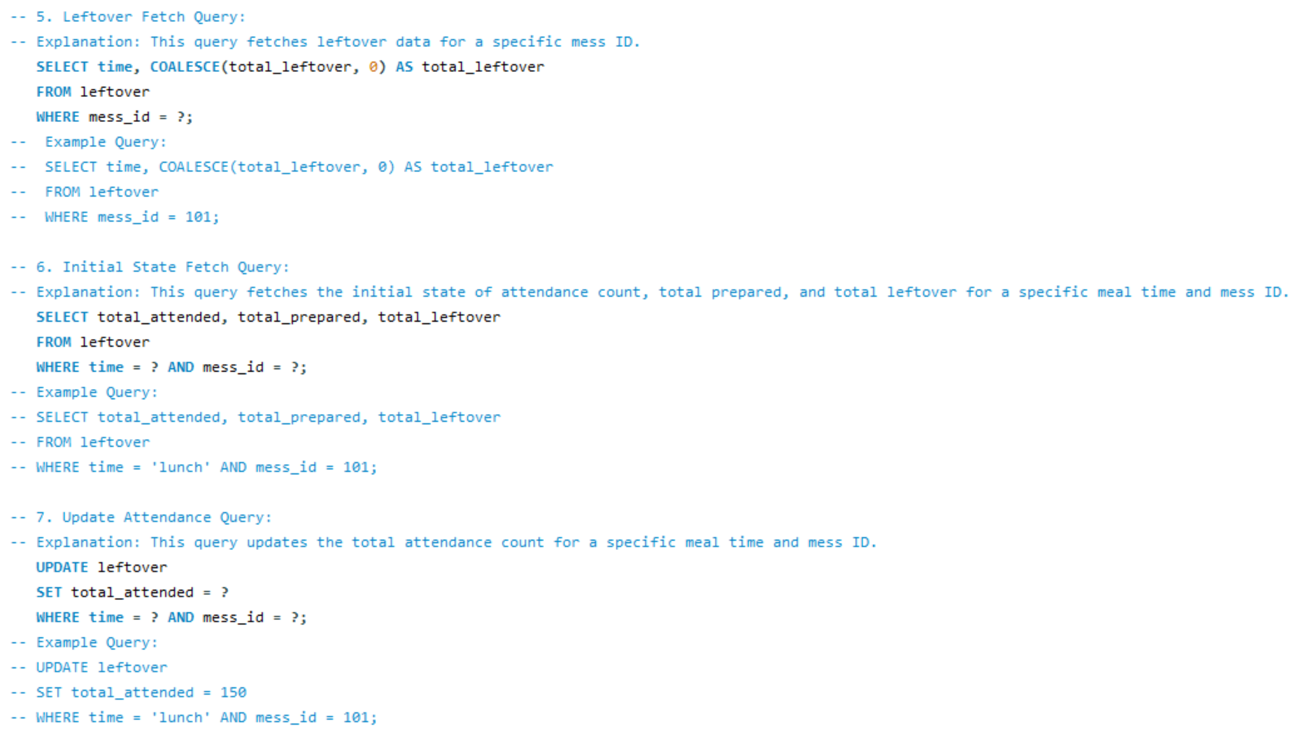
[✓ Breakfast] Lunch Dinner

Attendance Count : 80

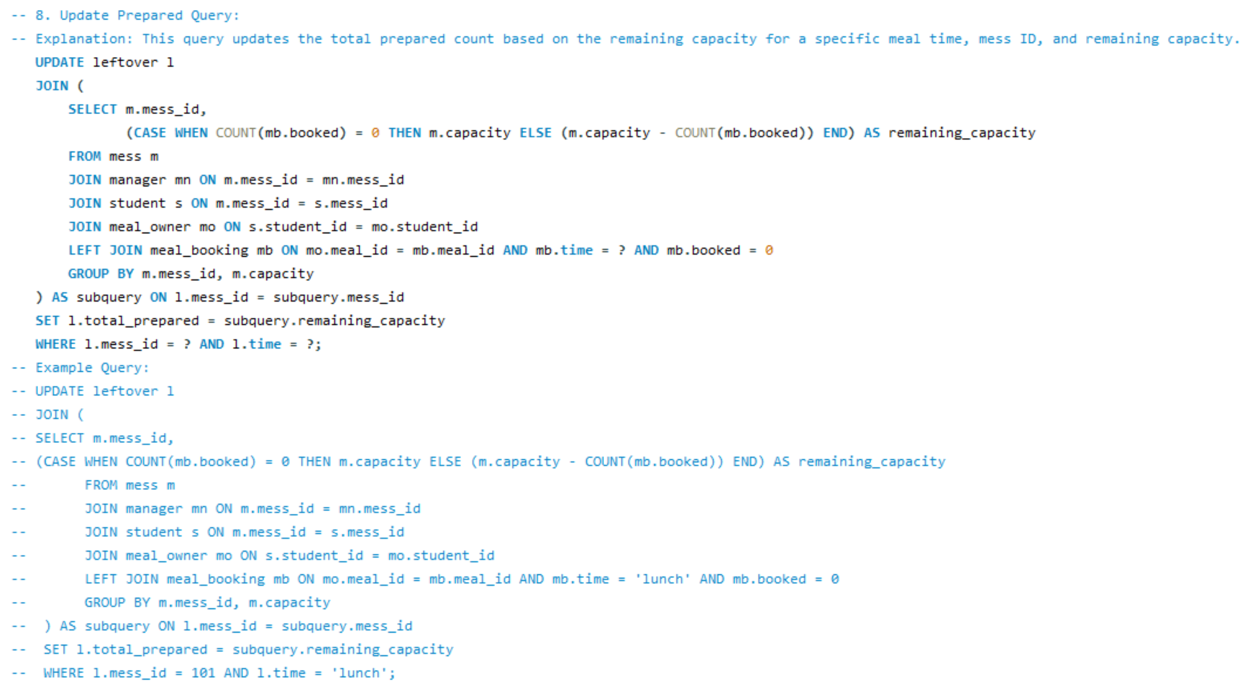
UPDATE

Meal Time	Leftover Count
breakfast	17
lunch	0
dinner	0

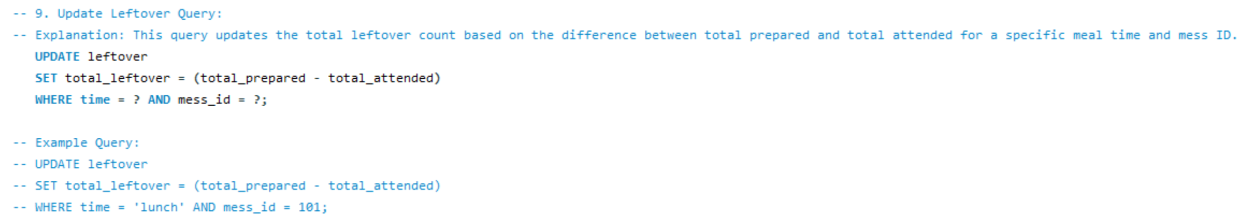
Queries Used -



```
-- 5. Leftover Fetch Query:
-- Explanation: This query fetches leftover data for a specific mess ID.
   SELECT time, COALESCE(total_leftover, 0) AS total_leftover
   FROM leftover
   WHERE mess_id = ?;
--  Example Query:
--  SELECT time, COALESCE(total_leftover, 0) AS total_leftover
--  FROM leftover
--  WHERE mess_id = 101;

-- 6. Initial State Fetch Query:
-- Explanation: This query fetches the initial state of attendance count, total prepared, and total leftover for a specific meal time and mess ID.
   SELECT total_attended, total_prepared, total_leftover
   FROM leftover
   WHERE time = ? AND mess_id = ?;
-- Example Query:
-- SELECT total_attended, total_prepared, total_leftover
-- FROM leftover
-- WHERE time = 'lunch' AND mess_id = 101;

-- 7. Update Attendance Query:
-- Explanation: This query updates the total attendance count for a specific meal time and mess ID.
   UPDATE leftover
   SET total_attended = ?
   WHERE time = ? AND mess_id = ?;
-- Example Query:
-- UPDATE leftover
-- SET total_attended = 150
-- WHERE time = 'lunch' AND mess_id = 101;
```



```
-- 8. Update Prepared Query:
-- Explanation: This query updates the total prepared count based on the remaining capacity for a specific meal time, mess ID, and remaining capacity.
   UPDATE leftover l
   JOIN (
       SELECT m.mess_id,
              (CASE WHEN COUNT(mb.booked) = 0 THEN m.capacity ELSE (m.capacity - COUNT(mb.booked)) END) AS remaining_capacity
       FROM mess m
       JOIN manager mn ON m.mess_id = mn.mess_id
       JOIN student s ON m.mess_id = s.mess_id
       JOIN meal_owner mo ON s.student_id = mo.student_id
       LEFT JOIN meal_booking mb ON mo.meal_id = mb.meal_id AND mb.time = ? AND mb.booked = 0
       GROUP BY m.mess_id, m.capacity
   ) AS subquery ON l.mess_id = subquery.mess_id
   SET l.total_prepared = subquery.remaining_capacity
   WHERE l.mess_id = ? AND l.time = ?;
-- Example Query:
-- UPDATE leftover l
-- JOIN (
-- SELECT m.mess_id,
-- (CASE WHEN COUNT(mb.booked) = 0 THEN m.capacity ELSE (m.capacity - COUNT(mb.booked)) END) AS remaining_capacity
--       FROM mess m
--       JOIN manager mn ON m.mess_id = mn.mess_id
--       JOIN student s ON m.mess_id = s.mess_id
--       JOIN meal_owner mo ON s.student_id = mo.student_id
--       LEFT JOIN meal_booking mb ON mo.meal_id = mb.meal_id AND mb.time = 'lunch' AND mb.booked = 0
--       GROUP BY m.mess_id, m.capacity
--  ) AS subquery ON l.mess_id = subquery.mess_id
--  SET l.total_prepared = subquery.remaining_capacity
--  WHERE l.mess_id = 101 AND l.time = 'lunch';
```

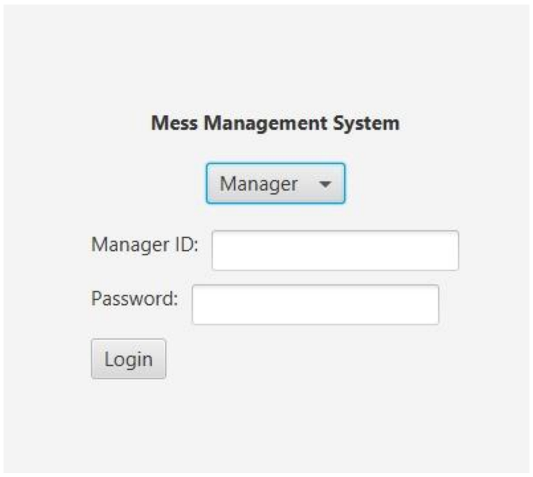


```
-- 9. Update Leftover Query:
-- Explanation: This query updates the total leftover count based on the difference between total prepared and total attended for a specific meal time and mess ID.
   UPDATE leftover
   SET total_leftover = (total_prepared - total_attended)
   WHERE time = ? AND mess_id = ?;

-- Example Query:
-- UPDATE leftover
-- SET total_leftover = (total_prepared - total_attended)
-- WHERE time = 'lunch' AND mess_id = 101;
```

7. **Mess-wise Management -** Manage mess services separately for each mess.

Snapshots :

The Manager needs to login through his Manager ID and password.


Mess Management System

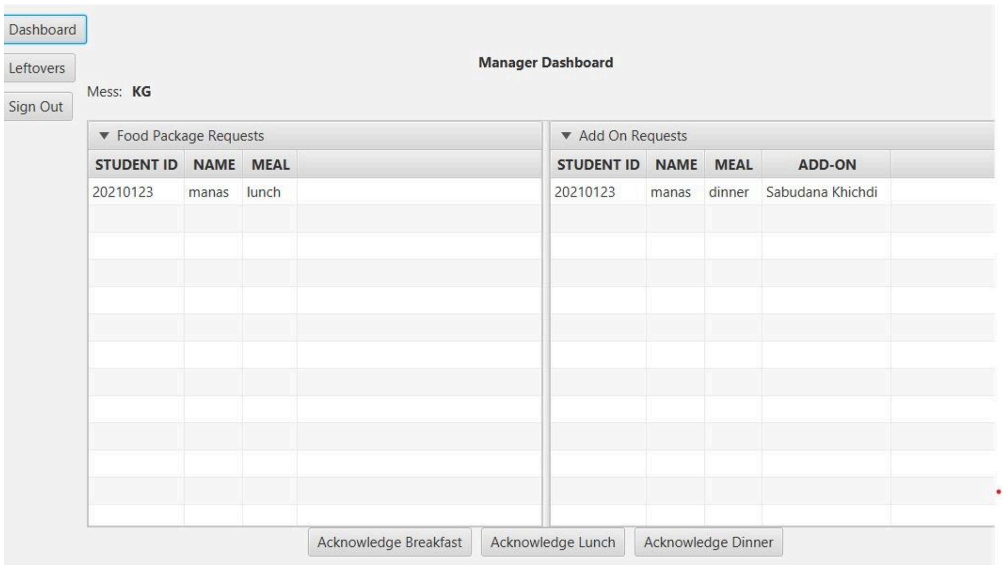
Manager

Manager ID:

Password:

Login

The Manager dashboard contains the Food Package Requests and Add On Requests tables showing details of the student, the meal asked for and add-ons if any.The manager acknowledges the respective order and resets the table values to zero.



Manager Dashboard

Mess: **KG**

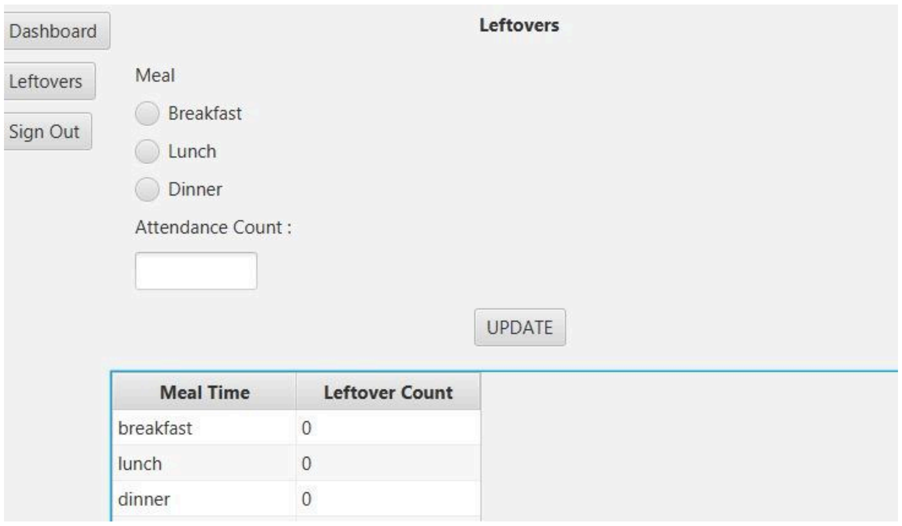
Food Package Requests

STUDENT ID	NAME	MEAL
20210123	manas	lunch

Add On Requests

STUDENT ID	NAME	MEAL	ADD-ON
20210123	manas	dinner	Sabudana Khichdi

The Leftovers section tracks the attendance count of the respective meal and updates it.



Meal Time	Leftover Count
breakfast	0
lunch	0
dinner	0

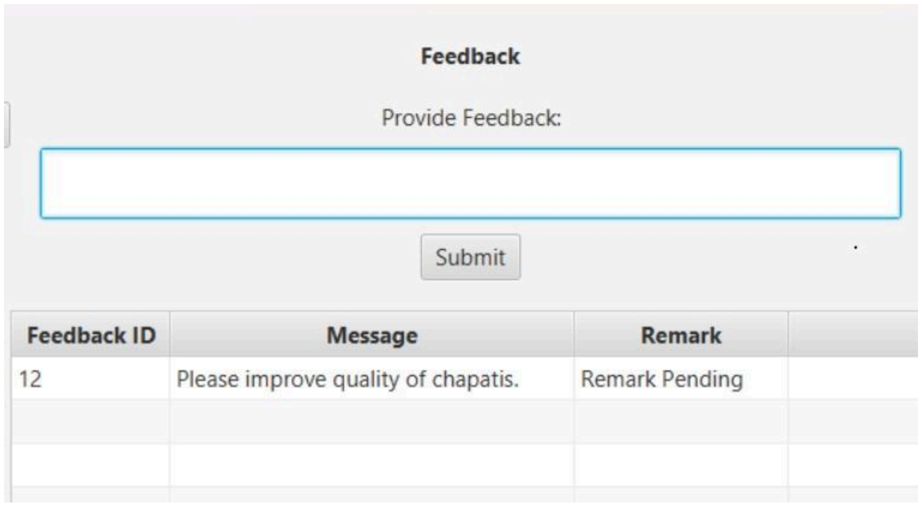
Query Used -

```
-- ManagerPanel queries

-- 1.Retrieve Mess ID Query:sql
-- Explanation: This query retrieves the mess ID, capacity, and name for the mess associated with a specific manager ID.
   SELECT mess_id, mess.capacity, mess_name
   FROM manager
   NATURAL JOIN mess
   WHERE manager_id = ?;
-- Example Query:
-- SELECT mess_id, mess.capacity, mess_name
-- FROM manager
-- NATURAL JOIN mess
-- WHERE manager_id = 101;

-- 2.Food Package Requests Query:
-- Explanation: This query fetches food package requests for a specific mess ID.
   SELECT r.student_id, s.name, r.time
   FROM requests r
   JOIN student s ON r.student_id = s.student_id
   WHERE s.mess_id = ? AND r.food_package = 1;
-- Example Query:
-- SELECT r.student_id, s.name, r.time
-- FROM requests r
-- JOIN student s ON r.student_id = s.student_id
-- WHERE s.mess_id = 101 AND r.food_package = 1
```

```sql
-- 3.Add-On Requests Query:
-- Explanation: This query retrieves add-on requests for a specific mess ID.
   SELECT r.student_id, s.name, r.time, r.addon
   FROM requests r
   JOIN student s ON r.student_id = s.student_id
   WHERE s.mess_id = ? AND r.addon != 0;
-- Example Query:
-- SELECT r.student_id, s.name, r.time, r.addon
-- FROM requests r
-- JOIN student s ON r.student_id = s.student_id
-- WHERE s.mess_id = 101 AND r.addon != 0;

-- 4. Acknowledgement Query:
-- Explanation: This query updates the acknowledgment status for food package and add-on requests based on the meal time.
   UPDATE requests r
   JOIN (
       SELECT r1.meal_id
       FROM requests r1
       JOIN meal_owner mo ON r1.meal_id = mo.meal_id
       JOIN student s ON mo.student_id = s.student_id
       WHERE s.mess_id = ? AND r1.time = ?
   ) AS subquery ON r.meal_id = subquery.meal_id
   SET r.food_package = CASE WHEN r.time = ? THEN 0 ELSE r.food_package END,
       r.addon = CASE WHEN r.time = ? THEN 0 ELSE r.addon END;
-- Example Query:
--  UPDATE requests r
--  JOIN (
--  SELECT r1.meal_id
--  FROM requests r1
--  JOIN meal_owner mo ON r1.meal_id = mo.meal_id
--  JOIN student s ON mo.student_id = s.student_id
--  WHERE s.mess_id = 101 AND r1.time = 'breakfast'
--  ) AS subquery ON r.meal_id = subquery.meal_id
--  SET r.food_package = CASE WHEN r.time = 'breakfast' THEN 0 ELSE r.food_package END,
--  r.addon = CASE WHEN r.time = 'breakfast' THEN 0 ELSE r.addon END;
```

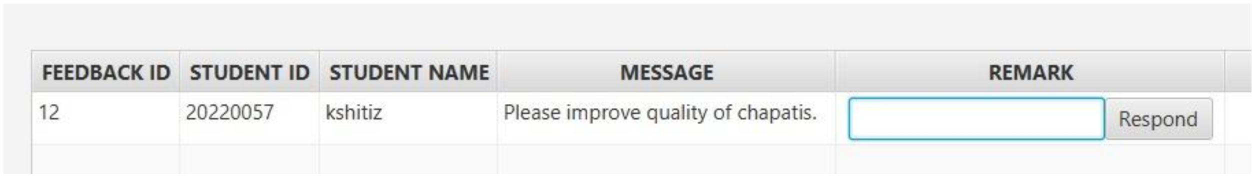
8. **Feedback System -** Allow students to rate and review the service provided by the mess and get response for it.
   Snapshots -

   Student Feedback Panel

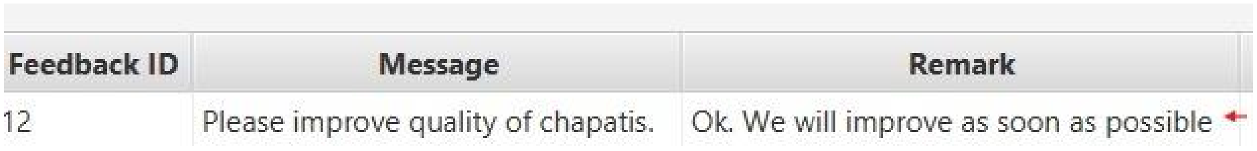


Feedback ID	Message	Remark
12	Please improve quality of chapatis.	Remark Pending

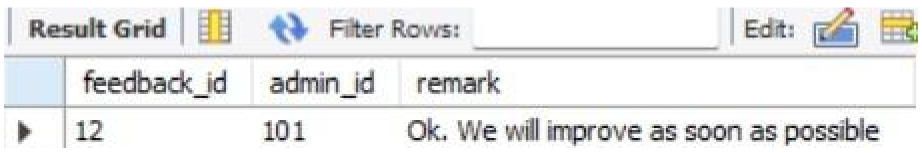
   Admin's Response



FEEDBACK ID	STUDENT ID	STUDENT NAME	MESSAGE	REMARK
12	20220057	kshitiz	Please improve quality of chapatis.	



Feedback ID	Message	Remark
12	Please improve quality of chapatis.	Ok. We will improve as soon as possible



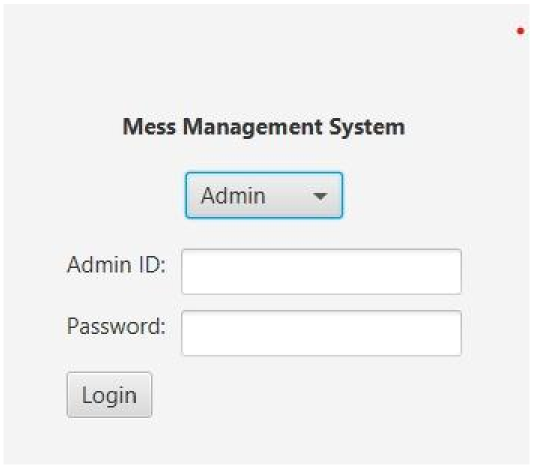
feedback_id	admin_id	remark
12	101	Ok. We will improve as soon as possible

Query Used -

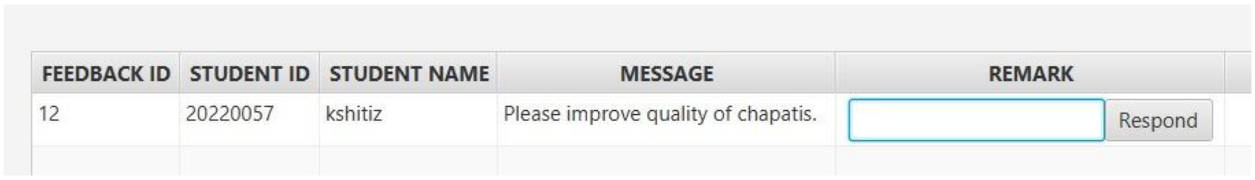
```
-- 7. Insert statement to add feedback to the feedback table
INSERT INTO feedback (student_id, message)
VALUES (?, ?);

-- Example: Submit feedback for student ID 456 with message 'The food quality is excellent'
-- INSERT INTO feedback (student_id, message)
-- VALUES (20220057, 'The food quality is excellent')
```

```
-- Query to fetch feedback along with student information
SELECT student.student_id, student.name, feedback.message
FROM feedback
NATURAL JOIN student;

-- Query to fetch feedback along with student information and admin responses
SELECT feedback.feedback_id, student.student_id, student.name AS student_name, feedback.message, responds.remark
FROM feedback
JOIN student ON student.student_id = feedback.student_id
LEFT JOIN responds ON feedback.feedback_id = responds.feedback_id;

-- Query to insert or update admin remarks on feedback responses
INSERT INTO responds (feedback_id, admin_id, remark)
VALUES (?, ?, ?)
ON DUPLICATE KEY UPDATE remark = VALUES(remark);
-- Example query
-- INSERT INTO responds (feedback_id, admin_id, remark)
-- VALUES (6, 1212, 'Your request will be fulfilled as soon as possible.');
-- ON DUPLICATE KEY UPDATE remark = VALUES(remark);
```

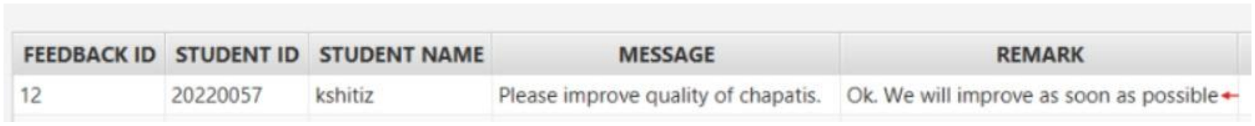
9. **Admin Panel -** An admin panel for system administrators to manage user's leave and view feedback and respond to them.
Snapshots-
Admin Login panel



Admin's Response

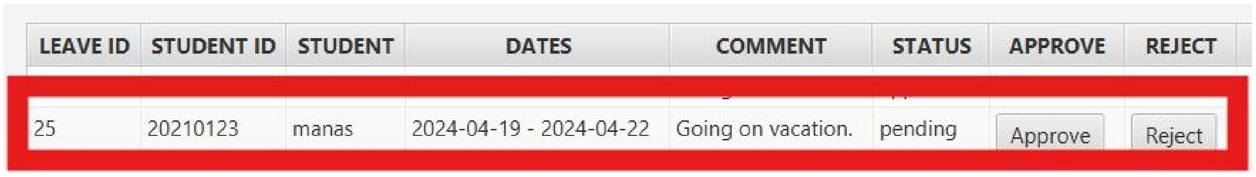


FEEDBACK ID	STUDENT ID	STUDENT NAME	MESSAGE	REMARK
12	20220057	kshitiz	Please improve quality of chapatis.	Respond

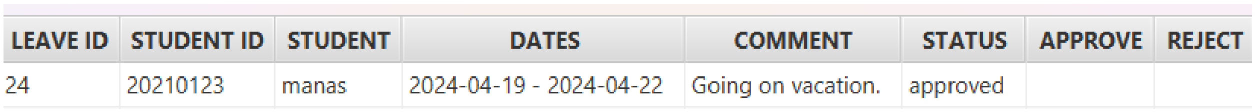


FEEDBACK ID	STUDENT ID	STUDENT NAME	MESSAGE	REMARK
12	20220057	kshitiz	Please improve quality of chapatis.	Ok. We will improve as soon as possible

Leave Approval

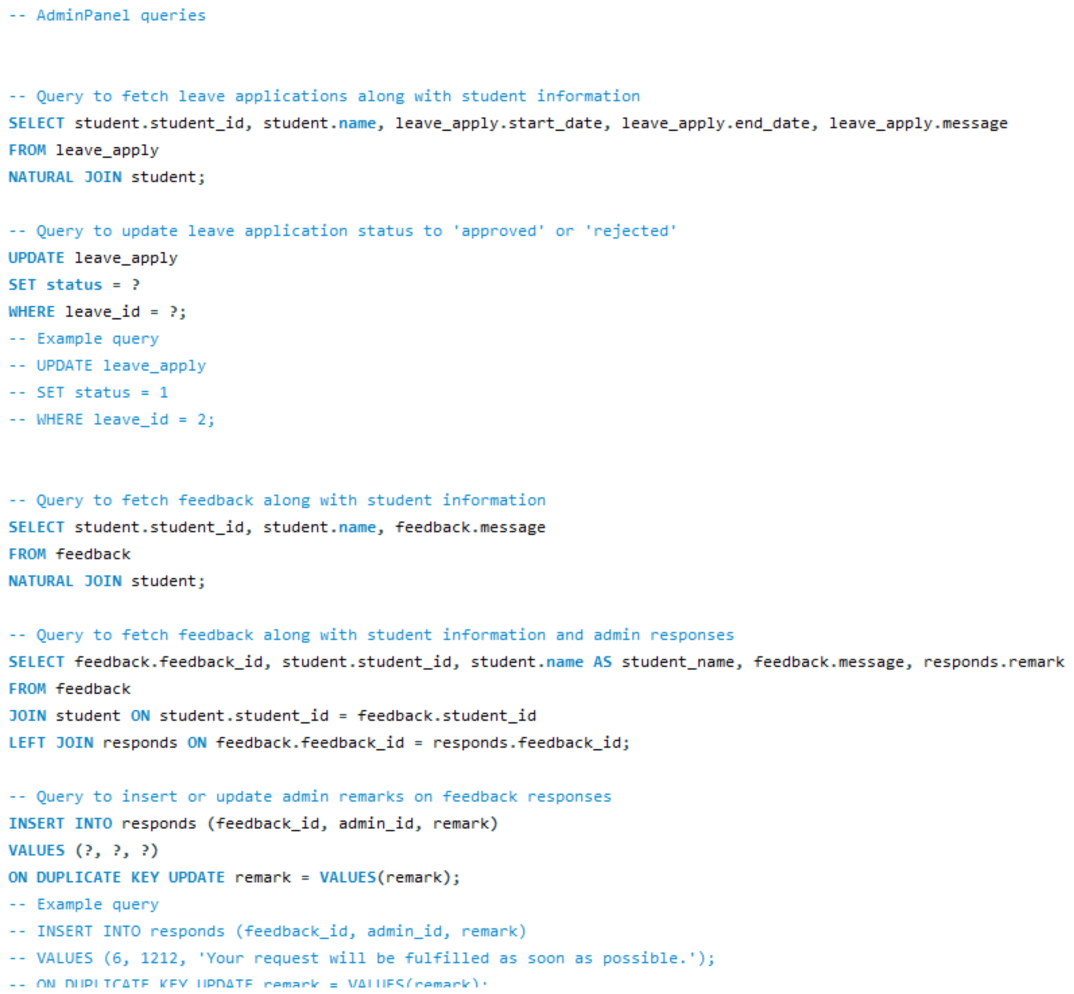


LEAVE ID	STUDENT ID	STUDENT	DATES	COMMENT	STATUS	APPROVE	REJECT
25	20210123	manas	2024-04-19 - 2024-04-22	Going on vacation.	pending	Approve	Reject

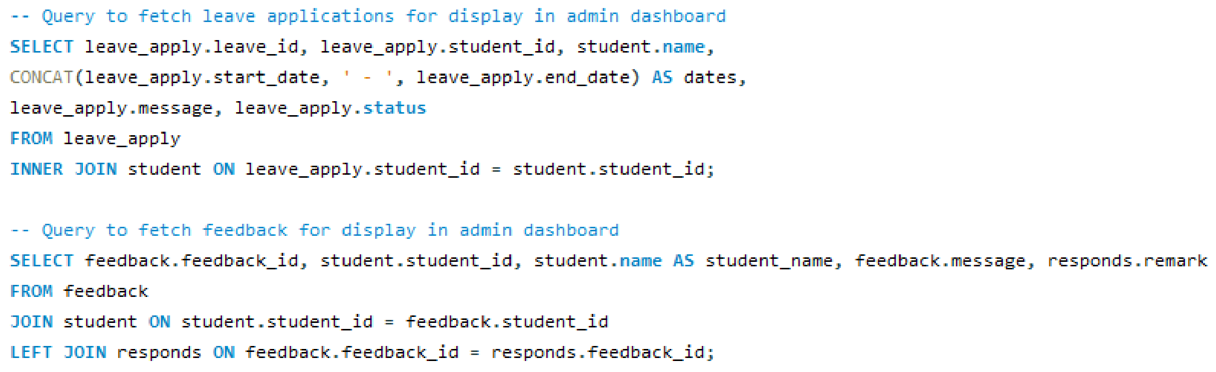


LEAVE ID	STUDENT ID	STUDENT	DATES	COMMENT	STATUS	APPROVE	REJECT
24	20210123	manas	2024-04-19 - 2024-04-22	Going on vacation.	approved		

Query Used -



```
-- AdminPanel queries


-- Query to fetch leave applications along with student information
SELECT student.student_id, student.name, leave_apply.start_date, leave_apply.end_date, leave_apply.message
FROM leave_apply
NATURAL JOIN student;

-- Query to update leave application status to 'approved' or 'rejected'
UPDATE leave_apply
SET status = ?
WHERE leave_id = ?;
-- Example query
-- UPDATE leave_apply
-- SET status = 1
-- WHERE leave_id = 2;


-- Query to fetch feedback along with student information
SELECT student.student_id, student.name, feedback.message
FROM feedback
NATURAL JOIN student;

-- Query to fetch feedback along with student information and admin responses
SELECT feedback.feedback_id, student.student_id, student.name AS student_name, feedback.message, responds.remark
FROM feedback
JOIN student ON student.student_id = feedback.student_id
LEFT JOIN responds ON feedback.feedback_id = responds.feedback_id;

-- Query to insert or update admin remarks on feedback responses
INSERT INTO responds (feedback_id, admin_id, remark)
VALUES (?, ?, ?)
ON DUPLICATE KEY UPDATE remark = VALUES(remark);
-- Example query
-- INSERT INTO responds (feedback_id, admin_id, remark)
-- VALUES (6, 1212, 'Your request will be fulfilled as soon as possible.');
-- ON DUPLICATE KEY UPDATE remark = VALUES(remark);
```



```
-- Query to fetch leave applications for display in admin dashboard
SELECT leave_apply.leave_id, leave_apply.student_id, student.name,
CONCAT(leave_apply.start_date, ' - ', leave_apply.end_date) AS dates,
leave_apply.message, leave_apply.status
FROM leave_apply
INNER JOIN student ON leave_apply.student_id = student.student_id;

-- Query to fetch feedback for display in admin dashboard
SELECT feedback.feedback_id, student.student_id, student.name AS student_name, feedback.message, responds.remark
FROM feedback
JOIN student ON student.student_id = feedback.student_id
LEFT JOIN responds ON feedback.feedback_id = responds.feedback_id;
```

Complex Queries -

1. This complex query updates the total prepared count based on the remaining capacity for a specific meal time, mess ID, and remaining capacity.

```
UPDATE leftover l
JOIN (
    SELECT m.mess_id,
           (CASE WHEN COUNT(mb.booked) = 0 THEN m.capacity ELSE (m.capacity - COUNT(mb.booked)) END) AS remaining_capacity
    FROM mess m
    JOIN manager mn ON m.mess_id = mn.mess_id
    JOIN student s ON m.mess_id = s.mess_id
    JOIN meal_owner mo ON s.student_id = mo.student_id
    LEFT JOIN meal_booking mb ON mo.meal_id = mb.meal_id AND mb.time = ? AND mb.booked = 0
    GROUP BY m.mess_id, m.capacity
) AS subquery ON l.mess_id = subquery.mess_id
SET l.total_prepared = subquery.remaining_capacity
WHERE l.mess_id = ? AND l.time = ?;
```

2. This complex query updates the acknowledgement status for food packages and add-on requests based on the meal time.

```
UPDATE requests r
JOIN (
    SELECT r1.meal_id
    FROM requests r1
    JOIN meal_owner mo ON r1.meal_id = mo.meal_id
    JOIN student s ON mo.student_id = s.student_id
    WHERE s.mess_id = ? AND r1.time = ?
) AS subquery ON r.meal_id = subquery.meal_id
SET r.food_package = CASE WHEN r.time = ? THEN 0 ELSE r.food_package END,
    r.addon = CASE WHEN r.time = ? THEN 0 ELSE r.addon END;
```

3. This complex query fetches student credentials including name, hostel, room, mess, and meal ID based on student ID.

```
SELECT s.name, s.hostel, s.room, m.mess_name, mo.meal_id
FROM student s
JOIN mess m ON s.mess_id = m.mess_id
JOIN meal_owner mo ON s.student_id = mo.student_id
WHERE s.student_id = ?;
```

4. This complex query to fetch leave applications for display in admin dashboard

```
SELECT leave_apply.leave_id, leave_apply.student_id, student.name,
CONCAT(leave_apply.start_date, ' - ', leave_apply.end_date) AS dates,
leave_apply.message, leave_apply.status
FROM leave_apply
INNER JOIN student ON leave_apply.student_id = student.student_id;
```

5. This Complex Query fetches feedback for display in admin dashboard

```
SELECT feedback.feedback_id, student.student_id, student.name AS student_name, feedback.message, responds.remark
FROM feedback
JOIN student ON student.student_id = feedback.student_id
LEFT JOIN responds ON feedback.feedback_id = responds.feedback_id;
```

Steps to Run Application -

(Requirements - MySQL Software, Eclipse IDE or Any other IDE for Java Implementation with JDK Version 1.8. JAR files should be included in the Project).

1. Install proper version of JavaFX and Java Connector in the IDE.
2. Create a Java Project and add these JAR files in the Project.
3. Import the code files in the src folder of code file Mess-Frontend.
4. For the MySQL part, open the Mess Management System.mwb in MySQL.
5. Insert the data for admin, mess and manager table. Queries for the same are written in initial part of List of Queries.sql
6. Now, run the MainApp.java as "Run as Application"
7. Mess Management System Interface will Appear.
8. Further details about application are written in ReadMe File.docx. Detailed explanation is given in this File.

1.7 Conclusions

Traditional campus mess systems are often burdened by inefficiencies, leading to frustration among students and contributing to unnecessary food wastage. These inefficiencies stem from manual processes, such as errors in meal booking, delays in processing leave applications, and challenges in collecting and addressing feedback effectively. As a result, students may encounter difficulties in securing meals according to their preferences and schedules, leading to dissatisfaction with the dining experience.

However, the introduction of the Mess Management System represents a significant improvement over traditional methods. By automating various processes, such as meal booking and leave applications, the system streamlines operations and reduces the likelihood of errors. Additionally, features such as efficient feedback collection mechanisms allow for better communication between students and administrators, leading to more responsive and tailored dining services.

Moreover, the system's ability to track and manage leftover food contributes to a reduction in food wastage, aligning with sustainability goals and promoting responsible resource utilization. By optimizing resource allocation and minimizing unnecessary food production, the Mess Management System not only improves operational efficiency but also reduces the environmental impact of campus dining.

Despite these advancements, one notable limitation of the system is its dependency on a live database to enable timely notifications. Without a real-time database connection, students may miss important updates regarding meal schedules, menu changes, or other relevant information. Therefore, ensuring a live database connection is crucial for maximizing the system's effectiveness in enhancing the dining experience for students.