Feedback & Attendance Management System using MySQL

Submitted To:

SureTrust

Submitted By:

Syam kumar Appikatla

G18 POWER BI & SQL

Date:
01-09-2025

/

Abstract

This project integrates **Google Form feedback data** with **meeting attendance records** using MySQL. The main objective is to validate whether the feedback submitted belongs to a user who actually attended the meeting.

The process involves:

- Collecting feedback via Google Forms
- Importing responses into MySQL
- Matching responses with user attendance
- Running a stored procedure that automatically validates and logs the results

If the feedback is valid, it is marked as **processed**; otherwise, it is stored with an **error message**. This project demonstrates **database design, SQL programming, data validation, and reporting**, which can be extended to real-time systems in future.

A stored procedure in MySQL ensures that only feedback from users who actually attended the meeting is marked as valid, while others are logged as errors.

The system also maintains a processing log for transparency and accountability.

By combining real-time data collection (Google Forms) and backend validation (MySQL), this project demonstrates a hybrid approach to feedback management that can be applied to educational institutions, corporate organizations, and training programs.

This solution improves data integrity, efficiency, and transparency.

Introduction:

Feedback plays a vital role in improving the quality of meetings, training sessions, and organizational processes. Traditionally, feedback collection and validation are carried out manually, which is time-consuming, error-prone, and difficult to scale.

In this project, we address these challenges by designing a *database-driven system* that automates the process. The workflow is as follows:

1. Users submit feedback via Google Forms.
2. The responses are stored in Google Sheets.
3. Data is imported into a MySQL database.
4. A stored procedure validates responses against attendance records.
5. Logs and reports are generated for analysis and decision-making.

This approach ensures that:

Feedback is only considered if the user attended the session.

Invalid entries are flagged and logged for review.

Administrators can view real-time participation reports.

The project showcases how **SQL databases** can be leveraged for data validation, automation, and reporting.

Content:

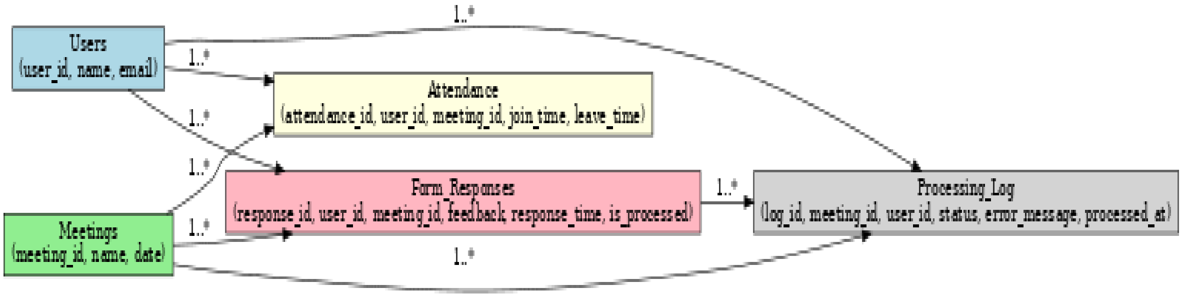
System Requirements

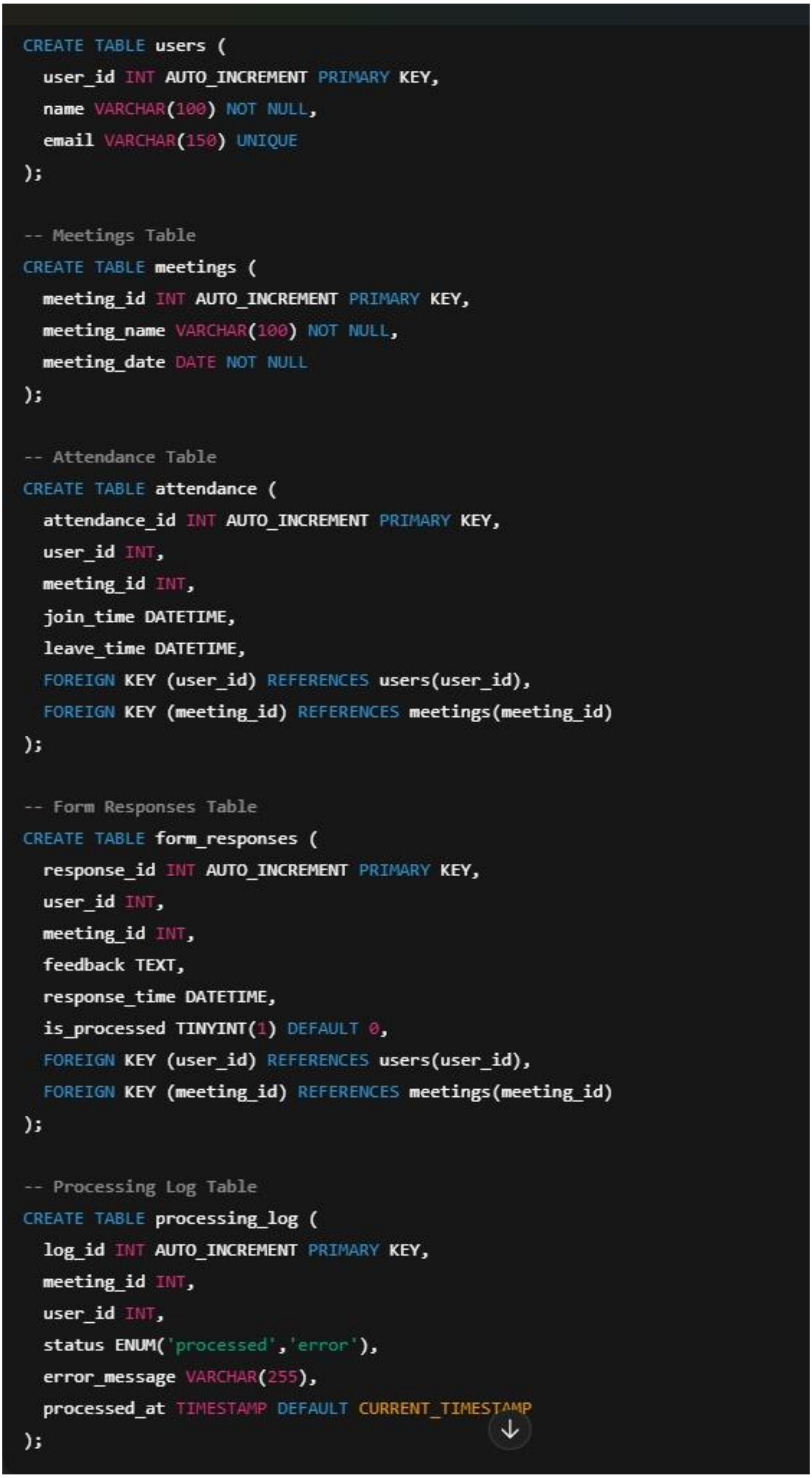
Hardware Requirements

- Processor: Intel i3 or higher
- RAM: Minimum 4 GB
- Storage: 100 MB free space

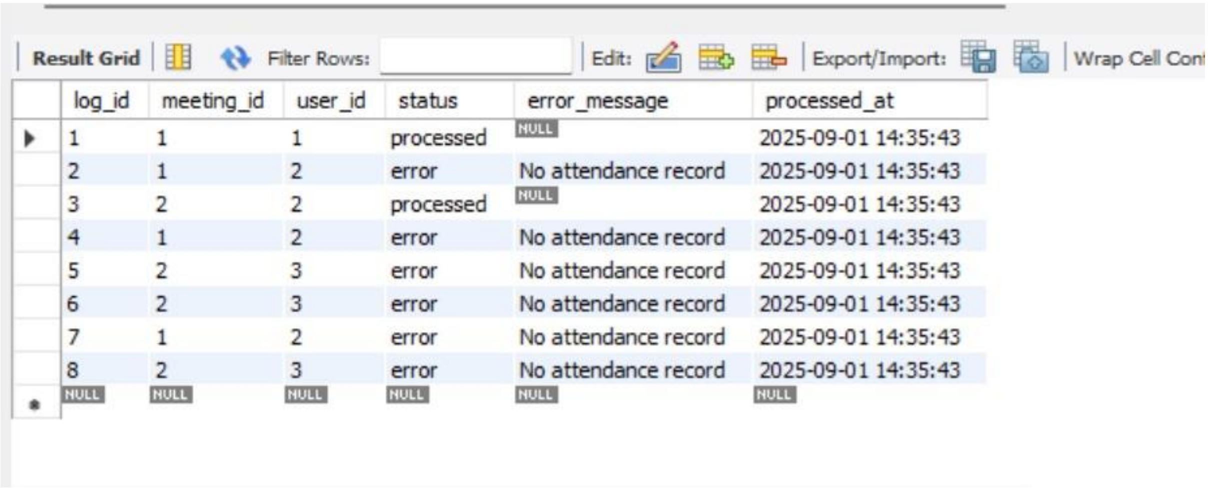
Software Requirements

- MySQL Workbench 8.0 or above
- MySQL Server 8.0 or above
- Python 3.10+ (optional for automation)
- Google Forms & Google Sheets


Users
(user_id, name, email)
1..*
1..*
1..*
Attendance
(attendance_id, user_id, meeting_id, join_time, leave_time)
1..*
Form_Responses
(response_id, user_id, meeting_id, feedback, response_time, is_processed)
1..*
Processing_Log
(log_id, meeting_id, user_id, status, error_message, processed_at)
Meetings
(meeting_id, name, date)
1..*
1..*



```sql
CREATE TABLE users (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(150) UNIQUE
);

-- Meetings Table
CREATE TABLE meetings (
  meeting_id INT AUTO_INCREMENT PRIMARY KEY,
  meeting_name VARCHAR(100) NOT NULL,
  meeting_date DATE NOT NULL
);

-- Attendance Table
CREATE TABLE attendance (
  attendance_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT,
  meeting_id INT,
  join_time DATETIME,
  leave_time DATETIME,
  FOREIGN KEY (user_id) REFERENCES users(user_id),
  FOREIGN KEY (meeting_id) REFERENCES meetings(meeting_id)
);

-- Form Responses Table
CREATE TABLE form_responses (
  response_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT,
  meeting_id INT,
  feedback TEXT,
  response_time DATETIME,
  is_processed TINYINT(1) DEFAULT 0,
  FOREIGN KEY (user_id) REFERENCES users(user_id),
  FOREIGN KEY (meeting_id) REFERENCES meetings(meeting_id)
);

-- Processing Log Table
CREATE TABLE processing_log (
  log_id INT AUTO_INCREMENT PRIMARY KEY,
  meeting_id INT,
  user_id INT,
  status ENUM('processed','error'),
  error_message VARCHAR(255),
  processed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```



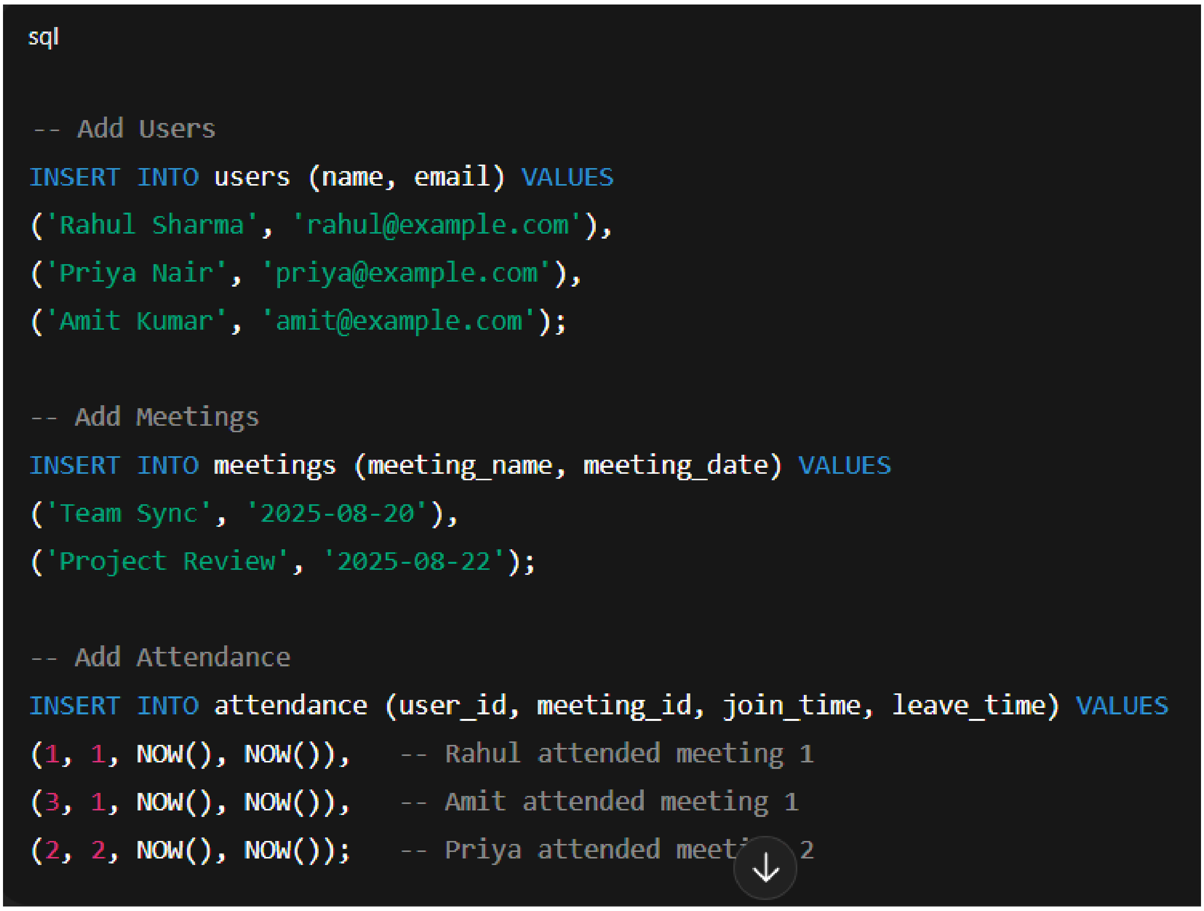
log_id	meeting_id	user_id	status	error_message	processed_at
1	1	1	processed	NULL	2025-09-01 14:35:43
2	1	2	error	No attendance record	2025-09-01 14:35:43
3	2	2	processed	NULL	2025-09-01 14:35:43
4	1	2	error	No attendance record	2025-09-01 14:35:43
5	2	3	error	No attendance record	2025-09-01 14:35:43
6	2	3	error	No attendance record	2025-09-01 14:35:43
7	1	2	error	No attendance record	2025-09-01 14:35:43
8	2	3	error	No attendance record	2025-09-01 14:35:43
NULL	NULL	NULL	NULL	NULL	NULL

Database Schema

The system consists of 6 main tables:

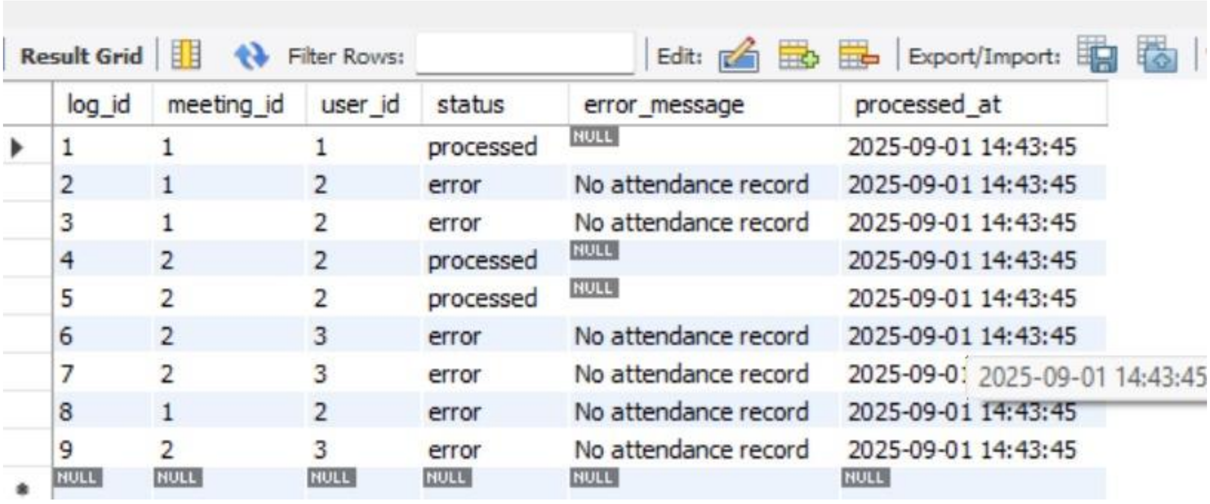
- **users** – stores participant information
- **meetings** – stores meeting details
- **attendance** – records attendance of users
- **form_responses_raw** – stores raw Google Form data
- **form_responses** – stores processed form responses
- **processing_log** – logs validation results

Data Insertion



sql

```sql
-- Add Users
INSERT INTO users (name, email) VALUES
('Rahul Sharma', 'rahul@example.com'),
('Priya Nair', 'priya@example.com'),
('Amit Kumar', 'amit@example.com');

-- Add Meetings
INSERT INTO meetings (meeting_name, meeting_date) VALUES
('Team Sync', '2025-08-20'),
('Project Review', '2025-08-22');

-- Add Attendance
INSERT INTO attendance (user_id, meeting_id, join_time, leave_time) VALUES
(1, 1, NOW(), NOW()),   -- Rahul attended meeting 1
(3, 1, NOW(), NOW()),   -- Amit attended meeting 1
(2, 2, NOW(), NOW());   -- Priya attended meeting 2
```



log_id	meeting_id	user_id	status	error_message	processed_at
1	1	1	processed	NULL	2025-09-01 14:43:45
2	1	2	error	No attendance record	2025-09-01 14:43:45
3	1	2	error	No attendance record	2025-09-01 14:43:45
4	2	2	processed	NULL	2025-09-01 14:43:45
5	2	2	processed	NULL	2025-09-01 14:43:45
6	2	3	error	No attendance record	2025-09-01 14:43:45
7	2	3	error	No attendance record	2025-09-01 14:43:45
8	1	2	error	No attendance record	2025-09-01 14:43:45
9	2	3	error	No attendance record	2025-09-01 14:43:45
NULL	NULL	NULL	NULL	NULL	NULL

Importing Google Form Data

Import Google Form Data

The **feedback** is collected through **Google Forms**, which automatically stores responses into a **Google Sheet**.
To use this data inside MySQL, the sheet is exported as a **CSV file** and then imported into MySQL using the **Table Data Import Wizard** in MySQL Workbench.

Step 1: Export from Google Sheets

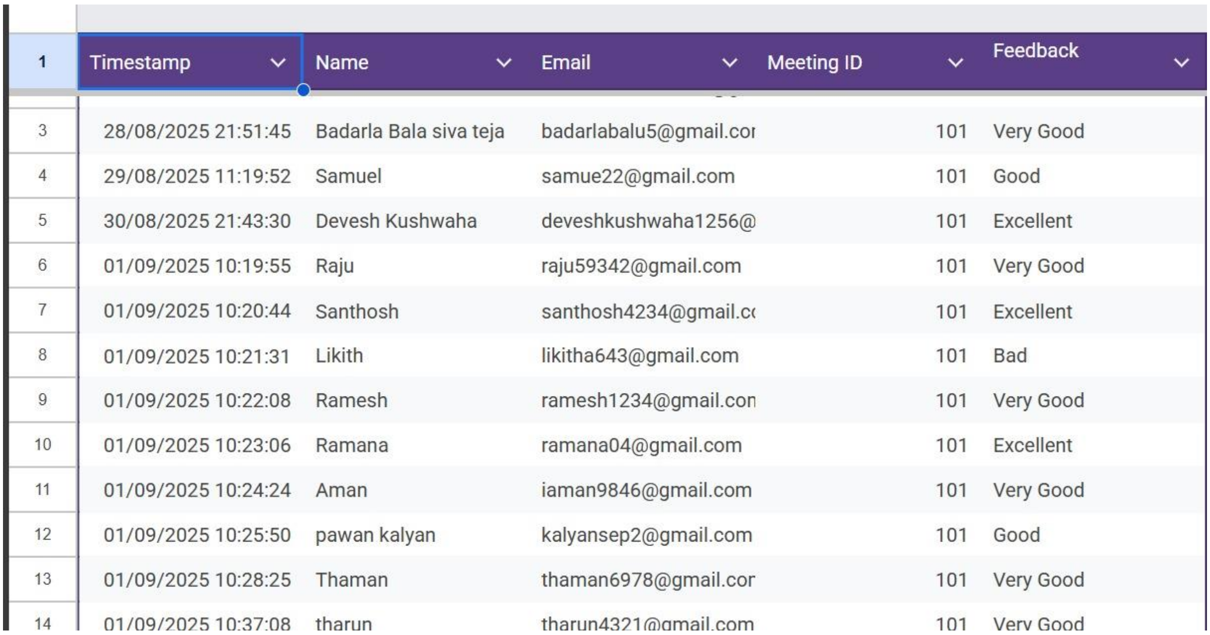
- Open the Google Form responses sheet.
- Click on **File → Download → CSV**.

Step 2: Import into MySQL

- In MySQL Workbench → Right-click database `suretrust_project`.
- Select **Table Data Import Wizard**.
- Choose your CSV file.
- Import it into a new table called **`form_responses_raw`**.

Step 3: Verify Data Import

```
-- Check first 10 records from raw responses
SELECT * FROM form_responses_raw LIMIT 10;
```



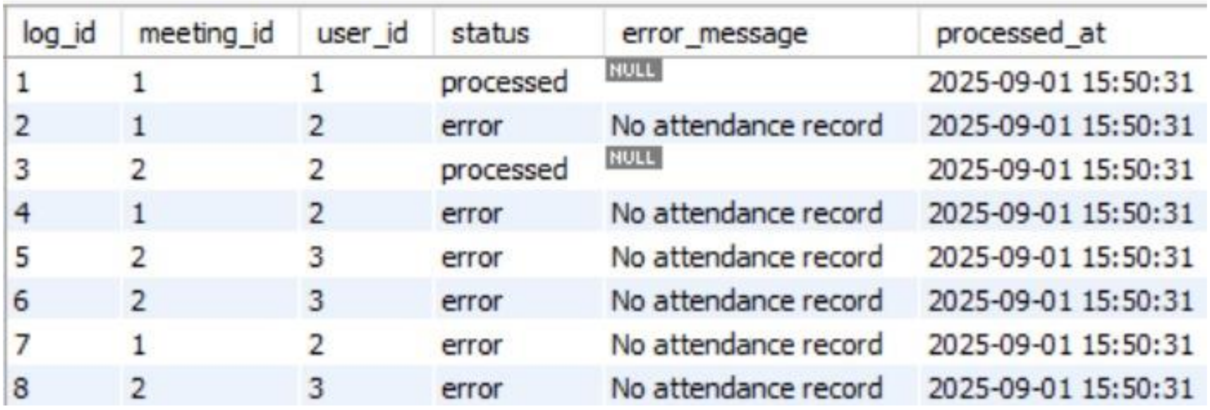
1	Timestamp	Name	Email	Meeting ID	Feedback
3	28/08/2025 21:51:45	Badarla Bala siva teja	badarlabalu5@gmail.cor	101	Very Good
4	29/08/2025 11:19:52	Samuel	samue22@gmail.com	101	Good
5	30/08/2025 21:43:30	Devesh Kushwaha	deveshkushwaha1256@	101	Excellent
6	01/09/2025 10:19:55	Raju	raju59342@gmail.com	101	Very Good
7	01/09/2025 10:20:44	Santhosh	santhosh4234@gmail.c	101	Excellent
8	01/09/2025 10:21:31	Likith	likitha643@gmail.com	101	Bad
9	01/09/2025 10:22:08	Ramesh	ramesh1234@gmail.con	101	Very Good
10	01/09/2025 10:23:06	Ramana	ramana04@gmail.com	101	Excellent
11	01/09/2025 10:24:24	Aman	iaman9846@gmail.com	101	Very Good
12	01/09/2025 10:25:50	pawan kalyan	kalyansep2@gmail.com	101	Good
13	01/09/2025 10:28:25	Thaman	thaman6978@gmail.cor	101	Very Good
14	01/09/2025 10:37:08	tharun	tharun4321@gmail.com	101	Very Good

Moving Data to Main Table

Step 4: Insert into Processed Table

We move data from `form_responses_raw` to `form_responses`.
The **user_id** is matched using the **email ID**.

```
INSERT INTO form_responses (user_id, meeting_id, feedback, response_time)

SELECT u.user_id, 1, r.Feedback, r.Timestamp

FROM form_responses_raw r

JOIN users u ON u.email = r.Email;
```



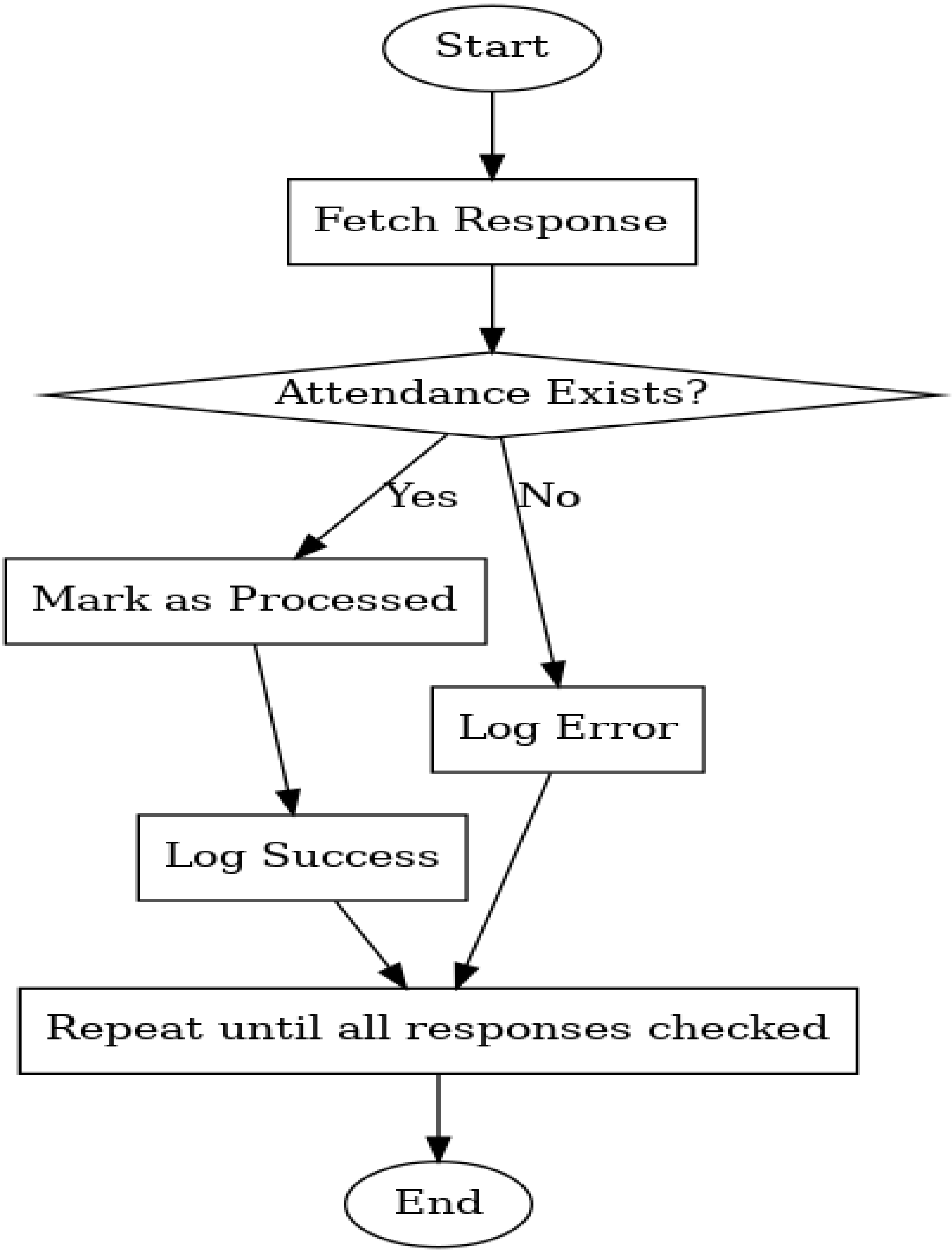
log_id	meeting_id	user_id	status	error_message	processed_at
1	1	1	processed	NULL	2025-09-01 15:50:31
2	1	2	error	No attendance record	2025-09-01 15:50:31
3	2	2	processed	NULL	2025-09-01 15:50:31
4	1	2	error	No attendance record	2025-09-01 15:50:31
5	2	3	error	No attendance record	2025-09-01 15:50:31
6	2	3	error	No attendance record	2025-09-01 15:50:31
7	1	2	error	No attendance record	2025-09-01 15:50:31
8	2	3	error	No attendance record	2025-09-01 15:50:31

Stored Procedure for Processing Responses

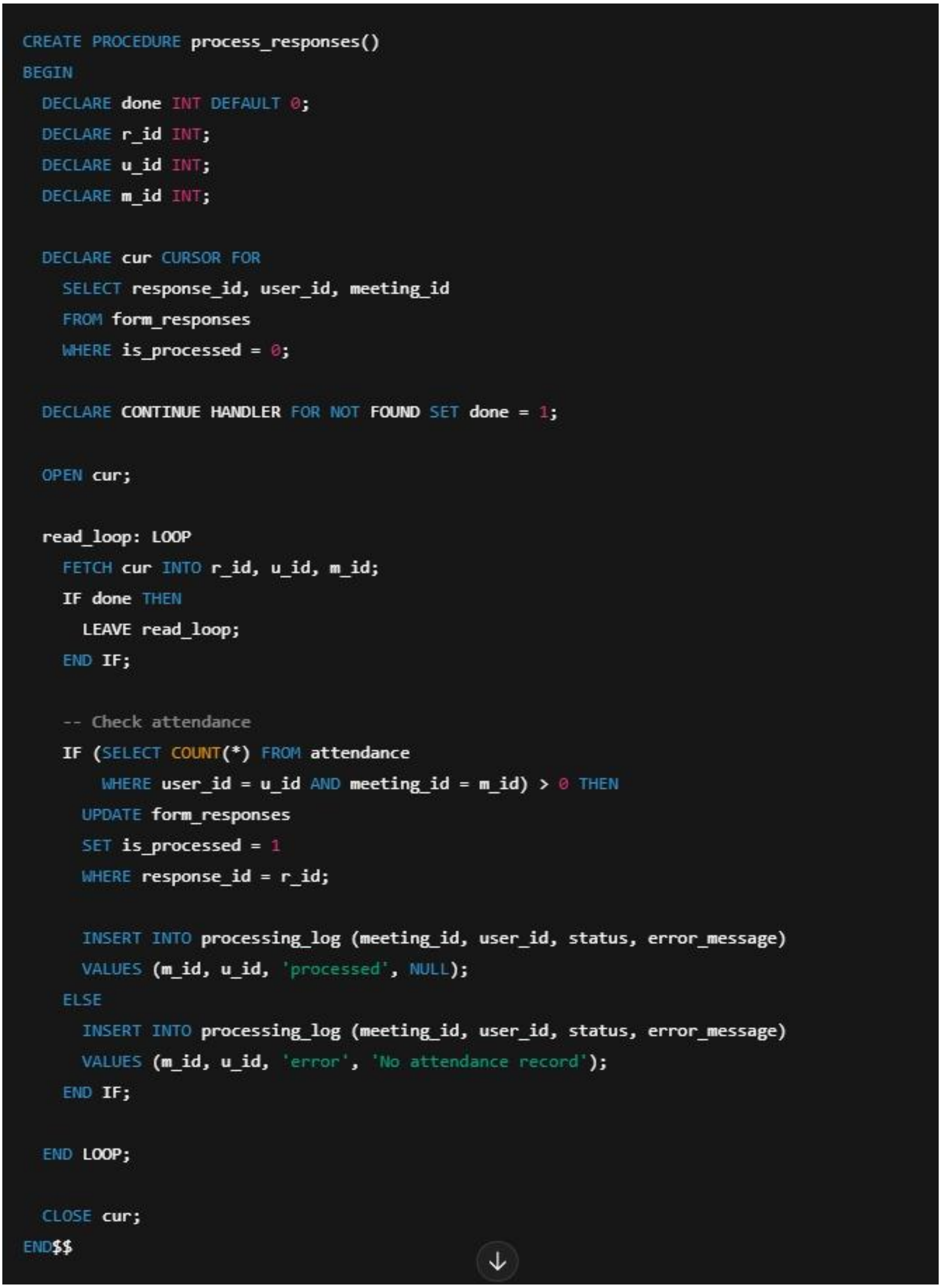
Stored Procedure Logic:

- **Fetch unprocessed responses** from `form_responses`.
- **Check attendance** of the user for the corresponding meeting.
- If valid → mark response as **processed** and log it as **success**.
- If invalid → log it as **error** with message *“No attendance record”*.

Flowchart of Logic

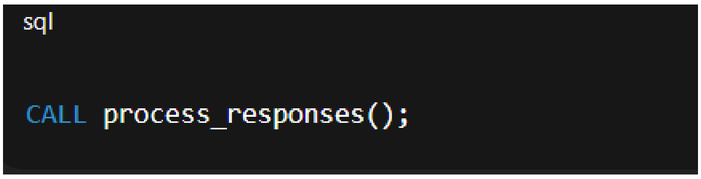


SCRIPT:



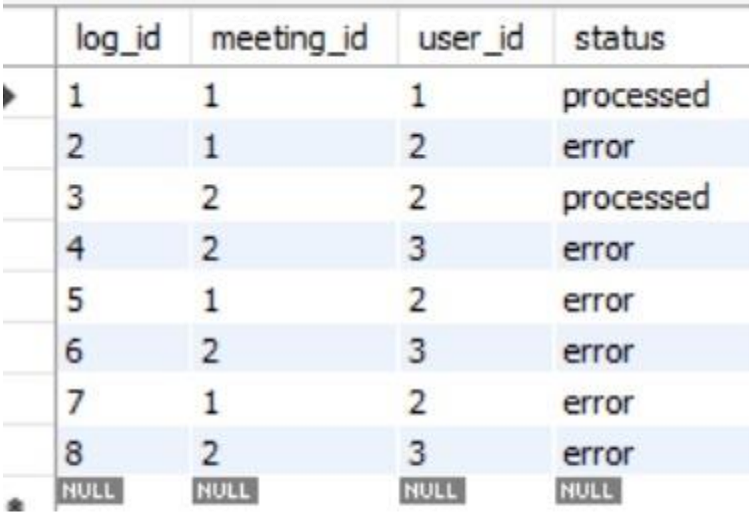
```sql
CREATE PROCEDURE process_responses()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE r_id INT;
  DECLARE u_id INT;
  DECLARE m_id INT;

  DECLARE cur CURSOR FOR
    SELECT response_id, user_id, meeting_id
    FROM form_responses
    WHERE is_processed = 0;

  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur;

  read_loop: LOOP
    FETCH cur INTO r_id, u_id, m_id;
    IF done THEN
      LEAVE read_loop;
    END IF;

    -- Check attendance
    IF (SELECT COUNT(*) FROM attendance
        WHERE user_id = u_id AND meeting_id = m_id) > 0 THEN
      UPDATE form_responses
      SET is_processed = 1
      WHERE response_id = r_id;

      INSERT INTO processing_log (meeting_id, user_id, status, error_message)
      VALUES (m_id, u_id, 'processed', NULL);
    ELSE
      INSERT INTO processing_log (meeting_id, user_id, status, error_message)
      VALUES (m_id, u_id, 'error', 'No attendance record');
    END IF;

  END LOOP;

  CLOSE cur;
END$$
```

Execution:



sql

```sql
CALL process_responses();
```



log_id	meeting_id	user_id	status
1	1	1	processed
2	1	2	error
3	2	2	processed
4	2	3	error
5	1	2	error
6	2	3	error
7	1	2	error
8	2	3	error
NULL	NULL	NULL	NULL

Results and Reports

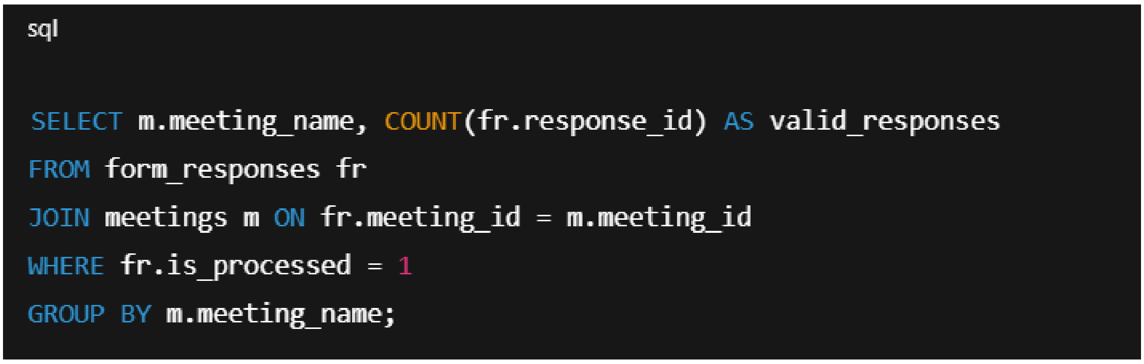
Reports Generated:

After processing the feedback responses, we analyze the results to identify:

1. Total valid responses per meeting.
2. Total errors (invalid responses) per meeting.
3. User-level participation analysis.

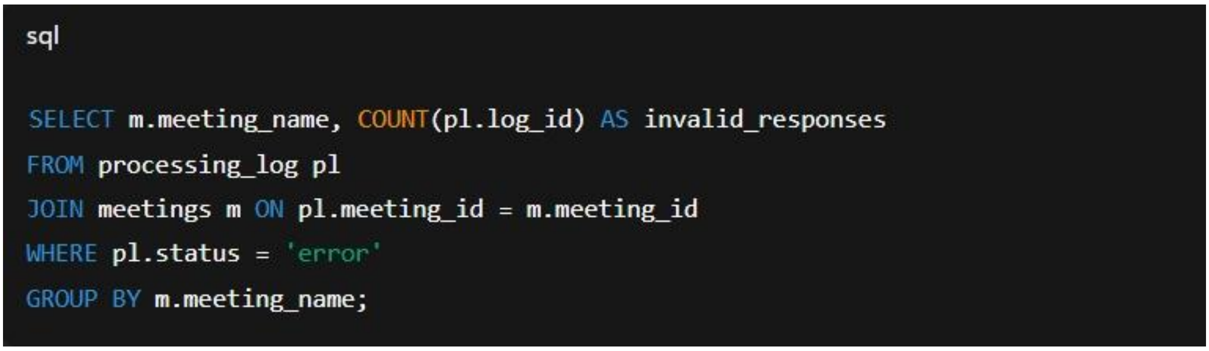
SQL Queries

1. Valid Responses per Meeting



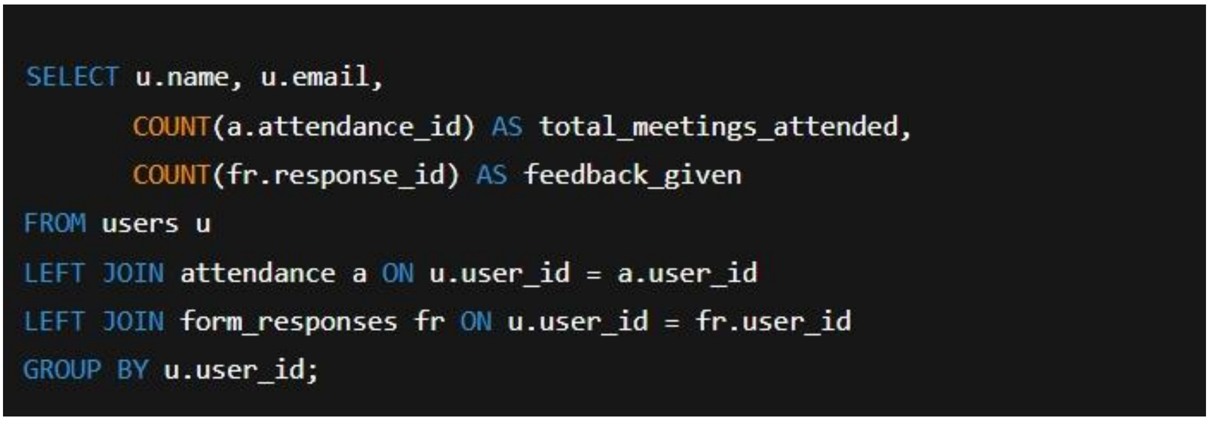
```sql
SELECT m.meeting_name, COUNT(fr.response_id) AS valid_responses
FROM form_responses fr
JOIN meetings m ON fr.meeting_id = m.meeting_id
WHERE fr.is_processed = 1
GROUP BY m.meeting_name;
```

2. Invalid Responses per Meeting

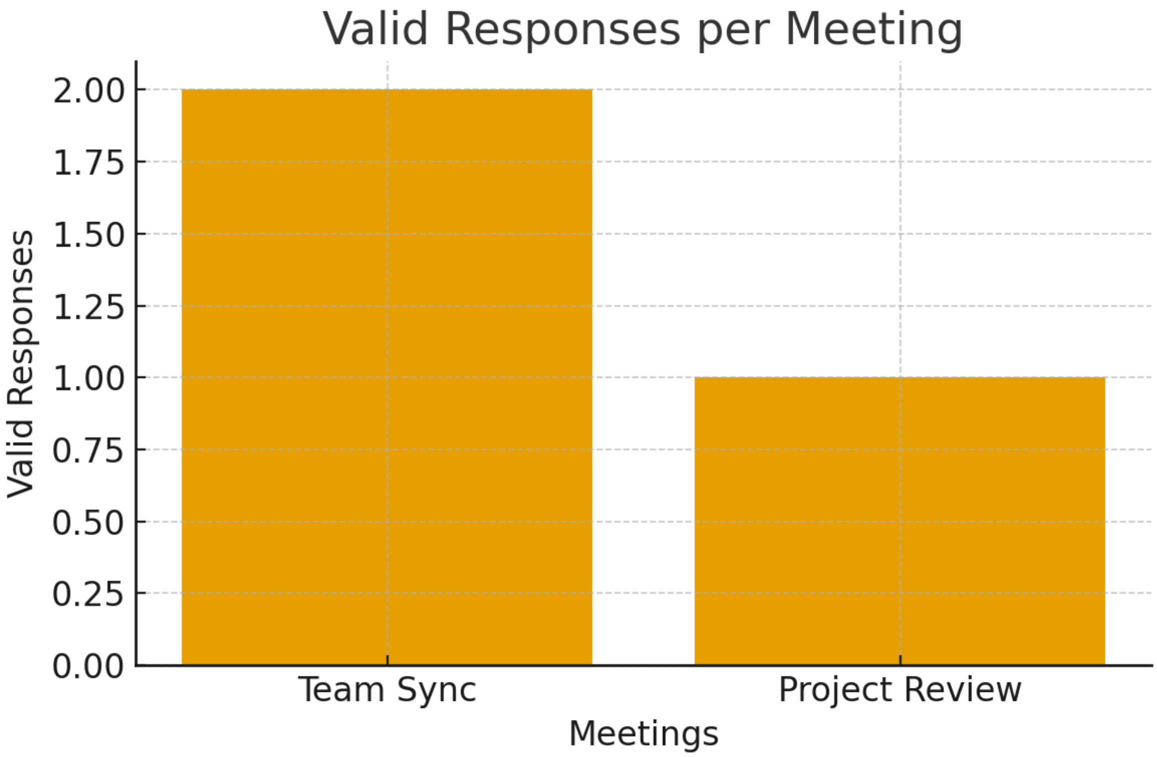


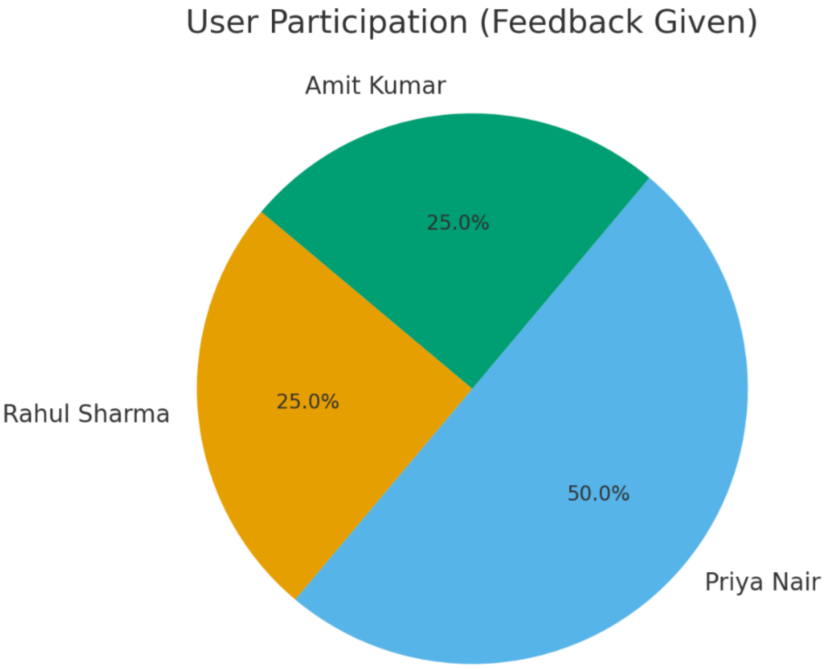
```sql
SELECT m.meeting_name, COUNT(pl.log_id) AS invalid_responses
FROM processing_log pl
JOIN meetings m ON pl.meeting_id = m.meeting_id
WHERE pl.status = 'error'
GROUP BY m.meeting_name;
```

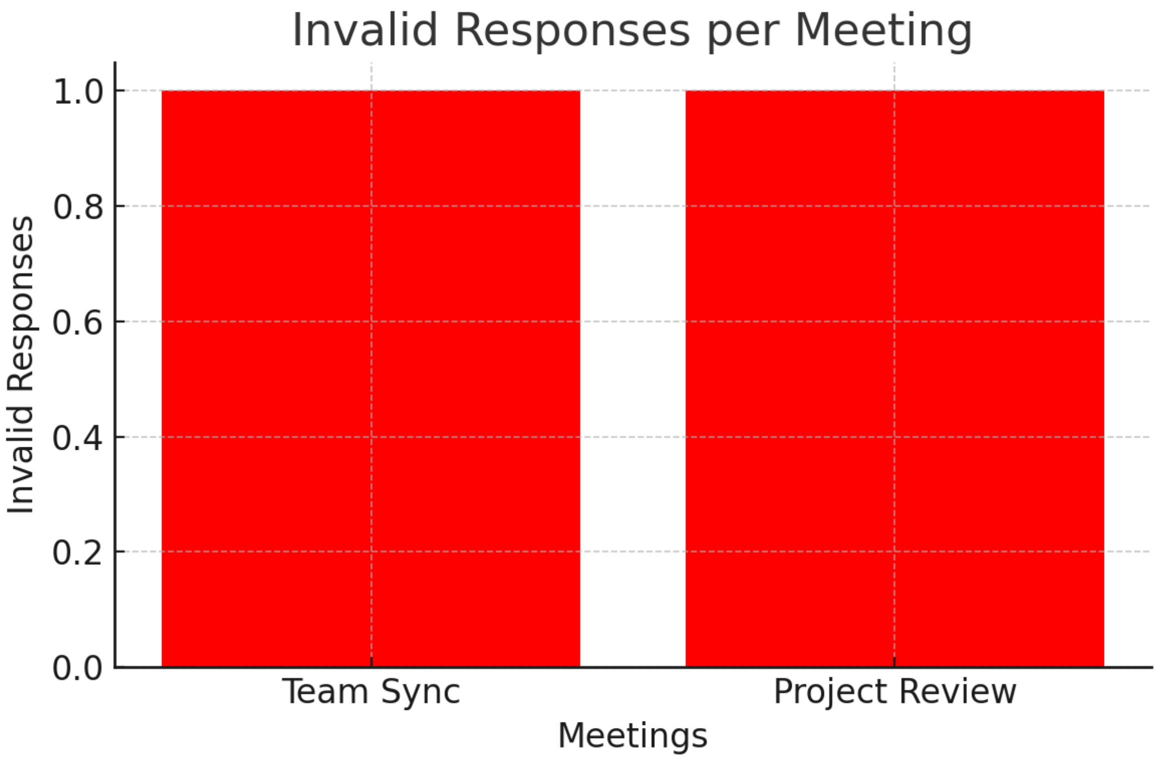
3.User Participation Report



```
SELECT u.name, u.email,
       COUNT(a.attendance_id) AS total_meetings_attended,
       COUNT(fr.response_id) AS feedback_given
FROM users u
LEFT JOIN attendance a ON u.user_id = a.user_id
LEFT JOIN form_responses fr ON u.user_id = fr.user_id
GROUP BY u.user_id;
```




User Participation (Feedback Given)
Amit Kumar
25.0%
Rahul Sharma
25.0%
50.0%
Priya Nair


Invalid Responses per Meeting
Invalid Responses
1.0
0.8
0.6
0.4
0.2
0.0
Team Sync
Project Review
Meetings

Conclusion and Future Scope

Conclusion

This project successfully demonstrates how feedback validation can be automated using MySQL database design and stored procedures. By integrating Google Form data with attendance records, the system ensures that only genuine participants' feedback is considered valid.

Key achievements:

- Designed a relational database with Users, Meetings, Attendance, and Feedback tables.
- Imported real-time Google Form data into MySQL.
- Implemented a stored procedure to validate responses automatically.
- Generated logs for both valid and invalid feedback entries.
- Produced analytical reports on participation and meeting engagement.

The solution shows that **SQL-based systems** can effectively replace manual validation, reduce errors, and improve transparency in feedback collection.

Future Scope

The project can be further extended with:

1. **Automation via Python** – Automatically fetch Google Sheets data into MySQL using scripts.
2. **Dashboard Integration** – Create a real-time dashboard (using Power BI, Tableau, or Flask Web App).

3. **Email Notifications** – Send automatic alerts to participants with invalid feedback.
4. **Scalability** – Extend system to handle multiple organizations and larger datasets.

Limitations

- Current implementation requires manual import of Google Form data into MySQL.
- The validation process works only if emails in Google Form exactly match the database records.
- Reporting is SQL-based; visualization requires export to external tools.