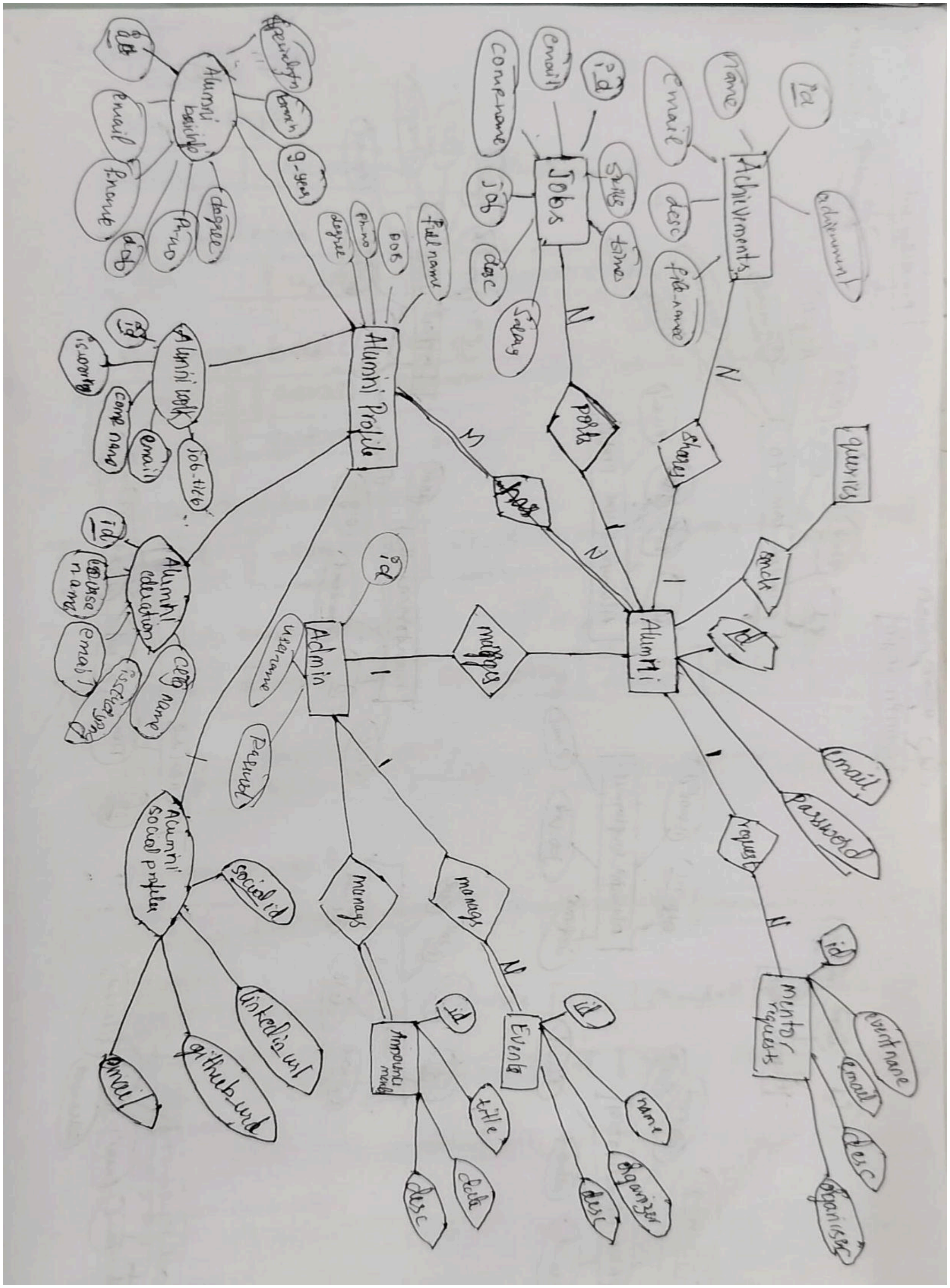
PROJECT NAME

PESU ALUMNIVERSE

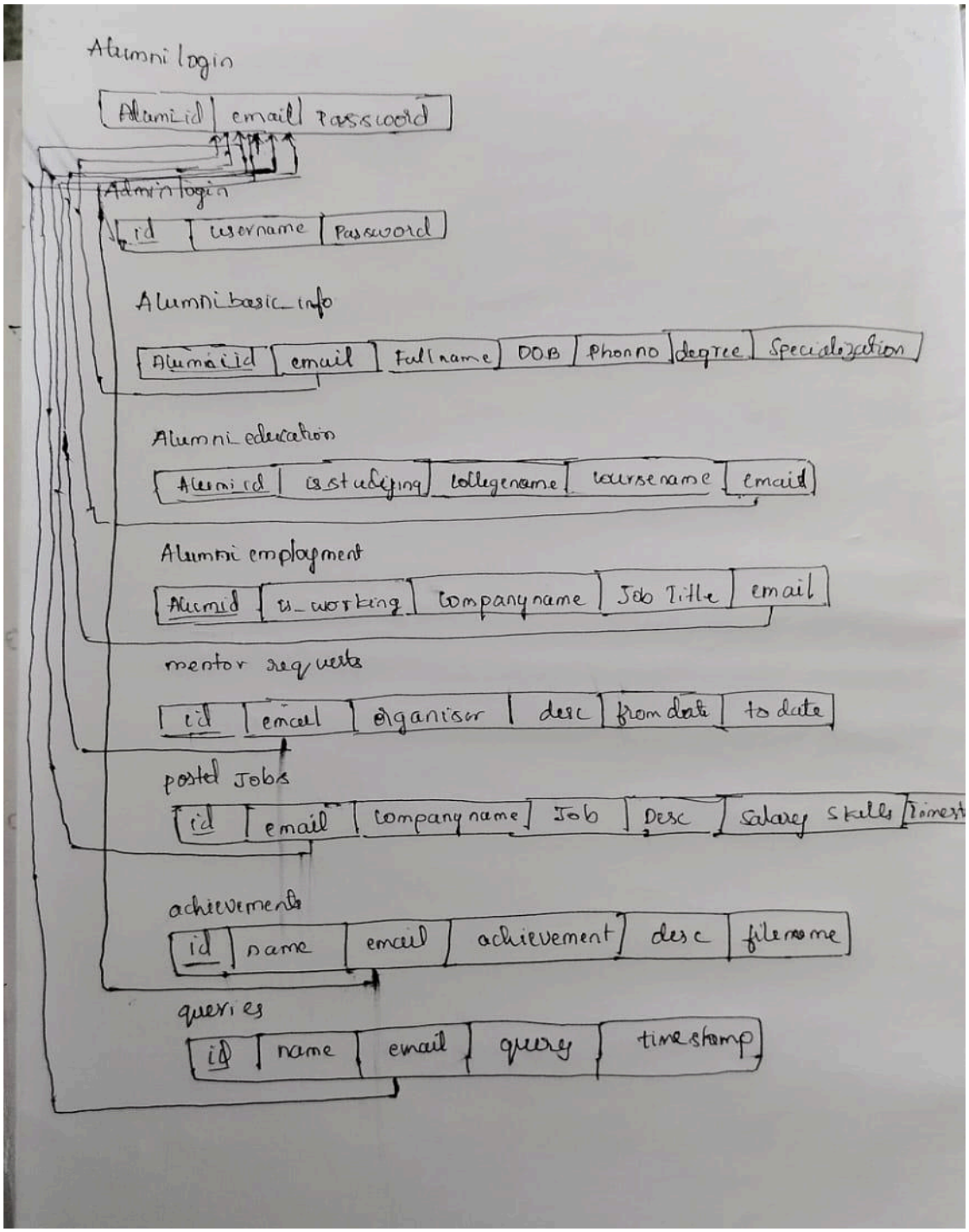
TEAM

MOHAN E	PES1UG22CS357
MOHAMMED KAIF	PES1UG22CS355

ER DIAGRAM:


Alumni Profile
Alumni
Admin
Jobs
Achivements
Alumni social profile
Mentor requests
Events

RELATIONAL SCHEMA:



Tables used:

```
[MariaDB [alumni_portal]> show tables;
 ERROR 2006 (HY000): MySQL server has gone away
 No connection. Trying to reconnect...
 Connection id:    202
 Current database: alumni_portal

 +-------------------------+
 | Tables_in_alumni_portal |
 +-------------------------+
 | achievements            |
 | adminlogin              |
 | alumni_basic_info       |
 | alumni_education        |
 | alumni_employment       |
 | alumni_social_profiles  |
 | alumnilogin             |
 | announcements           |
 | events                  |
 | mentorrequests          |
 | postedjobs              |
 | queries                 |
 +-------------------------+
 12 rows in set (0.055 sec)

 MariaDB [alumni_portal]>
```

Procedure used:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `DisplayAlumniProfile` (IN `alumniEmail`
VARCHAR(50))    BEGIN
   DECLARE alumni_id INT;


   SELECT alumni_id, full_name, dob, phone_num, degree, specialization, branch,
graduation_year
```

```
   INTO alumni_id, @full_name, @dob, @phone_num, @degree, @specialization, @branch,
@graduation_year
   FROM alumni_basic_info
   WHERE email = alumniEmail;


   SELECT COALESCE(company_name, 'N/A') AS company_name,
          COALESCE(job_title, 'N/A') AS job_title,
          COALESCE(is_working, 0) AS is_working
   INTO @company_name, @job_title, @is_working
   FROM alumni_employment
   WHERE alumni_employment.email = alumniEmail;


   SELECT COALESCE(linkedin_url, 'N/A') AS linkedin_url,
          COALESCE(github_url, 'N/A') AS github_url
   INTO @linkedin_url, @github_url
   FROM alumni_social_profiles
   WHERE alumni_social_profiles.email = alumniEmail;


   SELECT COALESCE(college_name, 'N/A') AS college_name,
          COALESCE(course_name, 'N/A') AS course_name,
          COALESCE(is_studying, 0) AS is_studying
   INTO @college_name, @course_name, @is_studying
   FROM alumni_education
   WHERE alumni_education.email = alumniEmail;


   SELECT 'Alumni Profile' AS Section,
          @full_name AS Full_Name,
          @dob AS Date_of_Birth,
          @phone_num AS Phone_Number,
          @degree AS Degree,
          @specialization AS Specialization,
          @branch AS Branch,
          @graduation_year AS Graduation_Year,
          @company_name AS Company,
          @job_title AS Job_Title,
          @is_working AS Is_Working,
          @linkedin_url AS LinkedIn,
          @github_url AS GitHub,
```

```
        @college_name AS College_Name,
        @course_name AS Course_Name,
        @is_studying AS Is_Studying;

END$$
```

The procedure `DisplayAlumniProfile` retrieves and displays an alumni's profile information based on their email. It gathers personal, employment, social, and educational details from various tables, filling in missing values with defaults like "N/A" or `0`. Finally, it outputs a formatted profile summary with these details.

Function used:

```
--
CREATE DEFINER=`root`@`localhost` FUNCTION `reassign_announcement_ids` () RETURNS
VARCHAR(255) CHARSET utf8mb4 COLLATE utf8mb4_general_ci DETERMINISTIC BEGIN
   DECLARE new_id INT DEFAULT 1;
   DECLARE msg VARCHAR(255);


   DECLARE done INT DEFAULT 0;
   DECLARE announcement_id INT;
   DECLARE announcement_cursor CURSOR FOR
       SELECT id FROM announcements ORDER BY id;


   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

   OPEN announcement_cursor;

   read_loop: LOOP
       FETCH announcement_cursor INTO announcement_id;
       IF done THEN
           LEAVE read_loop;
       END IF;
```

```
        UPDATE announcements SET id = new_id WHERE id = announcement_id;
        SET new_id = new_id + 1;
    END LOOP;

    CLOSE announcement_cursor;

    SET msg = 'Deleted and Announcement IDs reassigned successfully!';
    RETURN msg;
END$$
```

The function `reassign_announcement_ids` reassigns consecutive IDs to records in the `announcements` table, starting from 1. It uses a cursor to iterate through the existing `id`s in order and updates each row with a new sequential ID. After reassigning all IDs, it returns a success message.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `reassign_event_ids` () RETURNS
VARCHAR(255) CHARSET utf8mb4 COLLATE utf8mb4_general_ci DETERMINISTIC BEGIN
    DECLARE msg VARCHAR(255);


    SET @new_id := 0;


    UPDATE events
    SET id = (@new_id := @new_id + 1)
    ORDER BY id;

    SET msg = 'Event IDs reassigned successfully!';
    RETURN msg;
END$$

DELIMITER ;
```

The `reassign_event_ids` function renumbers the IDs in the `events` table sequentially, starting from 1. It updates each row's `id` in order and returns a message saying "Event IDs reassigned successfully!"

Triggers used:

```
--
-- Triggers `events`
--
DELIMITER $$
CREATE TRIGGER `before_insert_event` BEFORE INSERT ON `events` FOR EACH ROW BEGIN
   DECLARE max_id INT;


   SELECT COALESCE(MAX(id), 0) + 1 INTO max_id FROM events;


   SET NEW.id = max_id;
END
$$
DELIMITER ;
```

The `before_insert_event` trigger ensures that, before a new row is inserted into the `events` table, its `id` is set to the next available number. It calculates this by finding the maximum `id` currently in the table, adding 1 to it, and assigning this value to the new row's `id` field. This helps maintain sequential IDs automatically.

Join Operations used:

```
SELECT
   abi.*, ae.*, aemp.*, asp.*
FROM
   alumni_basic_info abi
LEFT JOIN
   alumni_education ae ON abi.alumni_id = ae.alumni_id
```

```
LEFT JOIN
   alumni_employment aemp ON abi.alumni_id = aemp.alumni_id
LEFT JOIN
   alumni_social_profiles asp ON abi.alumni_id = asp.alumni_id
   WHERE abi.branch LIKE ?
or abi.graduation_year = ?
or abi.alumni_id = ?
```

This SQL query retrieves comprehensive alumni data from four related tables (`alumni_basic_info`, `alumni_education`, `alumni_employment`, `alumni_social_profiles`) by joining them on `alumni_id`. It uses a `LEFT JOIN` to ensure all records from `alumni_basic_info` are included, even if there are no matching records in the other tables. The `WHERE`clause filters the results to show only those alumni whose `branch` matches a partial string, or whose `graduation_year` or `alumni_id` matches specific values. This setup enables flexible retrieval based on multiple search criteria in a single query.

Aggregate Queries:

UPDATE alumni_social_profiles
SET linkedin_url = ?, github_url = ?
WHERE alumni_id = ?

Check if alumni_social_profile and update

UPDATE alumni_education
SET is_studying = ?, college_name = ?, course_name = ?
WHERE alumni_id = ?
Update alumni current education details college name and course name
UPDATE alumni_employment
SET is_working = ?, company_name = ?, job_title = ?
WHERE alumni_id = ?

Update Alumni where he is working Company name and Job title

UPDATE alumni_basic_info
SET full_name = ?, dob = ?, phone_num = ?, degree = ?, specialization = ?, branch = ?, graduation_year = ?
WHERE alumni_id = ?

Update Alumni personal basic information Full name DOB,Phone number,Degree,Specialization,branch and Graduation Year