Database Design Document

Gym Management Database Design Document
Rishab Shukla – 002825936

Database Purpose:

Membership Management: All gym members' data, such as personal information, contact details, and membership access, is centrally stored in the database. This enables efficient management of gym entries, subscriptions, and attendance tracking.

Trainer Management: Details about gym staff and trainers, including their roles and session assignments, are stored to ensure proper allocation and scheduling.

Session and Booking System: Tracks gym sessions, trainer assignments, and member bookings. This system enables automated seat management and prevents double bookings.

Attendance Tracking: Logs gym entries while verifying membership access. Attendance data helps track peak usage times and plan resource allocation.

Equipment Management: Monitors the gym's equipment inventory, usage, and maintenance history. Ensures timely repairs and replacements to maintain safety and usability.

Payroll Management: Calculates gym staff salaries biweekly based on logged hours. Automates payroll reporting using stored functions and views.

Live Member Analytics: Tracks real-time gym occupancy and member activity, ensuring compliance with capacity rules and improving service planning.

Business Problems Addressed

Access Control: Ensures only authorized members can access the gym using a role-based access system and real-time attendance tracking.

Trainer Allocation: Assigns trainers to gym sessions and members based on their roles and availability.

Session Booking: Prevents overbooking by limiting seats and ensuring one seat per session per member.

Equipment Maintenance: Tracks last service dates and assigns staff for equipment maintenance, ensuring optimal equipment availability.

Payroll Errors: Automates salary calculation based on actual hours worked, reducing manual errors.

Real-Time Analytics: Provides live insights into member activity, occupancy, and session attendance for effective gym management.

Business Rules:

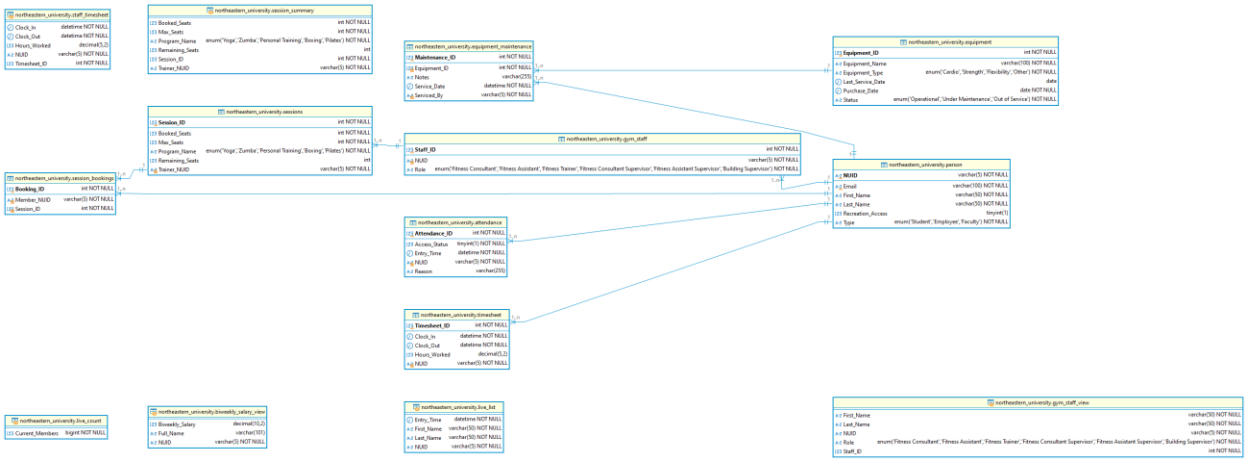
- A member may attend zero or more sessions.
- A trainer may conduct one or more sessions.
- A session must have one trainer and can be attended by zero or more members.
- A gym staff member can log a maximum of 20 hours per week.
- Equipment must be serviced only by authorized staff members.

Entities and Relationships

Entities

Entity Name	Description	Relationships
Person	Stores details of all individuals related to the gym (members, trainers, staff).	Relates to Gym_Staff and Attendance.
Gym_Staff	Tracks roles and assignments of gym staff members.	Linked to Sessions and Timesheet.
Attendance	Logs entries and access statuses of members.	Relates to Person.
Sessions	Manages gym sessions, their trainers, and capacities.	Linked to Bookings and Attendance.
Session_Bookings	Tracks individual bookings for sessions.	Relates to Members and Sessions.
Equipment	Stores inventory details of gym equipment.	Linked to Maintenance.
Equipment_Maintenance	Logs maintenance activities on equipment.	Relates to Equipment and Gym_Staff.
Timesheet	Tracks gym staff clock-in/clock-out times for payroll purposes.	Relates to Gym_Staff.

ER Diagram



Stored Procedures:

1. **CheckAndLogAttendance:** Verifies gym access and logs attendance.

```
128
129     DELIMITER $$
130
131     CREATE PROCEDURE CheckAndLogAttendance(IN person_nuid VARCHAR(5))
132     BEGIN
133         DECLARE has_access BOOLEAN;
134         DECLARE person_exists INT;
135
136         -- Check if the person exists
137         SELECT COUNT(*) INTO person_exists
138         FROM Person
139         WHERE NUID = person_nuid;
140
141         -- If the person does not exist
142         IF person_exists = 0 THEN
143             SIGNAL SQLSTATE '45000'
144             SET MESSAGE_TEXT = 'Person does not exist in the system.';
145         ELSE
146             -- Check Recreation_Access if the person exists
147             SELECT Recreation_Access INTO has_access
148             FROM Person
149             WHERE NUID = person_nuid;
150
151             -- If the person does not have access
152             IF has_access = FALSE THEN
153                 SIGNAL SQLSTATE '45000'
154                 SET MESSAGE_TEXT = 'No gym access.';
155             ELSE
156                 -- If the person has access, log attendance
157                 INSERT INTO Attendance (NUID, Access_Status)
158                 VALUES (person_nuid, TRUE);
159             END IF;
160         END IF;
161     END $$
162
163     DELIMITER ;
```

2. **BookSession**:

   Books a session for a member while checking seat availability.

```
DELIMITER $$

CREATE PROCEDURE BookSession(
    IN member_nuid VARCHAR(5),
    IN session_id INT
)
BEGIN
    DECLARE booking_id INT; -- Variable to store Booking_ID

    -- Start the transaction
    START TRANSACTION;

    -- Check if the member has already booked this session
    SELECT Booking_ID INTO booking_id
    FROM Session_Bookings
    WHERE Session_ID = session_id AND Member_NUID = member_nuid
    LIMIT 1;

    -- Debugging: Output booking_id
    SELECT booking_id AS BookingID;

    IF booking_id IS NULL THEN
        -- Insert the booking record (triggers handle seat availability and updates)
        INSERT INTO Session_Bookings (Session_ID, Member_NUID)
        VALUES (session_id, member_nuid);

        -- Commit the transaction if everything succeeds
        COMMIT;
    ELSE
        -- Rollback and signal error
        ROLLBACK;
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Member has already booked this session.';
    END IF;

END $$

DELIMITER ;
```

Triggers

1. **Generate_NUID**:

   Automatically generates unique IDs for individuals.

```
19    DELIMITER $$
20
21 •  CREATE TRIGGER Generate_NUID
22    BEFORE INSERT ON Person
23    FOR EACH ROW
24    BEGIN
25        DECLARE random_suffix INT;
26        SET random_suffix = FLOOR(1000 + (RAND() * 9000));
27        SET NEW.NUID = LPAD(random_suffix, 5, '0');
28    END $$
29
30    DELIMITER ;
31
```

2. **Check_Seat_Availability**:

   Prevents overbooking of gym sessions.

```
321   DELIMITER $$
322
323 • CREATE TRIGGER Check_Seat_Availability
324   BEFORE INSERT ON Session_Bookings
325   FOR EACH ROW
326   BEGIN
327       DECLARE remaining_seats INT;
328
329       -- Retrieve the remaining seats for the session
330       SELECT Remaining_Seats INTO remaining_seats
331       FROM Sessions
332       WHERE Session_ID = NEW.Session_ID;
333
334       -- If no seats are available, raise an error
335       IF remaining_seats <= 0 THEN
336           SIGNAL SQLSTATE '45000'
337           SET MESSAGE_TEXT = 'No seats available for this session.';
338       END IF;
339
340       -- Increment Booked_Seats in the Sessions table
341       UPDATE Sessions
342       SET Booked_Seats = Booked_Seats + 1
343       WHERE Session_ID = NEW.Session_ID;
344   END $$
345
346   DELIMITER ;
347
```

Views

1. **Live_Count**:
   Displays the current count of members in the gym.
   SELECT * FROM Live_Count;

```
523 ●  CREATE VIEW Live_Count AS
524    SELECT
525        COUNT(*) AS Current_Members -- Count of members currently in the gym
526    FROM
527        Attendance
528    WHERE
529         Entry_Time >= NOW() - INTERVAL 2 HOUR; -- Filter for the last 2 hours
530
531 ●  SELECT * FROM Live_Count;
532
533 ●  CREATE VIEW Live_List AS
534    SELECT
535        a.NUID,
536        p.First_Name,
537        p.Last_Name,
538        a.Entry_Time
539    FROM
540        Attendance a
541    JOIN
542        Person p ON a.NUID = p.NUID
543    WHERE
544        a.Access_Status = TRUE
545        AND a.Entry_Time >= NOW() - INTERVAL 2 HOUR;
546
547 ●  SELECT * FROM Live_List;
```

2. **Biweekly_Salary_View**:
   Calculates staff salaries based on timesheet data.

```
510 •  CREATE VIEW Biweekly_Salary_View AS
511    SELECT
512        g.NUID,
513        CONCAT(p.First_Name, ' ', p.Last_Name) AS Full_Name,
514        Calculate_Biweekly_Salary(g.NUID) AS Biweekly_Salary
515    FROM
516        Gym_Staff g
517    JOIN
518        Person p ON g.NUID = p.NUID;
519
520
521 •  SELECT * FROM Biweekly_Salary_View;
522
523 •  CREATE VIEW Live_Count AS
524    SELECT
525        COUNT(*) AS Current_Members -- Count of members currently in the gym
526    FROM
527        Attendance
528    WHERE
529         Entry_Time >= NOW() - INTERVAL 2 HOUR; -- Filter for the last 2 hours
```

Functions:

1. **Biweekly_Salary:**

   Used to calculate the Biweekly salary of the Gym employees

```
DELIMITER $$

CREATE FUNCTION Calculate_Biweekly_Salary(staff_nuid VARCHAR(5))
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE total_hours DECIMAL(10, 2);
    DECLARE salary DECIMAL(10, 2);
    DECLARE pay_rate DECIMAL(5, 2) DEFAULT 15.00; -- Fixed pay rate of $15/hour

    -- Calculate total hours worked in the last 14 days
    SELECT COALESCE(SUM(TIMESTAMPDIFF(MINUTE, Clock_In, Clock_Out) / 60), 0)
    INTO total_hours
    FROM Timesheet
    WHERE NUID = staff_nuid AND Clock_In >= DATE_SUB(CURDATE(), INTERVAL 14 DAY);

    -- Calculate the salary
    SET salary = total_hours * pay_rate;

    RETURN salary;
END $$

DELIMITER ;
```

Normalization:

1NF: Removed multi-valued attributes and ensured atomic data.

2NF: Eliminated partial dependencies (e.g., moved session details to separate tables).

3NF: Eliminated transitive dependencies by separating maintenance logs and staff details.

Future Enhancements:

- Integration with gym management apps for better member engagement.
- Adding gamification features for workout tracking.
- Implementing predictive analytics for gym trends and peak usage.
- Adding IoT integration for equipment usage monitoring.