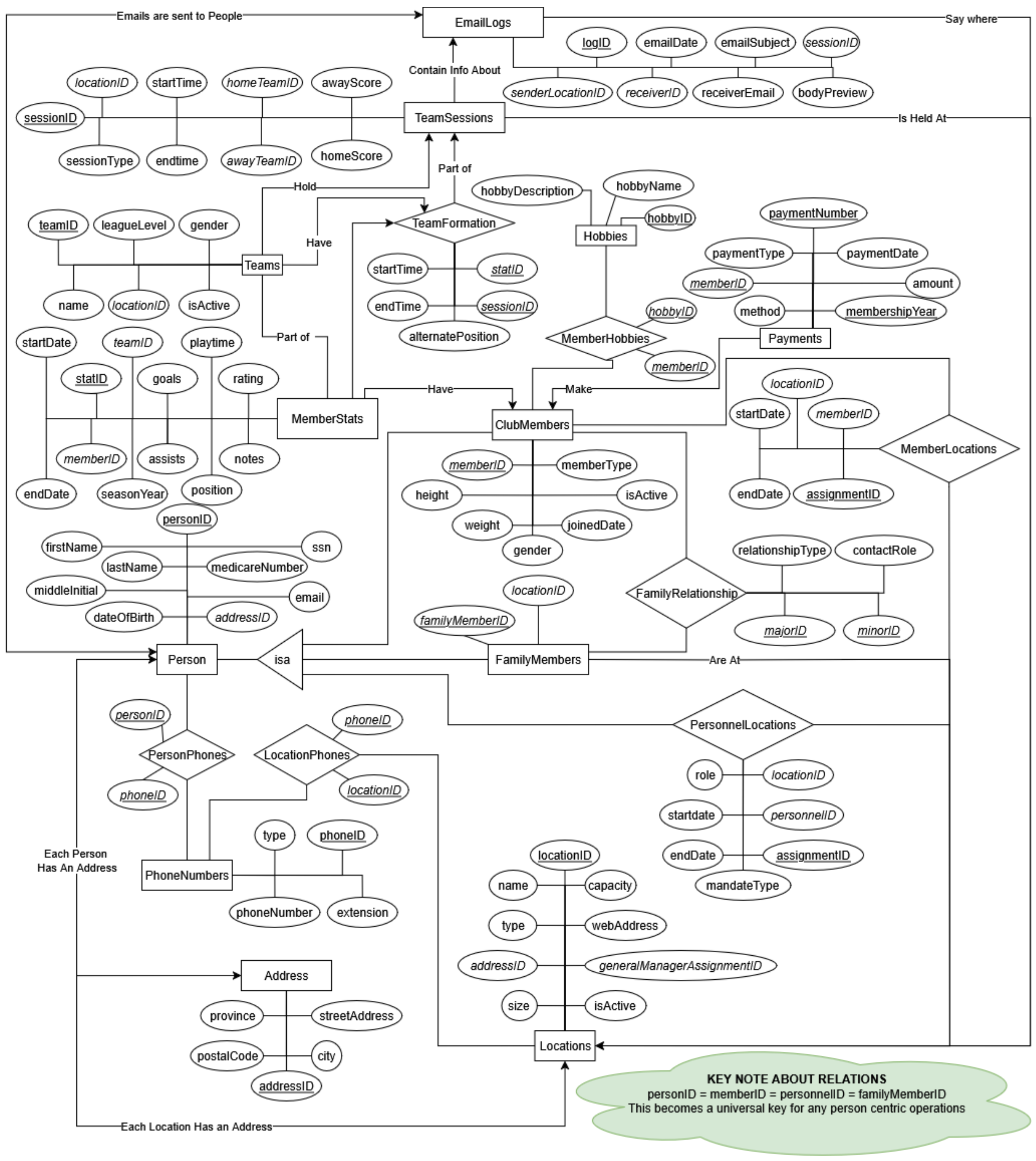
Comp 353 – Files & Databases
Summer 2025
Main Project

Group Account - stc353_1 - Comp_353_CC_G05_STC
Anas Bhar - 40299171
Ricky Germain - 29447644
Mark Gourley - 40326147
Jalal Zakaria - 40265485

August 6, 2025

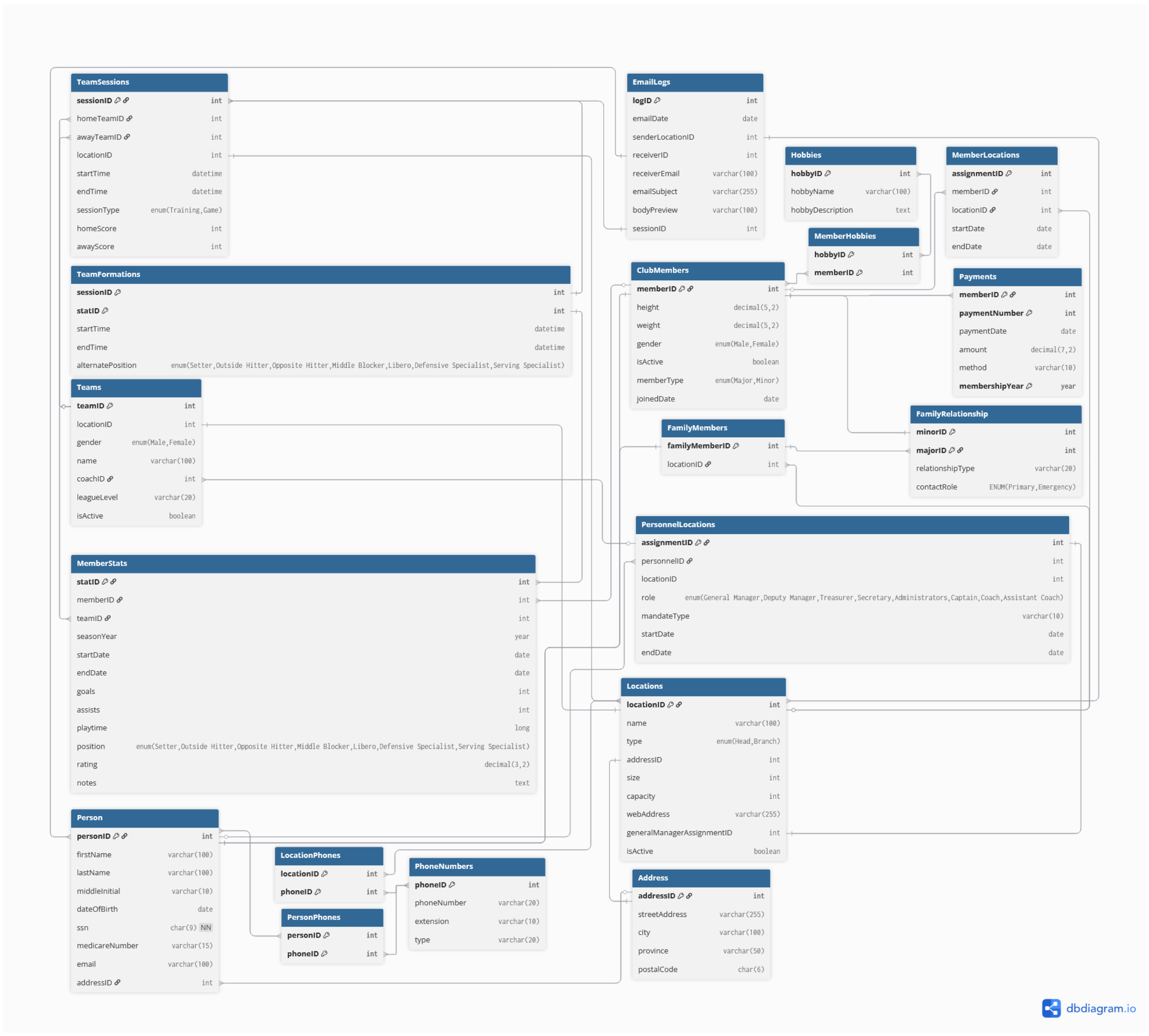
E/R Diagram:



KEY NOTE ABOUT RELATIONS
personID = memberID = personnelID = familyMemberID
This becomes a universal key for any person centric operations

Constraints

Since the warm up project, we've attempted to streamline out tables while adding the newly requested functionality. To that end, the biggest change is making personID a sort of universal ID for accessing the various tables a person may be part of. The direct links are shown here, but this could feasibly streamline queries. Several constraints are better handled by SQL/Database triggers, further in this document. Other key connections are handled better in this database diagram, created on dbdiagram.io



Database Schema

SQL Queries - Table Creation

Core Infrastructure Tables

```
-- Address: Centralized address management for locations and persons
CREATE TABLE Address (
    addressID INT PRIMARY KEY AUTO_INCREMENT,
    streetAddress VARCHAR(255),
    city VARCHAR(100),
    province VARCHAR(50),
    postalCode CHAR(6),
    CHECK (postalCode REGEXP '^[A-Z][0-9][A-Z][0-9][A-Z][0-9]$')
);

-- PhoneNumbers: Phone number registry with extension and type support
CREATE TABLE PhoneNumbers (
    phoneID INT PRIMARY KEY AUTO_INCREMENT,
    phoneNumber VARCHAR(20),
    extension VARCHAR(10),
    type VARCHAR(20)
);

-- Hobbies: Master list of recreational activities for member profiles
CREATE TABLE Hobbies (
    hobbyID int PRIMARY KEY AUTO_INCREMENT,
    hobbyName VARCHAR(100),
    hobbyDescription TEXT
);
```

Person Management

```
-- Person: Universal person registry - foundation for all roles (members,
    personnel, family)
CREATE TABLE Person (
    personID INT PRIMARY KEY AUTO_INCREMENT,
    firstName VARCHAR(100),
    lastName VARCHAR(100),
    middleInitial VARCHAR(10),
    dateOfBirth DATE,
    ssn CHAR(9) NOT NULL UNIQUE,
    medicareNumber VARCHAR(15) UNIQUE,
    email VARCHAR(100),
    addressID INT REFERENCES Address(addressID)
              ON DELETE RESTRICT ON UPDATE CASCADE,
    CHECK (ssn REGEXP '^[0-9]{9}$'),
    CHECK (DATEDIFF(CURDATE(), dateOfBirth) >= 11*365)
);
```

Location And Branch Structure

```
-- Locations: Club facilities (Head office and branch locations)
CREATE TABLE Locations (
    locationID INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100),
    type ENUM('Head', 'Branch'),
    addressID INT REFERENCES Address(addressID),
    size INT,
    capacity INT,
    webAddress VARCHAR(255),
    generalManagerAssignmentID INT,
    isActive BOOLEAN
);
```

Member and Family Management

```
-- FamilyMembers: Adults responsible for minor members
CREATE TABLE FamilyMembers (
    familyMemberID INT PRIMARY KEY,
    locationID INT,
    FOREIGN KEY (familyMemberID) REFERENCES Person(personID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (locationID) REFERENCES Locations(locationID)
        ON DELETE RESTRICT ON UPDATE CASCADE
);

-- ClubMembers: Active volleyball club participants
CREATE TABLE ClubMembers (
    memberID INT PRIMARY KEY REFERENCES Person(personID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    height DECIMAL(5,2),
    weight DECIMAL(5,2),
    gender ENUM('Male', 'Female'),
    isActive BOOLEAN,
    memberType ENUM('Major', 'Minor'),
    joinedDate DATE
);
```

Contact Information Management

```
-- PersonPhones: Links persons to their phone numbers
CREATE TABLE PersonPhones (
    personID INT REFERENCES Person(personID),
    phoneID INT REFERENCES PhoneNumbers(phoneID),
    PRIMARY KEY (personID, phoneID)
);

-- LocationPhones: Links locations to their contact numbers
CREATE TABLE LocationPhones (
    locationID INT REFERENCES Locations(locationID),
    phoneID INT REFERENCES PhoneNumbers(phoneID),
    PRIMARY KEY (locationID, phoneID)
);
```

Contact Information Management

```
-- PersonnelLocations: Staff assignments and roles at club locations
CREATE TABLE PersonnelLocations (
    assignmentID INT PRIMARY KEY AUTO_INCREMENT,
    personnelID INT REFERENCES Person(personID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    locationID INT REFERENCES Locations(locationID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    role ENUM('General␣Manager', 'Deputy␣Manager', 'Treasurer', 'Secretary',
        'Administrator', 'Captain', 'Coach', 'Assistant␣Coach', 'Other'),
    mandateType ENUM('Volunteer', 'Salaried'),
    startDate DATE,
    endDate DATE
);

-- FamilyRelationship: Family connections between members and guardians
CREATE TABLE FamilyRelationship (
    minorID INT REFERENCES ClubMembers(memberID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    majorID INT REFERENCES FamilyMembers(familyMemberID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    relationshipType ENUM('Father', 'Mother', 'Grandfather', 'Grandmother',
        'Tutor', 'Partner', 'Friend', 'Other'),
    contactRole ENUM('Primary','Emergency') NOT NULL DEFAULT 'Primary',
    PRIMARY KEY (minorID, majorID)
);

-- MemberLocations: Member location assignments with transfer history
CREATE TABLE MemberLocations (
    assignmentID INT AUTO_INCREMENT PRIMARY KEY,
    memberID INT REFERENCES ClubMembers(memberID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    locationID INT REFERENCES Locations(locationID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    startDate DATE,
    endDate DATE,
    UNIQUE KEY unique_member_period (memberID, startDate),
    CHECK (endDate IS NULL OR endDate >= startDate)
);

-- MemberHobbies: Member recreational activity tracking
CREATE TABLE MemberHobbies (
    memberID INT REFERENCES ClubMembers(memberID),
    hobbyID int REFERENCES Hobbies(hobbyID),
    PRIMARY KEY (memberID, hobbyID)
);
```

Stat / Team Management

```sql
-- Teams: Volleyball teams organized by location and gender
CREATE TABLE Teams (
    teamID INT PRIMARY KEY AUTO_INCREMENT,
    locationID INT REFERENCES Locations(locationID),
    gender ENUM('Male', 'Female'),
    name VARCHAR(100),
    coachID INT PersonnelLocations(assignmentID),
    leagueLevel VARCHAR(20),
    isActive BOOLEAN
);

-- MemberStats: Performance tracking and position assignments
CREATE TABLE MemberStats (
    statID INT PRIMARY KEY,
    memberID INT REFERENCES ClubMembers (memberID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    teamID INT REFERENCES Teams (teamID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    seasonYear YEAR,
    startDate DATE,
    endDate DATE,
    goals int DEFAULT 0,
    assists int DEFAULT 0,
    playtime bigint DEFAULT 0, -- in seconds
    position enum('Setter','Outside␣Hitter','Opposite␣Hitter','Middle␣
       Blocker','Libero','Defensive␣Specialist','Serving␣Specialist'),
    rating decimal(3,2), -- skill assessment (0.00 to 10.00)
    notes text, -- for admin/coaching staff observations
    -- Constraint: Rating must be between 0.00 and 10.00
    CONSTRAINT chk_rating_range CHECK (rating >= 0.00 AND rating <= 10.00)
);
```

Financial Management

```sql
-- Payments: Member fee tracking with installment support
CREATE TABLE Payments (
    memberID INT NOT NULL REFERENCES ClubMembers(memberID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    membershipYear YEAR NOT NULL,
    paymentNumber INT NOT NULL,
    paymentDate DATE NOT NULL,
    amount DECIMAL(7,2) NOT NULL,
    paymentMethod ENUM('Cash', 'Debit', 'Credit', 'Cheque', 'Other') NOT NULL,
    PRIMARY KEY (memberID, membershipYear, paymentNumber),
    CHECK (paymentNumber >= 1),
    CHECK (amount > 0)
);
```

Session and Formation Management

```
-- TeamSessions: Training sessions and games with scheduling
CREATE TABLE TeamSessions (
    sessionID INT PRIMARY KEY AUTO_INCREMENT,
    homeTeamID INT REFERENCES Teams(teamID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    awayTeamID INT REFERENCES Teams(teamID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    -- Trigger used to make this the homeTeam location if no locationID given on
        insertion
    locationID INT REFERENCES Locations(locationID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    startTime DATETIME,
    endTime DATETIME,
    sessionType ENUM('Training', 'Game'),
    homeScore INT,
    awayScore INT,
    CHECK (endTime IS NULL OR endTime > startTime)
);

-- TeamFormations: Player assignments for specific sessions
CREATE TABLE TeamFormations (
    statID INT REFERENCES MemberStats(statID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    sessionID INT REFERENCES TeamSessions(sessionID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    memberStartDate DATE,
    startTime DATETIME,
    endTime DATETIME,
    alternatePosition ENUM('Setter', 'Outside␣Hitter', 'Opposite␣Hitter', 'Middle␣
        Blocker', 'Libero', 'Defensive␣Specialist', 'Serving␣Specialist'),
    PRIMARY KEY (statID, sessionID)
);
```

Communication Logging

```
-- EmailLogs: System-generated email notification tracking
CREATE TABLE EmailLogs (
    logID INT PRIMARY KEY AUTO_INCREMENT,
    emailDate DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    senderLocationID int REFERENCES Locations(locationID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    receiverID int REFERENCES Person(personID)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    receiverEmail varChar(100),
    emailSubject VARCHAR(255) NOT NULL,
    bodyPreview VARCHAR(100), -- First 100 characters of body
    sessionID int REFERENCES TeamSessions(sessionID)
        ON DELETE RESTRICT ON UPDATE CASCADE
);
```

SQL Queries - Constraints and Indexes

Constraints and Indexes

```sql
-- Add foreign key constraints after all tables are created
ALTER TABLE Locations
ADD CONSTRAINT fk_gm_assignment
FOREIGN KEY (generalManagerAssignmentID)
REFERENCES PersonnelLocations(assignmentID);

-- INDEXES for performance (after all tables are created)
CREATE INDEX idx_member_active ON ClubMembers(isActive);
CREATE INDEX idx_member_type ON ClubMembers(memberType);
CREATE INDEX idx_session_start_time ON TeamSessions(startTime);
CREATE INDEX idx_payment_year ON Payments(membershipYear);
CREATE INDEX idx_assignment_active ON MemberLocations(memberID, endDate);
CREATE INDEX idx_personnel_active ON PersonnelLocations(personnelID, endDate);
CREATE INDEX idx_person_ssn ON Person(ssn);
CREATE INDEX idx_person_medicare ON Person(medicareNumber);
```

Triggers for Business Rules

1) Handle Payments and Active Status

handle_payment_and_active_status

```sql
DELIMITER //
CREATE TRIGGER handle_payment_and_active_status
AFTER INSERT ON Payments
FOR EACH ROW
BEGIN
    DECLARE total_paid DECIMAL(7,2);
    DECLARE required_fee DECIMAL(7,2);
    DECLARE member_type_val ENUM('Major', 'Minor');
    DECLARE new_total DECIMAL(7,2);
    DECLARE warning_msg TEXT;

    -- Get member type
    SELECT memberType INTO member_type_val
    FROM ClubMembers
    WHERE memberID = NEW.memberID;

    -- Determine required fee
    SET required_fee = CASE
        WHEN member_type_val = 'Minor' THEN 100.00
        WHEN member_type_val = 'Major' THEN 200.00
        ELSE 0.00
    END;

    -- Sum existing payments for the year (including this one via NEW.amount)
    SELECT COALESCE(SUM(amount), 0)
    INTO total_paid
    FROM Payments
    WHERE memberID = NEW.memberID
      AND membershipYear = NEW.membershipYear;
    SET new_total = total_paid; -- already includes NEW.amount because AFTER
        INSERT

    -- Determine messaging
    IF new_total > required_fee THEN
        SET warning_msg = CONCAT(
            'Membership␣paid␣in␣full.␣Excess␣payment␣of␣$',
            FORMAT(new_total - required_fee, 2),
            '␣will␣be␣treated␣as␣a␣donation␣to␣the␣club.␣Thank␣you␣for␣your␣
                generosity!'
        );
        SIGNAL SQLSTATE '01000' SET MESSAGE_TEXT = warning_msg;
    ELSEIF new_total = required_fee THEN
        SET warning_msg = 'Payment␣accepted.␣Membership␣is␣now␣paid␣in␣full␣for␣
            this␣year.';
        SIGNAL SQLSTATE '01000' SET MESSAGE_TEXT = warning_msg;
    ELSE
        -- Partial or arrears situation: determine number of prior payments
        DECLARE payment_count INT;
        SELECT COUNT(*) INTO payment_count
        FROM Payments
        WHERE memberID = NEW.memberID
          AND membershipYear = NEW.membershipYear;

        IF payment_count >= 4 THEN
            SET warning_msg = CONCAT(
                'Warning:␣Account␣will␣remain␣in␣arrears.␣Outstanding␣balance:␣$',
                FORMAT(required_fee - new_total, 2),
                '.␣Member␣will␣remain␣inactive␣until␣full␣payment␣received.'
            );
        ELSE
            SET warning_msg = CONCAT(
                'Partial␣payment␣accepted.␣Outstanding␣balance:␣$',
                FORMAT(required_fee - new_total, 2),
                '.␣Member␣will␣remain␣inactive␣until␣full␣payment␣received.'
            );
        END IF;
        SIGNAL SQLSTATE '01000' SET MESSAGE_TEXT = warning_msg;
    END IF;

    -- Update active status (done after messaging so the signal doesn't preempt
        this)
    UPDATE ClubMembers
    SET isActive = (new_total >= required_fee)
    WHERE memberID = NEW.memberID;
END//
DELIMITER ;
```

2) Prevent a member from being on two formations within 3 hours of each other

prevent_conflicting_team_formations

```
DELIMITER //
CREATE TRIGGER prevent_conflicting_team_formations
BEFORE INSERT ON TeamFormations
FOR EACH ROW
BEGIN
    DECLARE new_start DATETIME;
    DECLARE new_end DATETIME;

    -- Get the start/end of the session we're inserting into
    SELECT startTime, endTime
    INTO new_start, new_end
    FROM TeamSessions
    WHERE sessionID = NEW.sessionID;

    IF EXISTS (
        SELECT 1
        FROM TeamFormations tf
        JOIN TeamSessions ts ON tf.sessionID = ts.sessionID
        WHERE tf.statID = NEW.statID
          AND tf.sessionID != NEW.sessionID
          AND (
              -- Overlap: both have defined intervals and they intersect
              (new_start IS NOT NULL AND new_end IS NOT NULL
               AND ts.startTime IS NOT NULL AND ts.endTime IS NOT NULL
               AND NOT (ts.endTime <= new_start OR new_end <= ts.startTime))
              -- Existing ends less than 3 hours before new starts
              OR (ts.endTime IS NOT NULL AND new_start IS NOT NULL
                  AND ts.endTime <= new_start
                  AND TIMESTAMPDIFF(HOUR, ts.endTime, new_start) < 3)
              -- New ends less than 3 hours before existing starts
              OR (new_end IS NOT NULL AND ts.startTime IS NOT NULL
                  AND new_end <= ts.startTime
                  AND TIMESTAMPDIFF(HOUR, new_end, ts.startTime) < 3)
              -- Existing has NULL endTime (open) and its start is within 3 hours
                  before new start
              OR (ts.endTime IS NULL AND new_start IS NOT NULL
                  AND ts.startTime IS NOT NULL
                  AND ts.startTime <= new_start
                  AND TIMESTAMPDIFF(HOUR, ts.startTime, new_start) < 3)
              -- New has NULL endTime and its start is within 3 hours before
                  existing start
              OR (new_end IS NULL AND ts.startTime IS NOT NULL
                  AND new_start IS NOT NULL
                  AND new_start <= ts.startTime
                  AND TIMESTAMPDIFF(HOUR, new_start, ts.startTime) < 3)
          )
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Conflicting␣formation:␣overlapping␣or␣less␣than␣3␣
            hours␣separation␣for␣this␣player';
    END IF;
END//
DELIMITER ;
```

3) Ensure only one active member location assignment

ensure_single_active_member_assignment

```
DELIMITER //
CREATE TRIGGER ensure_single_active_member_assignment
BEFORE INSERT ON MemberLocations
FOR EACH ROW
BEGIN
    -- Check if there's already an active assignment
    DECLARE active_count INT DEFAULT 0;

    SELECT COUNT(*) INTO active_count
    FROM MemberLocations
    WHERE memberID = NEW.memberID
        AND endDate IS NULL
        AND assignmentID != NEW.assignmentID;

    -- If there's an active assignment, raise an error
    IF active_count > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Member already has an active location assignment. 
           Please end the current assignment first.';
    END IF;
END//
DELIMITER ;
```

4) When a member becomes inactive, close their location assignments and stats

close_member_records_on_inactive

```
DELIMITER //
CREATE TRIGGER close_member_records_on_inactive
AFTER UPDATE ON ClubMembers
FOR EACH ROW
BEGIN
    -- If member changed from active to inactive
    IF OLD.isActive = TRUE AND NEW.isActive = FALSE THEN

        -- Close active location assignments
        UPDATE MemberLocations
        SET endDate = CURDATE()
        WHERE memberID = NEW.memberID
            AND endDate IS NULL;

        -- Close active member stats for all teams
        UPDATE MemberStats
        SET endDate = CURDATE()
        WHERE memberID = NEW.memberID
            AND endDate IS NULL;
    END IF;
END//
DELIMITER ;
```

5) Prevent adding inactive members to team formations

prevent_inactive_member_formation

```
DELIMITER //
CREATE TRIGGER prevent_inactive_member_formation
BEFORE INSERT ON TeamFormations
FOR EACH ROW
BEGIN
    DECLARE member_id INT;
    DECLARE is_active BOOLEAN;
    DECLARE has_active_location INT;
    DECLARE stats_open INT;

    -- Resolve club member from statID
    SELECT memberID INTO member_id
    FROM MemberStats
    WHERE statID = NEW.statID;

    -- Check active flag
    SELECT isActive INTO is_active
    FROM ClubMembers
    WHERE memberID = member_id;
    IF is_active IS NULL OR is_active = FALSE THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Cannot␣add␣inactive␣member␣to␣team␣formation';
    END IF;

    -- Check active location
    SELECT COUNT(*) INTO has_active_location
    FROM MemberLocations
    WHERE memberID = member_id AND endDate IS NULL;
    IF has_active_location = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Member␣has␣no␣active␣location␣assignment';
    END IF;

    -- Ensure the stats record itself is open
    SELECT COUNT(*) INTO stats_open
    FROM MemberStats
    WHERE statID = NEW.statID AND endDate IS NULL;
    IF stats_open = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'MemberStats␣record␣is␣closed;␣cannot␣assign␣to␣
            formation';
    END IF;
END//
DELIMITER ;
```

6) Ensure team formation members belong to the same location as the team

validate_member_team_location

```
DELIMITER //
CREATE TRIGGER validate_member_team_location
BEFORE INSERT ON TeamFormations
FOR EACH ROW
BEGIN
    DECLARE member_id INT;
    DECLARE team_id INT;
    DECLARE member_location_id INT;
    DECLARE team_location_id INT;

    -- Resolve the member and team from the stats record
    SELECT memberID, teamID
    INTO member_id, team_id
    FROM MemberStats
    WHERE statID = NEW.statID
      AND endDate IS NULL
    LIMIT 1;

    -- Get member's current active location
    SELECT mla.locationID
    INTO member_location_id
    FROM MemberLocations mla
    WHERE mla.memberID = member_id
      AND mla.endDate IS NULL
    LIMIT 1;

    -- Get team's location
    SELECT t.locationID
    INTO team_location_id
    FROM Teams t
    WHERE t.teamID = team_id
    LIMIT 1;

    -- Enforce same location
    IF member_location_id IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Member␣has␣no␣active␣location␣assignment';
    ELSEIF team_location_id IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Team␣has␣no␣assigned␣location';
    ELSEIF member_location_id != team_location_id THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Member␣must␣be␣assigned␣to␣the␣same␣location␣as␣the␣
            team';
    END IF;
END//
DELIMITER ;
```

7) Validate team session scheduling conflicts

validate_session_scheduling

```
DELIMITER //
CREATE TRIGGER validate_session_scheduling
BEFORE INSERT ON TeamSessions
FOR EACH ROW
BEGIN
    DECLARE conflict_count INT DEFAULT 0;

    -- Check for scheduling conflicts at the same location
    SELECT COUNT(*) INTO conflict_count
    FROM TeamSessions
    WHERE locationID = NEW.locationID
        AND DATE(startTime) = DATE(NEW.startTime)
        AND (
            (NEW.startTime BETWEEN startTime AND endTime) OR
            (NEW.endTime BETWEEN startTime AND endTime) OR
            (startTime BETWEEN NEW.startTime AND NEW.endTime)
        );

    -- Allow some overlap but warn about potential conflicts
    IF conflict_count > 2 THEN
        SIGNAL SQLSTATE '01000'
        SET MESSAGE_TEXT = 'Warning:␣Multiple␣sessions␣scheduled␣at␣same␣location␣
            and␣time␣-␣check␣for␣conflicts';
    END IF;
END//
DELIMITER ;
```

8) Set default location from home team if no location specified

set_default_session_location

```
DELIMITER //
CREATE TRIGGER set_default_session_location
BEFORE INSERT ON TeamSessions
FOR EACH ROW
BEGIN
    DECLARE team_location_id INT;

    IF NEW.locationID IS NULL THEN
        SELECT locationID INTO team_location_id
        FROM Teams
        WHERE teamID = NEW.homeTeamID;

        SET NEW.locationID = team_location_id;
    END IF;
END//
DELIMITER ;
```

9) Gender validation for teams

enforce_same_gender_team

```
DELIMITER //
CREATE TRIGGER enforce_same_gender_team
BEFORE INSERT ON TeamFormations
FOR EACH ROW
BEGIN
    DECLARE team_gender ENUM('Male', 'Female');
    DECLARE member_gender ENUM('Male', 'Female');

    -- Get team's gender
    SELECT gender INTO team_gender
    FROM Teams t
    JOIN MemberStats ms ON t.teamID = ms.teamID
    WHERE ms.statID = NEW.statID;

    -- Get member's gender
    SELECT cm.gender INTO member_gender
    FROM MemberStats ms
    JOIN ClubMembers cm ON ms.memberID = cm.memberID
    WHERE ms.statID = NEW.statID;

    IF team_gender != member_gender THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Player␣gender␣must␣match␣team␣gender';
    END IF;
END//
DELIMITER ;
```

10) Ensure minor members have family association

validate_minor_family_association

```
DELIMITER //
CREATE TRIGGER validate_minor_family_association
BEFORE UPDATE ON ClubMembers
FOR EACH ROW
BEGIN
    IF NEW.memberType = 'Minor' THEN
        IF NOT EXISTS (
            SELECT 1 FROM FamilyRelationship
            WHERE minorID = NEW.memberID
        ) THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Minor␣members␣must␣be␣associated␣with␣a␣family␣
                member';
        END IF;
    END IF;
END//
DELIMITER ;
```

11) Set default position for team formation if not specified

set_default_formation_position

```
DELIMITER //
CREATE TRIGGER set_default_formation_position
BEFORE INSERT ON TeamFormations
FOR EACH ROW
BEGIN
    DECLARE usual_position VARCHAR(50);

    IF NEW.alternatePosition IS NULL THEN
        -- Get the member's usual position from their most recent stats record
        SELECT ms.position INTO usual_position
        FROM MemberStats ms
        WHERE ms.statID = NEW.statID
          AND ms.endDate IS NULL  -- Current/active stats record
        LIMIT 1;

        -- If no current stats record, get their most recent position
        IF usual_position IS NULL THEN
            SELECT ms.position INTO usual_position
            FROM MemberStats ms
            WHERE ms.statID = NEW.statID
            ORDER BY ms.seasonYear DESC, ms.startDate DESC
            LIMIT 1;
        END IF;

        -- Set the alternate position to their usual position
        IF usual_position IS NOT NULL THEN
            SET NEW.alternatePosition = usual_position;
        END IF;
    END IF;
END//
DELIMITER ;
```

12) Person must be at least 11 to be inserted into the system

person_age_insert

```
DELIMITER //
CREATE TRIGGER person_age_insert
BEFORE INSERT ON Person
FOR EACH ROW
BEGIN
    IF NEW.dateOfBirth IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'dateOfBirth␣cannot␣be␣NULL';
    END IF;
    IF TIMESTAMPDIFF(YEAR, NEW.dateOfBirth, CURDATE()) < 11 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Person␣must␣be␣at␣least␣11␣years␣old';
    END IF;
END//
DELIMITER ;
```

Scheduled Events

Enable even scheduler

```
SET GLOBAL event_scheduler = ON;
```

Handling member aging

Daily check on member ages

```
DELIMITER //
CREATE EVENT IF NOT EXISTS daily_promote_new_majors
ON SCHEDULE EVERY 1 DAY
DO
BEGIN
    UPDATE ClubMembers cm
    JOIN Person p ON cm.memberID = p.personID
    SET cm.memberType = 'Major'
    WHERE cm.memberType = 'Minor'
      AND TIMESTAMPDIFF(YEAR, p.dateOfBirth, CURDATE()) >= 18;
END//
DELIMITER ;
```

Weekly Email Requirement

Stored Procedure to notify members of upcoming sessions

```
DELIMITER //
CREATE PROCEDURE notify_members_for_session(IN inputSessionID INT)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE member_id INT;
    DECLARE member_fname VARCHAR(100);
    DECLARE member_lname VARCHAR(100);
    DECLARE member_email VARCHAR(100);
    DECLARE member_assigned_position VARCHAR(50);
    DECLARE coach_fname VARCHAR(100);
    DECLARE coach_lname VARCHAR(100);
    DECLARE coach_email VARCHAR(100);
    DECLARE team_name_from_formation VARCHAR(100);
    DECLARE email_subject TEXT;
    DECLARE email_body TEXT;
    DECLARE session_address VARCHAR(500);
    DECLARE member_cursor CURSOR FOR
        SELECT DISTINCT
            ms.memberID,
            p.firstName,
            p.lastName,
            p.email,
            tf.alternatePosition,
            t.name
        FROM TeamFormations tf
            JOIN MemberStats ms ON tf.statID = ms.statID
            JOIN ClubMembers cm ON ms.memberID = cm.memberID
            JOIN Person p ON cm.memberID = p.personID
            JOIN Teams t ON ms.teamID = t.teamID
        WHERE tf.sessionID = inputSessionID;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    -- Get session address
    SELECT CONCAT(a.streetAddress, ',␣', a.city, ',␣', a.province, '␣',
       a.postalCode)
    INTO session_address
    FROM Address a
        JOIN Locations l ON a.addressID = l.addressID
        JOIN TeamSessions ts ON ts.locationID = l.locationID
    WHERE ts.sessionID = inputSessionID;

    OPEN member_cursor;
    read_loop: LOOP
        FETCH member_cursor INTO
            member_id,
            member_fname,
            member_lname,
            member_email,
            member_assigned_position,
            team_name_from_formation;

        IF done THEN
            LEAVE read_loop;
        END IF;
        -- Get coach info
        SELECT p.firstName, p.lastName, p.email
        INTO coach_fname, coach_lname, coach_email
        FROM TeamSessions ts
            JOIN Teams t ON (ts.homeTeamID = t.teamID OR ts.awayTeamID = t.teamID)
            JOIN PersonnelLocations pla ON t.coachID = pla.assignmentID
            JOIN Person p ON pla.personnelID = p.personID
        WHERE ts.sessionID = inputSessionID
        LIMIT 1;
        -- Build email subject and body
        SET email_subject = CONCAT(
            team_name_from_formation, '␣',
            DAYNAME((SELECT startTime FROM TeamSessions WHERE sessionID =
               inputSessionID)), '␣',
            DATE_FORMAT((SELECT startTime FROM TeamSessions WHERE sessionID =
               inputSessionID), '%d-%b-%Y'), '␣',
            TIME_FORMAT((SELECT startTime FROM TeamSessions WHERE sessionID =
               inputSessionID), '%H:%i'), '␣',
            (SELECT sessionType FROM TeamSessions WHERE sessionID =
               inputSessionID), '␣session'
        );

        SET email_body = CONCAT(
            'Dear␣', member_fname, '␣', member_lname, ',', CHAR(10),
            'You␣are␣assigned␣as␣', member_assigned_position, '␣for␣the␣upcoming␣
               ',
            (SELECT sessionType FROM TeamSessions WHERE sessionID =
               inputSessionID), '␣session␣on␣',
            DATE_FORMAT((SELECT startTime FROM TeamSessions WHERE sessionID =
               inputSessionID), '%M␣%d,␣%Y'),
            '␣at␣', TIME_FORMAT((SELECT startTime FROM TeamSessions WHERE
               sessionID = inputSessionID), '%h:%i␣%p'), '.', CHAR(10),
            'Your␣team␣for␣this␣session␣is:␣', team_name_from_formation, '.',
               CHAR(10),
            'Head␣Coach:␣', coach_fname, '␣', coach_lname,
            '␣(', coach_email, ').', CHAR(10),
            'Location:␣', session_address, '.', CHAR(10),
            'Please␣arrive␣15␣minutes␣early␣for␣warm-up.␣See␣you␣there!'
        );

        INSERT INTO EmailLogs (senderLocationID, receiverID, receiverEmail,
           emailSubject, bodyPreview, sessionID)
        VALUES (
            (SELECT locationID FROM TeamSessions WHERE sessionID =
               inputSessionID),
            member_id,
            member_email,
            email_subject,
            LEFT(email_body, 100),
            inputSessionID
        );

    END LOOP;

    CLOSE member_cursor;
END//
DELIMITER ;
```

Weekly event to run the stored procedure

```
DELIMITER //
CREATE EVENT IF NOT EXISTS weekly_session_notification
ON SCHEDULE EVERY 1 WEEK
STARTS CURRENT_TIMESTAMP + INTERVAL 1 MINUTE
DO
BEGIN
    DECLARE session_cursor_done INT DEFAULT FALSE;
    DECLARE session_id INT;

    DECLARE cur_sessions CURSOR FOR
        SELECT sessionID
        FROM TeamSessions
        WHERE startTime BETWEEN CURDATE()
            AND DATE_ADD(CURDATE(), INTERVAL 7 DAY);

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET session_cursor_done = TRUE;

    OPEN cur_sessions;

    read_sessions: LOOP
        FETCH cur_sessions INTO session_id;
        IF session_cursor_done THEN
            LEAVE read_sessions;
        END IF;

        CALL notify_members_for_session(session_id);
    END LOOP;

    CLOSE cur_sessions;
END//
DELIMITER ;
```

BCNF verification / reasonings

Attributes and Primary Keys

First, we verify the attributes and Primary Keys:

Table 1: Tables with their Attributes and Primary Keys (part 1)

TableName	Attributes	Primary Key
Address	streetAddress city province postalCode	addressID [surrogate]
Person	firstName lastName middleInitial dateOfBirth ssn medicareNumber email addressID	personID [surrogate]
PhoneNumbers	phoneNumber extension type	phoneID [surrogate]
Hobbies	hobbyName hobbyDescription	hobbyID [surrogate]
ClubMembers	height weight isActive memberType joinedDate	memberID [from personID] [surrogate]
Locations	name type addressID size capacity webAddress generalManagerAssignmentID isActive	locationID [surrogate]
Teams	locationID gender name coachID leagueLevel isActive	teamID [surrogate]

Table 2: Tables with their Attributes and Primary Keys (part 2)

TableName	Attributes	Primary Key
Payments	memberID paymentNumber paymentDate amount method membershipYear	(memberID, membershipYear, paymentNumber) [natural]
PersonnelLocations	personnelID locationID role mandateType startDate endDate	assignmentID [surrogate]
EmailLogs	emailDate senderLocationID receiverID receiverEmail emailSubject bodyPreview sessionID	logID [surrogate]
TeamFormations	statID sessionID startTime endTime alternatePosition	(statID, sessionID) [natural]
MemberStats	memberID teamID seasonYear startDate endDate goals assists playtime position rating notes	statID [surrogate]
MemberLocations	memberID locationID startDate endDate	assignmentID [surrogate]
FamilyMembers	familyMemberID locationID	familyMemberID [from personID] [surrogate]
PersonPhones	personID phoneID	(personID, phoneID) [natural]
LocationPhones	locationID phoneID	(locationID, phoneID) [natural]
FamilyRelationship	minorID majorID relationshipType contatctRole	(minorID, majorID) [natural]

Table 3: Tables with their Attributes and Primary Keys (part 3)

TableName	Attributes	Primary Key
MemberHobbies	memberID hobbyID	(memberID, hobbyID) [natural]
TeamSessions	homeTeamID awayTeamID locationID startTime endTime sessionType homeScore awayScore	sessionID [surrogate]

Candidate Keys (Natural)

Table 4: Natural Candidate Keys

TableName	Candidate Keys (Natural Keys only)
Address	(streetAddress, city, province)
	(streetAddress, postalCode)
Person	(ssn)
	(medicareNumber)
PhoneNumbers	(phoneNumber, extension)
Hobbies	(hobbyName)
ClubMembers	None [see discussion below]
Locations	(name)
Teams	(name, locationID)
Payments	(memberID, membershipYear, paymentNumber)
PersonnelLocations	(personnelID, locationID, startDate)
EmailLogs	(emailDate, senderLocationID, receiverID)
TeamFormations	(statID, sessionID)
MemberStats	(memberID, teamID, seasonYear, startDate)
MemberLocations	(memberID, locationID, startDate)
FamilyMembers	(familyMemberID)
PersonPhones	(personID, phoneID)
LocationPhones	(locationID, phoneID)
FamilyRelationship	(minorID, majorID)
MemberHobbies	(memberID, hobbyID)
TeamSessions	(homeTeamID, awayTeamID, startTime)

Functional Dependencies

Table 5: Functional Dependencies (including those not informing candidate keys)

TableName	Functional Dependencies
Address	(streetAddress, city, province) → (postalCode) (streetAddress, postalCode) → (city, province)
	(postalCode) → (city, province) [violates BCNF]
Person	(ssn) → (firstName, lastName, middleInitial, dateOfBirth, medicareNumber, email, addressID)
	(medicareNumber) → (firstName, lastName, middleInitial, dateOfBirth, ssn, email, addressID)
PhoneNumbers	(phoneNumber, extension) → type
Hobbies	(hobbyName) → (hobbyDescription)
ClubMembers	None [see discussion below]
Locations	(name) → (type, addressID, size, capacity, webAddress, generalManagerAssignmentID, isActive)
Teams	(name, locationID) → (gender, coachID, leagueLevel, isActive)
Payments	(memberID, membershipYear, paymentNumber) → (paymentDate, amount, method)
PersonnelLocations	(personnelID, locationID, startDate) → (role, mandateType, endDate)
EmailLogs	(emailDate, senderLocationID, receiverID) → (receiverEmail, emailSubject, bodyPreview, sessionID)
TeamFormations	(statID, sessionID) → (startTime, endTime, alternatePosition)
MemberStats	(memberID, teamID, seasonYear, startDate) → (endDate, goals, assists, playtime, position, rating, notes)
MemberLocations	(memberID, locationID, startDate) → (endDate)
FamilyMembers	(familyMemberID) → (locationID)
PersonPhones	only trivial ones
LocationPhones	only trivial ones
FamilyRelationship	(minorID, majorID) → (relationshipType, contactRole)
MemberHobbies	only trivial ones
TeamSessions	(homeTeamID, awayTeamID, startTime) → (locationID, endTime, sessionType, homeScore, awayScore)

Comments on FDs

- Address
  1) This table is in 3NF but not in BCNF due to the functional dependency: (postalCode) $\rightarrow$ (city, province), as the LHS is not a candidate key, while the RHS is part of a candidate key.
- Person
  1) (SSN) contains private information, and it likely goes against Canada's Privacy Act to use this as a unique identifier.
  2) Not all minors have a (SSN); not all majors necessarily have a SSN.
  3) (medicareNumber) contains private information, and it likely goes against Canada's Privacy Act to use this as a unique identifier. Also, not all members have a Quebec (medicareNumber) if they are from another province and are temporarily residing in Montreal as out of province Canadians or even foreigners.
  4) A couple could use the same (email), so (email) alone is not a candidate key. It is also vulnerable to modifications.
  5) The only attributes guaranteed to be unique are (SSN) and (medicareNumber). In this case, we opt for a surrogate key to prevent privacy leaks.
- PhoneNumbers
  1) For the proposed candidate key to be valid, extension must have a default value for when (phoneNumber) has no (extension). A primary key or part of a primary key cannot be null.
  2) (phoneNumber) alone is not enough to determine phone (type) due to the case where the same (phoneNumber) has several extensions, as some extensions could be for mobiles, others for landlines.
- ClubMembers
  1) ClubMember is a Person by inheritance, so the primary key memberID directly references personID from Person table.
  2) Without a memberID, none of the attributes in the same row are guaranteed unique. Any combination of attributes will not make up a key, because two members could have the same height, weight, status, member type and join date. Therefore, memberID in this scenario is a much needed surrogate key to guarantee 1NF.
- Locations
  1) Each branch of the MVC should have a unique name, therefore (name) can constitute a candidate key.
  2) (addressID) is also unique. No two branches should be registered at the same address.
  3) Although (webAddress) should be unique, it is vulnerable to modification (mutable) and therefore not a good candidate key despite its potential for being functionally determinant of the rest of the table.
- Teams
  1) The same team name could be used at different branch locations, but likely not at the same location. Therefore, adding the (locationID) to team name can form a candidate key.
- Payments
  1) (paymentDate) is YYYY-MM-DD and (membershipYear) is YYYY. This does not violate 2NF by a partial dependency between prime attribute (membershipYear) and non-prime attribute (paymentDate), because a (paymentDate) for a (membershipYear) is not necessarily restricted to the same calendar year. A member can pay in advance or be in arrears, thus decoupling the YYYY values of each attribute from each other.

Breakdown of Responsibilities

ID	**Task Type**	**Task Description**	**Assigned to**
1	Database Normalization	Develop an E/R diagram to represent the conceptual database design for the above application.	Mark
2		Mark or express various constraints (keys, functional dependencies, cardinalities of the relationships, etc.). Identify any constraints that are not captured by the E/R diagram.	Mark
3		Convert your E/R diagram into a relational database schema. Make refinements to the DB schema if necessary. Identify various integrity constraints such as primary keys, foreign keys, functional dependencies, and referential constraints. Make sure that your database schema is at least in 3NF.	Mark
4		Are all your relations in the database in BCNF? (Explain which ones and why not)	Ricky
5		For any relation in your database, if it is not in BCNF, then show that it is in 3NF.	Ricky
6		Create at least one trigger to execute some of the requirements specified in the description above.	Mark
7		General review of WU project DB design	Ricky/Mark
1	Procedures Queries for Insertion Deletion Updates	Create/Delete/Edit/Display a Location.	Ricky/Mark
2		Create/Delete/Edit/Display a Personnel.	Ricky/Mark
3		Create/Delete/Edit/Display a FamilyMember (Primary/Secondary).	Ricky/Mark
4		Create/Delete/Edit/Display a ClubMember (Major/Minor).	Ricky/Mark
5		Create/Delete/Edit/Display a TeamFormation.	Ricky/Mark
6		Assign/Delete/Edit a club member to a team formation.	Ricky/Mark
7		Make payment for a club member.	Ricky/Mark

ID	Task Type	Task Description	Assigned to
8	SQL Sample Queries (DDL, DML)	Get complete details for every location in the system	Anas
9		Get family member and associated club members details	Anas
10		Get team formations for a location within a date range	Anas
11		Get inactive club members with multiple locations and 2+ years membership	Anas
12		Team formation report with date range parameter	Anas
13		Get active club members never assigned to any formation	Anas
14		Get active major members who joined as minors	Anas
15		Get active club members only assigned as setters	Anas
16		Get active club members assigned to all positions in games	Anas
17		Get family members who are coaches at a specific location	Anas
18		Get active club members who never lost a game	Anas
19		Get volunteer personnel who are family members of minor club members	Anas
20		You should show the trigger(s) used by your system. Explain the trigger(s) used and their benefits.	Mark
21		You need to demonstrate the integrity of all the requirements provided in the description. Example, the system should not allow a user to assign a player on two different formation sessions at the same time or on a conflicting time (less than three hours difference).	Mark
22		You need to demonstrate the generation of emails, and the logs of the emails produced by the system.	Mark
1	Miscellaneous	Web Interface/GUI design	Jalal/Ricky/Mark
2		Update E/R diagram	Mark
3		Assemble Final Report (preferably in Latex but not mandatory)	Mark

SQL Procedures and Queries

We've used several saved SQL queries and procedures for inserting and deleting records, to allow for serverside control of such events.

Insertion Procedures

1. InsertAddress

1. InsertAddress

```
DELIMITER //
CREATE  PROCEDURE InsertAddress(
    IN p_street_address VARCHAR(255),
    IN p_city VARCHAR(100),
    IN p_province VARCHAR(50),
    IN p_postal_code CHAR(6),
    OUT p_address_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_address_id INT DEFAULT NULL;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_address_id = -1;
    END;

    START TRANSACTION;

    -- Validate postal code format
    IF p_postal_code IS NOT NULL AND NOT (p_postal_code REGEXP
       '^[A-Z][0-9][A-Z][0-9][A-Z][0-9]$') THEN
        SET p_address_id = -1;
        SET p_result_message = 'Invalid postal code format. Must be A1A1A1';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if address exists (case-insensitive for city/province)
    SELECT addressID INTO v_address_id
    FROM Address
    WHERE streetAddress = p_street_address
    AND LOWER(city) = LOWER(p_city)
    AND LOWER(province) = LOWER(p_province)
    AND postalCode = p_postal_code
    LIMIT 1;

    -- If address doesn't exist, create it
    IF v_address_id IS NULL THEN
        INSERT INTO Address (streetAddress, city, province, postalCode)
        VALUES (p_street_address, p_city, p_province, p_postal_code);
        SET v_address_id = LAST_INSERT_ID();
        SET p_result_message = 'Address created successfully';
    ELSE
```

```
        SET p_result_message = 'Address␣already␣exists';
    END IF;

    SET p_address_id = v_address_id;
    COMMIT;
END //
DELIMITER ;
```

2. InsertPhone

2. InsertPhone

```
DELIMITER //
CREATE  PROCEDURE InsertPhone(
    IN p_phone_number VARCHAR(20),
    IN p_phone_extension VARCHAR(10),
    IN p_phone_type VARCHAR(20),
    OUT p_phone_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_phone_id INT DEFAULT NULL;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_phone_id = -1;
    END;

    START TRANSACTION;

    -- Basic phone number validation
    IF p_phone_number IS NULL OR LENGTH(TRIM(p_phone_number)) = 0 THEN
        SET p_phone_id = -1;
        SET p_result_message = 'Phone␣number␣cannot␣be␣empty';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if phone exists
    SELECT phoneID INTO v_phone_id
    FROM PhoneNumbers
    WHERE phoneNumber = p_phone_number
    AND (extension = p_phone_extension OR (extension IS NULL AND
       p_phone_extension IS NULL))
    LIMIT 1;

    -- If phone doesn't exist, create it
    IF v_phone_id IS NULL THEN
        INSERT INTO PhoneNumbers (phoneNumber, extension, type)
        VALUES (p_phone_number, p_phone_extension, COALESCE(p_phone_type,
           'Cell'));
        SET v_phone_id = LAST_INSERT_ID();
        SET p_result_message = 'Phone␣number␣created␣successfully';
    ELSE
        SET p_result_message = 'Phone␣number␣already␣exists';
    END IF;
    SET p_phone_id = v_phone_id;
    COMMIT;
END //
DELIMITER ;
```

3. InsertPerson

3. InsertPerson

```
DELIMITER //
CREATE  PROCEDURE InsertPerson(
    IN p_first_name VARCHAR(100),
    IN p_last_name VARCHAR(100),
    IN p_middle_initial VARCHAR(10),
    IN p_date_of_birth DATE,
    IN p_ssn CHAR(9),
    IN p_medicare_number VARCHAR(15),
    IN p_email VARCHAR(100),
    IN p_street_address VARCHAR(255),
    IN p_city VARCHAR(100),
    IN p_province VARCHAR(50),
    IN p_postal_code CHAR(6),
    IN p_phone_number VARCHAR(20),
    IN p_phone_extension VARCHAR(10),
    IN p_phone_type VARCHAR(20),
    OUT p_person_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_address_id INT DEFAULT NULL;
    DECLARE v_phone_id INT DEFAULT NULL;
    DECLARE v_existing_person_id INT DEFAULT NULL;
    DECLARE v_address_message VARCHAR(255);
    DECLARE v_phone_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_person_id = -1;
    END;

    START TRANSACTION;

    -- Validate required fields
    IF p_ssn IS NULL OR LENGTH(TRIM(p_ssn)) != 9 OR NOT (p_ssn REGEXP
       '^[0-9]{9}$') THEN
        SET p_person_id = -1;
        SET p_result_message = 'Invalid␣SSN␣format.␣Must␣be␣9␣digits';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if person already exists by SSN
    SELECT personID INTO v_existing_person_id
    FROM Person
    WHERE ssn = p_ssn
```

```
LIMIT 1;

IF v_existing_person_id IS NOT NULL THEN
    SET p_person_id = v_existing_person_id;
    SET p_result_message = 'Person␣with␣this␣SSN␣already␣exists';
    COMMIT;
    LEAVE proc_label;
END IF;

-- Check if person already exists by Medicare Number (if provided)
IF p_medicare_number IS NOT NULL THEN
    SELECT personID INTO v_existing_person_id
    FROM Person
    WHERE medicareNumber = p_medicare_number
    LIMIT 1;

    IF v_existing_person_id IS NOT NULL THEN
        SET p_person_id = v_existing_person_id;
        SET p_result_message = 'Person␣with␣this␣Medicare␣Number␣already␣
            exists';
        COMMIT;
        LEAVE proc_label;
    END IF;
END IF;

-- Handle Address using enhanced InsertAddress procedure
IF p_street_address IS NOT NULL THEN
    CALL InsertAddress(p_street_address, p_city, p_province, p_postal_code,
        v_address_id, v_address_message);
    IF v_address_id = -1 THEN
        SET p_person_id = -1;
        SET p_result_message = CONCAT('Address␣error:␣', v_address_message);
        ROLLBACK;
        LEAVE proc_label;
    END IF;
END IF;

-- Handle Phone Number using enhanced InsertPhone procedure
IF p_phone_number IS NOT NULL THEN
    CALL InsertPhone(p_phone_number, p_phone_extension, p_phone_type,
        v_phone_id, v_phone_message);
    IF v_phone_id = -1 THEN
        SET p_person_id = -1;
        SET p_result_message = CONCAT('Phone␣error:␣', v_phone_message);
        ROLLBACK;
        LEAVE proc_label;
    END IF;
END IF;

-- Insert the Person
INSERT INTO Person (
    firstName, lastName, middleInitial, dateOfBirth,
    ssn, medicareNumber, email, addressID
)
VALUES (
    p_first_name, p_last_name, p_middle_initial, p_date_of_birth,
    p_ssn, p_medicare_number, p_email, v_address_id
);
```

```
    SET p_person_id = LAST_INSERT_ID();

    -- Link phone to person (if phone was provided)
    IF v_phone_id IS NOT NULL THEN
        INSERT IGNORE INTO PersonPhones (personID, phoneID) VALUES (p_person_id,
            v_phone_id);
    END IF;

    SET p_result_message = 'Person␣created␣successfully';
    COMMIT;
END //
DELIMITER ;
```

4. InsertLocation

4. InsertLocation

```
DELIMITER //
CREATE  PROCEDURE InsertLocation(
-- Address information
    IN p_street_address VARCHAR(255),
    IN p_city VARCHAR(100),
    IN p_province VARCHAR(50),
    IN p_postal_code CHAR(6),
    -- Location information
    IN p_location_name VARCHAR(100),
    IN p_location_type ENUM('Head', 'Branch'),
        IN p_size INT,
    IN p_capacity INT,
    IN p_web_address VARCHAR(255),
    -- Phone information
        IN p_phone_number VARCHAR(20),
    IN p_phone_extension VARCHAR(10),
    IN p_phone_type VARCHAR(20),
    -- Outputs
    OUT p_location_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_address_id INT DEFAULT NULL;
    DECLARE v_phone_id INT DEFAULT NULL;
    DECLARE v_existing_location_id INT DEFAULT NULL;
    DECLARE v_address_message VARCHAR(255);
    DECLARE v_phone_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_location_id = -1;
    END;

    START TRANSACTION;

    -- Validate required fields
    IF p_location_name IS NULL OR LENGTH(TRIM(p_location_name)) = 0 THEN
        SET p_location_id = -1;
        SET p_result_message = 'Location␣name␣cannot␣be␣empty';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if location already exists
    SELECT locationID INTO v_existing_location_id
    FROM Locations
    WHERE name = p_location_name
    LIMIT 1;

    IF v_existing_location_id IS NOT NULL THEN
        SET p_location_id = v_existing_location_id;
        SET p_result_message = 'Location␣already␣exists';
        COMMIT;
        LEAVE proc_label;
    END IF;

    -- Handle Address using enhanced InsertAddress procedure
    IF p_street_address IS NOT NULL THEN
        CALL InsertAddress(p_street_address, p_city, p_province, p_postal_code,
            v_address_id, v_address_message);
        IF v_address_id = -1 THEN
            SET p_location_id = -1;
            SET p_result_message = CONCAT('Address␣error:␣', v_address_message);
            ROLLBACK;
            LEAVE proc_label;
        END IF;
    END IF;

    -- Handle Phone Number using enhanced InsertPhone procedure
    IF p_phone_number IS NOT NULL THEN
        CALL InsertPhone(p_phone_number, p_phone_extension, p_phone_type,
            v_phone_id, v_phone_message);
        IF v_phone_id = -1 THEN
            SET p_location_id = -1;
            SET p_result_message = CONCAT('Phone␣error:␣', v_phone_message);
            ROLLBACK;
            LEAVE proc_label;
        END IF;
    END IF;

    -- Insert the Location
    INSERT INTO Locations (
        name, type, addressID, size, capacity, webAddress,
        generalManagerAssignmentID, isActive
    )
    VALUES (
        p_location_name, p_location_type, v_address_id, p_size,
        p_capacity, p_web_address, NULL, 1
    );

    SET p_location_id = LAST_INSERT_ID();

    -- Link phone to location (if phone was provided)
    IF v_phone_id IS NOT NULL THEN
        INSERT IGNORE INTO LocationPhones (locationID, phoneID) VALUES
            (p_location_id, v_phone_id);
    END IF;
    SET p_result_message = 'Location␣created␣successfully';
    COMMIT;
END //
DELIMITER ;
```

5A. InsertPersonnelAssignment

5A. InsertPersonnelAssignment

```
DELIMITER //
CREATE  PROCEDURE InsertPersonnelAssignment(
    IN p_person_id INT,
    IN p_location_id INT,
    IN p_role ENUM('General␣Manager','Deputy␣
       Manager','Treasurer','Secretary','Administrator','Captain','Coach','Assistant␣
       Coach','Other'),
    IN p_mandate_type ENUM('Volunteer','Salaried'),
    IN p_start_date DATE,
    IN p_end_date DATE,
    OUT p_assignment_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_assignment_id INT DEFAULT NULL;
    DECLARE v_existing_assignment INT DEFAULT NULL;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_assignment_id = -1;
    END;

    START TRANSACTION;

    -- Validate required fields
    IF p_person_id IS NULL THEN
        SET p_assignment_id = -1;
        SET p_result_message = 'Person␣ID␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_location_id IS NULL THEN
        SET p_assignment_id = -1;
        SET p_result_message = 'Location␣ID␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_role IS NULL THEN
        SET p_assignment_id = -1;
        SET p_result_message = 'Role␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate that person and location exist (simplified checks)
```

```
IF NOT EXISTS (SELECT 1 FROM Person WHERE personID = p_person_id) THEN
    SET p_assignment_id = -1;
    SET p_result_message = 'Person␣does␣not␣exist';
    ROLLBACK;
    LEAVE proc_label;
END IF;

IF NOT EXISTS (SELECT 1 FROM Locations WHERE locationID = p_location_id AND
    isActive = 1) THEN
    SET p_assignment_id = -1;
    SET p_result_message = 'Location␣does␣not␣exist␣or␣is␣inactive';
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Validate start date
IF p_start_date IS NULL THEN
    SET p_start_date = CURDATE();
END IF;

-- Validate end date (if provided)
IF p_end_date IS NOT NULL AND p_end_date < p_start_date THEN
    SET p_assignment_id = -1;
    SET p_result_message = 'End␣date␣cannot␣be␣before␣start␣date';
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Check if this person already has an active assignment at this location
    with this role
SELECT assignmentID INTO v_existing_assignment
FROM PersonnelLocations
WHERE personnelID = p_person_id
AND locationID = p_location_id
AND role = p_role
AND (endDate IS NULL OR endDate > CURDATE())
LIMIT 1;

IF v_existing_assignment IS NOT NULL THEN
    SET p_assignment_id = v_existing_assignment;
    SET p_result_message = 'Personnel␣assignment␣already␣exists␣for␣this␣
        person/location/role';
    COMMIT;
    LEAVE proc_label;
END IF;

-- Special validation for General Manager role (only one active per location)
IF p_role = 'General␣Manager' THEN
    IF EXISTS (
        SELECT 1 FROM PersonnelLocations
        WHERE locationID = p_location_id
        AND role = 'General␣Manager'
        AND (endDate IS NULL OR endDate > CURDATE())
    ) THEN
        SET p_assignment_id = -1;
        SET p_result_message = 'Location␣already␣has␣an␣active␣General␣
            Manager';
        ROLLBACK;
        LEAVE proc_label;
        END IF;
    END IF;

    -- Create Personnel Assignment
    INSERT INTO PersonnelLocations (
        personnelID, locationID, role, mandateType, startDate, endDate
    )
    VALUES (
        p_person_id, p_location_id, p_role,
        COALESCE(p_mandate_type, 'Volunteer'), p_start_date, p_end_date
    );

    SET v_assignment_id = LAST_INSERT_ID();

    -- If this is a General Manager assignment, update the location's GM reference
    IF p_role = 'General␣Manager' THEN
        UPDATE Locations
        SET generalManagerAssignmentID = v_assignment_id
        WHERE locationID = p_location_id;
    END IF;

    SET p_assignment_id = v_assignment_id;
    SET p_result_message = 'Personnel␣assignment␣created␣successfully';

    COMMIT;
END //
DELIMITER ;
```

5B. InsertPersonnel

5B. InsertPersonnel

```
DELIMITER //
CREATE  PROCEDURE InsertPersonnel(
    -- Person details
    IN p_first_name VARCHAR(100),
    IN p_last_name VARCHAR(100),
    IN p_middle_initial VARCHAR(10),
    IN p_date_of_birth DATE,
    IN p_ssn CHAR(9),
    IN p_medicare_number VARCHAR(15),
    IN p_email VARCHAR(100),
    IN p_street_address VARCHAR(255),
    IN p_city VARCHAR(100),
    IN p_province VARCHAR(50),
    IN p_postal_code CHAR(6),
    IN p_phone_number VARCHAR(20),
    IN p_phone_extension VARCHAR(10),
    IN p_phone_type VARCHAR(20),
    -- Personnel assignment details
    IN p_location_id INT,
    IN p_role ENUM('General␣Manager','Deputy␣
       Manager','Treasurer','Secretary','Administrator','Captain','Coach','Assistant␣
       Coach','Other'),
    IN p_mandate_type ENUM('Volunteer','Salaried'),
    IN p_start_date DATE,
    IN p_end_date DATE,
    OUT p_person_id INT,
    OUT p_assignment_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_person_id INT DEFAULT NULL;
    DECLARE v_assignment_id INT DEFAULT NULL;
    DECLARE v_person_message VARCHAR(255);
    DECLARE v_assignment_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_person_id = -1;
        SET p_assignment_id = -1;
    END;

    START TRANSACTION;

    -- Create/Find Person using the InsertPerson procedure
    CALL InsertPerson(
        p_first_name, p_last_name, p_middle_initial, p_date_of_birth,
        p_ssn, p_medicare_number, p_email,
        p_street_address, p_city, p_province, p_postal_code,
        p_phone_number, p_phone_extension, p_phone_type,
        v_person_id, v_person_message
    );

    IF v_person_id = -1 THEN
        SET p_person_id = -1;
        SET p_assignment_id = -1;
        SET p_result_message = CONCAT('Person creation failed: ',
            v_person_message);
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Create Personnel Assignment using the InsertPersonnelAssignment procedure
    CALL InsertPersonnelAssignment(
        v_person_id, p_location_id, p_role, p_mandate_type,
        p_start_date, p_end_date,
        v_assignment_id, v_assignment_message
    );

    IF v_assignment_id = -1 THEN
        SET p_person_id = -1;
        SET p_assignment_id = -1;
        SET p_result_message = CONCAT('Personnel assignment failed: ',
            v_assignment_message);
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    SET p_person_id = v_person_id;
    SET p_assignment_id = v_assignment_id;
    SET p_result_message = CONCAT('Personnel created successfully - Person: ',
       v_person_message, '; Assignment: ', v_assignment_message);

    COMMIT;
```

```
END //
DELIMITER ;
```

6A. InsertClubMember

6A. InsertClubMember

```
DELIMITER //
CREATE  PROCEDURE InsertClubMember(
    IN p_person_id INT,
    IN p_height DECIMAL(5,2),
    IN p_weight DECIMAL(5,2),
    IN p_gender ENUM('Male','Female'),
    IN p_member_type ENUM('Major','Minor'),
    IN p_location_id INT,
    IN p_family_member_id INT,
    IN p_relationship_type
       ENUM('Father','Mother','Grandfather','Grandmother','Sibling','Tutor','Partner','Friend',
    IN p_contact_role ENUM('Primary','Emergency'),
    OUT p_member_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_is_minor BOOLEAN DEFAULT FALSE;
    DECLARE v_current_date DATE DEFAULT CURDATE();
    DECLARE v_assignment_id INT DEFAULT NULL;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_member_id = -1;
    END;

    START TRANSACTION;

    -- Validate required fields
    IF p_person_id IS NULL THEN
        SET p_member_id = -1;
        SET p_result_message = 'Person␣ID␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_member_type IS NULL THEN
        SET p_member_id = -1;
        SET p_result_message = 'Member␣type␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate that person exists
    IF NOT EXISTS (SELECT 1 FROM Person WHERE personID = p_person_id) THEN
        SET p_member_id = -1;
        SET p_result_message = 'Person␣does␣not␣exist';
        ROLLBACK;
        LEAVE proc_label;
    END IF;
```

```sql
-- Validate location exists if provided
IF p_location_id IS NOT NULL THEN
    IF NOT EXISTS (SELECT 1 FROM Locations WHERE locationID = p_location_id
        AND isActive = 1) THEN
        SET p_member_id = -1;
        SET p_result_message = 'Location␣does␣not␣exist␣or␣is␣inactive';
        ROLLBACK;
        LEAVE proc_label;
    END IF;
END IF;

-- Check if person is already a club member
IF EXISTS (SELECT 1 FROM ClubMembers WHERE memberID = p_person_id) THEN
    SET p_member_id = p_person_id;
    SET p_result_message = 'Person␣is␣already␣a␣club␣member';
    COMMIT;
    LEAVE proc_label;
END IF;

-- For minors, validate family member relationship
IF p_member_type = 'Minor' THEN
    SET v_is_minor = TRUE;

    -- Family member is required for minors
    IF p_family_member_id IS NULL THEN
        SET p_member_id = -1;
        SET p_result_message = 'Family␣member␣ID␣is␣required␣for␣minor␣
            members';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate that family member exists
    IF NOT EXISTS (SELECT 1 FROM FamilyMembers WHERE familyMemberID =
        p_family_member_id) THEN
        SET p_member_id = -1;
        SET p_result_message = 'Family␣member␣does␣not␣exist';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate relationship type is provided
    IF p_relationship_type IS NULL THEN
        SET p_member_id = -1;
        SET p_result_message = 'Relationship␣type␣is␣required␣for␣minor␣
            members';
        ROLLBACK;
        LEAVE proc_label;
    END IF;
END IF;

-- For majors, family member is optional but validate if provided
IF p_member_type = 'Major' AND p_family_member_id IS NOT NULL THEN
    -- Validate that family member exists
    IF NOT EXISTS (SELECT 1 FROM FamilyMembers WHERE familyMemberID =
        p_family_member_id) THEN
        SET p_member_id = -1;
        SET p_result_message = 'Family␣member␣does␣not␣exist';
```

```
            ROLLBACK;
            LEAVE proc_label;
        END IF;
    END IF;

    -- Create Club Member record
    INSERT INTO ClubMembers (
        memberID, height, weight, gender, isActive, memberType, joinedDate
    )
    VALUES (
        p_person_id, p_height, p_weight, p_gender, 1, p_member_type,
           v_current_date
    );

    SET p_member_id = p_person_id;

    -- Create MemberLocations record if location was provided
    IF p_location_id IS NOT NULL THEN
        INSERT INTO MemberLocations (
            memberID, locationID, startDate, endDate
        )
        VALUES (
            p_person_id, p_location_id, v_current_date, NULL
        );

        SET v_assignment_id = LAST_INSERT_ID();
    END IF;

    -- If this is a minor or a major with family member, create relationship
    IF (v_is_minor OR (p_member_type = 'Major' AND p_family_member_id IS NOT
       NULL)) THEN
        -- Default to 'Primary' if contact role not specified
        SET p_contact_role = COALESCE(p_contact_role, 'Primary');

        -- For minors, relationship type is required
        -- For majors with family member, use 'Other' if not specified
        IF p_member_type = 'Major' AND p_relationship_type IS NULL THEN
            SET p_relationship_type = 'Other';
        END IF;

        INSERT INTO FamilyRelationship (
            minorID, majorID, relationshipType, contactRole
        )
        VALUES (
            CASE WHEN v_is_minor THEN p_person_id ELSE NULL END,
            p_family_member_id,
            p_relationship_type,
            p_contact_role
        )
        ON DUPLICATE KEY UPDATE
            relationshipType = COALESCE(p_relationship_type, relationshipType),
            contactRole = COALESCE(p_contact_role, contactRole);
    END IF;

    SET p_result_message = CONCAT(UCASE(p_member_type), '␣club␣member␣created␣
       successfully');
    IF p_location_id IS NOT NULL THEN
        SET p_result_message = CONCAT(p_result_message, '␣with␣location␣
           assignment');
    END IF;

    COMMIT;
END //
DELIMITER ;
```

6B. InsertClubMemberWithPerson

6B. InsertClubMemberWithperson

```
DELIMITER //
CREATE  PROCEDURE InsertClubMemberWithPerson(
    -- Person details
    IN p_first_name VARCHAR(100),
    IN p_last_name VARCHAR(100),
    IN p_middle_initial VARCHAR(10),
    IN p_date_of_birth DATE,
    IN p_ssn CHAR(9),
    IN p_medicare_number VARCHAR(15),
    IN p_email VARCHAR(100),
    IN p_street_address VARCHAR(255),
    IN p_city VARCHAR(100),
    IN p_province VARCHAR(50),
    IN p_postal_code CHAR(6),
    IN p_phone_number VARCHAR(20),
    IN p_phone_extension VARCHAR(10),
    IN p_phone_type VARCHAR(20),
    -- Club member details
    IN p_height DECIMAL(5,2),
    IN p_weight DECIMAL(5,2),
    IN p_gender ENUM('Male','Female'),
    IN p_member_type ENUM('Major','Minor'),
    IN p_location_id INT,
    IN p_family_member_id INT,
    IN p_relationship_type
       ENUM('Father','Mother','Grandfather','Grandmother','Sibling','Tutor','Partner','Friend',
    IN p_contact_role ENUM('Primary','Emergency'),
    -- Outputs
    OUT p_person_id INT,
    OUT p_member_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_person_id INT DEFAULT NULL;
    DECLARE v_member_id INT DEFAULT NULL;
    DECLARE v_person_message VARCHAR(255);
    DECLARE v_member_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_person_id = -1;
        SET p_member_id = -1;
    END;

    START TRANSACTION;
    -- Create/Find Person using the InsertPerson procedure
    CALL InsertPerson(
        p_first_name, p_last_name, p_middle_initial, p_date_of_birth,
        p_ssn, p_medicare_number, p_email,
        p_street_address, p_city, p_province, p_postal_code,
        p_phone_number, p_phone_extension, p_phone_type,
        v_person_id, v_person_message
    );

    IF v_person_id = -1 THEN
        SET p_person_id = -1;
        SET p_member_id = -1;
        SET p_result_message = CONCAT('Person creation failed: ',
            v_person_message);
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Create Club Member using the InsertClubMember procedure
    CALL InsertClubMember(
        v_person_id, p_height, p_weight, p_gender, p_member_type,
        p_location_id, p_family_member_id, p_relationship_type, p_contact_role,
        v_member_id, v_member_message
    );

    IF v_member_id = -1 THEN
        SET p_person_id = -1;
        SET p_member_id = -1;
        SET p_result_message = CONCAT('Club member creation failed: ',
            v_member_message);
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    SET p_person_id = v_person_id;
    SET p_member_id = v_member_id;
    SET p_result_message = CONCAT('Club member created successfully - Person: ',
        v_person_message, '; Member: ', v_member_message);

    COMMIT;
END //
DELIMITER ;
```

7. InsertTeam

7. InsertTeam

```
DELIMITER //
CREATE  PROCEDURE InsertTeam(
    IN p_location_id INT,
    IN p_gender ENUM('Male','Female'),
    IN p_team_name VARCHAR(100),
    IN p_coach_id INT,
    IN p_league_level VARCHAR(20),
    OUT p_team_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
```

```
DECLARE v_location_exists INT DEFAULT 0;
DECLARE v_coach_exists INT DEFAULT 0;
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK;
    GET DIAGNOSTICS CONDITION 1
        p_result_message = MESSAGE_TEXT;
    SET p_team_id = -1;
END;

START TRANSACTION;

-- Validate required fields
IF p_location_id IS NULL THEN
    SET p_team_id = -1;
    SET p_result_message = 'Location␣ID␣is␣required';
    ROLLBACK;
    LEAVE proc_label;
END IF;

IF p_gender IS NULL THEN
    SET p_team_id = -1;
    SET p_result_message = 'Gender␣is␣required';
    ROLLBACK;
    LEAVE proc_label;
END IF;

IF p_team_name IS NULL OR TRIM(p_team_name) = '' THEN
    SET p_team_id = -1;
    SET p_result_message = 'Team␣name␣is␣required';
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Validate location exists
SELECT COUNT(*) INTO v_location_exists
FROM Locations
WHERE locationID = p_location_id AND isActive = 1;

IF v_location_exists = 0 THEN
    SET p_team_id = -1;
    SET p_result_message = 'Location␣does␣not␣exist␣or␣is␣inactive';
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Validate coach exists if provided
IF p_coach_id IS NOT NULL THEN
    SELECT COUNT(*) INTO v_coach_exists
    FROM PersonnelLocations
    WHERE assignmentID = p_coach_id AND (endDate IS NULL OR endDate >
        CURDATE());

    IF v_coach_exists = 0 THEN
        SET p_team_id = -1;
        SET p_result_message = 'Coach␣assignment␣does␣not␣exist␣or␣is␣
            inactive';
        ROLLBACK;
        LEAVE proc_label;
        END IF;
    END IF;

    -- Create the team
    INSERT INTO Teams (
        locationID, gender, name, coachID, leagueLevel, isActive
    )
    VALUES (
        p_location_id, p_gender, p_team_name, p_coach_id, p_league_level, 1
    );

    SET p_team_id = LAST_INSERT_ID();
    SET p_result_message = CONCAT('Team "', p_team_name, '" created successfully 
       with ID ', p_team_id);

    COMMIT;
END //
DELIMITER ;
```

8. InsertSession

8. InsertSession

```
DELIMITER //
CREATE  PROCEDURE InsertSession(
    IN p_home_team_id INT,
    IN p_away_team_id INT,
    IN p_location_id INT,
    IN p_start_time DATETIME,
    IN p_end_time DATETIME,
    IN p_session_type ENUM('Training','Game'),
    IN p_home_score INT,
    IN p_away_score INT,
    OUT p_session_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_home_team_exists INT DEFAULT 0;
    DECLARE v_away_team_exists INT DEFAULT 0;
    DECLARE v_location_exists INT DEFAULT 0;
    DECLARE v_same_team INT DEFAULT 0;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_session_id = -1;
    END;

    START TRANSACTION;

    -- Validate required fields
    IF p_home_team_id IS NULL THEN
        SET p_session_id = -1;
        SET p_result_message = 'Home Team ID is required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;
```

```
IF p_away_team_id IS NULL THEN
    SET p_session_id = -1;
    SET p_result_message = 'Away␣Team␣ID␣is␣required';
    ROLLBACK;
    LEAVE proc_label;
END IF;

IF p_location_id IS NULL THEN
    SET p_session_id = -1;
    SET p_result_message = 'Location␣ID␣is␣required';
    ROLLBACK;
    LEAVE proc_label;
END IF;

IF p_start_time IS NULL THEN
    SET p_session_id = -1;
    SET p_result_message = 'Start␣time␣is␣required';
    ROLLBACK;
    LEAVE proc_label;
END IF;

IF p_session_type IS NULL THEN
    SET p_session_id = -1;
    SET p_result_message = 'Session␣type␣is␣required';
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Check if teams are different
IF p_home_team_id = p_away_team_id THEN
    SET p_session_id = -1;
    SET p_result_message = 'Home␣and␣Away␣teams␣cannot␣be␣the␣same';
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Validate teams exist
SELECT COUNT(*) INTO v_home_team_exists
FROM Teams
WHERE teamID = p_home_team_id AND isActive = 1;

SELECT COUNT(*) INTO v_away_team_exists
FROM Teams
WHERE teamID = p_away_team_id AND isActive = 1;

IF v_home_team_exists = 0 THEN
    SET p_session_id = -1;
    SET p_result_message = 'Home␣Team␣does␣not␣exist␣or␣is␣inactive';
    ROLLBACK;
    LEAVE proc_label;
END IF;

IF v_away_team_exists = 0 THEN
    SET p_session_id = -1;
    SET p_result_message = 'Away␣Team␣does␣not␣exist␣or␣is␣inactive';
    ROLLBACK;
    LEAVE proc_label;
END IF;
```

```
-- Validate location exists
SELECT COUNT(*) INTO v_location_exists
FROM Locations
WHERE locationID = p_location_id AND isActive = 1;

IF v_location_exists = 0 THEN
    SET p_session_id = -1;
    SET p_result_message = 'Location␣does␣not␣exist␣or␣is␣inactive';
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Validate time logic if end time is provided
IF p_end_time IS NOT NULL AND p_end_time <= p_start_time THEN
    SET p_session_id = -1;
    SET p_result_message = 'End␣time␣must␣be␣after␣start␣time␣if␣provided';
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Validate scores only if provided for games
IF p_session_type = 'Game' THEN
    IF (p_home_score IS NOT NULL AND p_away_score IS NULL) OR
       (p_home_score IS NULL AND p_away_score IS NOT NULL) THEN
        SET p_session_id = -1;
        SET p_result_message = 'Both␣scores␣must␣be␣provided␣or␣both␣must␣be␣
            NULL␣for␣games';
        ROLLBACK;
        LEAVE proc_label;
    END IF;
ELSE -- Training session
    IF p_home_score IS NOT NULL OR p_away_score IS NOT NULL THEN
        SET p_session_id = -1;
        SET p_result_message = 'Scores␣should␣not␣be␣provided␣for␣training␣
            sessions';
        ROLLBACK;
        LEAVE proc_label;
    END IF;
END IF;

-- Create the session
INSERT INTO TeamSessions (
    homeTeamID, awayTeamID, locationID, startTime, endTime,
    sessionType, homeScore, awayScore
)
VALUES (
    p_home_team_id, p_away_team_id, p_location_id, p_start_time,
    p_end_time, p_session_type, p_home_score, p_away_score
);

SET p_session_id = LAST_INSERT_ID();

-- Generate appropriate success message
IF p_session_type = 'Game' THEN
    IF p_home_score IS NOT NULL THEN
        SET p_result_message = CONCAT('Game␣session␣created␣successfully␣with␣
            ID␣', p_session_id,
                                            '␣(Score:␣', COALESCE(p_home_score,
                                                'NULL'), '-',
                                            COALESCE(p_away_score, 'NULL'), ')');
        ELSE
            SET p_result_message = CONCAT('Game␣session␣created␣successfully␣with␣
                ID␣', p_session_id,
                                            '␣(Scores␣not␣yet␣recorded)');
        END IF;
    ELSE
        SET p_result_message = CONCAT('Training␣session␣created␣successfully␣with␣
            ID␣', p_session_id);
    END IF;

    COMMIT;
END //
DELIMITER ;
```

9. InsertMemberStats

9. InsertMemberStats

```
DELIMITER //
CREATE  PROCEDURE InsertMemberStats(
    IN p_member_id INT,
    IN p_team_id INT,
    IN p_season_year YEAR,
    IN p_start_date DATE,
    IN p_end_date DATE,
    IN p_position ENUM('Setter','Outside␣Hitter','Opposite␣Hitter','Middle␣
        Blocker','Libero','Defensive␣Specialist','Serving␣Specialist'),
    OUT p_stat_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_member_exists INT DEFAULT 0;
    DECLARE v_team_exists INT DEFAULT 0;
    DECLARE v_active_assignment INT DEFAULT 0;
    DECLARE v_current_stat_id INT DEFAULT NULL;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_stat_id = -1;
    END;

    START TRANSACTION;

    -- Validate required fields
    IF p_member_id IS NULL THEN
        SET p_stat_id = -1;
        SET p_result_message = 'Member␣ID␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_team_id IS NULL THEN
        SET p_stat_id = -1;
        SET p_result_message = 'Team␣ID␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_season_year IS NULL THEN
        SET p_stat_id = -1;
        SET p_result_message = 'Season␣year␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_start_date IS NULL THEN
        SET p_stat_id = -1;
        SET p_result_message = 'Start␣date␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_position IS NULL THEN
        SET p_stat_id = -1;
        SET p_result_message = 'Position␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate member exists and is active
    SELECT COUNT(*) INTO v_member_exists
    FROM ClubMembers
    WHERE memberID = p_member_id AND isActive = 1;

    IF v_member_exists = 0 THEN
        SET p_stat_id = -1;
        SET p_result_message = 'Member␣does␣not␣exist␣or␣is␣inactive';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate team exists and is active
    SELECT COUNT(*) INTO v_team_exists
    FROM Teams
    WHERE teamID = p_team_id AND isActive = 1;

    IF v_team_exists = 0 THEN
        SET p_stat_id = -1;
        SET p_result_message = 'Team␣does␣not␣exist␣or␣is␣inactive';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check for date validity
    IF p_end_date IS NOT NULL AND p_end_date < p_start_date THEN
        SET p_stat_id = -1;
        SET p_result_message = 'End␣date␣cannot␣be␣before␣start␣date';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check for existing active assignment for this member-team-season
    SELECT COUNT(*) INTO v_active_assignment
    FROM MemberStats
    WHERE memberID = p_member_id
      AND teamID = p_team_id
      AND seasonYear = p_season_year
      AND (endDate IS NULL OR endDate > CURDATE());

    IF v_active_assignment > 0 THEN
        SET p_stat_id = -1;
        SET p_result_message = 'Member␣already␣has␣an␣active␣assignment␣for␣this␣
            team/season';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Create the member stats record
    INSERT INTO MemberStats (
        memberID, teamID, seasonYear, startDate, endDate, position,
        goals, assists, playtime, rating, notes
    )
    VALUES (
        p_member_id, p_team_id, p_season_year, p_start_date, p_end_date,
            p_position,
        0, 0, 0, NULL, NULL
    );

    SET p_stat_id = v_current_stat_id;
    SET p_result_message = CONCAT('Member␣', p_member_id, '␣successfully␣assigned␣
        to␣team␣', p_team_id,
                                   '␣for␣season␣', p_season_year, '␣as␣',
                                       p_position);

    COMMIT;
END //
DELIMITER ;
```

10. InsertPlayerFormation

10. InsertPlayerFormation

```
DELIMITER //
CREATE  PROCEDURE InsertPlayerFormation(
    IN p_session_id INT,
    IN p_member_id INT,
    IN p_alternate_position ENUM('Setter','Outside␣Hitter','Opposite␣
       Hitter','Middle␣Blocker','Libero','Defensive␣Specialist','Serving␣
       Specialist'),
    IN p_start_time DATETIME,
    IN p_end_time DATETIME,
    OUT p_formation_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_session_exists INT DEFAULT 0;
    DECLARE v_member_exists INT DEFAULT 0;
    DECLARE v_team_id INT DEFAULT NULL;
    DECLARE v_regular_position VARCHAR(50) DEFAULT NULL;
    DECLARE v_stat_id INT DEFAULT NULL;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_formation_id = -1;
    END;

    START TRANSACTION;

    -- Validate required fields
    IF p_session_id IS NULL THEN
        SET p_formation_id = -1;
        SET p_result_message = 'Session␣ID␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_member_id IS NULL THEN
        SET p_formation_id = -1;
        SET p_result_message = 'Member␣ID␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate session exists
    SELECT COUNT(*) INTO v_session_exists
    FROM TeamSessions
    WHERE sessionID = p_session_id;

    IF v_session_exists = 0 THEN
        SET p_formation_id = -1;
        SET p_result_message = CONCAT('Session␣with␣ID␣', p_session_id, '␣does␣
           not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate member exists and is active
    SELECT COUNT(*) INTO v_member_exists
    FROM ClubMembers
    WHERE memberID = p_member_id AND isActive = 1;

    IF v_member_exists = 0 THEN
        SET p_formation_id = -1;
        SET p_result_message = CONCAT('Member␣with␣ID␣', p_member_id, '␣does␣not␣
           exist␣or␣is␣inactive');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Get member's team and regular position from current stats
    SELECT ms.teamID, ms.position INTO v_team_id, v_regular_position
    FROM MemberStats ms
    WHERE ms.memberID = p_member_id
      AND (ms.endDate IS NULL OR ms.endDate >= CURDATE())
    ORDER BY ms.startDate DESC
    LIMIT 1;
```

```
IF v_team_id IS NULL THEN
    SET p_formation_id = -1;
    SET p_result_message = CONCAT('Member␣with␣ID␣', p_member_id, '␣is␣not␣
        currently␣assigned␣to␣any␣team');
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Check if member is already in this session's formation
IF EXISTS (
    SELECT 1 FROM TeamFormations
    WHERE sessionID = p_session_id AND statID IN (
        SELECT statID FROM MemberStats WHERE memberID = p_member_id
    )
) THEN
    SET p_formation_id = -1;
    SET p_result_message = CONCAT('Member␣with␣ID␣', p_member_id, '␣is␣
        already␣in␣this␣session␣formation');
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Get the member's current stat record
SELECT statID INTO v_stat_id
FROM MemberStats
WHERE memberID = p_member_id
  AND (endDate IS NULL OR endDate >= CURDATE())
ORDER BY startDate DESC
LIMIT 1;

-- Validate time logic if both times provided
IF p_start_time IS NOT NULL AND p_end_time IS NOT NULL AND p_end_time <=
    p_start_time THEN
    SET p_formation_id = -1;
    SET p_result_message = 'End␣time␣must␣be␣after␣start␣time␣if␣both␣are␣
        provided';
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Determine position to use (alternate or regular)
SET @position_to_use = COALESCE(p_alternate_position, v_regular_position);

IF @position_to_use IS NULL THEN
    SET p_formation_id = -1;
    SET p_result_message = 'No␣position␣specified␣and␣member␣has␣no␣regular␣
        position␣assigned';
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Insert the formation record
INSERT INTO TeamFormations (
    statID, sessionID, memberStartDate, startTime, endTime, alternatePosition
)
VALUES (
    v_stat_id, p_session_id, CURDATE(), p_start_time, p_end_time,
    CASE WHEN p_alternate_position IS NULL THEN NULL ELSE
        p_alternate_position END
    );

    SET p_formation_id = LAST_INSERT_ID();

    SET p_result_message = CONCAT('Member␣', p_member_id, '␣added␣to␣session␣',
       p_session_id, '␣as␣',
                                IF(p_alternate_position IS NULL,
                                   CONCAT('regular␣', v_regular_position),
                                   CONCAT('alternate␣', p_alternate_position)));

    COMMIT;
END //
DELIMITER ;
```

11. InsertPayment

11. InsertPayment

```
DELIMITER //
CREATE  PROCEDURE InsertPayment(
    IN p_member_id INT,
    IN p_membership_year YEAR,
    IN p_payment_date DATE,
    IN p_amount DECIMAL(7,2),
    IN p_payment_method ENUM('Cash','Debit','Credit','Cheque','Other'),
    OUT p_payment_number INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_member_exists INT DEFAULT 0;
    DECLARE v_payment_number INT DEFAULT NULL;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_payment_number = -1;
    END;

    START TRANSACTION;

    -- Validate input parameters
    IF p_member_id IS NULL OR p_member_id <= 0 THEN
        SET p_payment_number = -1;
        SET p_result_message = 'Invalid␣member␣ID';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_membership_year IS NULL THEN
        SET p_payment_number = -1;
        SET p_result_message = 'Membership␣year␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_payment_date IS NULL THEN
        SET p_payment_number = -1;
        SET p_result_message = 'Payment␣date␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_amount IS NULL OR p_amount <= 0 THEN
        SET p_payment_number = -1;
        SET p_result_message = 'Payment␣amount␣must␣be␣greater␣than␣0';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_payment_method IS NULL THEN
        SET p_payment_number = -1;
        SET p_result_message = 'Payment␣method␣is␣required';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if member exists
    SELECT COUNT(*) INTO v_member_exists
    FROM ClubMembers
    WHERE memberID = p_member_id;

    IF v_member_exists = 0 THEN
        SET p_payment_number = -1;
        SET p_result_message = 'Member␣ID␣does␣not␣exist';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Insert payment (paymentNumber will be handled by trigger)
    INSERT INTO Payments (memberID, membershipYear, paymentDate, amount,
        paymentMethod)
    VALUES (p_member_id, p_membership_year, p_payment_date, p_amount,
        p_payment_method);

    -- Get the payment number that was assigned by the trigger
    SELECT paymentNumber INTO v_payment_number
    FROM Payments
    WHERE memberID = p_member_id
    AND membershipYear = p_membership_year
    AND paymentDate = p_payment_date
    AND amount = p_amount
    AND paymentMethod = p_payment_method
    ORDER BY paymentNumber DESC
    LIMIT 1;

    SET p_payment_number = v_payment_number;
    SET p_result_message = 'Payment␣inserted␣successfully';

    COMMIT;
END //
DELIMITER ;
```

12A. InsertFamily

12A. InsertFamily

```
DELIMITER //
CREATE  PROCEDURE InsertFamily(
    IN p_minor_id INT,
    IN p_major_id INT,
    IN p_location_id INT,
    IN p_relationship VARCHAR(50),
    IN p_contact_role ENUM('Primary','Emergency'),
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_family_member_id INT;
    DECLARE v_family_member_message VARCHAR(255);
    DECLARE v_relationship_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Step 1: Insert the major as a family member
    CALL InsertFamilyMember(p_major_id, p_location_id, v_family_member_id,
        v_family_member_message);

    -- Check for errors from the first sub-procedure
    IF v_family_member_id = -1 THEN
        SET p_result_message = CONCAT('Failed␣to␣insert␣family␣member:␣',
            v_family_member_message);
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Step 2: Insert the relationship between the minor and the new major
    CALL InsertFamilyRelationship(p_minor_id, p_major_id, p_relationship,
        p_contact_role, v_relationship_message);

    -- Check for errors from the second sub-procedure
    IF v_relationship_message NOT LIKE 'Family␣relationship␣created/updated␣
        successfully%' THEN
        SET p_result_message = CONCAT('Family␣member␣inserted␣successfully,␣but␣
            failed␣to␣create␣relationship:␣', v_relationship_message);
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- If both calls succeeded
    SET p_result_message = CONCAT('New␣family␣member␣(', p_major_id, ')␣and␣
        relationship␣created␣successfully:␣', v_relationship_message);
    COMMIT;
END //
DELIMITER ;
```

12B. InsertFamilyMember

12A. InsertFamily

```
DELIMITER //
CREATE  PROCEDURE InsertFamilyMember(
    IN p_person_id INT,
    IN p_location_id INT,
    OUT p_family_member_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_person_exists INT DEFAULT 0;
    DECLARE v_location_exists INT DEFAULT 0;
    DECLARE v_is_family_member INT DEFAULT 0;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_family_member_id = -1;
    END;

    START TRANSACTION;

    -- Check if the person exists
    SELECT COUNT(*) INTO v_person_exists FROM Person WHERE personID = p_person_id;
    IF v_person_exists = 0 THEN
        SET p_family_member_id = -1;
        SET p_result_message = CONCAT('Person␣with␣ID␣', p_person_id, '␣does␣not␣
            exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if the location exists
    SELECT COUNT(*) INTO v_location_exists FROM Locations WHERE locationID =
        p_location_id;
    IF v_location_exists = 0 THEN
        SET p_family_member_id = -1;
        SET p_result_message = CONCAT('Location␣with␣ID␣', p_location_id, '␣does␣
            not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if the person is already a family member
    SELECT COUNT(*) INTO v_is_family_member FROM FamilyMembers WHERE
        familyMemberID = p_person_id;
    IF v_is_family_member > 0 THEN
        SET p_family_member_id = p_person_id;
        SET p_result_message = CONCAT('Person␣with␣ID␣', p_person_id, '␣is␣
            already␣a␣family␣member');
        COMMIT;
        LEAVE proc_label;
    END IF;

    -- Insert the new family member
    INSERT INTO FamilyMembers (familyMemberID, locationID)
    VALUES (p_person_id, p_location_id);
    SET p_family_member_id = p_person_id;
    SET p_result_message = 'Family␣member␣inserted␣successfully';
    COMMIT;
END //
DELIMITER ;
```

12C. InsertFamilyRelationship

12C. InsertFamilyRelationship

```
DELIMITER //
CREATE  PROCEDURE InsertFamilyRelationship(
    IN p_minor_id INT,
    IN p_major_id INT,
    IN p_relationship VARCHAR(50),
    IN p_contact_role ENUM('Primary','Emergency'),
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_minor_exists INT DEFAULT 0;
    DECLARE v_major_is_family_member INT DEFAULT 0;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_minor_id IS NULL OR p_major_id IS NULL THEN
        SET p_result_message = 'Minor␣and␣Major␣IDs␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_minor_id = p_major_id THEN
        SET p_result_message = 'A␣person␣cannot␣be␣a␣major␣for␣themselves';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate ENUM values
    IF p_relationship IS NOT NULL AND NOT FIND_IN_SET(p_relationship,
       'Father,Mother,Grandfather,Grandmother,Sibling,Tutor,Partner,Friend,Other')
       > 0 THEN
        SET p_result_message = CONCAT('Invalid␣relationship␣type:␣',
            p_relationship);
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_contact_role IS NOT NULL AND NOT FIND_IN_SET(p_contact_role,
       'Primary,Emergency') > 0 THEN
        SET p_result_message = CONCAT('Invalid␣contact␣role:␣', p_contact_role);
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if the minor person exists
    SELECT COUNT(*) INTO v_minor_exists FROM Person WHERE personID = p_minor_id;
    IF v_minor_exists = 0 THEN
        SET p_result_message = CONCAT('Minor␣with␣ID␣', p_minor_id, '␣does␣not␣
            exist␣as␣a␣person.');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if the major exists in the FamilyMembers table
    SELECT COUNT(*) INTO v_major_is_family_member FROM FamilyMembers WHERE
       familyMemberID = p_major_id;
    IF v_major_is_family_member = 0 THEN
        SET p_result_message = CONCAT('Major␣with␣ID␣', p_major_id, '␣is␣not␣a␣
            family␣member.␣Please␣add␣them␣first.');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Insert or update the relationship
    INSERT INTO FamilyRelationship (minorID, majorID, relationship, contactRole)
    VALUES (p_minor_id, p_major_id, p_relationship, p_contact_role)
    ON DUPLICATE KEY UPDATE
        relationship = VALUES(relationship),
        contactRole = VALUES(contactRole);

    SET p_result_message = CONCAT('Family␣relationship␣created/updated␣
       successfully␣between␣Minor␣ID␣', p_minor_id,
                                 '␣and␣Major␣ID␣', p_major_id);
    COMMIT;
END //
DELIMITER ;
```

Deletion Procedures

1. DeleteAddress

1. DeleteAddress

```
DELIMITER //
CREATE  PROCEDURE DeleteAddress(
    IN p_address_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_address_exists INT DEFAULT 0;
    DECLARE v_reference_count INT DEFAULT 0;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;
    START TRANSACTION;
    -- Validate input
    IF p_address_id IS NULL THEN
        SET p_result_message = 'Address␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if address exists
    SELECT COUNT(*) INTO v_address_exists
    FROM Address
    WHERE addressID = p_address_id;

    IF v_address_exists = 0 THEN
        SET p_result_message = CONCAT('Address␣with␣ID␣', p_address_id, '␣does␣
            not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check for references in Person table
    SELECT COUNT(*) INTO v_reference_count
    FROM Person
    WHERE addressID = p_address_id;

    IF v_reference_count > 0 THEN
        SET p_result_message = CONCAT('Cannot␣delete␣-␣Address␣is␣referenced␣by␣
            ', v_reference_count, '␣person␣records');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check for references in Locations table
    SELECT COUNT(*) INTO v_reference_count
    FROM Locations
    WHERE addressID = p_address_id;

    IF v_reference_count > 0 THEN
        SET p_result_message = CONCAT('Cannot␣delete␣-␣Address␣is␣referenced␣by␣
            ', v_reference_count, '␣location␣records');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- If we get here, safe to delete
    DELETE FROM Address WHERE addressID = p_address_id;

    SET p_result_message = CONCAT('Address␣with␣ID␣', p_address_id, '␣deleted␣
        successfully');
    COMMIT;
END //
DELIMITER ;
```

2. DeletePhone

2. DeletePhone

```
DELIMITER //
CREATE  PROCEDURE DeletePhone(
    IN p_phone_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_phone_exists INT DEFAULT 0;
    DECLARE v_reference_count INT DEFAULT 0;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_phone_id IS NULL THEN
        SET p_result_message = 'Phone␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if phone exists
    SELECT COUNT(*) INTO v_phone_exists
    FROM PhoneNumbers
    WHERE phoneID = p_phone_id;

    IF v_phone_exists = 0 THEN
        SET p_result_message = CONCAT('Phone␣with␣ID␣', p_phone_id, '␣does␣not␣
            exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check for references in PersonPhones
    SELECT COUNT(*) INTO v_reference_count
    FROM PersonPhones
    WHERE phoneID = p_phone_id;

    -- Check for references in LocationPhones
    SELECT COUNT(*) INTO v_reference_count
    FROM LocationPhones
    WHERE phoneID = p_phone_id;

    IF v_reference_count > 0 THEN
        SET p_result_message = CONCAT('Cannot␣delete␣-␣Phone␣is␣referenced␣by␣',
            v_reference_count, '␣records');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- If we get here, safe to delete
    DELETE FROM PhoneNumbers WHERE phoneID = p_phone_id;

    SET p_result_message = CONCAT('Phone␣with␣ID␣', p_phone_id, '␣deleted␣
        successfully');
    COMMIT;
END //
DELIMITER ;
```

3. DeleteLocation

3. DeleteLocation

```
DELIMITER //
CREATE  PROCEDURE DeleteLocation(
    IN p_location_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_address_id INT DEFAULT NULL;
    DECLARE v_phone_id INT DEFAULT NULL;
    DECLARE v_done INT DEFAULT FALSE;
    DECLARE v_phone_message VARCHAR(255) DEFAULT '';

    -- Cursor for all phones associated with this location
    DECLARE phone_cursor CURSOR FOR
        SELECT phoneID FROM LocationPhones WHERE locationID = p_location_id;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_done = TRUE;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;

        -- User-friendly constraint messages
        IF p_result_message LIKE '%a foreign key constraint fails%' THEN
            IF p_result_message LIKE '%TeamSessions%' THEN
                SET p_result_message = 'Cannot delete - Location is referenced by 
                    team sessions';
            ELSEIF p_result_message LIKE '%Teams%' THEN
                SET p_result_message = 'Cannot delete - Location is referenced by 
                    teams';
            ELSEIF p_result_message LIKE '%MemberLocations%' THEN
                SET p_result_message = 'Cannot delete - Location is referenced by 
                    member assignments';
            ELSEIF p_result_message LIKE '%FamilyMembers%' THEN
                SET p_result_message = 'Cannot delete - Location is referenced by 
                    family members';
            ELSEIF p_result_message LIKE '%PersonnelLocations%' THEN
                SET p_result_message = 'Cannot delete - Location is referenced by 
                    personnel assignments';
            ELSE
                SET p_result_message = 'Cannot delete - Location is referenced by 
                    other records';
            END IF;
        END IF;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_location_id IS NULL THEN
        SET p_result_message = 'Location ID cannot be NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;
```

```
-- Check if location exists and get its address
SELECT addressID INTO v_address_id
FROM Locations
WHERE locationID = p_location_id
LIMIT 1;

IF v_address_id IS NULL THEN
    SET p_result_message = CONCAT('Location␣with␣ID␣', p_location_id, '␣does␣
       not␣exist');
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Process all phone numbers associated with this location
OPEN phone_cursor;
phone_loop: LOOP
    FETCH phone_cursor INTO v_phone_id;
    IF v_done THEN
        LEAVE phone_loop;
    END IF;

    -- First remove the location-phone relationship
    DELETE FROM LocationPhones
    WHERE locationID = p_location_id AND phoneID = v_phone_id;

    -- Then try to delete the phone (will fail if still referenced elsewhere)
    CALL DeletePhone(v_phone_id, @phone_delete_result);

    IF @phone_delete_result NOT LIKE 'Phone␣with␣ID%deleted␣successfully' THEN
        -- Collect phone deletion messages without failing
        SET v_phone_message = CONCAT(v_phone_message, '␣Phone␣', v_phone_id,
           ':␣', @phone_delete_result, ';');
    END IF;
END LOOP;
CLOSE phone_cursor;

-- Now delete the location (let foreign keys handle constraints)
DELETE FROM Locations WHERE locationID = p_location_id;

-- Delete the associated address
CALL DeleteAddress(v_address_id, @address_delete_result);

IF @address_delete_result NOT LIKE 'Address␣with␣ID%deleted␣successfully' THEN
    SET p_result_message = CONCAT('Location␣deleted␣but␣address␣cleanup␣
       failed:␣', @address_delete_result);
    IF LENGTH(v_phone_message) > 0 THEN
        SET p_result_message = CONCAT(p_result_message, '␣Phone␣issues:␣',
           v_phone_message);
    END IF;
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Success message with any phone cleanup notes
SET p_result_message = CONCAT('Location␣with␣ID␣', p_location_id, '␣deleted␣
   successfully');
IF LENGTH(v_phone_message) > 0 THEN
    SET p_result_message = CONCAT(p_result_message, '␣(Note:␣',
       v_phone_message, ')');
    END IF;

    COMMIT;
END //
DELIMITER ;
```

4. DeletePerson

4. DeletePerson

```
DELIMITER //
CREATE  PROCEDURE DeletePerson(
    IN p_person_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_address_id INT DEFAULT NULL;
    DECLARE v_phone_id INT DEFAULT NULL;
    DECLARE v_done INT DEFAULT FALSE;
    DECLARE v_phone_message VARCHAR(255) DEFAULT '';
    DECLARE v_is_club_member INT DEFAULT 0;
    DECLARE v_is_family_member INT DEFAULT 0;
    DECLARE v_is_personnel INT DEFAULT 0;

    -- Cursor for all phones associated with this person
    DECLARE phone_cursor CURSOR FOR
        SELECT phoneID FROM PersonPhones WHERE personID = p_person_id;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_done = TRUE;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;

        -- User-friendly constraint messages
        IF p_result_message LIKE '%a␣foreign␣key␣constraint␣fails%' THEN
            IF p_result_message LIKE '%ClubMembers%' THEN
                SET p_result_message = 'Cannot␣delete␣-␣Person␣is␣a␣club␣member␣
                    (delete␣from␣ClubMembers␣first)';
            ELSEIF p_result_message LIKE '%FamilyMembers%' THEN
                SET p_result_message = 'Cannot␣delete␣-␣Person␣is␣a␣family␣member␣
                    (delete␣from␣FamilyMembers␣first)';
            ELSEIF p_result_message LIKE '%PersonnelLocations%' THEN
                SET p_result_message = 'Cannot␣delete␣-␣Person␣has␣personnel␣
                    assignments';
            ELSEIF p_result_message LIKE '%FamilyRelationship%' THEN
                SET p_result_message = 'Cannot␣delete␣-␣Person␣is␣referenced␣in␣
                    family␣relationships';
            ELSEIF p_result_message LIKE '%MemberStats%' THEN
                SET p_result_message = 'Cannot␣delete␣-␣Person␣has␣member␣
                    statistics␣records';
            ELSE
                SET p_result_message = 'Cannot␣delete␣-␣Person␣is␣referenced␣by␣
                    other␣records';
            END IF;
        END IF;
    END;
```

```
START TRANSACTION;

-- Validate input
IF p_person_id IS NULL THEN
    SET p_result_message = ’Person␣ID␣cannot␣be␣NULL’;
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Check if person exists and get their address
SELECT addressID INTO v_address_id
FROM Person
WHERE personID = p_person_id
LIMIT 1;

IF v_address_id IS NULL THEN
    SET p_result_message = CONCAT(’Person␣with␣ID␣’, p_person_id, ’␣does␣not␣
        exist’);
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Check for special roles that need to be handled first
SELECT COUNT(*) INTO v_is_club_member FROM ClubMembers WHERE memberID =
    p_person_id;
SELECT COUNT(*) INTO v_is_family_member FROM FamilyMembers WHERE
    familyMemberID = p_person_id;
SELECT COUNT(*) INTO v_is_personnel FROM PersonnelLocations WHERE personnelID
    = p_person_id;

IF v_is_club_member > 0 THEN
    SET p_result_message = ’Cannot␣delete␣-␣Person␣is␣a␣club␣member␣(delete␣
        from␣ClubMembers␣first)’;
    ROLLBACK;
    LEAVE proc_label;
END IF;

IF v_is_family_member > 0 THEN
    SET p_result_message = ’Cannot␣delete␣-␣Person␣is␣a␣family␣member␣(delete␣
        from␣FamilyMembers␣first)’;
    ROLLBACK;
    LEAVE proc_label;
END IF;

IF v_is_personnel > 0 THEN
    SET p_result_message = ’Cannot␣delete␣-␣Person␣has␣personnel␣assignments’;
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Process all phone numbers associated with this person
OPEN phone_cursor;
phone_loop: LOOP
    FETCH phone_cursor INTO v_phone_id;
    IF v_done THEN
        LEAVE phone_loop;
    END IF;
```

```
        -- First remove the person-phone relationship
        DELETE FROM PersonPhones
        WHERE personID = p_person_id AND phoneID = v_phone_id;

        -- Then try to delete the phone (will fail if still referenced elsewhere)
        CALL DeletePhone(v_phone_id, @phone_delete_result);

        IF @phone_delete_result NOT LIKE 'Phone␣with␣ID%deleted␣successfully' THEN
            -- Collect phone deletion messages without failing
            SET v_phone_message = CONCAT(v_phone_message, '␣Phone␣', v_phone_id,
                ':␣', @phone_delete_result, ';');
        END IF;
    END LOOP;
    CLOSE phone_cursor;

    -- Now delete the person record
    DELETE FROM Person WHERE personID = p_person_id;

    -- Delete the associated address
    CALL DeleteAddress(v_address_id, @address_delete_result);

    IF @address_delete_result NOT LIKE 'Address␣with␣ID%deleted␣successfully' THEN
        SET p_result_message = CONCAT('Person␣deleted␣but␣address␣cleanup␣failed:␣
            ', @address_delete_result);
        IF LENGTH(v_phone_message) > 0 THEN
            SET p_result_message = CONCAT(p_result_message, '␣Phone␣issues:␣',
                v_phone_message);
        END IF;
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Success message with any phone cleanup notes
    SET p_result_message = CONCAT('Person␣with␣ID␣', p_person_id, '␣deleted␣
        successfully');
    IF LENGTH(v_phone_message) > 0 THEN
        SET p_result_message = CONCAT(p_result_message, '␣(Note:␣',
            v_phone_message, ')');
    END IF;

    COMMIT;
END //
DELIMITER ;
```

5. DeleteFamilyMember

5. DeleteFamilyMember

```
DELIMITER //
CREATE  PROCEDURE DeleteFamilyMember(
    IN p_family_member_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_is_minor INT DEFAULT 0;
    DECLARE v_has_minors INT DEFAULT 0;
    DECLARE v_person_id INT DEFAULT NULL;
    DECLARE v_location_id INT DEFAULT NULL;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_family_member_id IS NULL THEN
        SET p_result_message = 'Family␣Member␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if this is actually a family member
    SELECT COUNT(*) INTO v_person_id
    FROM FamilyMembers
    WHERE familyMemberID = p_family_member_id;

    IF v_person_id = 0 THEN
        SET p_result_message = CONCAT('Person␣with␣ID␣', p_family_member_id, '␣is␣
            not␣a␣family␣member');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if this family member is a minor in any relationships
    SELECT COUNT(*) INTO v_is_minor
    FROM FamilyRelationship
    WHERE minorID = p_family_member_id;

    IF v_is_minor > 0 THEN
        SET p_result_message = CONCAT('Cannot␣delete␣-␣Person␣with␣ID␣',
            p_family_member_id,
                                      '␣is␣a␣minor␣in␣family␣relationships.␣Delete␣
                                        them␣from␣ClubMembers␣first.');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if this family member is the sole major for any minors
    SELECT COUNT(*) INTO v_has_minors
    FROM FamilyRelationship fr
    WHERE fr.majorID = p_family_member_id
    AND NOT EXISTS (
        SELECT 1 FROM FamilyRelationship fr2
        WHERE fr2.minorID = fr.minorID
        AND fr2.majorID != p_family_member_id
        AND fr2.contactRole = 'Primary'
    );

    IF v_has_minors > 0 THEN
        SET p_result_message = CONCAT('Cannot␣delete␣-␣Person␣with␣ID␣',
            p_family_member_id,
                                      '␣is␣the␣sole␣primary␣contact␣for␣',
                                        v_has_minors,
                                      '␣minor(s).␣Either␣delete␣the␣minor(s)␣first␣
                                         or␣assign␣a␣new␣primary␣contact.');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Get location ID before deletion for cleanup
    SELECT locationID INTO v_location_id
    FROM FamilyMembers
    WHERE familyMemberID = p_family_member_id;

    -- Delete from FamilyMembers first
    DELETE FROM FamilyMembers WHERE familyMemberID = p_family_member_id;

    -- Delete any remaining relationships where they were a major
    DELETE FROM FamilyRelationship WHERE majorID = p_family_member_id;

    -- Now try to delete the person record
    CALL DeletePerson(p_family_member_id, @person_delete_result);

    IF @person_delete_result NOT LIKE 'Person␣with␣ID%deleted␣successfully' THEN
        -- Person deletion failed, but family member record is already gone
        SET p_result_message = CONCAT('Family␣member␣record␣removed␣but␣person␣
           deletion␣failed:␣',
                                     @person_delete_result);
        COMMIT;
        LEAVE proc_label;
    END IF;

    -- If location was exclusively used by this family member, delete it
    IF v_location_id IS NOT NULL THEN
        IF NOT EXISTS (
            SELECT 1 FROM FamilyMembers
            WHERE locationID = v_location_id
            AND familyMemberID != p_family_member_id
        ) THEN
            CALL DeleteAddress(v_location_id, @address_delete_result);
            -- Address deletion success/failure doesn't affect our overall success
        END IF;
    END IF;

    SET p_result_message = CONCAT('Family␣member␣with␣ID␣', p_family_member_id, '␣
       deleted␣successfully');
    COMMIT;
END //
DELIMITER ;
```

6A. DeleteClubMember

6. DeleteClubMember

```
DELIMITER //
CREATE  PROCEDURE DeleteClubMember(
    IN p_member_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_member_type VARCHAR(10) DEFAULT NULL;
    DECLARE v_has_minors INT DEFAULT 0;
    DECLARE v_family_member_id INT DEFAULT NULL;
    DECLARE v_location_id INT DEFAULT NULL;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_result_message = CONCAT('Error␣during␣deletion:␣',
           p_result_message);
    END;

    START TRANSACTION;

    -- Validate input
    IF p_member_id IS NULL THEN
        SET p_result_message = 'Member␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Get member type and verify existence
    SELECT memberType INTO v_member_type
    FROM ClubMembers
    WHERE memberID = p_member_id;

    IF v_member_type IS NULL THEN
        SET p_result_message = CONCAT('Club␣member␣with␣ID␣', p_member_id, '␣does␣
           not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Get family member info if exists
    SELECT familyMemberID, locationID
    INTO v_family_member_id, v_location_id
    FROM FamilyMembers
    WHERE familyMemberID = p_member_id;

    -- Delete member-related records that have RESTRICT foreign key constraints
       first
    -- Delete TeamFormations that reference this member's stats
    DELETE tf FROM TeamFormations tf
    JOIN MemberStats ms ON tf.statID = ms.statID
    WHERE ms.memberID = p_member_id;

    DELETE FROM Payments WHERE memberID = p_member_id;
    DELETE FROM MemberStats WHERE memberID = p_member_id;
    DELETE FROM MemberLocations WHERE memberID = p_member_id;

    -- Handle different member types and relationship constraints
    IF v_member_type = 'Minor' THEN
        -- For minors, delete from ClubMembers (relationships will CASCADE)
        DELETE FROM ClubMembers WHERE memberID = p_member_id;

    ELSE -- Major member
        -- Check if this major is the sole primary contact for any minors
        SELECT COUNT(*) INTO v_has_minors
        FROM FamilyRelationship fr
        WHERE fr.majorID = p_member_id
        AND fr.contactRole = 'Primary'
        AND NOT EXISTS (
            SELECT 1 FROM FamilyRelationship fr2
            WHERE fr2.minorID = fr.minorID
            AND fr2.majorID != p_member_id
            AND fr2.contactRole = 'Primary'
        );

        IF v_has_minors > 0 THEN
            SET p_result_message = CONCAT('Cannot␣delete␣-␣Member␣with␣ID␣',
                p_member_id,
                                        '␣is␣the␣sole␣primary␣contact␣for␣',
                                           v_has_minors,
                                        '␣minor(s).␣Assign␣a␣new␣primary␣contact␣
                                           first.');
            ROLLBACK;
            LEAVE proc_label;
        END IF;

        -- For majors, delete from ClubMembers (relationships will CASCADE)
        DELETE FROM ClubMembers WHERE memberID = p_member_id;
    END IF;

    -- If they were a family member, handle family-specific cleanup
    IF v_family_member_id IS NOT NULL THEN
        DELETE FROM FamilyMembers WHERE familyMemberID = p_member_id;

        -- Check if location should be deleted (if no other family members use it)
        IF v_location_id IS NOT NULL THEN
            IF NOT EXISTS (
                SELECT 1 FROM FamilyMembers
                WHERE locationID = v_location_id
            ) THEN
                CALL DeleteLocation(v_location_id, @location_delete_result);
            END IF;
        END IF;
    END IF;

    -- Finally try to delete the person record
    CALL DeletePerson(p_member_id, @person_delete_result);

    IF @person_delete_result NOT LIKE 'Person␣with␣ID%deleted␣successfully' THEN
        IF @person_delete_result LIKE '%personnel␣assignments%' OR
           @person_delete_result LIKE '%family␣member%' THEN
            SET p_result_message = CONCAT('Club␣member␣with␣ID␣', p_member_id, '␣
                (', v_member_type,
                                        ')␣removed␣from␣club␣but␣person␣record␣
                                           retained␣due␣to␣other␣roles');
        ELSE
            SET p_result_message = CONCAT('Club␣member␣record␣removed␣but␣person␣
                deletion␣failed:␣',
                                        @person_delete_result);
        END IF;
        COMMIT;
        LEAVE proc_label;
    END IF;
    SET p_result_message = CONCAT('Club␣member␣with␣ID␣', p_member_id, '␣(',
        v_member_type,
                                  ')␣deleted␣successfully');
    COMMIT;
END //
DELIMITER ;
```

6B. UpdateClubMemberComplete

6B. UpdateClubMemberComplete

```
DELIMITER //
CREATE PROCEDURE UpdateClubMemberComplete(
    IN p_club_member_id INT,
    -- Person fields
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(50),
    IN p_middle_initial CHAR(1),
    IN p_date_of_birth DATE,
    IN p_ssn CHAR(9),
    IN p_medicare VARCHAR(20),
    IN p_email VARCHAR(100),
    -- Address fields
    IN p_street_address VARCHAR(255),
    IN p_city VARCHAR(100),
    IN p_province VARCHAR(50),
    IN p_postal_code CHAR(6),
    -- Phone fields
    IN p_phone_number VARCHAR(20),
    IN p_phone_extension VARCHAR(10),
    IN p_phone_type VARCHAR(20),
    -- Member fields
    IN p_height DECIMAL(5,2),
    IN p_weight DECIMAL(5,2),
    IN p_gender ENUM('Male','Female'),
    IN p_is_active BOOLEAN,
    -- Family relationship fields
    IN p_location_id INT,
    IN p_family_id INT,
    IN p_relationship VARCHAR(50),
    IN p_contact_role ENUM('Primary','Secondary','Emergency'),
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_person_id INT;
    DECLARE v_address_id INT;
    DECLARE v_phone_id INT;
    DECLARE v_new_address_id INT;
    DECLARE v_new_phone_id INT;
    DECLARE v_address_message VARCHAR(255);
    DECLARE v_phone_message VARCHAR(255);
    DECLARE v_member_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;
```

```
START TRANSACTION;

-- Get club member details
SELECT p.personID, p.addressID INTO v_person_id, v_address_id
FROM ClubMembers cm
JOIN Person p ON cm.personID = p.personID
WHERE cm.memberID = p_club_member_id;

IF v_person_id IS NULL THEN
    SET p_result_message = CONCAT('Club␣Member␣with␣ID␣', p_club_member_id, '␣
        does␣not␣exist');
    ROLLBACK;
    LEAVE proc_label;
END IF;

-- Get phone ID
SELECT phoneID INTO v_phone_id FROM PersonPhones WHERE personID = v_person_id
    LIMIT 1;

-- Update person details directly (simple fields)
IF p_first_name IS NOT NULL OR p_last_name IS NOT NULL OR p_middle_initial IS
    NOT NULL OR p_date_of_birth IS NOT NULL OR p_ssn IS NOT NULL OR p_medicare
    IS NOT NULL OR p_email IS NOT NULL THEN
    UPDATE Person
    SET firstName = COALESCE(p_first_name, firstName),
        lastName = COALESCE(p_last_name, lastName),
        middleInitial = COALESCE(p_middle_initial, middleInitial),
        dateOfBirth = COALESCE(p_date_of_birth, dateOfBirth),
        ssn = COALESCE(p_ssn, ssn),
        medicare = COALESCE(p_medicare, medicare),
        email = COALESCE(p_email, email)
    WHERE personID = v_person_id;
END IF;

-- Handle address updates
IF p_street_address IS NOT NULL OR p_city IS NOT NULL OR p_province IS NOT
    NULL OR p_postal_code IS NOT NULL THEN
    IF v_address_id IS NOT NULL THEN
        -- Update existing address
        CALL UpdateAddress(v_address_id, p_street_address, p_city,
            p_province, p_postal_code, v_new_address_id, v_address_message);

        IF v_new_address_id = -1 THEN
            SET p_result_message = v_address_message;
            ROLLBACK;
            LEAVE proc_label;
        END IF;

        -- Update person's address reference if new address was created
        IF v_new_address_id != v_address_id THEN
            UPDATE Person SET addressID = v_new_address_id WHERE personID =
                v_person_id;
        END IF;
    ELSE
        -- Create new address and link to person
        CALL InsertAddress(p_street_address, p_city, p_province,
            p_postal_code, v_new_address_id, v_address_message);
        IF v_new_address_id = -1 THEN
            SET p_result_message = v_address_message;
            ROLLBACK;
            LEAVE proc_label;
        END IF;

        UPDATE Person SET addressID = v_new_address_id WHERE personID =
            v_person_id;
    END IF;
END IF;

-- Handle phone updates
IF p_phone_number IS NOT NULL OR p_phone_extension IS NOT NULL OR
   p_phone_type IS NOT NULL THEN
    IF v_phone_id IS NOT NULL THEN
        -- Update existing phone
        CALL UpdatePhone(v_phone_id, p_phone_number, p_phone_extension,
            p_phone_type, v_new_phone_id, v_phone_message);

        IF v_new_phone_id = -1 THEN
            SET p_result_message = v_phone_message;
            ROLLBACK;
            LEAVE proc_label;
        END IF;

        -- Update person phone reference if new phone was created
        IF v_new_phone_id != v_phone_id THEN
            UPDATE PersonPhones SET phoneID = v_new_phone_id WHERE personID =
                v_person_id AND phoneID = v_phone_id;
        END IF;
    ELSE
        -- Create new phone and link to person
        CALL InsertPhone(p_phone_number, p_phone_extension, p_phone_type,
            v_new_phone_id, v_phone_message);

        IF v_new_phone_id = -1 THEN
            SET p_result_message = v_phone_message;
            ROLLBACK;
            LEAVE proc_label;
        END IF;

        INSERT INTO PersonPhones (personID, phoneID) VALUES (v_person_id,
            v_new_phone_id);
    END IF;
END IF;

-- Update club member details using existing procedure
IF p_height IS NOT NULL OR p_weight IS NOT NULL OR p_gender IS NOT NULL OR
   p_is_active IS NOT NULL THEN
    CALL UpdateClubMember(p_club_member_id, p_height, p_weight, p_gender,
        p_is_active, v_member_message);
END IF;

-- Update family relationship and location directly if provided
IF p_location_id IS NOT NULL THEN
    UPDATE ClubMembers SET locationID = p_location_id WHERE memberID =
        p_club_member_id;
END IF;
IF p_family_id IS NOT NULL OR p_relationship IS NOT NULL OR p_contact_role IS
   NOT NULL THEN
    UPDATE FamilyMemberRelationships
    SET familyMemberID = COALESCE(p_family_id, familyMemberID),
        relationship = COALESCE(p_relationship, relationship),
        contactRole = COALESCE(p_contact_role, contactRole)
    WHERE clubMemberID = p_club_member_id;
END IF;

SET p_result_message = 'Club␣Member␣updated␣successfully';
COMMIT;
END //
DELIMITER ;
```

7A. DeletePersonnelAssignment

7A. DeletePersonnelAssignment

```
DELIMITER //
CREATE  PROCEDURE DeletePersonnelAssignment(
    IN p_assignment_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_person_id INT DEFAULT NULL;
    DECLARE v_role VARCHAR(50) DEFAULT NULL;
    DECLARE v_location_id INT DEFAULT NULL;
    DECLARE v_team_id INT DEFAULT NULL;
    DECLARE v_is_gm INT DEFAULT 0;
    DECLARE v_is_coach INT DEFAULT 0;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_assignment_id IS NULL THEN
        SET p_result_message = 'Assignment␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Get assignment details
    SELECT pl.personnelID, pl.role, pl.locationID, t.teamID
    INTO v_person_id, v_role, v_location_id, v_team_id
    FROM PersonnelLocations pl
    LEFT JOIN Teams t ON t.coachID = pl.assignmentID
    WHERE pl.assignmentID = p_assignment_id
    AND (pl.endDate IS NULL OR pl.endDate > CURDATE())
    LIMIT 1;

    IF v_person_id IS NULL THEN
        SET p_result_message = CONCAT('Active␣assignment␣with␣ID␣',
           p_assignment_id, '␣not␣found');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if this is a GM or Coach
    IF v_role = 'General␣Manager' THEN
        SET v_is_gm = 1;
    ELSEIF v_role IN ('Coach', 'Assistant␣Coach') THEN
        SET v_is_coach = 1;
    END IF;

    -- Handle role-specific cleanup
    IF v_is_gm = 1 AND v_location_id IS NOT NULL THEN
        UPDATE Locations
        SET generalManagerAssignmentID = NULL
        WHERE locationID = v_location_id
        AND generalManagerAssignmentID = p_assignment_id;
    END IF;

    IF v_is_coach = 1 AND v_team_id IS NOT NULL THEN
        UPDATE Teams
        SET coachID = NULL
        WHERE teamID = v_team_id
        AND coachID = p_assignment_id;
    END IF;

    -- Delete the specific assignment
    DELETE FROM PersonnelLocations WHERE assignmentID = p_assignment_id;

    SET p_result_message = CONCAT('Assignment␣ID␣', p_assignment_id, '␣terminated␣
        successfully');
    IF v_is_gm = 1 THEN
        SET p_result_message = CONCAT(p_result_message, '␣(GM␣role␣cleared)');
    END IF;
    IF v_is_coach = 1 THEN
        SET p_result_message = CONCAT(p_result_message, '␣(Coach␣role␣cleared)');
    END IF;

    COMMIT;
END //
DELIMITER ;
```

7B. DeletePersonnelCompletely

7B. DeletePersonnelCompletely

```
DELIMITER //
CREATE  PROCEDURE DeletePersonnelCompletely(
    IN p_person_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_done INT DEFAULT FALSE;
    DECLARE v_assignment_id INT DEFAULT NULL;
    DECLARE v_assignment_message VARCHAR(255) DEFAULT '';
    DECLARE assignment_cursor CURSOR FOR
        SELECT assignmentID FROM PersonnelLocations
        WHERE personnelID = p_person_id
        AND (endDate IS NULL OR endDate > CURDATE());
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_done = TRUE;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_person_id IS NULL THEN
        SET p_result_message = 'Person␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if person exists in personnel
    IF NOT EXISTS (SELECT 1 FROM PersonnelLocations WHERE personnelID =
       p_person_id) THEN
        SET p_result_message = CONCAT('Person␣with␣ID␣', p_person_id, '␣has␣no␣
           personnel␣records');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Process all active assignments
    OPEN assignment_cursor;
    assignment_loop: LOOP
        FETCH assignment_cursor INTO v_assignment_id;
        IF v_done THEN
            LEAVE assignment_loop;
        END IF;

        -- Delete each assignment using the surgical procedure
        CALL DeletePersonnelAssignment(v_assignment_id, v_assignment_message);

        -- Collect any error messages
        IF v_assignment_message NOT LIKE 'Assignment␣ID%terminated␣successfully%'
           THEN
            SET p_result_message = CONCAT(IFNULL(p_result_message, ''),
                '␣Assignment␣', v_assignment_id, ':␣', v_assignment_message, ';');
        END IF;
    END LOOP;
    CLOSE assignment_cursor;

    -- Now try to delete the person record
    CALL DeletePerson(p_person_id, @person_delete_result);

    IF @person_delete_result NOT LIKE 'Person␣with␣ID%deleted␣successfully' THEN
        SET p_result_message = CONCAT(IFNULL(p_result_message, ''),
            '␣Person␣deletion␣failed:␣', @person_delete_result);
        COMMIT;
        LEAVE proc_label;
    END IF;

    SET p_result_message = CONCAT('All␣personnel␣records␣for␣ID␣', p_person_id,
                                        '␣deleted␣successfully', IFNULL(p_result_message,
                                            ''));

    COMMIT;
END //
DELIMITER ;
```

8A. DeleteMemberFormation

8A. DeleteMemberLocation

```
DELIMITER //
CREATE  PROCEDURE DeleteMemberFormation(
    IN p_formation_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_formation_exists INT DEFAULT 0;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_formation_id IS NULL THEN
        SET p_result_message = 'Formation␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if the formation record exists
    SELECT COUNT(*) INTO v_formation_exists
    FROM TeamFormations
    WHERE formationID = p_formation_id;

    IF v_formation_exists = 0 THEN
        SET p_result_message = CONCAT('Formation␣record␣with␣ID␣',
           p_formation_id, '␣does␣not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Delete the formation record
    DELETE FROM TeamFormations WHERE formationID = p_formation_id;

    SET p_result_message = CONCAT('Formation␣record␣with␣ID␣', p_formation_id, '␣
        deleted␣successfully');
    COMMIT;
END //
DELIMITER ;
```

8B. DeleteTeamFormation

8. DeleteTeamFormation

```
DELIMITER //
CREATE  PROCEDURE DeleteTeamFormation(
    IN p_session_id INT,
    IN p_team_id INT,
    OUT p_rows_deleted INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_session_exists INT DEFAULT 0;
    DECLARE v_team_exists INT DEFAULT 0;
    DECLARE v_formation_id INT;
    DECLARE v_delete_result VARCHAR(255);
    DECLARE v_done INT DEFAULT FALSE;

    DECLARE formation_cursor CURSOR FOR
        SELECT tf.formationID
        FROM TeamFormations tf
        JOIN MemberStats ms ON tf.statID = ms.statID
        WHERE tf.sessionID = p_session_id
        AND ms.teamID = p_team_id;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_done = TRUE;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_rows_deleted = -1;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_session_id IS NULL THEN
        SET p_rows_deleted = -1;
        SET p_result_message = 'Session␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    IF p_team_id IS NULL THEN
        SET p_rows_deleted = -1;
        SET p_result_message = 'Team␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate session exists
    SELECT COUNT(*) INTO v_session_exists
    FROM TeamSessions
    WHERE sessionID = p_session_id;

    IF v_session_exists = 0 THEN
        SET p_rows_deleted = -1;
        SET p_result_message = CONCAT('Session␣with␣ID␣', p_session_id, '␣does␣
            not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate team exists
    SELECT COUNT(*) INTO v_team_exists
    FROM Teams
    WHERE teamID = p_team_id;

    IF v_team_exists = 0 THEN
        SET p_rows_deleted = -1;
        SET p_result_message = CONCAT('Team␣with␣ID␣', p_team_id, '␣does␣not␣
            exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    SET p_rows_deleted = 0;
    OPEN formation_cursor;
    formation_loop: LOOP
        FETCH formation_cursor INTO v_formation_id;
        IF v_done THEN
            LEAVE formation_loop;
        END IF;

        -- Call atomized DeleteMemberFormation
        CALL DeleteMemberFormation(v_formation_id, v_delete_result);

        -- Check the result of the call
        IF v_delete_result LIKE 'Formation␣record␣with␣ID%' THEN
            SET p_rows_deleted = p_rows_deleted + 1;
        ELSE
            -- In case of an error during a sub-call, we can log it but continue
            -- For this example, we'll just append the error message
            SET p_result_message = CONCAT(IFNULL(p_result_message, ''), '␣Error␣
                deleting␣formation␣ID␣', v_formation_id, ':␣', v_delete_result,
                ';');
        END IF;
    END LOOP;
    CLOSE formation_cursor;

    IF p_rows_deleted = 0 THEN
        SET p_result_message = CONCAT('No␣formation␣records␣found␣for␣team␣',
            p_team_id,
                                      '␣in␣session␣', p_session_id);
    ELSE
        SET p_result_message = CONCAT('Deleted␣', p_rows_deleted, '␣formation␣
            records␣for␣team␣',
                                      p_team_id, '␣in␣session␣', p_session_id);
    END IF;

    COMMIT;
END //
DELIMITER ;
```

Search Procedures

1. Location Searches

SearchLocationByID

```
DELIMITER //
CREATE  PROCEDURE SearchLocationByID(
  IN p_locationID INT
)
BEGIN
  SELECT
    l.locationID,
    a.addressID,
    a.streetAddress,
    a.city,
    a.province,
    a.postalCode,
    l.name AS locationName,
    l.type AS locationType,
    l.size,
    l.capacity,
    l.webAddress,
    pn.phoneID,
    pn.phoneNumber,
    pn.extension,
    pn.type AS phoneType,
    l.isActive
  FROM Locations l
  LEFT JOIN Address a ON l.addressID = a.addressID
  LEFT JOIN LocationPhones lp ON l.locationID = lp.locationID
  LEFT JOIN PhoneNumbers pn ON lp.phoneID = pn.phoneID
  WHERE l.locationID = p_locationID;
END //
DELIMITER ;
```

ShowAllLocations

```
DELIMITER //
CREATE  PROCEDURE ShowAllLocations()
BEGIN
  SELECT
    l.locationID,
    a.streetAddress,
    a.city,
    a.province,
    a.postalCode,
    l.name AS locationName,
    l.type AS locationType,
    l.size,
    l.capacity,
    l.webAddress,
    pn.phoneNumber AS phoneNumber,
    pn.extension,
    pn.type AS phoneType,
    l.isActive
  FROM Locations l
  LEFT JOIN Address a ON l.addressID = a.addressID
  LEFT JOIN LocationPhones lp ON l.locationID = lp.locationID
  LEFT JOIN PhoneNumbers pn ON lp.phoneID = pn.phoneID
  ORDER BY l.locationID;
END //
DELIMITER ;
```

2. Personnel Searches

SearchPersonnelByAssignmentID

```
DELIMITER //
CREATE  PROCEDURE SearchPersonnelByAssignmentID(IN p_assignmentID INT)
BEGIN
  SELECT
    pl.assignmentID,
    pl.personnelID,
    p.firstName,
    p.lastName,
    p.middleInitial,
    p.dateOfBirth,
    p.ssn,
    p.medicareNumber AS medicare,
    p.email,
    a.streetAddress,
    a.city,
    a.province,
    a.postalCode,
    pn.phoneNumber,
    pn.extension,
    pn.type AS phoneType,
    pl.locationID,
    pl.role,
    pl.mandateType,
    pl.startDate,
    pl.endDate
  FROM PersonnelLocations pl
  JOIN Person p ON pl.personnelID = p.personID
  LEFT JOIN Address a ON p.addressID = a.addressID
  LEFT JOIN PersonPhones pp ON p.personID = pp.personID
  LEFT JOIN PhoneNumbers pn ON pp.phoneID = pn.phoneID
  WHERE pl.assignmentID = p_assignmentID;
END //
DELIMITER ;
```

ShowAllPersonnelAssignments

```
DELIMITER //
CREATE  PROCEDURE ShowAllPersonnelAssignments()
BEGIN
  SELECT
    pl.assignmentID,
    pl.personnelID,
    p.firstName,
    p.lastName,
    p.dateOfBirth,
    p.email,
    a.streetAddress,
    a.city,
    a.province,
    a.postalCode,
    pn.phoneNumber,
    pn.extension,
    pn.type AS phoneType,
    pl.locationID,
    pl.role,
    pl.mandateType,
    pl.startDate,
    pl.endDate
  FROM PersonnelLocations pl
  JOIN Person p ON pl.personnelID = p.personID
  LEFT JOIN Address a ON p.addressID = a.addressID
  LEFT JOIN PersonPhones pp ON p.personID = pp.personID
  LEFT JOIN PhoneNumbers pn ON pp.phoneID = pn.phoneID
  ORDER BY pl.assignmentID;
END //
DELIMITER ;
```

3. Family Member Searches

SearchFamilyMemberByID

```
DELIMITER //
CREATE  PROCEDURE SearchFamilyMemberByID(IN p_familyMemberID INT)
BEGIN
  SELECT
    fm.familyMemberID,
    fm.locationID,
    p.firstName,
    p.lastName,
    p.middleInitial,
    p.dateOfBirth,
    p.ssn,
    p.medicareNumber AS medicare,
    p.email,
    a.streetAddress,
    a.city,
    a.province,
    a.postalCode,
    pn.phoneNumber,
    pn.extension,
    pn.type AS phoneType
  FROM FamilyMembers fm
  JOIN Person p ON fm.familyMemberID = p.personID
  LEFT JOIN Address a ON p.addressID = a.addressID
  LEFT JOIN PersonPhones pp ON p.personID = pp.personID
  LEFT JOIN PhoneNumbers pn ON pp.phoneID = pn.phoneID
  WHERE fm.familyMemberID = p_familyMemberID;
END //
DELIMITER ;
```

ShowAllFamilyMembers

```
DELIMITER //
CREATE  PROCEDURE ShowAllFamilyMembers()
BEGIN
  SELECT
    fm.familyMemberID,
    fm.locationID,
    p.firstName,
    p.lastName,
    p.middleInitial,
    p.dateOfBirth,
    p.ssn,
    p.medicareNumber AS medicare,
    p.email,
    a.streetAddress,
    a.city,
    a.province,
    a.postalCode,
    pn.phoneNumber,
    pn.extension,
    pn.type AS phoneType
  FROM FamilyMembers fm
  JOIN Person p ON fm.familyMemberID = p.personID
  LEFT JOIN Address a ON p.addressID = a.addressID
  LEFT JOIN PersonPhones pp ON p.personID = pp.personID
  LEFT JOIN PhoneNumbers pn ON pp.phoneID = pn.phoneID
  ORDER BY fm.familyMemberID;
END //
DELIMITER ;
```

4. Club Member Searches

SearchClubMemberByID

```
DELIMITER //
CREATE  PROCEDURE SearchClubMemberByID(IN p_memberID INT)
BEGIN
  SELECT
    cm.memberID,
    p.firstName,
    p.lastName,
    p.middleInitial,
    p.dateOfBirth,
    p.ssn,
    p.medicareNumber AS medicare,
    p.email,
    a.streetAddress,
    a.city,
    a.province,
    a.postalCode,
    pn.phoneNumber,
    pn.extension,
    pn.type AS phoneType,
    cm.height,
    cm.weight,
    cm.gender,
    cm.isActive,
    cm.memberType,
    cm.joinedDate
  FROM ClubMembers cm
  JOIN Person p ON cm.memberID = p.personID
  LEFT JOIN Address a ON p.addressID = a.addressID
  LEFT JOIN PersonPhones pp ON p.personID = pp.personID
  LEFT JOIN PhoneNumbers pn ON pp.phoneID = pn.phoneID
  WHERE cm.memberID = p_memberID;
END //
DELIMITER ;

DELIMITER //
CREATE  PROCEDURE SearchClubMemberPayments(
    IN p_member_id INT
)
proc_label: BEGIN
    -- Check if the memberID exists in the ClubMembers table
    IF NOT EXISTS (SELECT 1 FROM ClubMembers WHERE memberID = p_member_id) THEN
        -- If the member does not exist, return an empty result set and a message
        SELECT 'Error:␣Member␣with␣this␣ID␣does␣not␣exist' AS message;
        LEAVE proc_label;
    END IF;

    -- If the member exists, select all payment records for that member
    SELECT
        memberID,
        membershipYear,
        paymentNumber,
        paymentDate,
        amount,
        paymentMethod
    FROM Payments
    WHERE memberID = p_member_id
    ORDER BY membershipYear DESC, paymentDate DESC;

END //
DELIMITER ;
```

ShowAllClubMembers

```
DELIMITER //
CREATE  PROCEDURE ShowAllClubMembers()
BEGIN
  SELECT
    cm.memberID,
    p.firstName,
    p.lastName,
    cm.height,
    cm.weight,
    cm.gender,
    cm.isActive,
    cm.memberType,
    cm.joinedDate
  FROM ClubMembers cm
  JOIN Person p ON cm.memberID = p.personID
  ORDER BY cm.memberID;
END //
DELIMITER ;
```

5. Team Formation Searches

SearchFormationByIDs

```
DELIMITER //
CREATE  PROCEDURE SearchFormationByIDs(
  IN p_statID INT,
  IN p_sessionID INT
)
BEGIN
  SELECT
    tf.statID,
    tf.sessionID,
    tf.memberStartDate,
    tf.startTime,
    tf.endTime,
    tf.alternatePosition,
    ms.memberID,
    ms.teamID,
    ms.seasonYear,
    ms.startDate AS msStart,
    ms.endDate AS msEnd,
    ms.goals,
    ms.assists,
    ms.playtime,
    ms.position AS regularPosition,
    ms.rating,
    ms.notes
  FROM TeamFormations tf
  JOIN MemberStats ms
    ON tf.statID = ms.statID
  WHERE tf.statID = p_statID
    AND tf.sessionID = p_sessionID;
END //
DELIMITER ;
```

ShowAllFormations

```
DELIMITER //
CREATE  PROCEDURE ShowAllFormations()
BEGIN
  SELECT
    tf.statID,
    tf.sessionID,
    tf.memberStartDate,
    tf.startTime,
    tf.endTime,
    tf.alternatePosition
  FROM TeamFormations tf
  ORDER BY tf.statID, tf.sessionID;
END //
DELIMITER ;
```

Update Procedures

1. UpdateAddress

1. DeleteAddress

```
DELIMITER //
CREATE PROCEDURE UpdateAddress(
    IN p_address_id INT,
    IN p_street_address VARCHAR(255),
    IN p_city VARCHAR(100),
    IN p_province VARCHAR(50),
    IN p_postal_code CHAR(6),
    OUT p_new_address_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_reference_count INT DEFAULT 0;
    DECLARE v_new_address_id INT DEFAULT NULL;
    DECLARE v_address_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_new_address_id = -1;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_address_id IS NULL THEN
        SET p_new_address_id = -1;
        SET p_result_message = 'Address␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if address exists
    IF NOT EXISTS (SELECT 1 FROM Address WHERE addressID = p_address_id) THEN
        SET p_new_address_id = -1;
        SET p_result_message = CONCAT('Address␣with␣ID␣', p_address_id, '␣does␣
            not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Count references to this address
    SELECT COUNT(*) INTO v_reference_count
    FROM (
        SELECT addressID FROM Person WHERE addressID = p_address_id
        UNION ALL
        SELECT addressID FROM Locations WHERE addressID = p_address_id
    ) AS refs;

    -- If multiple references, create new address using InsertAddress
    IF v_reference_count > 1 THEN
        CALL InsertAddress(
            p_street_address, p_city, p_province, p_postal_code,
            v_new_address_id, v_address_message
        );

        IF v_new_address_id = -1 THEN
            SET p_new_address_id = -1;
            SET p_result_message = CONCAT('Failed␣to␣create␣new␣address:␣',
                v_address_message);
            ROLLBACK;
            LEAVE proc_label;
        END IF;
        SET p_result_message = CONCAT('New␣address␣created␣with␣ID␣',
            v_new_address_id,
                                        '␣(original␣address␣has␣multiple␣
                                            references)');
    ELSE
        -- Update existing address
        UPDATE Address
        SET streetAddress = COALESCE(p_street_address, streetAddress),
            city = COALESCE(p_city, city),
            province = COALESCE(p_province, province),
            postalCode = COALESCE(p_postal_code, postalCode)
        WHERE addressID = p_address_id;

        SET v_new_address_id = p_address_id;
        SET p_result_message = CONCAT('Address␣with␣ID␣', p_address_id, '␣updated␣
            successfully');
    END IF;

    SET p_new_address_id = v_new_address_id;
    COMMIT;
END //
DELIMITER ;
```

2. UpdatePhone

2. UpdatePhone

```
DELIMITER //
CREATE PROCEDURE UpdatePhone(
    IN p_phone_id INT,
    IN p_phone_number VARCHAR(20),
    IN p_phone_extension VARCHAR(10),
    IN p_phone_type VARCHAR(20),
    OUT p_new_phone_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_reference_count INT DEFAULT 0;
    DECLARE v_new_phone_id INT DEFAULT NULL;
    DECLARE v_phone_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
        SET p_new_phone_id = -1;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_phone_id IS NULL THEN
        SET p_new_phone_id = -1;
        SET p_result_message = 'Phone␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;
    -- Check if phone exists
    IF NOT EXISTS (SELECT 1 FROM PhoneNumbers WHERE phoneID = p_phone_id) THEN
        SET p_new_phone_id = -1;
        SET p_result_message = CONCAT('Phone␣with␣ID␣', p_phone_id, '␣does␣not␣
            exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Count references to this phone
    SELECT COUNT(*) INTO v_reference_count
    FROM (
        SELECT phoneID FROM PersonPhones WHERE phoneID = p_phone_id
        UNION ALL
        SELECT phoneID FROM LocationPhones WHERE phoneID = p_phone_id
    ) AS refs;

    -- If multiple references, create new phone using InsertPhone
    IF v_reference_count > 1 THEN
        CALL InsertPhone(
            p_phone_number, p_phone_extension, p_phone_type,
            v_new_phone_id, v_phone_message
        );

        IF v_new_phone_id = -1 THEN
            SET p_new_phone_id = -1;
            SET p_result_message = CONCAT('Failed␣to␣create␣new␣phone:␣',
                v_phone_message);
            ROLLBACK;
            LEAVE proc_label;
        END IF;

        SET p_result_message = CONCAT('New␣phone␣created␣with␣ID␣',
            v_new_phone_id,
                                     '␣(original␣phone␣has␣multiple␣references)');
    ELSE
        -- Update existing phone
        UPDATE PhoneNumbers
        SET phoneNumber = COALESCE(p_phone_number, phoneNumber),
            extension = COALESCE(p_phone_extension, extension),
            type = COALESCE(p_phone_type, type)
        WHERE phoneID = p_phone_id;

        SET v_new_phone_id = p_phone_id;
        SET p_result_message = CONCAT('Phone␣with␣ID␣', p_phone_id, '␣updated␣
            successfully');
    END IF;

    SET p_new_phone_id = v_new_phone_id;
    COMMIT;
END //
DELIMITER ;
```

3A. UpdateLocation

3B. UpdateLocationComplete

```
DELIMITER //
CREATE PROCEDURE UpdateLocation(
    IN p_location_id INT,
    IN p_location_name VARCHAR(100),
    IN p_location_type ENUM('Head', 'Branch'),
    IN p_size INT,
    IN p_capacity INT,
    IN p_web_address VARCHAR(255),
    IN p_is_active BOOLEAN,
    OUT p_result_message VARCHAR(255)
proc_label: BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_location_id IS NULL THEN
        SET p_result_message = 'Location␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if location exists
    IF NOT EXISTS (SELECT 1 FROM Locations WHERE locationID = p_location_id) THEN
        SET p_result_message = CONCAT('Location␣with␣ID␣', p_location_id, '␣does␣
            not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Update location
    UPDATE Locations
    SET name = COALESCE(p_location_name, name),
        type = COALESCE(p_location_type, type),
        size = COALESCE(p_size, size),
        capacity = COALESCE(p_capacity, capacity),
        webAddress = COALESCE(p_web_address, webAddress),
        isActive = COALESCE(p_is_active, isActive)
    WHERE locationID = p_location_id;

    SET p_result_message = CONCAT('Location␣with␣ID␣', p_location_id, '␣updated␣
        successfully');
    COMMIT;
END //
DELIMITER ;
```

3B. UpdateLocationComplete

3B.UpdateLocationComplete

```
CREATE PROCEDURE UpdateLocationComplete(
    IN p_location_id INT,
    -- Address fields
    IN p_street_address VARCHAR(255),
    IN p_city VARCHAR(100),
    IN p_province VARCHAR(50),
    IN p_postal_code CHAR(6),
    -- Location fields
    IN p_location_name VARCHAR(100),
    IN p_location_type ENUM('Head', 'Branch'),
    IN p_size INT,
    IN p_capacity INT,
    IN p_web_address VARCHAR(255),
    IN p_is_active BOOLEAN,
    -- Phone fields
    IN p_phone_number VARCHAR(20),
    IN p_phone_extension VARCHAR(10),
    IN p_phone_type VARCHAR(20),
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_address_id INT;
    DECLARE v_phone_id INT;
    DECLARE v_new_address_id INT;
    DECLARE v_new_phone_id INT;
    DECLARE v_address_message VARCHAR(255);
    DECLARE v_phone_message VARCHAR(255);
    DECLARE v_location_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate location exists
    IF NOT EXISTS (SELECT 1 FROM Locations WHERE locationID = p_location_id) THEN
        SET p_result_message = CONCAT('Location␣with␣ID␣', p_location_id, '␣does␣
           not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Get current address and phone IDs
    SELECT addressID INTO v_address_id FROM Locations WHERE locationID =
       p_location_id;
    SELECT phoneID INTO v_phone_id FROM LocationPhones WHERE locationID =
       p_location_id LIMIT 1;

    -- Handle address updates
    IF p_street_address IS NOT NULL OR p_city IS NOT NULL OR p_province IS NOT
       NULL OR p_postal_code IS NOT NULL THEN
        IF v_address_id IS NOT NULL THEN
            -- Update existing address
            CALL UpdateAddress(v_address_id, p_street_address, p_city,
               p_province, p_postal_code, v_new_address_id, v_address_message);

            IF v_new_address_id = -1 THEN
                SET p_result_message = v_address_message;
                ROLLBACK;
                LEAVE proc_label;
            END IF;

            -- Update location's address reference if new address was created
            IF v_new_address_id != v_address_id THEN
                UPDATE Locations SET addressID = v_new_address_id WHERE
                    locationID = p_location_id;
            END IF;
        ELSE
            -- Create new address and link to location
            CALL InsertAddress(p_street_address, p_city, p_province,
                p_postal_code, v_new_address_id, v_address_message);

            IF v_new_address_id = -1 THEN
                SET p_result_message = v_address_message;
                ROLLBACK;
                LEAVE proc_label;
            END IF;

            UPDATE Locations SET addressID = v_new_address_id WHERE locationID =
                p_location_id;
        END IF;
    END IF;

    -- Handle phone updates
    IF p_phone_number IS NOT NULL OR p_phone_extension IS NOT NULL OR
        p_phone_type IS NOT NULL THEN
        IF v_phone_id IS NOT NULL THEN
            -- Update existing phone
            CALL UpdatePhone(v_phone_id, p_phone_number, p_phone_extension,
                p_phone_type, v_new_phone_id, v_phone_message);

            IF v_new_phone_id = -1 THEN
                SET p_result_message = v_phone_message;
                ROLLBACK;
                LEAVE proc_label;
            END IF;

            -- Update location phone reference if new phone was created
            IF v_new_phone_id != v_phone_id THEN
                UPDATE LocationPhones SET phoneID = v_new_phone_id WHERE
                    locationID = p_location_id AND phoneID = v_phone_id;
            END IF;
        ELSE
            -- Create new phone and link to location
            CALL InsertPhone(p_phone_number, p_phone_extension, p_phone_type,
                v_new_phone_id, v_phone_message);

            IF v_new_phone_id = -1 THEN
                SET p_result_message = v_phone_message;
                ROLLBACK;
                LEAVE proc_label;
            END IF;

            INSERT INTO LocationPhones (locationID, phoneID) VALUES
                (p_location_id, v_new_phone_id);
        END IF;
    END IF;
    -- Update location details using existing procedure
    IF p_location_name IS NOT NULL OR p_location_type IS NOT NULL OR p_size IS
       NOT NULL OR p_capacity IS NOT NULL OR p_web_address IS NOT NULL OR
       p_is_active IS NOT NULL THEN
        CALL UpdateLocation(p_location_id, p_location_name, p_location_type,
            p_size, p_capacity, p_web_address, p_is_active, v_location_message);
    END IF;

    SET p_result_message = 'Location␣updated␣successfully';
    COMMIT;
END //
DELIMITER ;
```

4. UpdatePerson

4. DeletePerson

```
DELIMITER //
CREATE PROCEDURE UpdatePerson(
    IN p_person_id INT,
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(50),
    IN p_middle_initial CHAR(1),
    IN p_date_of_birth DATE,
    IN p_ssn CHAR(9),
    IN p_medicare VARCHAR(20),
    IN p_email VARCHAR(100),
    -- Address fields
    IN p_street_address VARCHAR(255),
    IN p_city VARCHAR(100),
    IN p_province VARCHAR(50),
    IN p_postal_code CHAR(6),
    -- Phone fields
    IN p_phone_number VARCHAR(20),
    IN p_phone_extension VARCHAR(10),
    IN p_phone_type VARCHAR(20),
    OUT p_result_message VARCHAR(255)
proc_label: BEGIN
    DECLARE v_address_id INT;
    DECLARE v_phone_id INT;
    DECLARE v_new_address_id INT;
    DECLARE v_new_phone_id INT;
    DECLARE v_address_message VARCHAR(255);
    DECLARE v_phone_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_person_id IS NULL THEN
        SET p_result_message = 'Person␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if person exists
    IF NOT EXISTS (SELECT 1 FROM Person WHERE personID = p_person_id) THEN
        SET p_result_message = CONCAT('Person␣with␣ID␣', p_person_id, '␣does␣not␣
            exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Get current address and phone IDs
    SELECT addressID INTO v_address_id FROM Person WHERE personID = p_person_id;
    SELECT phoneID INTO v_phone_id FROM PersonPhones WHERE personID = p_person_id
       LIMIT 1;

    -- Update person details directly (simple fields)
    UPDATE Person
    SET firstName = COALESCE(p_first_name, firstName),
        lastName = COALESCE(p_last_name, lastName),
        middleInitial = COALESCE(p_middle_initial, middleInitial),
        dateOfBirth = COALESCE(p_date_of_birth, dateOfBirth),
        ssn = COALESCE(p_ssn, ssn),
        medicare = COALESCE(p_medicare, medicare),
        email = COALESCE(p_email, email)
    WHERE personID = p_person_id;

    -- Handle address updates
    IF p_street_address IS NOT NULL OR p_city IS NOT NULL OR p_province IS NOT
       NULL OR p_postal_code IS NOT NULL THEN
        IF v_address_id IS NOT NULL THEN
            -- Update existing address
            CALL UpdateAddress(v_address_id, p_street_address, p_city,
               p_province, p_postal_code, v_new_address_id, v_address_message);

            IF v_new_address_id = -1 THEN
                SET p_result_message = v_address_message;
                ROLLBACK;
                LEAVE proc_label;
            END IF;

            -- Update person's address reference if new address was created
            IF v_new_address_id != v_address_id THEN
                UPDATE Person SET addressID = v_new_address_id WHERE personID =
                   p_person_id;
            END IF;
        ELSE
            -- Create new address and link to person
            CALL InsertAddress(p_street_address, p_city, p_province,
               p_postal_code, v_new_address_id, v_address_message);

            IF v_new_address_id = -1 THEN
                SET p_result_message = v_address_message;
                ROLLBACK;
                LEAVE proc_label;
            END IF;

            UPDATE Person SET addressID = v_new_address_id WHERE personID =
               p_person_id;
        END IF;
    END IF;

    -- Handle phone updates
    IF p_phone_number IS NOT NULL OR p_phone_extension IS NOT NULL OR
        p_phone_type IS NOT NULL THEN
        IF v_phone_id IS NOT NULL THEN
            -- Update existing phone
            CALL UpdatePhone(v_phone_id, p_phone_number, p_phone_extension,
                p_phone_type, v_new_phone_id, v_phone_message);

            IF v_new_phone_id = -1 THEN
                SET p_result_message = v_phone_message;
                ROLLBACK;
                LEAVE proc_label;
            END IF;

            -- Update person phone reference if new phone was created
            IF v_new_phone_id != v_phone_id THEN
                UPDATE PersonPhones SET phoneID = v_new_phone_id WHERE personID =
                    p_person_id AND phoneID = v_phone_id;
            END IF;
        ELSE
            -- Create new phone and link to person
            CALL InsertPhone(p_phone_number, p_phone_extension, p_phone_type,
                v_new_phone_id, v_phone_message);

            IF v_new_phone_id = -1 THEN
                SET p_result_message = v_phone_message;
                ROLLBACK;
                LEAVE proc_label;
            END IF;

            INSERT INTO PersonPhones (personID, phoneID) VALUES (p_person_id,
                v_new_phone_id);
        END IF;
    END IF;

    SET p_result_message = CONCAT('Person␣with␣ID␣', p_person_id, '␣updated␣
        successfully');
    COMMIT;
END //
DELIMITER ;
```

5A. UpdateFamilyMember

5A. UpdateFamilyMemberComplete

```
DELIMITER //
CREATE PROCEDURE UpdateFamilyMember(
    IN p_family_member_id INT,
    IN p_location_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_current_location_id INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
```

```
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_family_member_id IS NULL THEN
        SET p_result_message = 'Family␣Member␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if family member exists
    SELECT locationID INTO v_current_location_id
    FROM FamilyMembers
    WHERE familyMemberID = p_family_member_id;

    IF v_current_location_id IS NULL THEN
        SET p_result_message = CONCAT('Family␣Member␣with␣ID␣',
            p_family_member_id, '␣does␣not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate location if provided
    IF p_location_id IS NOT NULL AND NOT EXISTS (
        SELECT 1 FROM Locations WHERE locationID = p_location_id AND isActive = 1
    ) THEN
        SET p_result_message = 'Location␣does␣not␣exist␣or␣is␣inactive';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Update family member
    UPDATE FamilyMembers
    SET locationID = COALESCE(p_location_id, locationID)
    WHERE familyMemberID = p_family_member_id;

    SET p_result_message = CONCAT('Family␣Member␣with␣ID␣', p_family_member_id, '␣
        updated␣successfully');
    COMMIT;
END //
DELIMITER ;
```

5B. UpdateFamilyMemberComplete

5B. UpdateFamilyMemberComplete

```
DELIMITER //
CREATE PROCEDURE UpdateFamilyMemberComplete(
    IN p_family_member_id INT,
    -- Person fields
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(50),
    IN p_middle_initial CHAR(1),
    IN p_date_of_birth DATE,
    IN p_ssn CHAR(9),
    IN p_medicare VARCHAR(20),
    IN p_email VARCHAR(100),
    -- Address fields
    IN p_street_address VARCHAR(255),
    IN p_city VARCHAR(100),
    IN p_province VARCHAR(50),
    IN p_postal_code CHAR(6),
    -- Phone fields
    IN p_phone_number VARCHAR(20),
    IN p_phone_extension VARCHAR(10),
    IN p_phone_type VARCHAR(20),
    -- Family info
    IN p_location_id INT,
    OUT p_result_message VARCHAR(255))
proc_label: BEGIN
    DECLARE v_person_id INT;
    DECLARE v_family_message VARCHAR(255);
    DECLARE v_person_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Get person ID from family member
    SELECT fm.personID INTO v_person_id
    FROM FamilyMembers fm
    WHERE fm.familyMemberID = p_family_member_id;

    IF v_person_id IS NULL THEN
        SET p_result_message = CONCAT('Family Member with ID ',
           p_family_member_id, ' does not exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Update person details using the centralized procedure
    CALL UpdatePerson(
        v_person_id,
        p_first_name,
        p_last_name,
        p_middle_initial,
        p_date_of_birth,
        p_ssn,
        p_medicare,
        p_email,
        p_street_address,
        p_city,
        p_province,
        p_postal_code,
        p_phone_number,
        p_phone_extension,
        p_phone_type,
        v_person_message
    );
    -- Update family member location using existing procedure
    IF p_location_id IS NOT NULL THEN
        CALL UpdateFamilyMember(p_family_member_id, p_location_id,
            v_family_message);
    END IF;

    SET p_result_message = 'Family␣Member␣updated␣successfully';
    COMMIT;
END //
DELIMITER ;
```

6A. UpdateClubMember

6A. UpdateClubMember

```
DELIMITER //
CREATE PROCEDURE UpdateClubMember(
    IN p_member_id INT,
    IN p_height DECIMAL(5,2),
    IN p_weight DECIMAL(5,2),
    IN p_gender ENUM('Male','Female'),
    IN p_is_active BOOLEAN,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_current_member_type ENUM('Major','Minor');
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_member_id IS NULL THEN
        SET p_result_message = 'Member␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Get current member type
    SELECT memberType INTO v_current_member_type
    FROM ClubMembers
    WHERE memberID = p_member_id;

    IF v_current_member_type IS NULL THEN
        SET p_result_message = CONCAT('Club␣Member␣with␣ID␣', p_member_id, '␣does␣
            not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Update club member
    UPDATE ClubMembers
    SET height = COALESCE(p_height, height),
        weight = COALESCE(p_weight, weight),
        gender = COALESCE(p_gender, gender),
        isActive = COALESCE(p_is_active, isActive)
    WHERE memberID = p_member_id;

    SET p_result_message = CONCAT('Club␣Member␣with␣ID␣', p_member_id, '␣updated␣
        successfully');
    COMMIT;
END //
DELIMITER ;
```

6B. UpdateClubMemberComplete

6A. UpdateClubMember

```
DELIMITER //
CREATE PROCEDURE UpdateClubMemberComplete(
    IN p_club_member_id INT,
    -- Person fields
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(50),
    IN p_middle_initial CHAR(1),
    IN p_date_of_birth DATE,
    IN p_ssn CHAR(9),
    IN p_medicare VARCHAR(20),
    IN p_email VARCHAR(100),
    -- Address fields
    IN p_street_address VARCHAR(255),
    IN p_city VARCHAR(100),
    IN p_province VARCHAR(50),
    IN p_postal_code CHAR(6),
    -- Phone fields
    IN p_phone_number VARCHAR(20),
    IN p_phone_extension VARCHAR(10),
    IN p_phone_type VARCHAR(20),
    -- Member fields
    IN p_height DECIMAL(5,2),
    IN p_weight DECIMAL(5,2),
    IN p_gender ENUM('Male','Female'),
    IN p_is_active BOOLEAN,
    -- Family relationship fields
    IN p_location_id INT,
    IN p_family_id INT,
    IN p_relationship VARCHAR(50),
    IN p_contact_role ENUM('Primary','Secondary','Emergency'),
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_person_id INT;
    DECLARE v_member_message VARCHAR(255);
    DECLARE v_person_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;
    START TRANSACTION;

    -- Get person ID from club member
    SELECT personID INTO v_person_id
    FROM ClubMembers
    WHERE memberID = p_club_member_id;

    IF v_person_id IS NULL THEN
        SET p_result_message = CONCAT('Club␣Member␣with␣ID␣', p_club_member_id, '␣
            does␣not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Update person details using the centralized procedure
    CALL UpdatePerson(
        v_person_id,
        p_first_name,
        p_last_name,
        p_middle_initial,
        p_date_of_birth,
        p_ssn,
        p_medicare,
        p_email,
        p_street_address,
        p_city,
        p_province,
        p_postal_code,
        p_phone_number,
        p_phone_extension,
        p_phone_type,
        v_person_message
    );

    -- Update club member specific fields
    IF p_height IS NOT NULL OR p_weight IS NOT NULL OR p_gender IS NOT NULL OR
       p_is_active IS NOT NULL THEN
        CALL UpdateClubMember(p_club_member_id, p_height, p_weight, p_gender,
            p_is_active, v_member_message);
    END IF;

    -- Update family relationship and location directly if provided
    IF p_location_id IS NOT NULL THEN
        UPDATE ClubMembers SET locationID = p_location_id WHERE memberID =
            p_club_member_id;
    END IF;

    IF p_family_id IS NOT NULL OR p_relationship IS NOT NULL OR p_contact_role IS
       NOT NULL THEN
        UPDATE FamilyMemberRelationships
        SET familyMemberID = COALESCE(p_family_id, familyMemberID),
            relationship = COALESCE(p_relationship, relationship),
            contactRole = COALESCE(p_contact_role, contactRole)
        WHERE clubMemberID = p_club_member_id;
    END IF;

    SET p_result_message = 'Club␣Member␣updated␣successfully';
    COMMIT;
END //
```

```
DELIMITER ;
```

7A. UpdatePersonnelAssignment

7A. UpdatePersonnelAssignment

```
DELIMITER //
CREATE PROCEDURE UpdatePersonnelAssignment(
    IN p_assignment_id INT,
    IN p_role ENUM('General␣Manager','Deputy␣
       Manager','Treasurer','Secretary','Administrator','Captain','Coach','Assistant␣
       Coach','Other'),
    IN p_mandate_type ENUM('Volunteer','Salaried'),
    IN p_start_date DATE,
    IN p_end_date DATE,
    OUT p_result_message VARCHAR(255)
proc_label: BEGIN
    DECLARE v_current_location_id INT;
    DECLARE v_current_role VARCHAR(50);
    DECLARE v_new_gm_assignment_id INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_assignment_id IS NULL THEN
        SET p_result_message = 'Assignment␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Get current assignment details
    SELECT locationID, role INTO v_current_location_id, v_current_role
    FROM PersonnelLocations
    WHERE assignmentID = p_assignment_id;

    IF v_current_location_id IS NULL THEN
        SET p_result_message = CONCAT('Assignment␣with␣ID␣', p_assignment_id, '␣
            does␣not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate end date if provided
    IF p_end_date IS NOT NULL AND p_start_date IS NOT NULL AND p_end_date <
       p_start_date THEN
        SET p_result_message = 'End␣date␣cannot␣be␣before␣start␣date';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Special handling for General Manager role changes
    IF COALESCE(p_role, v_current_role) = 'General␣Manager' AND v_current_role !=
        'General␣Manager' THEN
        -- Check if location already has a GM
        IF EXISTS (
            SELECT 1 FROM PersonnelLocations
            WHERE locationID = v_current_location_id
            AND role = 'General␣Manager'
            AND (endDate IS NULL OR endDate > CURDATE())
            AND assignmentID != p_assignment_id
        ) THEN
            SET p_result_message = 'Location␣already␣has␣an␣active␣General␣
                Manager';
            ROLLBACK;
            LEAVE proc_label;
        END IF;
    END IF;

    -- Update the assignment
    UPDATE PersonnelLocations
    SET role = COALESCE(p_role, role),
        mandateType = COALESCE(p_mandate_type, mandateType),
        startDate = COALESCE(p_start_date, startDate),
        endDate = p_end_date  -- Allow setting to NULL
    WHERE assignmentID = p_assignment_id;

    -- If this is now a GM assignment, update the location's GM reference
    IF COALESCE(p_role, v_current_role) = 'General␣Manager' THEN
        UPDATE Locations
        SET generalManagerAssignmentID = p_assignment_id
        WHERE locationID = v_current_location_id;
    -- If this was a GM assignment but no longer is, clear the reference
    ELSEIF v_current_role = 'General␣Manager' AND COALESCE(p_role,
        v_current_role) != 'General␣Manager' THEN
        UPDATE Locations
        SET generalManagerAssignmentID = NULL
        WHERE locationID = v_current_location_id
        AND generalManagerAssignmentID = p_assignment_id;
    END IF;

    SET p_result_message = CONCAT('Assignment␣with␣ID␣', p_assignment_id, '␣
        updated␣successfully');
    COMMIT;
END //
DELIMITER ;
```

7B. UpdatePersonnelComplete

7B. UpdatePersonnelComplete

```
DELIMITER //
CREATE PROCEDURE UpdatePersonnelComplete(
    IN p_assignment_id INT,
    -- Person fields
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(50),
    IN p_middle_initial CHAR(1),
    IN p_date_of_birth DATE,
    IN p_ssn CHAR(9),
    IN p_medicare VARCHAR(20),
    IN p_email VARCHAR(100),
    -- Address fields
    IN p_street_address VARCHAR(255),
    IN p_city VARCHAR(100),
    IN p_province VARCHAR(50),
    IN p_postal_code CHAR(6),
    -- Phone fields
    IN p_phone_number VARCHAR(20),
    IN p_phone_extension VARCHAR(10),
    IN p_phone_type VARCHAR(20),
    -- Assignment fields
    IN p_location_id INT,
    IN p_role ENUM('General␣Manager','Deputy␣
       Manager','Treasurer','Secretary','Administrator','Captain','Coach','Assistant␣
       Coach','Other'),
    IN p_mandate_type ENUM('Volunteer','Salaried'),
    IN p_start_date DATE,
    IN p_end_date DATE,
    OUT p_result_message VARCHAR(255)
) proc_label: BEGIN
    DECLARE v_person_id INT;
    DECLARE v_assignment_message VARCHAR(255);
    DECLARE v_person_message VARCHAR(255);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Get person ID from assignment
    SELECT pl.personID INTO v_person_id
    FROM PersonnelLocations pl
    WHERE pl.assignmentID = p_assignment_id;

    IF v_person_id IS NULL THEN
        SET p_result_message = CONCAT('Personnel␣assignment␣with␣ID␣',
           p_assignment_id, '␣does␣not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Update person details using the centralized procedure
    CALL UpdatePerson(
        v_person_id,
        p_first_name,
        p_last_name,
        p_middle_initial,
        p_date_of_birth,
        p_ssn,
        p_medicare,
        p_email,
        p_street_address,
        p_city,
        p_province,
        p_postal_code,
        p_phone_number,
        p_phone_extension,
        p_phone_type,
        v_person_message
    );

    -- Update assignment using existing procedure
    IF p_role IS NOT NULL OR p_mandate_type IS NOT NULL OR p_start_date IS NOT
       NULL OR p_end_date IS NOT NULL THEN
        CALL UpdatePersonnelAssignment(p_assignment_id, p_role, p_mandate_type,
            p_start_date, p_end_date, v_assignment_message);
    END IF;

    -- Update assignment location directly if provided
    IF p_location_id IS NOT NULL THEN
        UPDATE PersonnelLocations SET locationID = p_location_id WHERE
            assignmentID = p_assignment_id;
    END IF;

    SET p_result_message = 'Personnel␣updated␣successfully';
    COMMIT;
END //
DELIMITER ;
```

8. UpdateTeamFormation

8.UpdateTeamFormation

```
DELIMITER //
CREATE PROCEDURE UpdateTeamFormation(
    IN p_formation_id INT,
    IN p_alternate_position ENUM('Setter','Outside␣Hitter','Opposite␣
       Hitter','Middle␣Blocker','Libero','Defensive␣Specialist','Serving␣
       Specialist'),
    IN p_start_time DATETIME,
    IN p_end_time DATETIME,
    OUT p_result_message VARCHAR(255)
proc_label: BEGIN
    DECLARE v_current_stat_id INT;
    DECLARE v_current_session_id INT;
    DECLARE v_current_member_id INT;
    DECLARE v_regular_position VARCHAR(50);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_formation_id IS NULL THEN
        SET p_result_message = 'Formation␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;
    -- Get current formation details
    SELECT tf.statID, tf.sessionID, ms.memberID, ms.position
    INTO v_current_stat_id, v_current_session_id, v_current_member_id,
       v_regular_position
    FROM TeamFormations tf
    JOIN MemberStats ms ON tf.statID = ms.statID
    WHERE tf.formationID = p_formation_id;

    IF v_current_stat_id IS NULL THEN
        SET p_result_message = CONCAT('Formation␣with␣ID␣', p_formation_id, '␣
           does␣not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Validate time logic if both times provided
    IF p_start_time IS NOT NULL AND p_end_time IS NOT NULL AND p_end_time <=
       p_start_time THEN
        SET p_result_message = 'End␣time␣must␣be␣after␣start␣time␣if␣both␣are␣
           provided';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Update formation
    UPDATE TeamFormations
    SET alternatePosition = CASE
                            WHEN p_alternate_position IS NULL THEN
                                alternatePosition
                            WHEN p_alternate_position = v_regular_position THEN
                                NULL
                            ELSE p_alternate_position
                            END,
        startTime = COALESCE(p_start_time, startTime),
        endTime = p_end_time  -- Allow setting to NULL
    WHERE formationID = p_formation_id;

    SET p_result_message = CONCAT('Formation␣with␣ID␣', p_formation_id, '␣updated␣
       successfully');
    COMMIT;
END //
DELIMITER ;
```

9. UpdateMemberStats

9. UpdateMemberStats

```
DELIMITER //
CREATE PROCEDURE UpdateMemberStats(
    IN p_stat_id INT,
    IN p_goals INT,
    IN p_assists INT,
    IN p_playtime INT,
    IN p_position ENUM('Setter','Outside␣Hitter','Opposite␣Hitter','Middle␣
       Blocker','Libero','Defensive␣Specialist','Serving␣Specialist'),
    IN p_rating DECIMAL(3,1),
    IN p_notes TEXT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_stat_exists INT DEFAULT 0;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_stat_id IS NULL THEN
        SET p_result_message = 'Stat␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if the stat record exists
    SELECT COUNT(*) INTO v_stat_exists FROM MemberStats WHERE statID = p_stat_id;
    IF v_stat_exists = 0 THEN
        SET p_result_message = CONCAT('Stat␣record␣with␣ID␣', p_stat_id, '␣does␣
            not␣exist');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Update the member stats using COALESCE for non-null values
    UPDATE MemberStats
    SET
        goals = COALESCE(p_goals, goals),
        assists = COALESCE(p_assists, assists),
        playtime = COALESCE(p_playtime, playtime),
        position = COALESCE(p_position, position),
        rating = COALESCE(p_rating, rating),
        notes = CASE
                    WHEN p_notes IS NOT NULL THEN CONCAT(notes, '\n[', NOW(), ']␣
                        ', p_notes)
                    ELSE notes
                END
    WHERE statID = p_stat_id;

    SET p_result_message = CONCAT('Member␣stats␣for␣ID␣', p_stat_id, '␣updated␣
        successfully');
    COMMIT;
END //
DELIMITER ;
```

Misc

TerminateAssignment

```
DELIMITER //
CREATE  PROCEDURE TerminateAssignment(
    IN p_assignment_id INT,
    OUT p_result_message VARCHAR(255)
)
proc_label: BEGIN
    DECLARE v_person_id INT DEFAULT NULL;
    DECLARE v_role VARCHAR(50) DEFAULT NULL;
    DECLARE v_location_id INT DEFAULT NULL;
    DECLARE v_team_id INT DEFAULT NULL;
    DECLARE v_is_gm INT DEFAULT 0;
    DECLARE v_is_coach INT DEFAULT 0;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        GET DIAGNOSTICS CONDITION 1
            p_result_message = MESSAGE_TEXT;
    END;

    START TRANSACTION;

    -- Validate input
    IF p_assignment_id IS NULL THEN
        SET p_result_message = 'Assignment␣ID␣cannot␣be␣NULL';
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Get assignment details
    SELECT pl.personnelID, pl.role, pl.locationID, t.teamID
    INTO v_person_id, v_role, v_location_id, v_team_id
    FROM PersonnelLocations pl
    LEFT JOIN Teams t ON t.coachID = pl.assignmentID
    WHERE pl.assignmentID = p_assignment_id
    AND (pl.endDate IS NULL OR pl.endDate > CURDATE())
    LIMIT 1;

    IF v_person_id IS NULL THEN
        SET p_result_message = CONCAT('Active␣assignment␣with␣ID␣',
           p_assignment_id, '␣not␣found');
        ROLLBACK;
        LEAVE proc_label;
    END IF;

    -- Check if this is a GM or Coach
    IF v_role = 'General␣Manager' THEN
        SET v_is_gm = 1;
    ELSEIF v_role IN ('Coach', 'Assistant␣Coach') THEN
        SET v_is_coach = 1;
    END IF;

    -- Handle role-specific cleanup
    IF v_is_gm = 1 AND v_location_id IS NOT NULL THEN
        UPDATE Locations
        SET generalManagerAssignmentID = NULL
        WHERE locationID = v_location_id
        AND generalManagerAssignmentID = p_assignment_id;
    END IF;

    IF v_is_coach = 1 AND v_team_id IS NOT NULL THEN
        UPDATE Teams
        SET coachID = NULL
        WHERE teamID = v_team_id
        AND coachID = p_assignment_id;
    END IF;

    -- Terminate the specific assignment
    UPDATE PersonnelLocations
    SET endDate = CURDATE()
    WHERE assignmentID = p_assignment_id;

    SET p_result_message = CONCAT('Assignment␣ID␣', p_assignment_id, '␣terminated␣
        successfully');
    IF v_is_gm = 1 THEN
        SET p_result_message = CONCAT(p_result_message, '␣(GM␣role␣cleared)');
    END IF;
    IF v_is_coach = 1 THEN
        SET p_result_message = CONCAT(p_result_message, '␣(Coach␣role␣cleared)');
    END IF;

    COMMIT;
END //
DELIMITER ;
```

Queries

8) Get complete details for every location in the system

GetAllLocationDetails

```
DELIMITER //
CREATE PROCEDURE GetAllLocationDetails()
BEGIN
    SELECT
        ad.streetAddress,
        ad.city,
        ad.province,
        ad.postalCode,
        pn.phoneNumber,
        l.webAddress,
        l.type,
        l.capacity,
        CONCAT(p.firstName, '␣', p.lastName) AS GeneralManagerName,
        COUNT(CASE WHEN cm.memberType = 'Minor' THEN 1 END) AS numMinors,
        COUNT(CASE WHEN cm.memberType = 'Major' THEN 1 END) AS numMajors,
        COUNT(DISTINCT t.teamID) AS numTeams
    FROM Locations l
        JOIN Address ad ON l.addressID = ad.addressID
        JOIN LocationPhones lp ON l.locationID = lp.locationID
        JOIN PhoneNumbers pn ON lp.phoneID = pn.phoneID
        -- If no manager, location will still appear
        LEFT JOIN PersonnelLocations pl ON l.generalManagerAssignmentID =
            pl.assignmentID
        LEFT JOIN Person p ON pl.personnelID = p.personID
        -- If no member, location will still appear
        LEFT JOIN MemberLocations ml ON l.locationID = ml.locationID
        -- If no Minor or major, location will still appear
        LEFT JOIN ClubMembers cm on ml.memberID = cm.memberID
        -- If no teams, location will still appear
        LEFT JOIN Teams t on l.locationID = t.locationID
    GROUP BY
        ad.streetAddress, ad.city, ad.province, ad.postalCode,
        pn.phoneNumber, l.webAddress, l.type, l.capacity,
        p.firstName, p.lastName
    ORDER BY
        ad.province ASC,
        ad.city ASC;
END //
DELIMITER ;
```

9) Get family member and associated club members details

GetFamilyMemberDetails

```
DELIMITER //
CREATE PROCEDURE GetFamilyMemberDetails ( IN familyMemberID INT )
BEGIN
    SELECT
        -- Secondary family member details
        secP . firstName AS secondaryFirstName ,
        secP . lastName AS secondaryLastName ,
        secPhone . phoneNumber AS secondaryPhoneNumber ,
        -- Associated club member details
        cm.memberID AS membershipNumber ,
        mP.firstName AS memberFirstName ,
        mP.lastName AS memberLastName ,
        mP.dateOfBirth ,
        mP.ssn ,
        mP.medicareNumber ,
        memPhone.phoneNumber AS memberPhoneNumber ,
        addr.streetAddress ,
        addr.city ,
        addr.province ,
        addr.postalCode ,
        fr.relationshipType
    FROM FamilyMembers fm
        -- Join to secondary person's personal info
        JOIN Person secP ON fm.familyMemberID = secP.personID
        -- Join to secondary person's phone
        LEFT JOIN PersonPhones secPP ON secPP.personID = secP.personID
        LEFT JOIN PhoneNumbers secPhone ON secPhone.phoneID = secPP.phoneID
        -- Get all minor members (club members) associated with this family member
        JOIN FamilyRelationship fr ON fr.majorID = fm.familyMemberID
        JOIN ClubMembers cm ON cm.memberID = fr.minorID
        JOIN Person mP ON cm.memberID = mP.personID
        -- Member phone
        LEFT JOIN PersonPhones memPP ON memPP.personID = mP.personID
        LEFT JOIN PhoneNumbers memPhone ON memPhone.phoneID = memP.phoneID
        -- Member address
        LEFT JOIN Address addr ON mP.addressID = addr.addressID
    WHERE
        fm.familyMemberID = familyMemberID ;
END //
DELIMITER ;
```

10) Get team formations for a location within a date range

GetLocationTeamFormations

```
DELIMITER //
CREATE PROCEDURE GetLocationTeamFormations(
    IN p_locationID INT,
    IN p_startDate DATE,
    IN p_endDate DATE
)
BEGIN
    SELECT
        COALESCE(CONCAT(coach.firstName, '␣', coach.lastName), 'No␣Coach␣
            Assigned') AS HeadCoachName,
        ts.startTime AS startDateSession,
        a.streetAddress,
        ts.sessionType,
        CONCAT(ht.name, '␣vs␣', COALESCE(at.name, 'TBD')) AS TeamsName,
        ts.homeScore,
        ts.awayScore,
        player.firstName AS PlayerFirstName,
        player.lastName AS PlayerLastName,
        COALESCE(tf.alternatePosition, ms.position) AS role,
        'Home' as teamRole
    FROM TeamSessions ts
        JOIN Locations l ON ts.locationID = l.locationID  -- Sessions at this
            location
        JOIN Address a ON l.addressID = a.addressID
        JOIN Teams ht ON ts.homeTeamID = ht.teamID
        LEFT JOIN Teams at ON ts.awayTeamID = at.teamID
        LEFT JOIN PersonnelLocations coach_assignment ON ht.coachID =
            coach_assignment.assignmentID
        LEFT JOIN Person coach ON coach_assignment.personnelID = coach.personID
        JOIN TeamFormations tf ON ts.sessionID = tf.sessionID
        JOIN MemberStats ms ON tf.statID = ms.statID
        JOIN ClubMembers cm ON ms.memberID = cm.memberID
        JOIN Person player ON cm.memberID = player.personID
    WHERE
        ts.locationID = p_locationID AND
        DATE(ts.startTime) BETWEEN p_startDate AND p_endDate

    ORDER BY
        startDateSession ASC, PlayerLastName ASC;
END //
DELIMITER ;
```

11) Get inactive club members with multiple locations and 2+ years membership

GetInactiveMultiLocationMembers

```
DELIMITER //
CREATE PROCEDURE GetInactiveMultiLocationMembers()
BEGIN
    SELECT
        cm.memberID AS clubMembershipNumber,
        p.firstName,
        p.lastName
    FROM ClubMembers cm
        JOIN Person p ON cm.memberID = p.personID
        JOIN MemberLocations ml ON cm.memberID = ml.memberID
    WHERE cm.isActive = FALSE
        AND DATEDIFF(CURDATE(), cm.joinedDate) >= 730
    GROUP BY
        cm.memberID, p.firstName, p.lastName
    HAVING
        COUNT(DISTINCT ml.locationID) >= 2;
END //
DELIMITER ;
```

12) Team formation report with date range parameter

GetTeamFormationReport

```
DELIMITER //
CREATE PROCEDURE GetTeamFormationReport(IN startDate DATE, IN endDate DATE)
BEGIN
    SELECT
        l.name AS LocationName,
        COUNT(DISTINCT ts.sessionID) AS totalSessions,
        COUNT(DISTINCT CASE WHEN ts.sessionType = 'Training' THEN ts.sessionID
           END) AS totalTrainingSessions,
        COUNT(DISTINCT CASE WHEN ts.sessionType = 'Training' THEN ms.memberID
           END) AS totalTrainingPlayers,
        COUNT(DISTINCT CASE WHEN ts.sessionType = 'Game' THEN ts.sessionID END)
           AS totalGameSessions,
        COUNT(DISTINCT CASE WHEN ts.sessionType = 'Game' THEN ms.memberID END) AS
           totalGamePlayers
    FROM Locations l
        JOIN TeamSessions ts ON l.locationID = ts.locationID
        JOIN Teams t ON ts.homeTeamID = t.teamID
        LEFT JOIN TeamFormations tf ON ts.sessionID = tf.sessionID
        LEFT JOIN MemberStats ms ON tf.statID = ms.statID
    WHERE
        ts.startTime BETWEEN startDate AND endDate
    GROUP BY
        l.locationID, l.name
    HAVING
        COUNT(CASE WHEN ts.sessionType = 'Game' THEN ts.sessionID END) >= 4
    ORDER BY
        totalGameSessions DESC;
END //
DELIMITER ;
```

13) Get active club members never assigned to any formation

GetMembersNeverAssigned

```
DELIMITER //
CREATE PROCEDURE GetMembersNeverAssigned()
BEGIN
    SELECT
        cm.memberID,
        p.firstName,
        p.lastName,
        TIMESTAMPDIFF(YEAR, p.dateOfBirth, CURDATE()) AS age,
        ph.phoneNumber,
        p.email,
        l.name AS locationName
    FROM ClubMembers cm
        JOIN Person p ON cm.memberID = p.personID
        JOIN MemberLocations ml ON cm.memberID = ml.memberID AND ml.endDate IS
            NULL
        JOIN Locations l ON ml.locationID = l.locationID
        JOIN PersonPhones pp ON p.personID = pp.personID
        JOIN PhoneNumbers ph ON pp.phoneID = ph.phoneID
    WHERE
        cm.isActive = TRUE
        -- Exclude members who have ever participated in a session
        AND cm.memberID NOT IN (
            SELECT DISTINCT ms.memberID
            FROM MemberStats ms
                JOIN TeamFormations tf ON ms.statID = tf.statID
        )
    ORDER BY
        l.name ASC,
        age ASC;
END //
DELIMITER ;
```

14) Get active major members who joined as minors

GetMajorMembersJoinedAsMinors

```sql
DELIMITER //
CREATE PROCEDURE GetMajorMembersJoinedAsMinors()
BEGIN
    SELECT
        cm.memberID,
        p.firstName,
        p.lastName,
        cm.joinedDate,
        TIMESTAMPDIFF(YEAR, p.dateOfBirth, CURDATE()) AS age,
        pn.phoneNumber,
        p.email,
        l.name
    FROM ClubMembers cm
        JOIN Person p ON cm.memberID = p.personID
        JOIN PersonPhones pp ON p.personID = pp.personID
        JOIN PhoneNumbers pn ON pp.phoneID = pn.phoneID
        JOIN MemberLocations ml ON cm.memberID = ml.memberID AND ml.endDate IS
           NULL
        JOIN Locations l ON ml.locationID = l.locationID
    WHERE
        cm.isActive = TRUE
        AND cm.memberType = 'Major'
        AND TIMESTAMPDIFF(YEAR, p.dateOfBirth, cm.joinedDate) < 18
    ORDER BY
        l.name ASC,
        age ASC;
END //
DELIMITER ;
```

15) Get active club members only assigned as setters

GetSetterOnlyMembers

```
DELIMITER //
CREATE PROCEDURE GetSetterOnlyMembers()
BEGIN
    SELECT
        cm.memberID,
        p.firstName,
        p.lastName,
        TIMESTAMPDIFF(YEAR, p.dateOfBirth, CURDATE()) AS age,
        pn.phoneNumber,
        p.email,
        l.name AS locationName
    FROM ClubMembers cm
        JOIN Person p ON cm.memberID = p.personID
        JOIN PersonPhones pp ON p.personID = pp.personID
        JOIN PhoneNumbers pn ON pp.phoneID = pn.phoneID
        JOIN MemberLocations ml ON cm.memberID = ml.memberID AND ml.endDate IS
            NULL
        JOIN Locations l ON ml.locationID = l.locationID
    WHERE
        cm.isActive = TRUE
        AND cm.memberID IN (
            SELECT ms.memberID
            FROM MemberStats ms
                JOIN TeamFormations tf ON ms.statID = tf.statID
                JOIN TeamSessions ts ON tf.sessionID = ts.sessionID
            WHERE ts.sessionType IN ('Training', 'Game') AND tf.alternatePosition
                = 'Setter'
            GROUP BY ms.memberID
        ) AND cm.memberID NOT IN (
            SELECT ms.memberID
            FROM MemberStats ms
                JOIN TeamFormations tf ON ms.statID = tf.statID
                JOIN TeamSessions ts ON tf.sessionID = ts.sessionID
            WHERE ts.sessionType IN ('Training', 'Game')
                AND tf.alternatePosition <> 'Setter'
        )
    ORDER BY
        l.name ASC,
        cm.memberID ASC;
END //
DELIMITER ;
```

16) Get active club members assigned to all positions in games

GetAllPositionMembers

```
DELIMITER //
CREATE PROCEDURE GetAllPositionMembers()
BEGIN
    SELECT
        cm.memberID,
        p.firstName,
        p.lastName,
        TIMESTAMPDIFF(YEAR, p.dateOfBirth, CURDATE()) AS age,
        pn.phoneNumber,
        p.email,
        l.name
    FROM ClubMembers cm
        JOIN Person p ON cm.memberID = p.personID
        JOIN PersonPhones pp ON p.personID = pp.personID
        JOIN PhoneNumbers pn ON pp.phoneID = pn.phoneID
        JOIN MemberLocations ml ON cm.memberID = ml.memberID AND ml.endDate IS
           NULL
        JOIN Locations l ON ml.locationID = l.locationID
    WHERE
        cm.isActive = TRUE AND cm.memberID IN (
            SELECT ms.memberID
            FROM MemberStats ms
                JOIN TeamFormations tf ON ms.statID = tf.statID
                JOIN TeamSessions ts ON tf.sessionID = ts.sessionID
            WHERE ts.sessionType = 'Game'
            GROUP BY ms.memberID
            HAVING
                SUM(tf.alternatePosition = 'Setter') > 0 AND
                SUM(tf.alternatePosition = 'Outside␣Hitter') > 0 AND
                SUM(tf.alternatePosition = 'Opposite␣Hitter') > 0 AND
                SUM(tf.alternatePosition = 'Middle␣Blocker') > 0 AND
                SUM(tf.alternatePosition = 'Libero') > 0 AND
                SUM(tf.alternatePosition = 'Defensive␣Specialist') > 0 AND
                SUM(tf.alternatePosition = 'Serving␣Specialist') > 0
        )
    ORDER BY
        l.name ASC,
        cm.memberID ASC;
END //
DELIMITER ;
```

17) Get family members who are coaches at a specific location

GetFamilyMemberCoaches

```
DELIMITER //
CREATE PROCEDURE GetFamilyMemberCoaches(IN locationName VARCHAR(100))
BEGIN
    SELECT DISTINCT
        p.firstName,
        p.lastName,
        pn.phoneNumber
    FROM FamilyMembers fm
        JOIN Person p ON fm.familyMemberID = p.personID
        JOIN PersonPhones pp ON pp.personID = p.personID
        JOIN PhoneNumbers pn ON pn.phoneID = pp.phoneID
        -- Link to minors through FamilyRelationship
        JOIN FamilyRelationship fr ON fr.majorID = fm.familyMemberID
        JOIN ClubMembers cm ON cm.memberID = fr.minorID AND cm.isActive = TRUE
        -- Ensure minor is currently assigned to the same location as family
           member
        JOIN MemberLocations ml ON ml.memberID = cm.memberID
            AND ml.locationID = fm.locationID
            AND ml.endDate IS NULL
        -- Family member must be an active coach at that same location
        JOIN PersonnelLocations pl ON pl.personnelID = fm.familyMemberID
            AND pl.role = 'Coach'
            AND pl.endDate IS NULL
            AND pl.locationID = fm.locationID
        JOIN Teams t ON t.coachID = pl.assignmentID
            AND t.locationID = fm.locationID
        JOIN Locations l ON l.locationID = fm.locationID
    WHERE l.name = locationName;
END //
DELIMITER ;
```

18) Get active club members who never lost a game

GetUndefeatedMembers

```
DELIMITER //
CREATE PROCEDURE GetUndefeatedMembers()
BEGIN
    SELECT
        cm.memberID,
        p.firstName,
        p.lastName,
        TIMESTAMPDIFF(YEAR, p.dateOfBirth, CURDATE()) AS age,
        pn.phoneNumber,
        p.email,
        l.name AS locationName
    FROM ClubMembers cm
        JOIN Person p ON cm.memberID = p.personID
        LEFT JOIN PersonPhones pp ON p.personID = pp.personID
        LEFT JOIN PhoneNumbers pn ON pp.phoneID = pn.phoneID
        JOIN MemberLocations ml ON cm.memberID = ml.memberID
            AND ml.endDate IS NULL
        JOIN Locations l ON ml.locationID = l.locationID
    WHERE cm.isActive = TRUE
        AND cm.memberID IN (
            -- Members who played at least one game session
            SELECT DISTINCT ms.memberID
            FROM TeamFormations tf
                JOIN MemberStats ms ON tf.statID = ms.statID
                JOIN Teams t ON ms.teamID = t.teamID
                JOIN TeamSessions ts ON tf.sessionID = ts.sessionID
            WHERE ts.sessionType = 'Game'
                AND ts.homeScore IS NOT NULL
                AND ts.awayScore IS NOT NULL
        ) AND cm.memberID NOT IN (
            -- Members who lost any game they played
            SELECT DISTINCT ms.memberID
            FROM TeamFormations tf
                JOIN MemberStats ms ON tf.statID = ms.statID
                JOIN Teams t ON ms.teamID = t.teamID
                JOIN TeamSessions ts ON tf.sessionID = ts.sessionID
            WHERE ts.sessionType = 'Game'
                AND ts.homeScore IS NOT NULL
                AND ts.awayScore IS NOT NULL
                AND (
                    -- Member's team was home team and lost
                    (t.teamID = ts.homeTeamID AND ts.homeScore < ts.awayScore)
                    OR
                    -- Member's team was away team and lost
                    (t.teamID = ts.awayTeamID AND ts.awayScore < ts.homeScore)
                )
        )
    ORDER BY l.name ASC, cm.memberID ASC;
END //
DELIMITER ;
```

19) Get volunteer personnel who are family members of minor club members

GetVolunteerFamilyPersonnel

```
DELIMITER //
CREATE PROCEDURE GetVolunteerFamilyPersonnel()
BEGIN
    SELECT
        p.firstName,
        p.lastName,
        COUNT(DISTINCT fr.minorID) AS numberOfMinorMembers,
        pn.phoneNumber,
        p.email,
        l.name AS locationName,
        pl.role, -- role
        pl.mandateType -- tells us volunteer or not
    FROM PersonnelLocations pl
        JOIN Person p ON pl.personnelID = p.personID
        JOIN FamilyMembers fm ON fm.familyMemberID = pl.personnelID
        JOIN FamilyRelationship fr ON fr.majorID = fm.familyMemberID
        JOIN ClubMembers cm ON cm.memberID = fr.minorID
            AND cm.memberType = 'Minor'
        JOIN MemberLocations ml ON ml.memberID = cm.memberID
            AND (ml.endDate IS NULL OR ml.endDate > CURDATE())
        JOIN Locations l ON ml.locationID = l.locationID
            AND l.locationID = fm.locationID
        JOIN PersonPhones pp ON pp.personID = p.personID
        JOIN PhoneNumbers pn ON pn.phoneID = pp.phoneID
    WHERE pl.endDate IS NULL
        AND pl.mandateType = 'Volunteer'
    GROUP BY
        p.personID,
        p.firstName,
        p.lastName,
        pn.phoneNumber,
        p.email,
        l.name,
        pl.role
    ORDER BY
        locationName ASC,
        pl.role ASC,
        p.firstName ASC,
        p.lastName ASC;
END //
DELIMITER ;
```

Data Insertion Queries

Our full data insertions for the core tables

1) Address

Address

```
INSERT INTO `stc353_1`.`Address` (`addressID`, `streetAddress`, `city`,
    `province`, `postalCode`)
VALUES
('1', '4205␣Parc␣Avenue', 'Longueil', 'Quebec', 'H1A2B3'),
('2', '2180␣Rue␣du␣Clocher', 'Montreal', 'Quebec', 'H1E4K7'),
('3', '4530␣Avenue␣des␣Brumes', 'Westmount', 'Quebec', 'H2L3N8'),
('4', '3125␣Saint-Denis␣Street', 'Sainte-Anne-de-Bellevue', 'Quebec', 'H2T1R4'),
('5', '7720␣Belanger␣Street', 'Montreal', 'Quebec', 'H2V2P9'),
('6', '3775␣Decarie␣Boulevard', 'Montreal', 'Quebec', 'H3A1B6'),
('7', '7750␣Rue␣de␣la␣Baie', 'Westmount', 'Quebec', 'H3G4M2'),
('8', '3105␣Boulevard␣des␣Pins', 'Laval', 'Quebec', 'H3K2S1'),
('9', '2100␣Rene-Leveque␣Avenue', 'Brossard', 'Quebec', 'H3Y3L4'),
('10', '1542␣Rue␣des␣Erables', 'Mont-Royal', 'Quebec', 'H4C1T8'),
('11', '5600␣Saint-Hubert␣Street', 'Montreal', 'Quebec', 'H7N2M5'),
('12', '2285␣Saint-Urbain␣Street', 'Pierrefonds', 'Quebec', 'H7L3G1'),
('13', '1250␣Sainte-Catherine␣Street', 'Montreal', 'Quebec', 'H7V4K2'),
('14', '9270␣Avenue␣du␣Marche', 'Laval', 'Quebec', 'H7R1S7'),
('15', '8100␣Cote-des-Neiges␣Road', 'Dollard-Des␣Ormeaux', 'Quebec', 'H7X3P4'),
('16', '6700␣Beaubien␣Street', 'Longueil', 'Quebec', 'J4K1T2'),
('17', '1380␣Maisonneuve␣Boulevard', 'Longueil', 'Quebec', 'J4Y3M6'),
('18', '482␣Rue␣Saint-Hubert', 'Brossard', 'Quebec', 'J4B2N9'),
('19', '7810␣Henri-Bourassa␣Boulevard', 'Mont-Royal', 'Quebec', 'J3Y4K3'),
('20', '8450␣Monkland␣Avenue', 'Montreal', 'Quebec', 'J3Z2P8'),
('21', '6421␣Avenue␣du␣Canal', 'Montreal', 'Quebec', 'H4B2K1'),
('22', '4825␣Saint-Laurent␣Boulevard', 'Longueuil', 'Quebec', 'H2W3T8'),
('23', '1580␣Sherbrooke␣Street', 'Ouestmount', 'Quebec', 'H1P7G4'),
('24', '2555␣Ontario␣Street', 'Sainte-Anne-de-Bellevue', 'Quebec', 'H3J2V6'),
('25', '905␣Rue␣des␣Tournesols', 'Montreal', 'Quebec', 'H2X1K9'),
('26', '3999␣Papineau␣Avenue', 'Montreal', 'Quebec', 'H3C5R2'),
('27', '9045␣Rachel␣Street', 'Brossard', 'Quebec', 'H4N2Y8'),
('28', '1299␣Boulevard␣du␣Havre', 'Laval', 'Quebec', 'H7L3G5'),
('29', '9450␣Cremazie␣Street', 'Montreal', 'Quebec', 'H3M1T4'),
('30', '6930␣Villeray␣Street', 'Westmount', 'Quebec', 'H3Z2S7');
```

2) PhoneNumbers

PhoneNumbers

```
INSERT INTO `stc353_1`.`PhoneNumbers` (`phoneID`, `phoneNumber`, `extension`,
    `type`) VALUES
('1', '438␣210-0987', '3178', 'Other'),
('2', '514␣345-6789', NULL, 'Landline'),
('3', '438␣890-2109', '741', 'Landline'),
('4', '438␣234-8765', NULL, 'Work'),
('5', '514␣678-9012', '2045', 'Landline'),
('6', '514␣123-4567', NULL, 'Work'),
('7', '514␣234-5678', '101', 'Cell'),
('8', '438␣345-7654', '1001', 'Work'),
('9', '438␣123-9876', '315', 'Other'),
('10', '438␣901-1098', '6123', 'Other'),
('11', '438␣567-5432', '999', 'Cell'),
('12', '438␣678-4321', '8901', 'Landline'),
('13', '438␣789-3210', '203', 'Other'),
('14', '514␣789-0123', '678', 'Work'),
('15', '514␣567-8901', '427', 'Cell'),
('16', '514␣210-3456', '9456', 'Other'),
('17', '514␣321-4567', '9999', 'Landline'),
('18', '514␣456-7890', '853', 'Cell'),
('19', '514␣901-2345', '4289', 'Work'),
('20', '438␣456-6543', '906', 'Cell'),
('21', '514␣392-8471', NULL, 'Branch'),
('22', '514␣670-2384', NULL, 'Branch'),
('23', '514␣783-9405', NULL, 'Branch'),
('24', '514␣215-6730', NULL, 'Branch'),
('25', '514␣489-7529', NULL, 'Branch'),
('26', '438␣901-3764', NULL, 'Branch'),
('27', '438␣564-8290', NULL, 'Branch'),
('28', '438␣712-4058', NULL, 'Branch'),
('29', '438␣228-6937', NULL, 'Branch'),
('30', '438␣845-1072', NULL, 'Branch');
```

3) Hobbies

Hobbies

```
INSERT INTO `stc353_1`.`Hobbies` (`hobbyID`, `hobbyName`, `hobbyDescription`)
    VALUES ('1', 'Painting', 'water␣colour'),
('2', 'Gardening', 'working␣in␣the␣garden'),
('3', 'Hiking', 'walks␣on␣the␣mountain'),
('4', 'Knitting', 'wool␣crafts'),
('5', 'Photography', 'polaroids'),
('6', 'Cooking', 'culinary␣arts'),
('7', 'Fishing', 'leisure'),
('8', 'Reading', 'book␣club'),
('9', 'Gaming', 'multiplayer␣games'),
('10', 'Cycling', 'spinning␣classes'),
('11', 'Birdwatching', 'walks␣on␣the␣mountain');
```

4) Person

Person

```
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('1', 'Elodie', 'Tremblay', 'J', '1983-03-22', '278450932',
    'TREE83032271', 'elodie.tremblay83@gmail.com', '1');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('2',
    'Mathieu', 'Gagnon', '1990-11-05', '210674589', 'GAGM90110542',
    'm.gagnon@gouv.qc.ca', '2');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('3', 'Chloe', 'Bouchard', 'D', '2001-07-14', '245983716',
    'BOUC01571409', 'chloe.bouchard_art@yahoo.ca', '3');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('4',
    'Felix', 'Cote', '1978-09-30', '283460915', 'COTF78093088',
    'felix.cote1978@videotron.ca', '4');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('5', 'Maelle', 'Lavoie', 'F', '1988-02-17', '265193407',
    'LAVM88321754', 'maelle.lavoie@umontreal.ca', '5');
```

```
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('6',
    'Olivier', 'Pelletier', '2010-05-09', '214598703', 'PELO10050963',
    'olivier.pelletier.gaming@hotmail.com', '6');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('7',
    'Jade', 'Lefebvre', '1999-12-28', '205781432', 'LEFJ99522810',
    'jade.lefebvre@gouv.qc.ca', '7');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('8',
    'Charles', 'Morin', '1985-08-19', '297340158', 'MORC85081992',
    'charles.morin.hiking@outlook.com', '8');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('9', 'Camille', 'Dubois', 'P', '2013-03-01', '276309854',
    'DUBC13530147', 'camille.dubois23@usherbrooke.ca', '9');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('10', 'Raphael', 'Beaulieu', 'M␣N', '2007-10-12', '234875601',
    'BEAR07101236', 'r.beaulieu@mtl.qc.ca', '10');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('11', 'Emily', 'Johnson', 'A', '1984-02-11', '257491830',
    'JOHN84021166', 'emily.johnson84@gmail.com', '11');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('12',
    'Liam', 'Carter', '1991-09-23', '295731406', 'CARL91092312',
    'liam.carter@gouv.qc.ca', '12');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('13', 'Olivia', 'Bennett', 'M', '2000-05-07', '218603947',
    'BENO00555744', 'olivia.bennett_art@yahoo.ca', '13');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('14',
    'Ethan', 'Campbell', '1979-11-29', '274196305', 'CAME79112928',
    'ethan.campbell79@videotron.ca', '14');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('15', 'Grace', 'Thompson', 'L', '1987-07-16', '263085794',
    'THOG87571685', 'grace.thompson@utoronto.ca', '15');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('16',
    'Mason', 'Parker', '2012-10-04', '243971680', 'PARM12105407',
    'mason.parker.gaming@hotmail.com', '16');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('17',
    'Ava', 'Roberts', '1998-03-25', '226704193', 'ROBA98532539',
    'ava.roberts@gouv.qc.ca', '17');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('18', 'Daniel', 'Hughes', 'R', '1986-12-08', '215903478',
    'HUGD86120873', 'daniel.hughes.hiking@outlook.com', '18');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('19', 'Lily', 'Edwards', 'J', '2011-01-19', '286310542',
    'EDWL11516951', 'lily.edwards11@mcgill.ca', '2');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('20',
    'Noah', 'Sullivan', '2006-06-27', '278503419', 'SULN06562792',
    'noah.sullivan@mtl.qc.ca', '8');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('21', 'Harper', 'Mitchell', 'K', '1982-08-03', '294618037',
    'MITH82085324', 'harper.mitchell82@gmail.com', '1');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('22', 'James', 'Turner', 'D', '1993-04-22', '203179468',
    'TURJ93442208', 'james.turner@gouv.qc.ca', '8');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('23',
    'Charlotte', 'Anderson', '2003-02-14', '271840956', 'ANDC03526471',
    'charlotte.anderson_art@yahoo.ca', '4');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('24', 'Benjamin', 'Scott', 'H', '1977-09-05', '284175093',
    'SCOB77090556', 'benjamin.scott77@videotron.ca', '15');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('25', 'Sophia', 'Kelly', 'E', '1989-11-30', '269481035',
    'KELS89518034', 'sophia.kelly@ubc.ca', '2');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('26',
    'Lucas', 'Phillips', '2013-07-02', '239107584', 'PHIL14525267',
    'lucas.phillips.games@hotmail.com', '10');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('27', 'Amelia', 'Ward', 'T', '1995-05-18', '225694031',
    'WARA95556802', 'amelia.ward@gouv.qc.ca', '17');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('28', 'Owen', 'Fisher', 'B', '1981-03-09', '288540197',
    'FISH81030991', 'owen.fisher.hiking@outlook.com', '2');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `middleInitial`, `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`)
    VALUES ('29', 'Chloe', 'Morgan', 'S', '2010-12-21', '279364150',
    'MORG10527178', 'chloe.morgan10@concordia.ca', '1');
INSERT INTO `stc353_1`.`Person` (`personID`, `firstName`, `lastName`,
    `dateOfBirth`, `ssn`, `medicareNumber`, `email`, `addressID`) VALUES ('30',
    'Jack', 'Lawson', '2008-09-11', '244971603', 'LAWJ08596163',
    'jack.lawson@mtl.qc.ca', '1');
```

5) Location

Location

```
INSERT INTO `stc353_1`.`Locations` (`locationID`, `name`, `type`, `addressID`,
    `capacity`, `webAddress`, `generalManagerAssignmentID`, `isActive`) VALUES
    ('2', 'Montreal␣Volleyball␣Club␣-␣Plateau', 'Head', '21', '100',
    'plateau.mtlvolleyballclub.com', '1', '1');
INSERT INTO `stc353_1`.`Locations` (`locationID`, `name`, `type`, `addressID`,
    `capacity`, `webAddress`, `generalManagerAssignmentID`, `isActive`) VALUES
    ('3', 'Montreal␣Volleyball␣Club␣-␣Verdun', 'Branch', '22', '100',
    'verdun.mtlvolleyballclub.com', '4', '1');
INSERT INTO `stc353_1`.`Locations` (`locationID`, `name`, `type`, `addressID`,
    `capacity`, `webAddress`, `isActive`) VALUES ('4', 'Montreal␣Volleyball␣Club␣-␣
    Cote-des-Neiges', 'Branch', '23', '100',
    'cote-des-neiges.mtlvolleyballclub.com', '1');
INSERT INTO `stc353_1`.`Locations` (`locationID`, `name`, `type`, `addressID`,
    `capacity`, `webAddress`, `isActive`) VALUES ('5', 'Montreal␣Volleyball␣Club␣-␣
    Hochelaga', 'Branch', '24', '100', 'hochelaga.mtlvolleyballclub.com', '1');
INSERT INTO `stc353_1`.`Locations` (`locationID`, `name`, `type`, `addressID`,
    `capacity`, `webAddress`, `generalManagerAssignmentID`, `isActive`) VALUES
    ('6', 'Montreal␣Volleyball␣Club␣-␣Outremont', 'Branch', '25', '100',
    'outremont.mtlvolleyballclub.com', '8', '1');
INSERT INTO `stc353_1`.`Locations` (`locationID`, `name`, `type`, `addressID`,
    `capacity`, `webAddress`, `isActive`) VALUES ('7', 'Montreal␣Volleyball␣Club␣-␣
    Laval', 'Branch', '26', '100', 'laval.mtlvolleyballclub.com', '1');
INSERT INTO `stc353_1`.`Locations` (`locationID`, `name`, `type`, `addressID`,
    `capacity`, `webAddress`, `isActive`) VALUES ('8', 'Montreal␣Volleyball␣Club␣-␣
    Pierrefonds', 'Branch', '27', '100', 'pierrefonds.mtlvolleyballclub.com', '1');
INSERT INTO `stc353_1`.`Locations` (`locationID`, `name`, `type`, `addressID`,
    `capacity`, `webAddress`, `isActive`) VALUES ('9', 'Montreal␣Volleyball␣Club␣-␣
    Villeray', 'Branch', '28', '100', 'villeray.mtlvolleyballclub.com', '1');
INSERT INTO `stc353_1`.`Locations` (`locationID`, `name`, `type`, `addressID`,
    `capacity`, `webAddress`, `generalManagerAssignmentID`, `isActive`) VALUES
    ('10', 'Montreal␣Volleyball␣Club␣-␣Longueil', 'Branch', '29', '100',
    'longueil.mtlvolleyballclub.com', '12', '1');
INSERT INTO `stc353_1`.`Locations` (`locationID`, `name`, `type`, `addressID`,
    `capacity`, `webAddress`, `isActive`) VALUES ('11', 'Montreal␣Volleyball␣Club␣
    -␣Westmount', 'Branch', '30', '100', 'westmount.mtlvolleyballclub.com', '1');
```

6) FamilyMembers

FamilyMembers

```
INSERT INTO `stc353_1`.`FamilyMembers` (`familyMemberID`, `locationID`) VALUES
('1', '2'),
('2', '3'),
('4', '5'),
('5', '6'),
('7', '8'),
('8', '9'),
('11', '4'),
('12', '6'),
('14', '8'),
('15', '10'),
('17', '3'),
('18', '9'),
('21', '2'),
('22', '8'),
('24', '10'),
('25', '2'),
('28', '2'),
('3', '4'),
('6', '7'),
('9', '10'),
('10', '11'),
('13', '7'),
('16', '2'),
('19', '3'),
('20', '8'),
('23', '4'),
('26', '10'),
('27', '7'),
('29', '2'),
('30', '2');
```

7) ClubMembers

ClubMembers

```
INSERT INTO `stc353_1`.`ClubMembers` (`memberID`, `height`, `weight`, `gender`,
    `isActive`, `memberType`, `joinedDate`) VALUES
('1', '152', '56', 'Female', '1', 'Major', '2010-10-07'),
('2', '165', '72', 'Male', '1', 'Major', '2013-03-27'),
('3', '178', '68', 'Female', '1', 'Major', '2022-01-09'),
('4', '183', '81', 'Male', '1', 'Major', '2015-08-09'),
('5', '160', '59', 'Female', '1', 'Major', '2019-11-20'),
('6', '175', '90', 'Male', '1', 'Minor', '2024-05-17'),
('7', '169', '77', 'Female', '1', 'Major', '2021-03-08'),
('8', '182', '63', 'Male', '1', 'Major', '2020-07-29'),
('9', '158', '84', 'Female', '1', 'Minor', '2025-02-07'),
('10', '170', '95', 'Male', '1', 'Minor', '2021-03-19'),
('11', '188', '61', 'Female', '1', 'Major', '2002-08-04'),
('12', '176', '73', 'Male', '1', 'Major', '2009-04-21'),
('13', '163', '66', 'Female', '1', 'Major', '2017-10-23'),
('14', '180', '88', 'Male', '1', 'Major', '2015-12-05'),
('15', '155', '54', 'Female', '1', 'Major', '2003-07-25'),
('16', '172', '79', 'Male', '1', 'Minor', '2024-11-26'),
('17', '185', '69', 'Female', '1', 'Major', '2022-11-03'),
('18', '167', '102', 'Male', '1', 'Major', '2004-12-14'),
('19', '190', '58', 'Female', '1', 'Minor', '2022-02-24'),
('20', '174', '85', 'Male', '1', 'Major', '2017-05-22'),
('21', '159', '74', 'Female', '1', 'Major', '2018-09-12'),
('22', '177', '67', 'Male', '1', 'Major', '2006-06-18'),
('23', '168', '92', 'Female', '1', 'Major', '2020-07-11'),
('24', '181', '60', 'Male', '0', 'Major', '2022-04-05'),
('25', '162', '83', 'Female', '1', 'Major', '2010-05-16'),
('30', '184', '71', 'Male', '1', 'Minor', '2021-09-30');
```

8) PersonPhones

PersonPhones

```
INSERT INTO `stc353_1`.`PersonPhones` (`personID`, `phoneID`) VALUES
('1', '1'),
('21', '1'),
('29', '1'),
('30', '1'),
('2', '2'),
('19', '2'),
('25', '2'),
('28', '2'),
('3', '3'),
('4', '4'),
('23', '4'),
('5', '5'),
('6', '6'),
('7', '7'),
('8', '8'),
('20', '8'),
('22', '8'),
('9', '9'),
('10', '10'),
('26', '10'),
('11', '11'),
('12', '12'),
('13', '13'),
('14', '14'),
('15', '15'),
('24', '15'),
('16', '16'),
('17', '17'),
('27', '17'),
('18', '18');
```

7) LocationPhones

LocationPhones

```
INSERT INTO `stc353_1`.`LocationPhones` (`locationID`, `phoneID`) VALUES
('2', '21'),
('3', '22'),
('4', '23'),
('5', '24'),
('6', '25'),
('7', '26'),
('8', '27'),
('9', '28'),
('10', '29'),
('11', '30');
```

7) PersonnelLocations

PersonnelLocations

```
INSERT INTO `stc353_1`.`PersonnelLocations` (`assignmentID`, `personnelID`,
    `locationID`, `role`, `mandateType`, `startDate`) VALUES
 ('1', '16', '2', 'General␣Manager', 'Salaried', '2024-11-26'),
 ('2', '30', '2', 'Coach', 'Salaried', '2021-09-30'),
 ('3', '29', '2', 'Secretary', 'Salaried', '2024-11-20'),
 ('4', '19', '3', 'General␣Manager', 'Salaried', '2022-02-24'),
 ('5', '27', '3', 'Deputy␣Manager', 'Volunteer', '2010-05-20'),
 ('6', '28', '3', 'Coach', 'Salaried', '2002-01-05'),
 ('7', '4', '5', 'Coach', 'Salaried', '2015-08-09'),
 ('8', '5', '6', 'General␣Manager', 'Salaried', '2019-11-20'),
 ('9', '6', '7', 'Assistant␣Coach', 'Salaried', '2024-05-17'),
 ('10', '7', '8', 'Secretary', 'Salaried', '2021-03-08'),
 ('11', '18', '9', 'Treasurer', 'Salaried', '2004-12-14'),
 ('12', '15', '10', 'General␣Manager', 'Salaried', '2003-07-25'),
 ('13', '24', '10', 'Coach', 'Salaried', '2022-04-05'),
 ('14', '26', '11', 'Coach', 'Salaried', '2024-04-12');
```

8) FamilyRelationship

FamilyRelationship

```
INSERT INTO `stc353_1`.`FamilyRelationship` (`minorID`, `majorID`,
    `relationshipType`, `contactRole`) VALUES
('1', '18', 'Partner', 'Primary'),
('2', '19', 'Friend', 'Primary'),
('3', '20', 'Sibling', 'Primary'),
('4', '21', 'Partner', 'Primary'),
('5', '22', 'Friend', 'Primary'),
('6', '1', 'Mother', 'Primary'),
('6', '13', 'Friend', 'Emergency'),
('6', '16', 'Friend', 'Primary'),
('7', '5', 'Mother', 'Primary'),
('8', '23', 'Partner', 'Primary'),
('9', '3', 'Father', 'Primary'),
('9', '4', 'Mother', 'Emergency'),
('10', '9', 'Father', 'Primary'),
('10', '10', 'Mother', 'Emergency'),
('11', '24', 'Friend', 'Primary'),
('12', '25', 'Partner', 'Primary'),
('13', '26', 'Sibling', 'Primary'),
('14', '27', 'Friend', 'Primary'),
('15', '28', 'Partner', 'Primary'),
('16', '13', 'Mother', 'Primary'),
('16', '17', 'Friend', 'Emergency'),
('17', '29', 'Friend', 'Primary'),
('18', '30', 'Partner', 'Primary'),
('19', '2', 'Father', 'Primary'),
('19', '11', 'Mother', 'Emergency'),
('20', '12', 'Father', 'Primary'),
('21', '1', 'Friend', 'Primary'),
('22', '2', 'Partner', 'Primary'),
('23', '3', 'Friend', 'Primary'),
('24', '4', 'Partner', 'Primary'),
('25', '5', 'Friend', 'Primary'),
('30', '13', 'Mother', 'Primary'),
('30', '17', 'Friend', 'Emergency');
```

9) MemberLocations

ClubMembers

```
INSERT INTO `stc353_1`.`MemberLocations` (`assignmentID`, `memberID`,
    `locationID`, `startDate`) VALUES
('1', '11', '4', '2002-08-04'),
('2', '15', '10', '2003-07-25'),
('3', '18', '9', '2004-12-14'),
('4', '22', '9', '2006-06-18'),
('5', '12', '6', '2009-04-21'),
('6', '25', '3', '2010-05-16'),
('7', '1', '2', '2010-10-07'),
('8', '2', '3', '2013-03-27'),
('9', '4', '5', '2015-08-09'),
('10', '14', '8', '2015-12-05'),
('11', '20', '9', '2017-05-22'),
('12', '13', '7', '2017-10-23'),
('13', '21', '2', '2018-09-12'),
('14', '5', '6', '2019-11-20'),
('15', '23', '5', '2020-07-11'),
('16', '8', '9', '2020-07-29'),
('17', '7', '8', '2021-03-08'),
('18', '10', '11', '2021-03-19'),
('19', '30', '2', '2021-09-30'),
('20', '3', '4', '2022-01-09'),
('21', '19', '3', '2022-02-24'),
('22', '24', '11', '2022-04-05'),
('23', '17', '3', '2022-11-03'),
('24', '6', '7', '2024-05-17'),
('25', '16', '2', '2024-11-26'),
('26', '9', '10', '2025-02-07');
```

10) MemberHobbies

MemberHobbies

```
INSERT INTO `stc353_1`.`MemberHobbies` (`memberID`, `hobbyID`) VALUES
('1', '3'),
('1', '6'),
('1', '11'),
('3', '2'),
('3', '4'),
('3', '5'),
('3', '7'),
('3', '9'),
('4', '1'),
('4', '3'),
('4', '5'),
('4', '7'),
('4', '10'),
('5', '2'),
('5', '6'),
('6', '8'),
('6', '10'),
('7', '1'),
('7', '5'),
('7', '7'),
('7', '9'),
('8', '3'),
('8', '6'),
('9', '2'),
('9', '4'),
('9', '6'),
('9', '8'),
('9', '10'),
('10', '1'),
('10', '5'),
('10', '7'),
('11', '3'),
('11', '5'),
('11', '7'),
('11', '9'),
('11', '11'),
('12', '2'),
('12', '4'),
('12', '6'),
('12', '8'),
('12', '10'),
('13', '1'),
('13', '3'),
('13', '5'),
('14', '2'),
('14', '4'),
('14', '6'),
('14', '8'),
('14', '10'),
('15', '1'),
('15', '3'),
('15', '5'),
('15', '7'),
('15', '9'),
('15', '11'),
('16', '4'),
('16', '6'),
('16', '8'),
('17', '1'),
('17', '3'),
('17', '5'),
('17', '7'),
('17', '9'),
('18', '2'),
('18', '4'),
('18', '6'),
('18', '8'),
('18', '10'),
('19', '1'),
('19', '3'),
('19', '5'),
('20', '2'),
('20', '4'),
('20', '6'),
('20', '8'),
('20', '10'),
('21', '1'),
('21', '3'),
('21', '5'),
('21', '7'),
('21', '9'),
('21', '11'),
('22', '2'),
('22', '4'),
('22', '6'),
('22', '8'),
('22', '10'),
('23', '1'),
('23', '3'),
('23', '5'),
('23', '7'),
('24', '2'),
('24', '4'),
('24', '6'),
('24', '8'),
('25', '1'),
('25', '3'),
('25', '5'),
('25', '7'),
('25', '9'),
('25', '11'),
('30', '2'),
('30', '4'),
('30', '6'),
('30', '8'),
('30', '10');
```

11) Teams

Teams

```
INSERT INTO `stc353_1`.`Teams` (`teamID`, `locationID`, `gender`, `name`,
    `leagueLevel`, `isActive`) VALUES
('1', '10', 'Male', 'The␣Crosscourt␣Flyers', 'Intermediate', '1'),
('2', '7', 'Male', 'Les␣Etoiles␣du␣Volley', 'Beginner', '1'),
('3', '5', 'Female', 'Club␣de␣Volleyball␣Horizon', 'Advanced', '1'),
('4', '6', 'Female', 'Laval␣Thunder␣Serve', 'Intermediate', '1'),
('5', '2', 'Male', 'Montreal␣Spikers', 'Intermediate', '1'),
('6', '11', 'Female', 'Les␣Titans␣du␣Service', 'Advanced', '1'),
('7', '9', 'Male', 'Club␣de␣Volleyball␣Mont-Brillant', 'Intermediate', '1'),
('8', '3', 'Male', 'Les␣As␣du␣Filet', 'Intermediate', '1'),
('9', '8', 'Male', 'Westmount␣Net␣Warriors', 'Advanced', '1'),
('10', '4', 'Female', 'Riverfront␣Smashers', 'Beginner', '1');
```

12) MemberStats

ClubMembers

```
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`, `notes`) VALUES ('1', '1', '5', 2023, '2023-01-15', '142', '85',
    '75033', 'Outside␣Hitter', '8.75', 'Consistent␣offensive␣threat');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`, `notes`) VALUES ('2', '2', '8', 2023, '2023-02-10', '78', '210',
    '88841', 'Setter', '9.25', 'Excellent␣court␣vision');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('3', '3', '3', 2023, '2022-03-05', '165', '42', '79226',
    'Opposite␣Hitter', '8.5');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`, `notes`) VALUES ('4', '4', '4', 2023, '2023-01-20', '92', '68',
    '69030', 'Middle␣Blocker', '7.75', 'Strong␣at␣net');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('5', '5', '6', 2023, '2023-02-15', '56', '185', '75623',
    'Setter', '8');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`, `notes`) VALUES ('6', '6', '2', 2024, '2024-06-01', '38', '24',
    '40816', 'Libero', '6.5', 'Rookie␣season');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`, `notes`) VALUES ('7', '7', '3', 2023, '2021-04-10', '188', '35',
    '85210', 'Outside␣Hitter', '9', 'Team␣MVP');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('8', '8', '9', 2023, '2020-08-15', '105', '90', '82827',
    'Opposite␣Hitter', '8.25');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`, `notes`) VALUES ('9', '9', '1', 2024, '2025-03-01', '12', '8',
    '19210', 'Defensive␣Specialist', '5.75', 'Developing␣player');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`, `notes`) VALUES ('10', '10', '6', 2023, '2021-04-05', '201', '28',
    '87027', 'Outside␣Hitter', '9.5', 'League␣scoring␣leader');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`, `notes`) VALUES ('11', '11', '3', 2023, '2022-09-10', '154', '76',
    '84011', 'Middle␣Blocker', '8.75', 'All-Star␣selection');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('12', '12', '6', 2023, '2023-01-25', '67', '195', '81014',
    'Setter', '9');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('13', '13', '2', 2023, '2023-03-15', '45', '210', '76841',
    'Setter', '8.5');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('14', '14', '6', 2023, '2023-02-01', '178', '41', '83443',
    'Opposite␣Hitter', '8.25');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('15', '15', '1', 2023, '2023-01-10', '88', '62', '70808',
    'Middle␣Blocker', '7.5');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('16', '16', '5', 2024, '2024-12-10', '25', '15', '27054',
    'Serving␣Specialist', '6');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('17', '17', '8', 2023, '2022-12-05', '132', '87', '78022',
    'Outside␣Hitter', '8.5');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('18', '18', '9', 2023, '2023-01-30', '115', '94', '81609',
    'Opposite␣Hitter', '8');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('19', '19', '8', 2023, '2022-03-20', '42', '178', '74410',
    'Setter', '8.75');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('20', '20', '9', 2023, '2023-02-25', '165', '38', '84605',
    'Outside␣Hitter', '8.5');
```

```
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`, `notes`) VALUES ('21', '21', '5', 2023, '2023-03-01', '97', '203',
    '85254', 'Setter', '9.25', 'All-Star␣selection');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('22', '22', '9', 2023, '2023-01-15', '203', '45', '87653',
    'Outside␣Hitter', '9');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('23', '23', '4', 2023, '2023-02-10', '118', '72', '77430',
    'Opposite␣Hitter', '8.25');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('24', '24', '6', 2023, '2023-03-05', '28', '165', '72052',
    'Setter', '8.5');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('25', '25', '8', 2023, '2023-01-20', '142', '88', '80419',
    'Outside␣Hitter', '8.75');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('26', '30', '5', 2023, '2023-02-15', '18', '12', '22831',
    'Defensive␣Specialist', '6.25');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`, `notes`) VALUES ('27', '1', '5', 2024, '2023-01-15', '158', '92',
    '78055', 'Outside␣Hitter', '9', 'Improved␣defense');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`, `notes`) VALUES ('28', '2', '8', 2024, '2023-02-10', '85', '225',
    '90058', 'Setter', '9.5', 'League␣assist␣leader');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('29', '3', '3', 2024, '2022-03-05', '182', '50', '81010',
    'Opposite␣Hitter', '8.75');
INSERT INTO `stc353_1`.`MemberStats` (`statID`, `memberID`, `teamID`,
    `seasonYear`, `startDate`, `goals`, `assists`, `playtime`, `position`,
    `rating`) VALUES ('30', '4', '4', 2024, '2023-01-20', '105', '75', '72059',
    'Middle␣Blocker', '8.25');
```

13) Payments

Truncated this one, as we tried to simulate the data properly, so it's 100+ lines.

ClubMembers

```
INSERT INTO `stc353_1`.`Payments` (`memberID`, `membershipYear`, `paymentNumber`,
    `paymentDate`, `amount`, `paymentMethod`) VALUES
('1', 2010, '1', '2010-10-07', '200', 'Debit'),
('1', 2011, '1', '2011-10-07', '200', 'Debit'),
('1', 2012, '1', '2012-10-07', '200', 'Debit'),
('1', 2013, '1', '2013-10-07', '200', 'Cheque'),
('1', 2014, '1', '2014-10-07', '200', 'Debit'),
('1', 2015, '1', '2015-10-07', '200', 'Cash'),
('1', 2016, '1', '2016-10-07', '200', 'Cash'),
('1', 2017, '1', '2017-10-07', '200', 'Cheque'),
('1', 2018, '1', '2018-10-07', '200', 'Debit'),
('1', 2019, '1', '2019-10-07', '200', 'Debit'),
('1', 2020, '1', '2020-10-07', '200', 'Cheque'),
('1', 2020, '2', '2020-12-25', '50', 'Debit'),
('1', 2021, '1', '2021-10-07', '200', 'Cheque'),
('1', 2022, '1', '2022-10-07', '200', 'Cheque'),
('1', 2023, '1', '2023-10-07', '200', 'Debit'),
('1', 2024, '1', '2024-10-07', '200', 'Cash'),
('1', 2025, '1', '2025-01-11', '200', 'Debit'),
('1', 2025, '2', '2025-01-30', '100', 'Debit');
```