University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: ___2_____

Date: ___2023.10.19_________

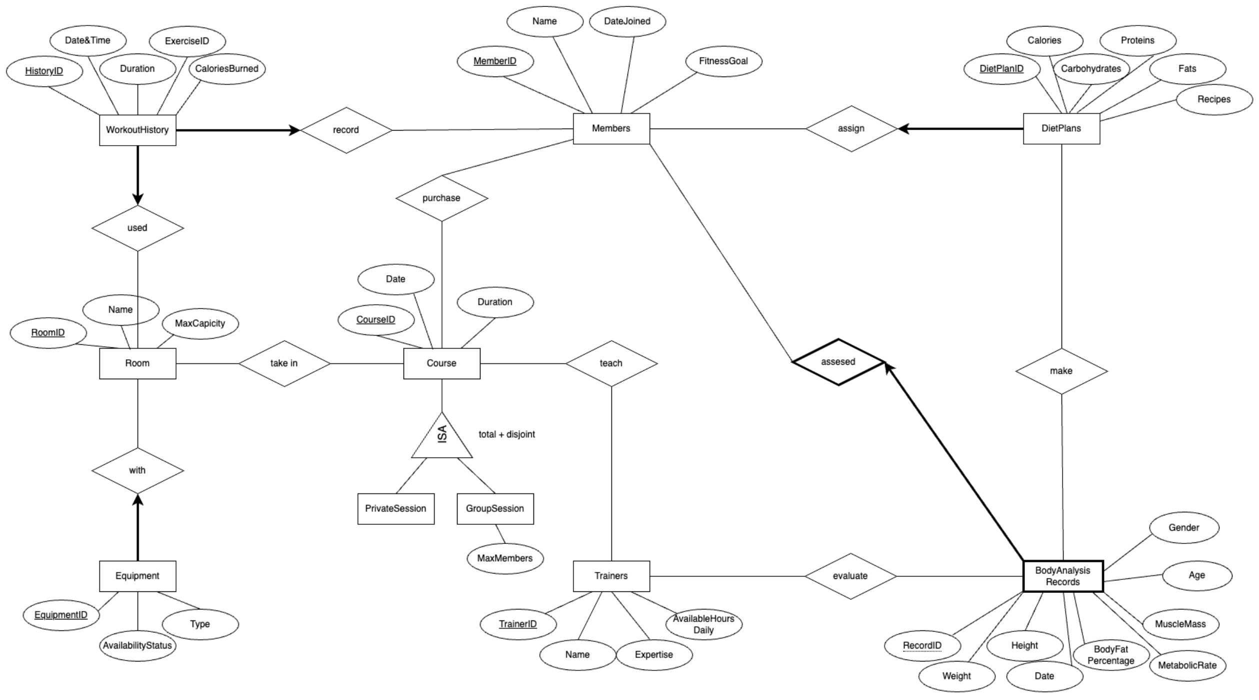
Group Number: _____92_____

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Jialu Jin	24403594	a2f3b	xyxxjinjialu@163.com
Camilla Ren	93534105	d5k5m	camillarr1002@gmail.com
Hao Jiang	58301110	o3f3l	a1181445408@126.com

2. A brief summary of your project.

Our project is centred around a comprehensive fitness and wellness management application. This application serves as a collaborative tool for gym members and trainers, allowing them to track, manage, and coordinate various aspects of a member's health journey. Key functionalities include storing member profiles, workout histories, and body measurements, enabling trainers to offer personalized advice and schedules, and ensuring gym equipment and space efficiency through detailed room and equipment management.

3. The ER diagram you are basing your item #3 (below) on.



As TA suggested, we delete the redundant MemberID and change it to HistoryID as part of the PK of "WorkoutHistory", so we won't have duplicate info. We add missing Total (assuming every course must be classified as either private or group) and Disjoint (since a course being both private and group at the same time isn't logical in most scenarios) constraints to the ISA relationship. We rename the partial key of "BodyAnalysisRecord", MemName to RecordID so it is not the duplicate of the PK of the master entity ("Member") and makes more sense.
In addition, we have removed the Duration key in the "DietPlan" as it is meaningless. We also rename the WorkoutRoutineID to ExerciseID in this entity so it is more reasonable and it makes more sense. We added new keys Gender and Age to the "BodyAnalysisRecord " entity so it contains more information in helping the body analysis, which is more reasonable and makes more sense. In the "Member" entity, we delete the keys Age and CurrentRountine because we have already used them in other entities and it is redundant. We change the relationship between the WorkoutHistory and Room from one-to-one into many-to-one since We can have histories of duplicated rooms used. Also, the equipment must be contained in rooms.

4. The schema derived from your ER diagram.

WorkoutHistory(<u>HistoryID</u>: integer, Date&Time: date-time, Duration: integer, CaloriesBurned: integer, ExerciseID: integer, **MemberID**: integer, **RoomID**: integer)

- PK: <u>HistoryID</u>
- CK: <u>HistoryID,</u> ExerciseID, **MemberID, RoomID**
- FK: **MemberID** REFERENCES Member, **RoomID** REFERENCES Room
- Constraints: <u>HistoryID,</u> **MemberID**, **RoomID,** ExerciseID are NOT NULL, Date&Time is UNIQUE

Trainer(<u>TrainerID</u>: integer, Name: char[30], Expertise: char[20], AvailableHoursDaily: time)

- PK: <u>TrainerID</u>
- CK: <u>TrainerID</u>
- Constraints: <u>TrainerID,</u> Name, AvailableHoursDaily are NOT NULL

Member(<u>MemberID</u>: integer, Name: char[30], DateJoined: date, FitnessGoal: char[30])

- PK: <u>MemberID</u>
- CK: <u>MemberID</u>
- Constraints: <u>MemberID,</u> Name, DatedJoined are NOT NULL

assessed_BosyAnalysisRecord(RecordID: integer, Weight: float, Height: float, BodyFatPercentage: float, Date: date, MetabolicRate: float, MuscleMass: float, Age: integer, Gender: char[10], **MemberID**: integer)
- PK: RecordID, **MemberID**
- CK: RecordID, **MemberID**
- FK: **MemberID** REFERENCES Member
- Constraints: **MemberID** is NOT NULL

DietPlan(DietPlanID: integer, Calories: integer, Carbohydrates: integer, Proteins: integer, Fats: integer, Recipes: char[100], **MemberID**: integer)
- PK: DietPlanID
- CK: DietPlanID, **MemberID**
- FK: **MemberID** REFERENCES Member
- Constraints: DietPlanID, Calories, Recipes, **MemberID** are NOT NULL

Course(CourseID: integer, Date: date, Price: float, Duration: integer)
- PK: CourseID
- CK: CourseID
- Constraints: CourseID, Date, Price, Duration are NOT NULL

PrivateSession(**CourseID:** integer)
- PK: **CourseID**
- CK: **CourseID**
- Constraints: **CourseID** is NOT NULL

GroupSession(MaxMembers: integer, **CourseID:** integer)
- PK: **CourseID**
- CK: **CourseID**
- Constraints: **CourseID** is NOT NULL

Room(RoomID: integer, Name: char[20], MaxCapacity: integer)
- PK: RoomID
- CK: RoomID
- Constraints: RoomID, MaxCapacity, Name, **HistoryID** are NOT NULL

Equipment(EquipmentID: integer, AvailabilityStatus: char[10], Type: char[20], **RoomID**: integer)
- PK: EquipmentID,
- CK: EquipmentID, **RoomID**
- FK: **RoomID** REFERENCES Room
- Constraints: EquipmentID, AvailabilityStatus, Type are NOT NULL

purchase(**MemberID**: integer, **CourseID**: integer)
- PK: **MemberID, CourseID**
- CK: **MemberID, CourseID**
- FK: **MemberID** REFERENCES Member, **CourseID** REFERENCES Course
- Constraints: **MemberID, CourseID** are NOT NULL

make(**RecordID**: integer, **MemberID**: integer, **DietPlanID**: integer)
- PK: **RecordID**, **MemberID**, **DietPlanID**
- CK:**RecordID**, **MemberID**, **DietPlanID**
- FK: **RecordID** RFERENCES BodyAnalysisRecord, **MemberID** REFERENCES Member, **DietPlanID** REFERENCES DietPlan
- Constraints: **MemberID**, **DietPlanID** are NOT NULL

evaluate(**RecordID**: integer, **MemberID**: integer, **TrainerID**: integer)
- PK: **RecordID**, **MemberID**, **TrainerID**
- CK: **RecordID**, **MemberID**, **TrainerID**
- FK: **RecordID** RFERENCES BodyAnalysisRecord, **MemberID** REFERENCES Member, **TrainerID** REFERENCES Trainer
- Constraints: **MemberID**, **TrainerID** are NOT NULL

teach(**TrainerID**: integer, **CourseID**: integer)
- PK: **CourseID**, **TrainerID**
- CK: **CourseID**, **TrainerID**
- FK: **TrainerID** REFERENCES Trainer, **CourseID** REFERENCES Course
- Constraints: **CourseID**, **TrainerID** are NOT NULL

take_in(**CourseID**: integer, **RoomID**: integer)
- PK: **CourseID**, **RoomID**
- CK: **CourseID**, **RoomID**
- FK: **TrainerID** REFERENCES Trainer, **RoomID** REFERENCES Room
- Constraints: **CourseID**, **RoomID** are NOT NULL

5. Functional Dependencies (FDs)

a. Identify the functional dependencies in your relations, including the ones involving all candidate keys (including the primary key).

PKs and CKs are considered functional dependencies and should be included in the list of FDs. You do not need to include trivial FDs such as A → A.

Note: In your list of FDs, there must be some kind of valid FD other than those identified by a PK or CK. If you observe that no relations have FDs other than the PK and CK(s), then you will have to intentionally add some (meaningful) attributes to show valid FDs. We want you to get a good normalization exercise. Your design must go through a normalization process.

WorkoutHistory:
HistoryID -> Date&Time, Duration, ExerciseID, CaloriesBurned, MemberID, RoomID
Duration, ExerciseID -> CaloriesBurned

Room:
RoomID -> Name, MaxCapacity

Equipment:
EquipmentID -> AvailabilityStatus, Type, RoomID

Course:
CourseID -> Date, Price, Duration

PrivateSessions:
CourseID -> Date, Price, Duration

GroupSession:
CourseID -> Date, Price, Duration, MaxMembers

Member:
MemberID -> Name, DateJoined, FitnessGoal

Trainer:
TrainerID -> Name, Expertise, AvailableHours Daily

DietPlan:
DietPlanID -> Calories, Carbohydrates, Proteins, Fats, Recipes, MemberID
Carbohydrates, Proteins, Fats -> Calories
Recipes -> Carbohydrates, Proteins, Fats, Calories

assessed_BodyAnalysisRecord:
MemberID, RecordID -> Weight, Height, Date, BodyFatPercentage, MetabolicRate, MuscleMass, Age, Gender
Weight, Height, Age -> MetabolicRate
Weight, Height, Age, Gender -> BodyFatPercentage

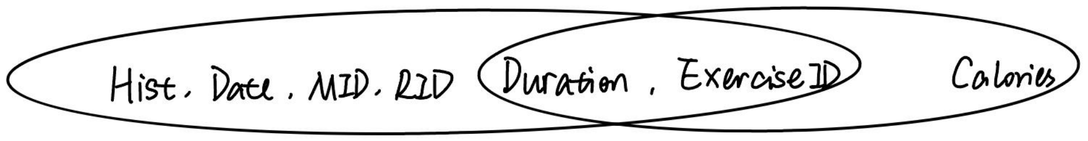
6. Normalization

a. Normalize each of your tables to be in 3NF or BCNF. Give the list of tables, their primary keys, their candidate keys, and their foreign keys after normalization.

You should show the steps taken for the decomposition. Should there be errors, and no work is shown, no partial credit can be awarded without steps shown. The format should be the same as Step 3, with tables listed similar to Table1(attr1:domain1, attr2:domain2, ...). ALL Tables must be listed, not only the ones post normalization.

We will check and normalize tables to be in BCNF.

$\{\text{Duration}, \text{ExerciseID}\}^+ = \{\text{Duration}, \text{ExerciseID}, \text{CaloriesBurnt}\}$



R_1 (HistoryID, Date & Time, Duration, ExerciseID, MemberID, RoomID)
R_2 (Duration, ExerciseID, CaloriesBurnt)

WorkoutHistory1(<u>HistoryID</u>: integer, Date&Time: date-time, **Duration**: integer, **ExerciseID**: integer, **MemberID**: integer, **RoomID**: integer)

- PK: <u>HistoryID</u>
- CK: <u>HistoryID</u>
- FK:
  (ExerciseID, Duration) REFERENCES WorkoutHistory2(ExerciseID, Duration)
  RoomID REFERENCES Room(RoomID)
  MemberID REFERENCES Member(MemberID)

WorkoutHistory2(ExerciseID: integer, Duration: integer, CaloriesBurned: integer)
- PK: ExerciseID, Duration
- CK: ExerciseID, Duration

Room(RoomID: integer, Name: char[20], MaxCapacity: integer)
- PK: RoomID
- CK: RoomID

Equipment(EquipmentID: integer, AvailabilityStatus: char[10], Type: char[20], **RoomID**: integer)
- PK: EquipmentID
- CK: EquipmentID, **RoomID**
- FK: **RoomID** REFERENCES Room(RoomID)

Course(CourseID: integer, Date: date, Price: float, Duration: integer)
- PK: CourseID
- CK: CourseID

PrivateSession(**CourseID:** integer)
- PK: **CourseID**
- CK: **CourseID**

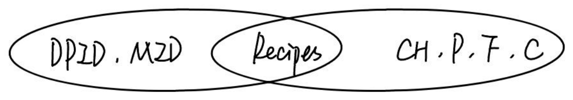
GroupSession(MaxMembers: integer, **CourseID:** integer)
- PK: **CourseID**
- CK: **CourseID**

Member(MemberID: integer, Name: char[30], DateJoined: date, FitnessGoal: char[30])
- PK: MemberID
- CK: MemberID

Trainer(TrainerID: integer, Name: char[30], Expertise: char[20], AvailableHoursDaily: time)
- PK: TrainerID
- CK: TrainerID

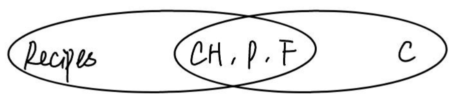
{Carbohydrates, Proteins, Fats}$^+$ = {Carbohydrates, Proteins, Fats, Calories}

Recipes$^+$ = {Carbohydrates, Proteins, Fats, Calories, Recipes}



R1 (Recipes, Carbohydrates, Proteins, Fats, Calories)

R2 (Recipes, DietPlanID, MemberID)



R3 (Carbohydrates, Proteins, Fats, Calories)

R4 (Carbohydrates, Proteins, Fats, Recipes)

Final Answer:

R2 (Recipes, DietPlanID, MemberID)

R3 (Carbohydrates, Proteins, Fats, Calories)

R4 (Carbohydrates, Proteins, Fats, Recipes)

(Abbreviations:
DPID: DietPlanID, MID: MemberID, CH: Carbohydrates, P: Proteins, F: Fats, C: Calories)

DietPlan2(<u>DietPlanID</u>: integer, Recipes: char[100], **MemberID**: integer)
- PK: <u>DietPlanID</u>
- CK: <u>DietPlanID</u>
- FK: **MemberID** REFERENCES Member(MemberID)

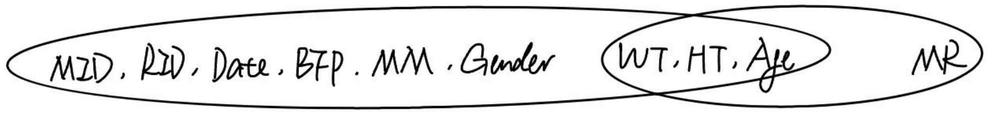
DietPlan3(<u>Carbohydrates</u>: integer, <u>Proteins</u>: integer, <u>Fats</u>: integer, Calories: integer)
- PK: <u>Carbohydrates, Proteins, Fats</u>
- CK: <u>Carbohydrates, Proteins, Fats</u>

DietPlan4(<u>Recipes</u>: char[100], **Carbohydrates**: integer, **Proteins**: integer, **Fats**: integer)
- PK: <u>Recipes</u>
- CK: <u>Recipes</u>
- FK: **(Carbohydrates, Proteins, Fats)** REFERENCES DietPlan3(Carbohydrates, Proteins, Fats)

{Weight, Height, Age}$^+$ = {Weight, Height, Age, MetabolicRate}

{Weight, Height, Age, Gender}$^+$

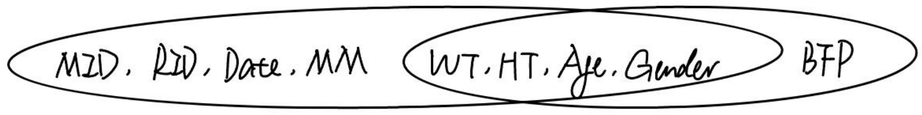
= {Weight, Height, Age, Gender, BodyFatPercentage, MetabolicRate}



R1 (MetabolicRate, Weight, Height, Age)

R2 (Weight, Height, Age, Member ID, RecordID, Date,

BodyFatPercentage, MuscleMass, Gender)



R3 (Weight, Height, Age, Gender, BodyFatPercentage)

R4 (Weight, Height, Age, Gender, Member ID, RecordID, Date, MuscleMass)

Final Answer:

R1 (MetabolicRate, Weight, Height, Age)

R3 (Weight, Height, Age, Gender, BodyFatPercentage)

R4 (Weight, Height, Age, Gender, Member ID, RecordID, Date, MuscleMass)

(Abbreviations:
MID: MemberID, RID: RecordID, BFP: BodyFatPercentage, MM: MuscleMass, WT: Weight, HT: Height, MR: MetabolicRate)

assessed_BodyAnalysisRecord1(<u>Age</u>: integer, <u>Weight</u>: float, <u>Height</u>: float, MetabolicRate: float)
- PK: <u>Weight, Height, Age</u>
- CK: <u>Weight, Height, Age</u>

assessed_BodyAnalysisRecord3(**<u>Age</u>**: integer, **<u>Weight</u>**: float, **<u>Height</u>**: float, <u>Gender</u>: char[10], BodyFatPercentage: float)
- PK: <u>Weight, Height, Age, Gender</u>
- CK: <u>Weight, Height, Age, Gender</u>
- FK:
  <u>(Age, Weight, Height)</u> REFERENCES assessed_BodyAnalysisRecord1(<u>Weight, Height, Age</u>)

assessed_BodyAnalysisRecord4(<u>RecordID</u>: integer, **Age**: integer, **Weight**: float, **Height**: float, **Gender**: char[10], Date: date, MuscleMass: float, MemberID: integer)
- PK: <u>RecordID</u>
- CK: <u>RecordID</u>
- FK:
  MemberID REFERENCES Member(MemberID)
  (Age, Weight, Height, Gender) REFERENCES assessed_BodyAnalysisRecord1(Age, Weight, Height, Gender)

purchase(**<u>MemberID</u>**: integer, **<u>CourseID</u>**: integer, Price: float)
- PK: **<u>MemberID, CourseID</u>**
- CK: **<u>MemberID, CourseID</u>**
- FK: **<u>MemberID</u>** REFERENCES Member(MemberID), **<u>CourseID</u>** REFERENCES Course(CourseID)

make(**<u>RecordID</u>**: integer, **<u>MemberID</u>**: integer, **<u>DietPlanID</u>**: integer)
- PK: **<u>RecordID</u>**, **<u>MemberID</u>**, **<u>DietPlanID</u>**
- CK: **<u>RecordID</u>**, **<u>MemberID</u>**, **<u>DietPlanID</u>**
- FK: **<u>RecordID</u>** RFERENCES BodyAnalysisRecord(RecordID), **<u>MemberID</u>** REFERENCES Member(MemberID), **<u>DietPlanID</u>** REFERENCES DietPlan(DietPlanID)

evaluate(**<u>RecordID</u>**: integer, **<u>MemberID</u>**: integer, **<u>TrainerID</u>**: integer)
- PK: **<u>RecordID</u>**, **<u>MemberID</u>**, **<u>TrainerID</u>**
- CK: **<u>RecordID</u>**, **<u>MemberID</u>**, **<u>TrainerID</u>**
- FK: **<u>RecordID</u>** RFERENCES BodyAnalysisRecord(RecordID), **<u>MemberID</u>** REFERENCES Member(MemberID), **<u>TrainerID</u>** REFERENCES Trainer(TrainerID)

teach(**<u>TrainerID</u>**: integer, **<u>CourseID</u>**: integer)
- PK: **<u>CourseID</u>**, **<u>TrainerID</u>**

- CK: **<u>CourseID</u>**, **<u>TrainerID</u>**
- FK: **<u>TrainerID</u>** REFERENCES Trainer(TrainerID), **<u>CourseID</u>** REFERENCES Course(CourseID)

take_in(**<u>CourseID</u>**: integer, **<u>RoomID</u>**: integer)
- PK: **<u>CourseID</u>**, **<u>RoomID</u>**
- CK: **<u>CourseID</u>**, **<u>RoomID</u>**
- FK: **<u>TrainerID</u>** REFERENCES Trainer(TrainerID), **<u>RoomID</u>** REFERENCES Room(RoomID)

7. The SQL DDL statements required to create all the tables from item #6.

The statements should use the appropriate foreign keys, primary keys, UNIQUE constraints, etc. Unless you know that you will always have exactly x characters for a given character, it is better to use the VARCHAR data type as opposed to a CHAR(Y). For example, UBC courses always use four characters to represent which department offers a course. In that case, you will want to use CHAR(4) for the department attribute in your SQL DDL statement. If you are trying to represent the name of a UBC course, you will want to use VARCHAR as the number of characters in a course name can vary greatly.

Since we will encounter problem by coding Date&Time as the attribute name, we change it into DateTime.

```
CREATE TABLE WorkoutHistory1(
        HistoryID INTEGER PRIMARY KEY NOT NULL,
        DateTime DATETIME NOT NULL,
        ExerciseID INTEGER,
        Duration INTEGER,
        RoomID INTEGER,
        MemberID INTEGER,
        FOREIGN KEY (ExerciseID, Duration) REFERENCES
            WorkoutHistory2(ExerciseID, Duration)
            ON DELETE NO ACTION,
        FOREIGN KEY (RoomID) REFERENCES
```

```
                Room(RoomID)
                ON DELETE NO ACTION,
        FOREIGN KEY (MemberID) REFERENCES
                Member(MemberID)
                ON DELETE NO ACTION);


CREATE TABLE WorkoutHistory2(
        ExerciseID INTEGER PRIMARY KEY,
        Duration INTEGER NOT NULL,
        CaloriesBurned INTEGER);


CREATE TABLE Room(
        RoomID INTEGER PRIMARY KEY,
        Name VARCHAR(20) NOT NULL,
        MaxCapacity INTEGER NOT NULL);


CREATE TABLE Equipment(
        EquipmentID INTEGER PRIMARY KEY,
        AvailabilityStatus VARCHAR(10) NOT NULL,
        Type VARCHAR(20) NOT NULL,
        RoomID INTEGER,
        FOREIGN KEY (RoomID) REFERENCES
                Room(RoomID)
                ON DELETE SET NULL);


CREATE TABLE Course(
        CourseID INTEGER PRIMARY KEY,
        Date DATE NOT NULL,
        Price FLOAT NOT NULL,
        Duration INTEGER NOT NULL);


CREATE TABLE PrivateSession(
        CourseID INTEGER PRIMARY KEY,
        FOREIGN KEY (CourseID) REFERENCES
                Course(CourseID)
                ON DELETE CASCADE);
```

```
CREATE TABLE GroupSession(
        CourseID INTEGER PRIMARY KEY,
        MaxMembers INTEGER,
        FOREIGN KEY (CourseID) REFERENCES
            Course(CourseID)
            ON DELETE CASCADE);


CREATE TABLE Member(
        MemberID INTEGER PRIMARY KEY,
        Name VARCHAR(30) NOT NULL,
        DateJoined DATE NOT NULL,
        FitnessGoal VARCHAR(30));


CREATE TABLE DietPlan2(
        DietPlanID INTEGER PRIMARY KEY,
        Recipes VARCHAR(100),
        MemberID INTEGER,
        FOREIGN KEY (MemberID) REFERENCES
            Member(MemberID)
            ON DELETE CASCADE
            ON UPDATE CASCADE);


CREATE TABLE DietPlan3(
        Carbohydrates INTEGER NOT NULL,
        Proteins INTEGER NOT NULL,
        Fats INTEGER NOT NULL,
        Calories INTEGER NOT NULL,
        PRIMARY KEY (Carbohydrates, Proteins, Fats));


CREATE TABLE DietPlan4(
        Recipes VARCHAR(100) PRIMARY KEY,
        Carbohydrates INTEGER,
        Proteins INTEGER,
        Fats INTEGER,
        FOREIGN KEY (Carbohydrates, Proteins, Fats) REFERENCES
```

```
          DietPlan3(Carbohydrates, Proteins, Fats)
          ON DELETE CASCADE
          ON UPDATE CASCADE);


CREATE TABLE Trainer(
      TrainerID INTEGER PRIMARY KEY,
      Name VARCHAR(30) NOT NULL,
      Expertise VARCHAR(20),
      AvailableHoursDaily TIME NOT NULL);

CREATE TABLE assessed_BodyAnalysisRecord1(
      Age INTEGER,
      Weight FLOAT,
      Height FLOAT,
      MetabolicRate FLOAT,
      PRIMARY KEY (Weight, Height, Age));


CREATE TABLE assessed_BodyAnalysisRecord3(
      Age INTEGER,
      Weight FLOAT,
      Height FLOAT,
      Gender VARCHAR(10),
      BodyFatPercentage FLOAT,
      PRIMARY KEY (Weight, Height, Age, Gender),
      FOREIGN KEY (Age, Weight, Height) REFERENCES
          assessed_BodyAnalysisRecord1(Age, Weight, Height)
          ON DELETE CASCADE
          ON UPDATE CASCADE));


CREATE TABLE assessed_BodyAnalysisRecord4(
      RecordID INTEGER PRIMARY KEY,
      Age INTEGER,
      Weight FLOAT,
      Height FLOAT,
      Gender VARCHAR(10),
      Date DATE,
      MuscleMass FLOAT,
      MemberID INTEGER,
          FOREIGN KEY (MemberID) REFERENCES
               Member(MemberID)
               ON DELETE CASCADE
               ON UPDATE CASCADE,
          FOREIGN KEY (Age, Weight, Height, Gender) REFERENCES
               assessed_BodyAnalysisRecord3(Age, Weight, Height, Gender)
               ON DELETE CASCADE
               ON UPDATE CASCADE));


CREATE TABLE purchase(
          MemberID INTEGER,
          CourseID INTEGER,
          Price INTEGER NOT NULL,
          PRIMARY KEY (MemberID, CourseID),
          FOREIGN KEY (MemberID, Price) REFERENCES
               Member(MemberID, Price)
               ON DELETE NO ACTION,
          FOREIGN KEY (CourseID) REFERENCES
               Course(CourseID)
               ON DELETE NO ACTION);


CREATE TABLE make(
          RecordID INTEGER,
          MemberID INTEGER,
          DietPlanID INTEGER,
          PRIMARY KEY (RecordID, MemberID, DietPlanID),
          FOREIGN KEY (RecordID) REFERENCES
               assessed_BodyAnalysisRecord4(RecordID)
               ON DELETE CASCADE
               ON UPDATE CASCADE,
          FOREIGN KEY (MemberID) REFERENCES
               Member(MemberID)
               ON DELETE CASCADE
               ON UPDATE CASCADE,
          FOREIGN KEY (DietPlanID) REFERENCES
               DietPlan(DietPlanID)
               ON DELETE SET NULL);
```

```
CREATE TABLE evaluate(
          RecordID INTEGER,
          MemberID INTEGER,
          TrainerID INTEGER,
          PRIMARY KEY (RecordID, MemberID, TrainerID),
          FOREIGN KEY (RecordID) REFERENCES
               assessed_BodyAnalysisRecord4(RecordID)
               ON DELETE CASCADE
               ON UPDATE CASCADE,
          FOREIGN KEY (MemberID) REFERENCES
               Member(MemberID)
               ON DELETE CASCADE
               ON UPDATE CASCADE,
          FOREIGN KEY (TrainerID) REFERENCES
               Trainer(TrainerID)
               ON DELETE CASCADE
               ON UPDATE CASCADE);


CREATE TABLE teach(
          TrainerID INTEGER,
          CourseID INTEGER,
          PRIMARY KEY (CourseID, TrainerID),
          FOREIGN KEY (TrainerID) REFERENCES
               Trainer(TrainerID)
               ON DELETE CASCADE
               ON UPDATE CASCADE,
          FOREIGN KEY (CourseID) REFERENCES
               Course(CourseID)
               ON DELETE CASCADE
               ON UPDATE CASCADE);


CREATE TABLE take_in(
          CourseID INTEGER,
          RoomID INTEGER,
          PRIMARY KEY (CourseID, RoomID),
          FOREIGN KEY (CourseID) REFERENCES
               Course(CourseID)
               ON DELETE CASCADE
               ON UPDATE CASCADE,
          FOREIGN KEY (RoomID) REFERENCES
               Room(RoomID)
               ON DELETE  DELETE NO ACTION
               ON UPDATE CASCADE);
```

8. INSERT statements to populate each table with at least 5 tuples.

You will likely want to have more than 5 tuples so that you can have meaningful queries later.

Note: Be consistent with the names used in your ER diagram, schema, and FDs. Make a note if the name has been intentionally changed.

Note: As you start analyzing these requirements, you may notice that certain details are missing. In this case, you may make any reasonable assumptions about them; but, if there is any uncertainty about some requirements, you should ask your project TA before proceeding further (or if it's more general in nature, post your question on Piazza). Furthermore, it is acceptable to modify your design from your original project proposal, because as you progress and start thinking more about the data and the queries that you want to answer from your application, it is normal to find that you need to modify the design (but don't go back and re-do or re-submit your project proposal).

WARNING: Do not start on the implementation portion of the project until you complete tutorial 6 /7/8. Tutorial 6/7/8 will offer a chance for you to try Oracle/Java, Oracle/PHP, and Oracle/Javascript. It is likely that you will not know what you enjoy working with until you finish these tutorials.
Check the milestone 2 assignment on Canvas for the grading rubric. Refer to the syllabus for information on late submission/penalty rules.

```
INSERT INTO WorkoutHistory1 (HistoryID, DateTime, ExerciseID, Duration, MemberID, RoomID)
VALUES
  (1, '2023-10-20 08:00:00', 101, 30, 1),
  (2, '2023-10-20 09:00:00', 102, 45, 2),
  (3, '2023-10-20 10:00:00', 103, 60, 3),
  (4, '2023-10-20 11:00:00', 101, 40, 4),
  (5, '2023-10-20 12:00:00', 104, 55, 5);

INSERT INTO WorkoutHistory2 (ExerciseID, Duration, CaloriesBurned) VALUES
```

```
    (101, 30, 150),
    (102, 45, 220),
    (103, 60, 300),
    (104, 55, 270),
    (105, 40, 190);

INSERT INTO Room (RoomID, Name, MaxCapacity) VALUES
    (1, 'Room A', 50),
    (2, 'Room B', 40),
    (3, 'Room C', 60),
    (4, 'Room D', 45),
    (5, 'Room E', 55);

INSERT INTO Equipment (EquipmentID, AvailabilityStatus, Type, RoomID) VALUES
    (101, 'Available', 'Treadmill', 1),
    (102, 'In Use', 'Elliptical', 2),
    (103, 'Available', 'Dumbbells', 3),
    (104, 'In Use', 'Exercise Bike', 4),
    (105, 'Available', 'Rowing Machine', 5);

INSERT INTO Course (CourseID, Date, Price, Duration) VALUES
    (1, '2023-10-20', 50.0, 60),
    (2, '2023-10-21', 40.0, 45),
    (3, '2023-10-22', 60.0, 75),
    (4, '2023-10-23', 55.0, 90),
    (5, '2023-10-24', 70.0, 70);

INSERT INTO PrivateSession (CourseID) VALUES
    (1),
    (2),
    (3),
    (4),
    (5);

INSERT INTO GroupSession (CourseID, MaxMembers) VALUES
    (6, 10),
    (7, 15),
    (8, 12),
    (9, 8),
    (10, 20);
```

```
INSERT INTO Member (MemberID, Name, DateJoined, FitnessGoal) VALUES
   (1, 'John Doe', '2023-01-15', 'Weight Loss'),
   (2, 'Jane Smith', '2023-03-20', 'Muscle Gain'),
   (3, 'Alice Johnson', '2023-05-10', 'Fitness Maintenance'),
   (4, 'Bob Brown', '2023-07-02', 'Cardiovascular Health'),
   (5, 'Eve Wilson', '2023-09-05', 'Strength Training');


INSERT INTO Trainer (TrainerID, Name, Expertise, AvailableHoursDaily) VALUES
   (1, 'Trainer 1', 'Strength Training', '08:00-12:00'),
   (2, 'Trainer 2', 'Yoga', '12:00-16:00'),
   (3, 'Trainer 3', 'Cardiovascular Health', '10:00-14:00'),
   (4, 'Trainer 4', 'CrossFit', '14:00-18:00'),
   (5, 'Trainer 5', 'Pilates', '16:00-20:00');

INSERT INTO DietPlan2 (DietPlanID, Recipes, MemberID) VALUES
   (1, 'Balanced Diet', 1),
   (2, 'Keto Diet', 2),
   (3, 'Vegan Diet', 3),
   (4, 'Paleo Diet', 4),
   (5, 'Low-Carb Diet', 5);

INSERT INTO DietPlan3 (Carbohydrates, Proteins, Fats, Calories) VALUES
   (100, 50, 30, 1200),
   (80, 60, 40, 1400),
   (60, 70, 50, 1500),
   (120, 40, 35, 1300),
   (90, 55, 45, 1350);

INSERT INTO DietPlan4 (Recipes, Carbohydrates, Proteins, Fats) VALUES
   ('Recipe 1', 50, 30, 1200),
   ('Recipe 2',60, 40, 1400),
   ('Recipe 3', 70, 50, 1500),
   ('Recipe 4', 40, 35, 1300),
   ('Recipe 5', 55, 45, 1350);

INSERT INTO assessed_BodyAnalysisRecord1 (Age, Weight, Height, MetabolicRate)
VALUES
   (25, 70.5, 175.0, 1500.0),
   (30, 68.2, 170.5, 1400.0),
   (35, 80.0, 180.0, 1600.0),
    (28, 65.5, 160.0, 1450.0),
    (40, 75.0, 170.0, 1550.0);

INSERT INTO assessed_BodyAnalysisRecord3 (Age, Weight, Height, Gender, BodyFatPercentage,
Date, MuscleMass, MemberID)
VALUES
    (25, 70.5, 175.0, 'Male', 18.5, '2023-01-15', 65.0, 1),
    (30, 68.2, 170.5, 'Female', 22.0, '2023-02-20', 60.5, 2),
    (35, 80.0, 180.0, 'Male', 15.2, '2023-03-10', 70.0, 3),
    (28, 65.5, 160.0, 'Male', 20.1, '2023-04-05', 58.0, 4),
    (40, 75.0, 170.0, 'Female', 19.8, '2023-05-12', 68.5, 5);

INSERT INTO assessed_BodyAnalysisRecord4 (RecordID, Age, Weight, Height, Gender, Date,
MuscleMass, MemberID)
VALUES
    (1, 25, 70.5, 175.0, 'Male', '2023-01-15', 65.0, 1),
    (2, 30, 68.2, 170.5, 'Female', '2023-02-20', 60.5, 2),
    (3, 35, 80.0, 180.0, 'Male', '2023-03-10', 70.0, 3),
    (4, 28, 65.5, 160.0, 'Male', '2023-04-05', 58.0, 4),
    (5, 40, 75.0, 170.0, 'Female', '2023-05-12', 68.5, 5);

INSERT INTO purchase (MemberID, CourseID, Price)
VALUES
    (1, 1, 50),
    (2, 2, 60),
    (3, 3, 55),
    (4, 4, 65),
    (5, 5, 70);

INSERT INTO make (RecordID, MemberID, DietPlanID)
VALUES
    (1, 1, 201),
    (2, 2, 202),
    (3, 3, 203),
    (4, 4, 204),
    (5, 5, 205);

INSERT INTO evaluate (RecordID, MemberID, TrainerID)
VALUES
    (1, 101, 201),
    (2, 102, 202),
    (3, 103, 203),
    (4, 104, 204),
    (5, 105, 205);

INSERT INTO teach (TrainerID, CourseID)
VALUES
    (201, 301),
    (202, 302),
    (203, 303),
    (204, 304),
    (205, 305);

INSERT INTO take_in (CourseID, RoomID)
VALUES
    (301, 401),
    (302, 402),
    (303, 403),
    (304, 404),
    (305, 405);
```