***Fitness Center Management***

## ***PROJECT PROPOSAL***

**Content, Scope and Objectives**

In today’s world, health and fitness plays a very crucial role in everybody’s life. Therefore, managing the fitness center to fulfill the needs of its members is extremely important.

Fitness centers provide various fitness facilities like equipment gymnasiums, class studios, sports recreation centers, etc. and other services to promote physical, social, and emotional health and wellness. The fitness center management will keep track of the fitness center employees, members, trainers, equipment and other facilities.

**Application Vision:** To create a fitness center management application where fitness centers can maintain information pertaining to memberships and schedules for classes, trainers, and amenities.

The objectives of the Fitness Center Management are to provide better customer service by maintaining:

• membership details of each member

• details of employees and equipment

• details of trainers and classes

Thisproject will be implemented in 3 sprints. Each sprint description and user stories will be mentioned.

## ***PROJECT ENVIRONMENT***

The project environment for the Fitness Center Management database will be phpMyAdmin (https://www.phpmyadmin.net/) using the Cloud9 development workspace in the cloud. The phpMyadmin will provide the interface for MySQL database management system through which data storage, retrieval and operations can be performed. Cloud9 is the best option as all team members already have an account and have the knowledge in how to perform operations in this application.

## ***HIGH LEVEL REQUIREMENTS***

### **Initial user roles**

|  |  |
| --- | --- |
| **User Role** | **Description** |
| Member | Member is a user who registers for classes in Fitness Center as well as signs up for a membership. |
| Trainer | Trainer is the person who trains members of Fitness Center. |
| Manager | Manager manages the trainer and other facilities of the Fitness Center. |

### **Initial user story descriptions**

|  |  |
| --- | --- |
| **Story ID** | **Story description** |
| US1 | Member should be able to view the type of classes, class schedule and trainer details of all the classes available at the fitness center. |
| US2 | Member should be able to view the classes that he/she enrolled to. |
| US3 | Manager should be able to add or remove the classes. |
| US4 | Manager should be able to assign trainer to the classes and schedule the classes. |
| US5 | Manager should be able to view all the members assigned to a trainer’s class |
| US6 | Trainer should be able to view his/her class schedule. |
| US7 | Trainer should be able to view the profile of all the members signed up for their class. |

## ***HIGH LEVEL CONCEPTUAL DESIGN***

Entities:

Member

Manager

Trainer

Membership

Location

Class

Class\_Schedule

Fitness\_Activity\_Type

Equipment

Facility\_Room

Relationships:

Member has Membership

Member views Class offered

Member views Class\_Schedule

Member views their enrolled Class\_Schedule

Member views class Trainer

Trainer trains Member

Trainer views Class assigned

Manager manages Facility

Manager adds Class

Manager removes Class

Manager assigns Trainer

Manager updates Trainer

Manager schedules Class

Manager views all Member assigned to Class

A Manager manages the fitness centre at a Location

Fitness\_Activity\_Type has Equipment associated to it

Fitness\_Activity\_Type has some Class

Class has a Class\_Schedule

Class has a Trainer

Class takes place in Facility\_Room

***Sprint-1***

## ***REQUIREMENTS***

|  |  |
| --- | --- |
| **Story ID** | **Story description** |
| US1 | As a Manager, I want to add a class(es). |
| US2 | As a Manager, I want to remove a class(es). |
| US3 | As a Manager, I should be able to assign trainer to the class(es). |
| US4 | As a Manager, I want to schedule a class(es). |
| US5 | As a Manager, I want to view all the members assigned to a trainer’s class. |
| US6 | As a Member, I want to view my class schedule. |
| US7 | As a Member, I want to view the trainer for the class I am taking. |
| US8 | As a Member, I want to see all the classes available at the fitness center so that I can choose a class. |
| US9 | As a Trainer, I want to view my class schedule. |
| US10 | As a Trainer, I should be able to view the profile of all the members signed up for a class. |

## ***CONCEPTUAL DESIGN***

Entity: **Employee (generalization)**

Attributes:

EmployeeID

Name [composite]

Last\_name

First\_name

Address [composite]

Street

City

State

Zip\_code

Phone\_number

Email\_ID

Annual\_salary

Entity: **Manager (specialization of Employee)**

Attributes:

EmployeeID

Role

Annual\_salary

Entity: **Trainer (specialization of Employee)**

Attributes:

EmployeeID

Role

Hourly\_Salary

Hours\_Worked

Entity: **Class**

Attributes:

ClassId

ClassName

Type

MaxNoOfStudents

Entity: **Schedule**

Attributes:

ScheduleID

Start\_time

End\_time

NoOfDays

Entity: **Member**

Attributes:

MemberID

Name [composite]

Last\_name

First\_name

Address [composite]

Street

City

State

Zip\_code

Phone\_number

Email

Relationship: **Manager** is an **Employee**

Cardinality: One to One

Participation:

Manager has total participation

Employee has partial participation

Relationship: **Trainer** is an **Employee**

Cardinality: One to One

Participation:

Trainer has total participation

Employee has partial participation

Relationship: **Manager** adds **Class**

Cardinality: One to Many

Participation:

Manager has partial participation

Class has total participation

Relationship: **Manager** assigns **Trainer**

Cardinality: One to Many

Participation:

Manager has partial participation

Trainer has total participation

Relationship: **Class** has **Trainer**

Cardinality: Many to One

Participation:

Trainer has partial participation

Class has total participation

Relationship: **Manager** schedules a **class**

Cardinality: One to Many

Participation:

Manager has partial participation

Class has partial participation

Relationship: **Member** registers to a **Class**

Cardinality: Many to Many

Participation:

Member has partial participation

Class has partial participation

Relationship: **Class** has **Schedule**

Cardinality: One to Many

Participation:

Class has partial participation

Schedule has total participation

## ***LOGICAL DESIGN***

Table: **Employee**

Columns:

EmployeeID (primary key)

Last\_name

First\_name

Street

City

State

Zip\_code

Phone number

Email\_ID

Primary key justification: EmployeeID is unique for Employee so it is a good primary key.

Assumption: EmployeeID number is uniquely assigned by the system when an employee is added.

Table: **Mananger**

Columns:

ManagerID (primary key also foreign key referencing to the EmployeeID in Employee table)

Role

Annual Salary

Primary key justification: ManagerID is unique for Manager so it is a good primary key.

Table: **Trainer**

Columns:

TrainerID (primary key also foreign key referencing to the EmployeeID in Employee table)

Role

Hourly\_Salary

Hours\_Worked

Primary key justification: TrainerID is unique for Trainer so it is a good primary key.

Table: **Class**

Columns:

ClassId

ClassName

MaxNoOfStudents

ClassAddedBy(foreign key referencing to ManagerID in Manager table)

TrainerID (foreign key referencing to TrainerID in Trainer table)

TrainerAssignedBy(foreign key referencing to ManagerID in Manager table)

Primary key justification: ClassID is unique for each class so it is a good primary key.

Assumption: ClassID number is uniquely assigned by the system when a new class is created.

Table: **Schedule**

Columns:

ScheduleID

Start\_time

End\_time

No\_Of\_Days

ClassId (foreign key referencing to ClassId in Class table)

ScheduledBy(foreign key referencing to ManagerID in Manager table)

Primary key justification: ScheduleID is unique for each class scheduled so it is a good primary key.

Assumption: ScheduleID number is uniquely assigned by the system when a new schedule is created.

Table: **Member**

Column:

MemberID

First\_name

Last\_name

Street

City

State

Zip\_code

Phone\_number

Email\_ID

Primary key justification: MemberID is unique for each member so it is a good primary key.

Assumption: MemberID number is uniquely assigned by the system when a new member is added.

Table: **Registration**

Column:

RegistrationID

ClassId( foreign key referencing ClassId in Class table)

MemberID (foreign key referencing MemberId in Member table)

Registration\_Date

Primary key justification: RegistrationID is unique for each member so it is a good primary key.

Assumption: RegistrationID number is uniquely assigned by the system when a new member is added.

## ***SQL QUERIES***

1. As a Member, I want to see all the classes available at the fitness center so that I can choose a class.

SELECT ClassName, IFNULL(Start\_time, '-') AS 'Start Time', IFNULL( End\_time,'-') 'End Time', IFNULL(No\_Of\_Days, '-') AS'No. Of Days', CONCAT(First\_Name, ' ', Last\_name) AS Trainer

FROM Schedule

RIGHT JOIN Class

USING (ClassID)

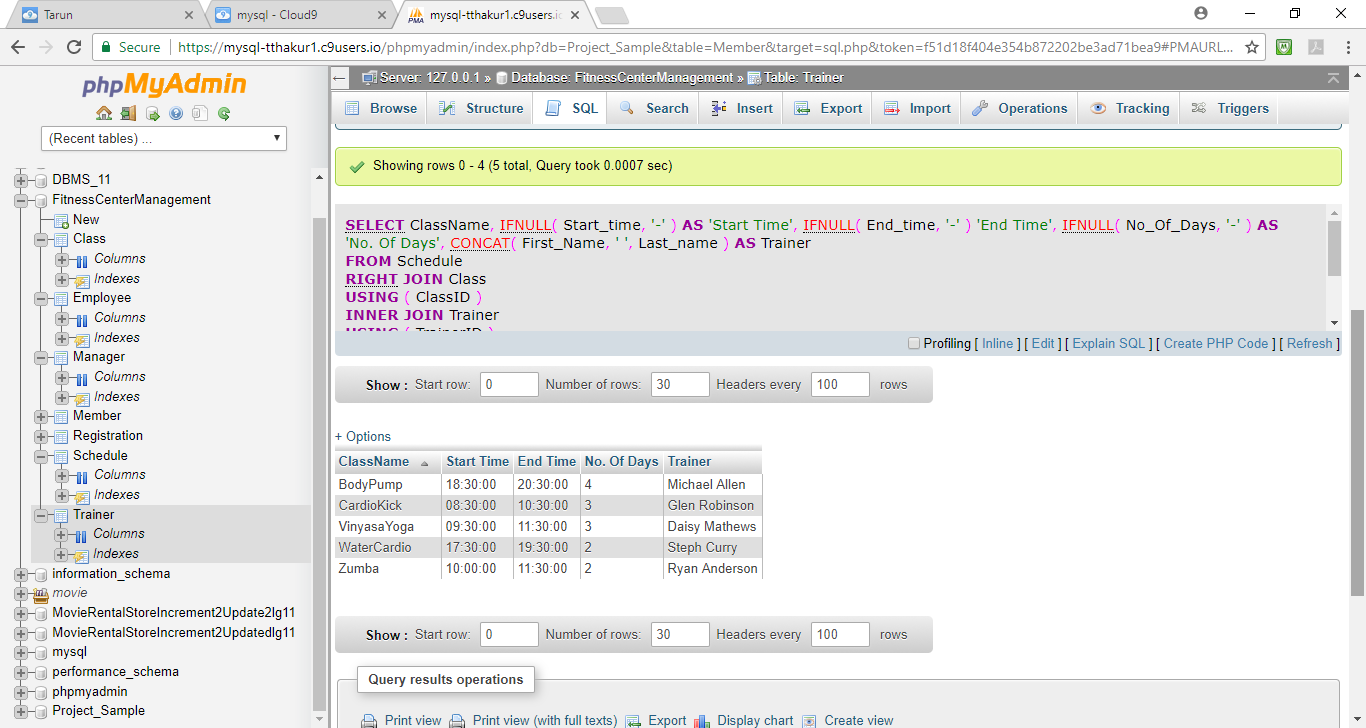
INNER JOIN Trainer

USING (TrainerID)

INNER JOIN Employee

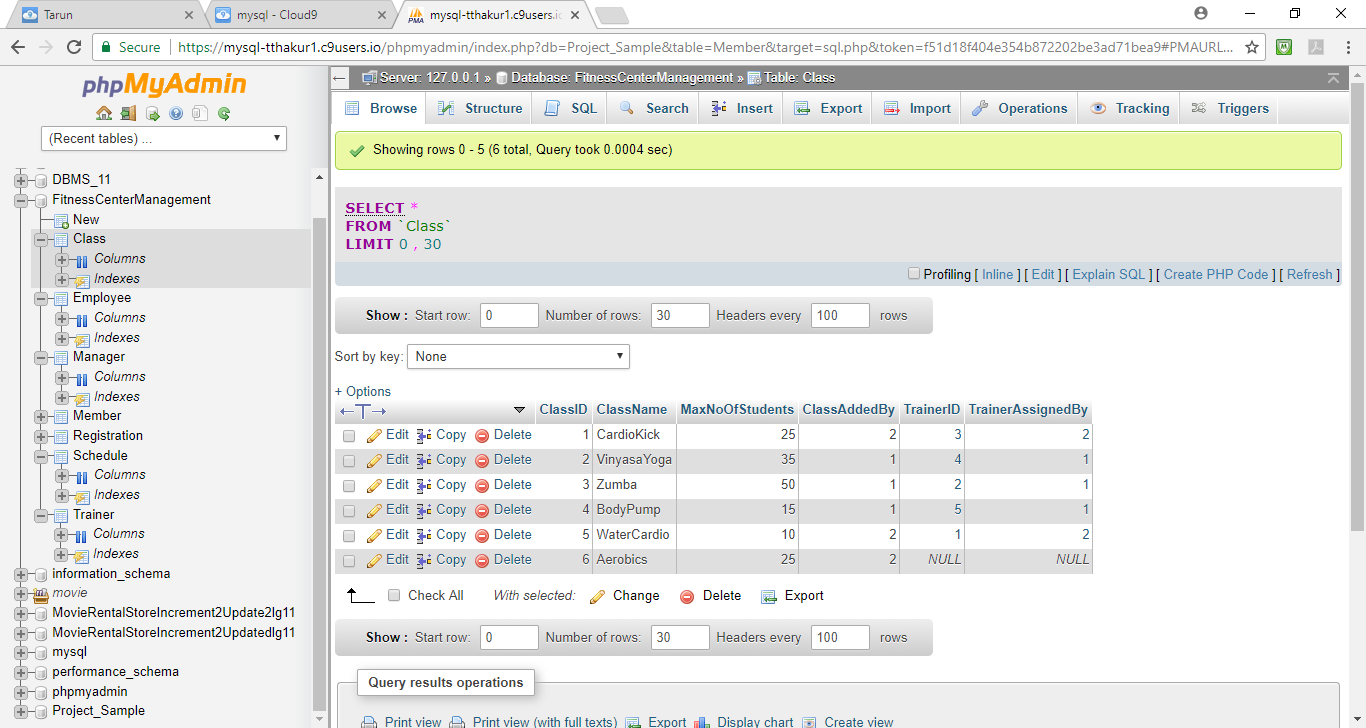
USING (EmployeeID)

ORDER BY ClassName;



2. As a Manager, I want to add a class(es).

INSERT INTO Class(ClassName, MaxNoOfStudents,ClassAddedBy) VALUES ('Aerobics',25,2);

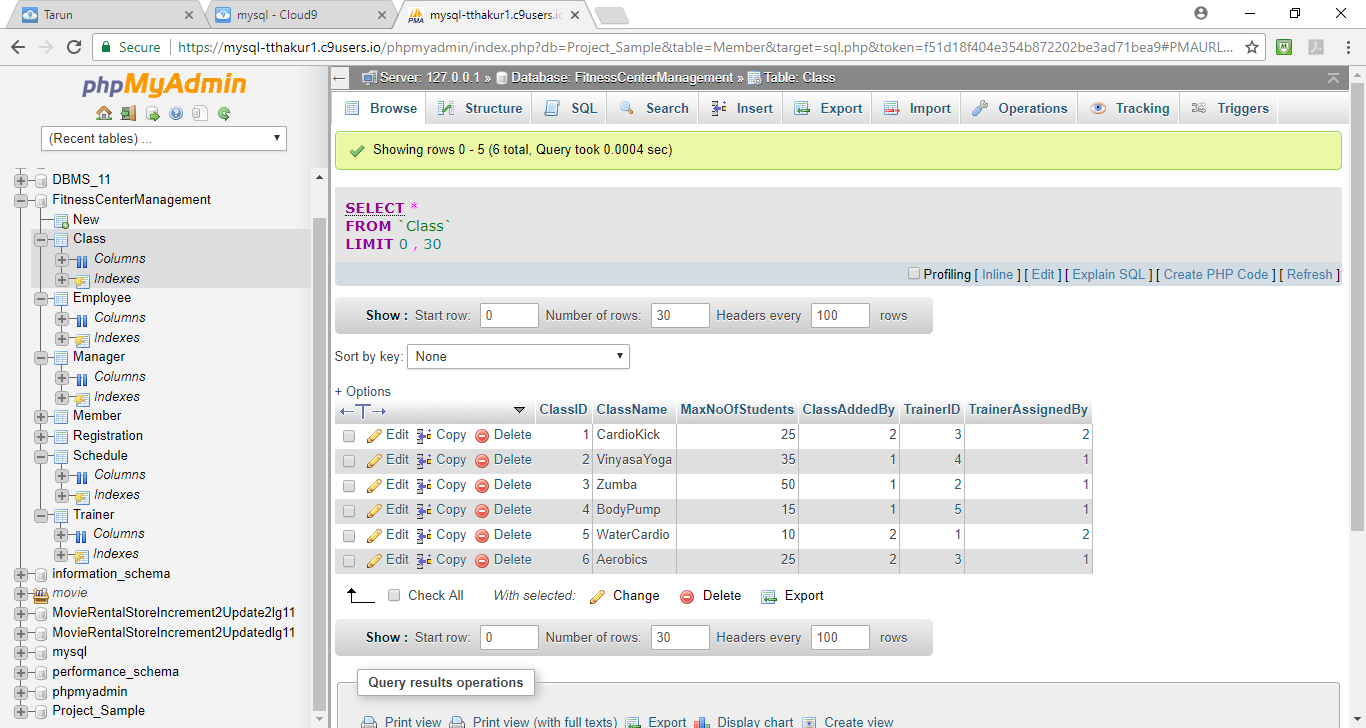


3. As a Manager, I should be able to assign trainer to the class(es).

UPDATE Class

SET TrainerID=3, TrainerAssignedBy=1

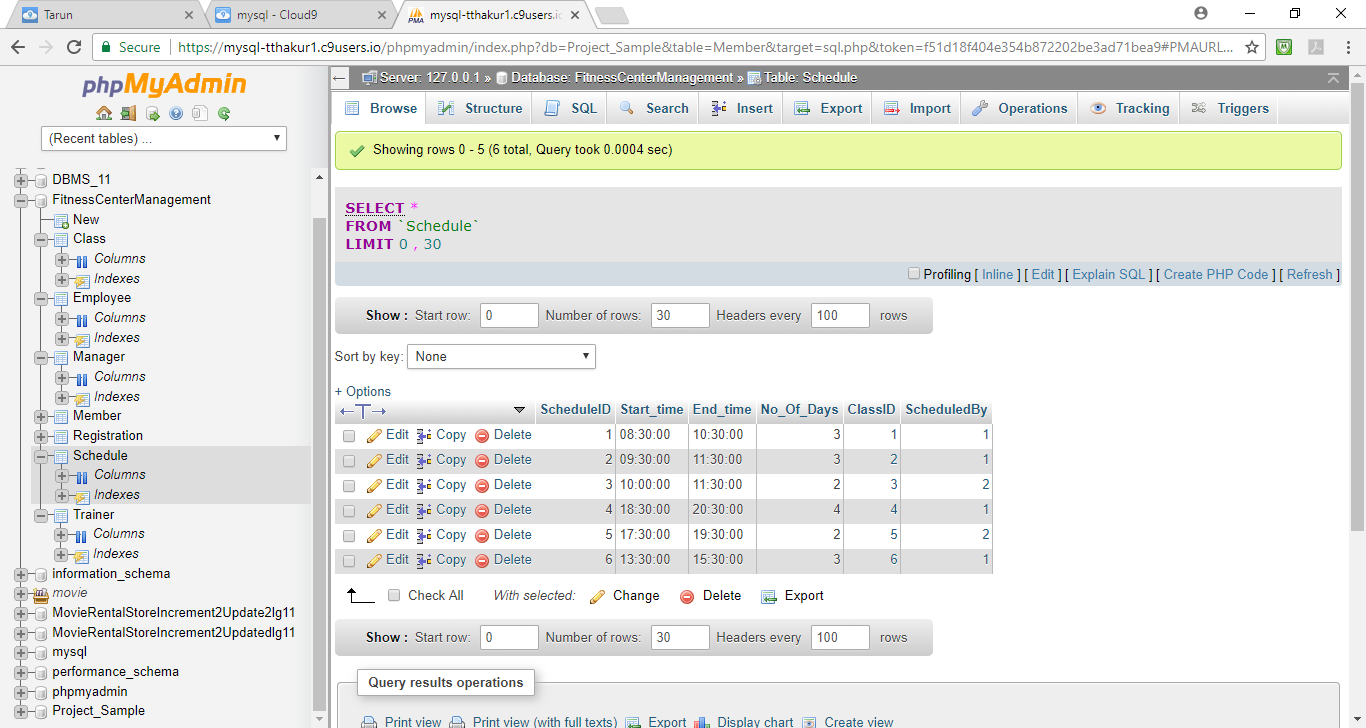
Where ClassName= 'Aerobics';



4. As a Manager, I want to schedule a class(es).

INSERT INTO Schedule (Start\_time,End\_time,No\_Of\_Days,ClassId,ScheduledBy)

VALUES ('13:30:00','15:30:00',3,6,1);



***Sprint 2***

## *REQUIREMENTS*

Below, User Stories marked yellow are for Sprint 1 and marked green are for Sprint 2.

|  |  |
| --- | --- |
| **Story ID** | **Story description** |
| US1 | As a Manager, I want to add a Fitness Course. |
| US2 | As a Manager, I want to remove a Fitness Course. |
| US3 | As a Manager, I should be able to assign trainer to the Fitness Course. |
| US4 | As a Manager, I want to schedule a Fitness Course. |
| US5 | As a Manager, I want to view all the members assigned to a Fitness Course. |
| US6 | As a Member, I want to view my weekly schedule. |
| US7 | As a Member, I want to view the trainer for the Fitness Course I am taking. |
| US8 | As a Member, I want to see all the Fitness Courses available at the fitness center so that I can choose a Fitness Course. |
| US9 | As a Trainer, I want to view my Course schedule. |
| US10 | As a Trainer, I should be able to view the profile of all the members signed up for a Fitness Course. |

## ***CONCEPTUAL DESIGN***

Entity: **Employee (generalization)**

Attributes:

EmployeeID

Name [composite]

Last\_name

First\_name

Address [composite]

Street

City

State

Zip\_code

Phone\_number

Email\_ID

Entity: **Manager (specialization of Employee)**

Attributes:

EmployeeID

Annual\_salary

Entity: **Trainer (specialization of Employee)**

Attributes:

EmployeeID

Hourly\_Salary

Hours\_Worked

Entity: **FitnessCourse**

Attributes:

FitnessCourseName

Type

MaxNoOfStudents

Entity: **WeeklySchedule**

Attributes:

Start\_time

End\_time

WeekDay

Entity: **Room**

Attributes:

RoomNumber

BuildingName

Entity: **Member**

Attributes:

Name [composite]

Last\_name

First\_name

Address [composite]

Street

City

State

Zip\_code

Phone\_number

Email

Relationship: **Manager** is an **Employee**

Manager is a disjoint specialization of the Entity Employee

Manager has total participation

Employee has partial participation

Relationship: **Trainer** is an **Employee**

Trainer is a disjoint specialization of the Entity Employee

Trainer has total participation

Employee has partial participation

Relationship: **Manager** adds **FitnessCourse**

Cardinality: Many to Many

Participation:

Manager has partial participation

FitnessCourse has total participation

Relationship: **Manager** assigns **Trainer**

Cardinality: Many to Many

Participation:

Manager has partial participation

Trainer has total participation

Relationship: **FitnessCourse** has **Trainer**

Cardinality: Many to Many

Participation:

Trainer has partial participation

FitnessCourse has total participation

Relationship: **FitnessCourse** is scheduled in a **Room**

Cardinality: Many to Many

Participation:

Room has partial participation

FitnessCourse has total participation

Relationship: **Manager** schedules a **FitnessCourse**

Cardinality: Many to Many

Participation:

Manager has partial participation

FitnessCourse has total participation

Relationship: **Member** registers to a **FitnessCourse**

Cardinality: Many to Many

Participation:

Member has partial participation

FitnessCourse has partial participation

Relationship: **FitnessCourse** has **Schedule**

Cardinality: Many to Many

Participation:

FitnessCourse has total participation

Schedule has partial participation

## ***LOGICAL DESIGN***

Table: **Employee**

Columns:

EmployeeID (primary key)

Last\_name

First\_name

Street

City

State

Zip\_code

Phone\_number

Email\_ID

Primary key justification: EmployeeID is unique for Employee so it is a good primary key.

Assumption: EmployeeID number is uniquely assigned by the system when an employee is added.

Table: **Mananger**

Columns:

ManagerID (primary key also foreign key referencing to the EmployeeID in Employee table)

Annual\_Salary

Primary key justification: ManagerID is the EmployeeID from the Employee table which is unique for each employee so it is a good primary key.

Table: **Trainer**

Columns:

TrainerID (primary key also foreign key referencing to the EmployeeID in Employee table)

Hourly\_Salary

Hours\_Worked

Primary key justification: TrainerID is the EmployeeID from the Employee table which is unique for each employee, so it is a good primary key.

Table: **FitnessCourse**

Columns:

FitnessCourseID

FitnessCourseName

Type

MaxNoOfStudents

FitnessCourseAddedBy(foreign key referencing to ManagerID in Manager table)

Primary key justification: FitnessCourseID is unique for each FitnessCourse so, it is a good primary key.

Assumption: FitnessCourseID number is uniquely assigned by the system when a new Course is created.

Table: **CourseTrainer**

Columns:

CourseTrainerID

FitnessCourseID (foreign key referencing to FitnessCourseID in FitnessCourse table)

TrainerID (foreign key referencing to TrainerID in Trainer table)

TrainerAssignedBy(foreign key referencing to ManagerID in Manager table)

Primary key justification: CourseTrainerID is unique for each Course and trainer combination so, it is a good primary key.

Table: **WeeklySchedule**

Columns:

WeeklyScheduleID

Start\_time

End\_time

Weekday

Primary key justification: WeeklyScheduleID is unique for each Schedule so it is a good primary key.

Table: **Room**

Columns:

RoomID

RoomNumber

BuildingName

Primary key justification: RoomID is unique for each Room so it is a good primary key.

Table: **CourseSchedule**

Columns:

CourseScheduleID

WeeklyScheduleID (foreign key referencing to WeeklyScheduleID in WeeklySchedule table)

FitnessCourseID (foreign key referencing to FitnessCourseID in FitnessCourse table)

RoomID (Foreign key referencing to RoomID in Room table)

ScheduledBy(foreign key referencing to ManagerID in Manager table)

Primary key justification: CourseScheduleID is unique for each class scheduled so it is a good primary key.

Table: **Member**

Column:

MemberID

First\_name

Last\_name

Street

City

State

Zip\_code

Phone\_number

Email\_ID

Primary key justification: MemberID is unique for each member so it is a good primary key.

Assumption: MemberID number is uniquely assigned by the system when a new member is added.

Table: **Registration**

Column:

RegistrationID

FitnessCourseID(foreign key referencing to FitnessCourseID in FitnessCourse table)

MemberID (foreign key referencing MemberID in Member table)

Registration\_Date

Primary key justification: RegistrationID is unique for each member so it is a good primary key.

Assumption: RegistrationID number is uniquely assigned by the system when a new member is added.

## ***VIEWS AND STORED PROGRAMS***

***Views:***

**View 1**: FitnessCourseScheduleDetails

Goal: The view contains the listing of all the fitness course schedule details including start and end time, day of week, and trainer so the member can choose a fitness course.

**View 2**: TotalMembersForFitnessCourse

Goal: The view contains a count of the number of members signed up for each fitness courses, so the manager can use the information to determine if they want to add a new fitness course or adjust fitness course schedules or trainers.

**View 3**: MemberListingForFitnessCourses

Goal: The view contains the listing of all the members signed up for all the fitness courses, so the manager can use the information to provide promotions to upgrade membership for frequent members.

***Stored Procedures:***

**Stored procedure 1**: GetFitnessCourseTypeListing

Parameters: Type

Goal: This will allow the fitness center manager to view all the fitness courses by course type to assist with fitness course addition decisions.

**Stored procedure 2**: GetTrainerFitnessCourses

Parameters: First\_Name,Last\_Name

Goal: This will provide the specific courses that are taken under specific trainers.

**Stored procedure 3**: GetMemberCourseDetails

Parameters: Email\_ID

Goal: This will allow the member to view all the fitness courses they are registered for along with the associated course details including start and stop times, day of week, and trainer by inputting their member Email ID.

**Stored procedure 4**: FitnessCourseAddition

Parameters: FitnessCourseName, Type, MaxNoOfStudents,

FitnessCourseAddedBy

Goal: This will allow the fitness center manager to add a new fitness course.

**Stored procedure 5**: FitnessCourseTrainerAssignment

Parameters: FitnessCourseName, TrainerEmailID

Goal: This will allow the fitness center manager to assign a trainer

course.

**Trigger**: N/A

**Event**: N/A

# ***Sprint 3 - Database design and implementation***

## ***REQUIREMENTS***

Below, User Stories marked yellow are for Sprint 1, marked turquoise are for Sprint 2 and marked green are for Sprint 3.

|  |  |
| --- | --- |
| Story ID | Story description |
| US1 | As a Manager, I want to add a Fitness Course. |
| US2 | As a Manager, I want to remove a Fitness Course. |
| US3 | As a Manager, I should be able to assign trainer to the Fitness Course. |
| US4 | As a Manager, I want to schedule a Fitness Course. |
| US5 | As a Manager, I want to view all the members assigned to a Fitness Course. |
| US6 | As a Member, I want to view my weekly schedule. |
| US7 | As a Member, I want to view the trainer for the Fitness Course I am taking. |
| US8 | As a Member, I want to see all the Fitness Courses available at the fitness center so that I can choose a Fitness Course. |
| US9 | As a Trainer, I want to view my Course schedule. |
| US10 | As a Trainer, I should be able to view the profile of all the members signed up for a Fitness Course. |

***CONCEPTUAL DESIGN***

Entity: **Employee (generalization)**

Attributes:

EmployeeID

Name [composite]

Last\_name

First\_name

Address\_Street

Phone\_number

Email\_ID

Entity: **Zipcode**

Attributes:

Zipcode

City

State

Entity: **Manager (specialization of Employee)**

Attributes:

EmployeeID

Annual\_salary

Entity: **Trainer (specialization of Employee)**

Attributes:

EmployeeID

Hourly\_Salary

Hours\_Worked

Entity: **FitnessCourse**

Attributes:

FitnessCourseName

MaxNoOfStudents

Entity: **FitnessCourseType**

Attributes:

FitnessCourseTypeID

TypeName

Entity: **WeeklySchedule**

Attributes:

Start\_time

End\_time

WeekDay

Entity: **Room**

Attributes:

RoomNumber

Entity: **Building**

Attributes:

BuildingName

Entity: **Member**

Attributes:

Name [composite]

Last\_name

First\_name

Address\_Street

Phone\_number

Email

Entity : **Status**

Attributes:

StatusID

StatusName

Relationship: **Employee** has a **Zipcode**

Cardinality: Many to One

Participation:

Employee has total participation

Zipcode has partial participation

Relationship: **Manager** is an **Employee**

Manager is a disjoint specialization of the Entity Employee

Manager has total participation

Employee has partial participation

Relationship: **Trainer** is an **Employee**

Trainer is a disjoint specialization of the Entity Employee

Trainer has total participation

Employee has partial participation

Relationship: **Employee** has a **Status**

Cardinality: Many to One

Participation:

Employee has total participation

Status has partial participation

Relationship: **Manager** adds/remove **FitnessCourse**

Cardinality: Many to Many

Participation:

Manager has partial participation

FitnessCourse has total participation

Relationship: **Manager** assigns **Trainer**

Cardinality: Many to Many

Participation:

Manager has partial participation

Trainer has total participation

Relationship: **FitnessCourse** has **FitnessCourseType**

Cardinality: Many to one

Participation:

FitnessCourseType has partial participation

FitnessCourse has total participation

Relationship: **FitnessCourse** has **Trainer**

Cardinality: Many to Many

Participation:

Trainer has partial participation

FitnessCourse has total participation

Relationship: **FitnessCourse** is scheduled in a **Room**

Cardinality: Many to Many

Participation:

Room has partial participation

FitnessCourse has total participation

Relationship: **Building** has a **Room**

Cardinality: Many to Many

Participation:

Building has total participation

Room has total participation

Relationship: **Manager** schedules a **FitnessCourse**

Cardinality: Many to Many

Participation:

Manager has partial participation

FitnessCourse has total participation

Relationship: **Member** registers to a **FitnessCourse**

Cardinality: Many to Many

Participation:

Member has partial participation

FitnessCourse has partial participation

Relationship: **Member** has a **Zipcode**

Cardinality: Many to One

Participation:

Member has total participation

Zipcode has partial participation

Relationship: **Member** has a **Status**

Cardinality: Many to One

Participation:

Member has total participation

Status has partial participation

Relationship: **FitnessCourse** has **Schedule**

Cardinality: Many to Many

Participation:

FitnessCourse has total participation

Schedule has partial participation

## ***LOGICAL DESIGN***

Table: **Employee**

Columns:

EmployeeID (primary key)

Last\_name

First\_name

Street

Zip\_code (foreign key referencing to Zip\_code in Zipcode table)

Phone\_number

Email\_ID

StatusID(foreign key referencing to StatusID in Status table)

Primary key justification: EmployeeID is unique for Employee so it is a good primary key.

Assumption: EmployeeID number is uniquely assigned by the system when an employee is added.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: EmployeeID

Justification: EmployeeID is a unique and not null column which can uniquely define each row in the employee table

Index 2: non-clustered

Columns: (First\_name, Last\_name)

Justification: Generally, employee details can be queried using the name of the Employee hence a non-clustered index is created on that.

Index 3: non-clustered

Columns: Zip\_code

Justification: Zip code can be used for queries to determine employee locations. Also, because it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Index 4: non-clustered

Columns: StatusID

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Table: **Zipcode**

Columns:

Zip\_code (primary key)

City

State

Primary key justification: Zip\_code is unique for a state and city so it is a good primary key.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: Zip\_code

Justification: Zip\_code is a unique and not null column which can uniquely define each row in the zipcode table

Index 2: non-clustered

Columns: City

Justification: Generally, city details can be queried using the name of the city hence a non-clustered index is created on that.

Index 3: non-clustered

Columns: State

Justification: Generally state details can be queried using the name of the state hence a non-clustered index is created on that.

Table: **Status**

Columns:

StatusID (primary key)

StatusName

Primary key justification: StatusID is unique for Status so it is a good primary key.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: StatusID

Justification: StatusID is a unique and not null column which can uniquely define each row in the employee table

Table: **Mananger**

Columns:

ManagerID (primary key also foreign key referencing to the EmployeeID in Employee table)

Annual\_Salary

Primary key justification: ManagerID is the EmployeeID from the Employee table which is unique for each employee so it is a good primary key.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: ManagerID

Justification: ManagerID is a unique and not null column which

can uniquely define each row in the manager table.

Index 2: non-clustered

Columns: Annual\_Salary

Justification: Generally, salary details can be queried using

salary buckets. Hence a non-clustered index is created on that.

Table: **Trainer**

Columns:

TrainerID (primary key also foreign key referencing to the EmployeeID in Employee table)

Hourly\_Salary

Hours\_Worked

Primary key justification: TrainerID is the EmployeeID from the Employee table which is unique for each employee, so it is a good primary key.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: TrainerID

Justification: TrainerID is a unique and not null column which can uniquely define each row in the Trainer table

Index 2: non-clustered

Columns: Hourly\_Salary

Justification: Generally total salary details can be queried using salary buckets, hence a non-clustered index is created on that.

Index 3: non-clustered

Columns: Hours\_Worked

Justification: Generally total salary details can be queried using salary buckets, hence a non-clustered index is created on that.

Table: **FitnessCourseType**

Columns:

FitnessCourseTypeID (primary key)

FitnessCourseTypeName

Primary key justification: FitnessCourseTypeID is unique for each FitnessCourseType so, it is a good primary key.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: FitnessCourseTypeID

Justification: FitnessCourseTypeID is a unique and not null column which can uniquely define each row in the Fitness Course Type table

Index 2: non-clustered

Columns: FitnessCourseTypeName

Justification: Generally, fitness course type details can be queried using fitness course type name hence a non-clustered index is created on that.

Table: **FitnessCourse**

Columns:

FitnessCourseID (primary key)

FitnessCourseName

FitnessCourseTypeID(foreign key referencing to FitnessCourseTypeID in FitnessCourseTypeID table)

MaxNoOfStudents

FitnessCourseAddedBy(foreign key referencing to ManagerID in Manager table)

Primary key justification: FitnessCourseID is unique for each FitnessCourse so, it is a good primary key.

Assumption: FitnessCourseID number is uniquely assigned by the system when a new Course is created.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: FitnessCourseID

Justification: FitnessCourseID is a unique and not null column which can uniquely define each row in the Fitness Course table

Index 2: non-clustered

Columns: FitnessCourseName

Justification: Generally, fitness course name details can be queried using fitness course name hence a non-clustered index is created on that.

Index 3: non-clustered

Columns: FitnessCourseAddedBy

Justification: Generally, fitness course added by details can be queried using manager ID hence a non-clustered index is created on that. Also, since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Index 4: non-clustered

Columns: FitnessCourseTypeID

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Table: **CourseTrainer**

Columns:

CourseTrainerID (primary key)

FitnessCourseID (foreign key referencing to FitnessCourseID in FitnessCourse table)

TrainerID (foreign key referencing to TrainerID in Trainer table)

TrainerAssignedBy(foreign key referencing to ManagerID in Manager table)

Primary key justification: CourseTrainerID is unique for each Course and trainer combination so, it is a good primary key.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: CourseTrainerID

Justification: CourseTrainerID is a unique and not null column which can uniquely define each row in the Fitness Course table.

Index 2: non-clustered

Columns: FitnessCourseID

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Index 3: non-clustered

Columns: TrainerID

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Index 4: non-clustered

Columns: TrainerAssignedBy

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Table: **WeeklySchedule**

Columns:

WeeklyScheduleID(primary key)

Start\_time

End\_time

Weekday

Primary key justification: WeeklyScheduleID is unique for each Schedule so it is a good primary key.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: WeeklyScheduleID

Justification: WeeklyScheduleID is a unique and not null column which can uniquely define each row in the WeeklySchedule table

Index 2: non-clustered

Columns: Start\_time, End\_time and Weekday

Justification: if a user wants to search a particular schedule based on the time or day this index will be helpful to improve the performance.

Table: **Building**

Columns:

BuildingID (primary key)

BuildingName

Primary key justification: BuildingID is unique for each Building so it is a good primary key.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: BuildingID

Justification: BuildingID is a unique and not null column which can uniquely define each row in the Building table

Index 2: non-clustered

Columns: BuildingName

Justification: Generally building name details can be queried using building name hence a non-clustered index is created on that.

Table: **Room**

Columns:

RoomID (primary key)

RoomNumber

BuildingID (foreign key referencing to the BuildingID in Building table)

Primary key justification: RoomID is unique for each Room so it is a good primary key.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: RoomID

Justification: RoomID is a unique and not null column which can uniquely define each row in the Building table

Index 2: non-clustered

Columns: BuildingID

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Table: **CourseSchedule**

Columns:

CourseScheduleID(primary key)

WeeklyScheduleID (foreign key referencing to WeeklyScheduleID in WeeklySchedule table)

FitnessCourseID (foreign key referencing to FitnessCourseID in FitnessCourse table)

TrainerID (foreign key referencing to TrainerID in Trainer table)

RoomID (Foreign key referencing to RoomID in Room table)

ScheduledBy(foreign key referencing to ManagerID in Manager table)

Primary key justification: CourseScheduleID is unique for each class scheduled so it is a good primary key.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: CourseScheduleID

Justification: CourseScheduleID is a unique and not null column which can uniquely define each row in the Course Schedule table.

Index 2: non-clustered

Columns: WeeklyScheduleID

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Index 3: non-clustered

Columns: FitnessCourseID

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Index 4: non-clustered

Columns: RoomID

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Index 5: non-clustered

Columns: TrainerID

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Index 4: non-clustered

Columns: ScheduledBy

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Table: **Member**

Column:

MemberID (primary key)

First\_name

Last\_name

Street

Zip\_code(foreign key referencing to Zip\_code in Zipcode table)

Phone\_number

Email\_ID

StatusID((foreign key referencing to StatusID in Status table)

Primary key justification: MemberID is unique for each member so it is a good primary key.

Assumption: MemberID number is uniquely assigned by the system when a new member is added.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: MemberID

Justification: MemberID is a unique and not null column which can uniquely define each row in the member table.

Index 2: non-clustered

Columns: Email\_ID

Justification: Generally, member details can be queried using the Email\_ID of the member, hence a non-clustered index is created on that.

Index 3: non-clustered

Columns: Zip\_code

Justification: Zip code can be used for queries to determine employee locations. Also, because it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Index 4: non-clustered

Columns: StatusID

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Table: **Registration**

Column:

RegistrationID (primary key)

FitnessCourseID(foreign key referencing to FitnessCourseID in FitnessCourse table)

MemberID (foreign key referencing MemberID in Member table)

Registration\_Date

Primary key justification: RegistrationID is unique for each member so it is a good primary key.

Assumption: RegistrationID number is uniquely assigned by the system when a new member is added.

Highest normalization level: 4NF (we do not have an, trivial multivalued dependencies hence, it is in 4NF.)

Indexes:

Index 1: clustered

Columns: RegistrationID

Justification: RegistrationID is a unique and not null column which can uniquely define each row in the Registration table.

Index 2: non-clustered

Columns: FitnessCourseID

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.

Index 3: non-clustered

Columns: MemberID

Justification: Since it is a foreign key, an index should be created to simplify joins and when the primary key is updated, it will be easy to update the foreign key.