University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 2

Date: 10/20/2022

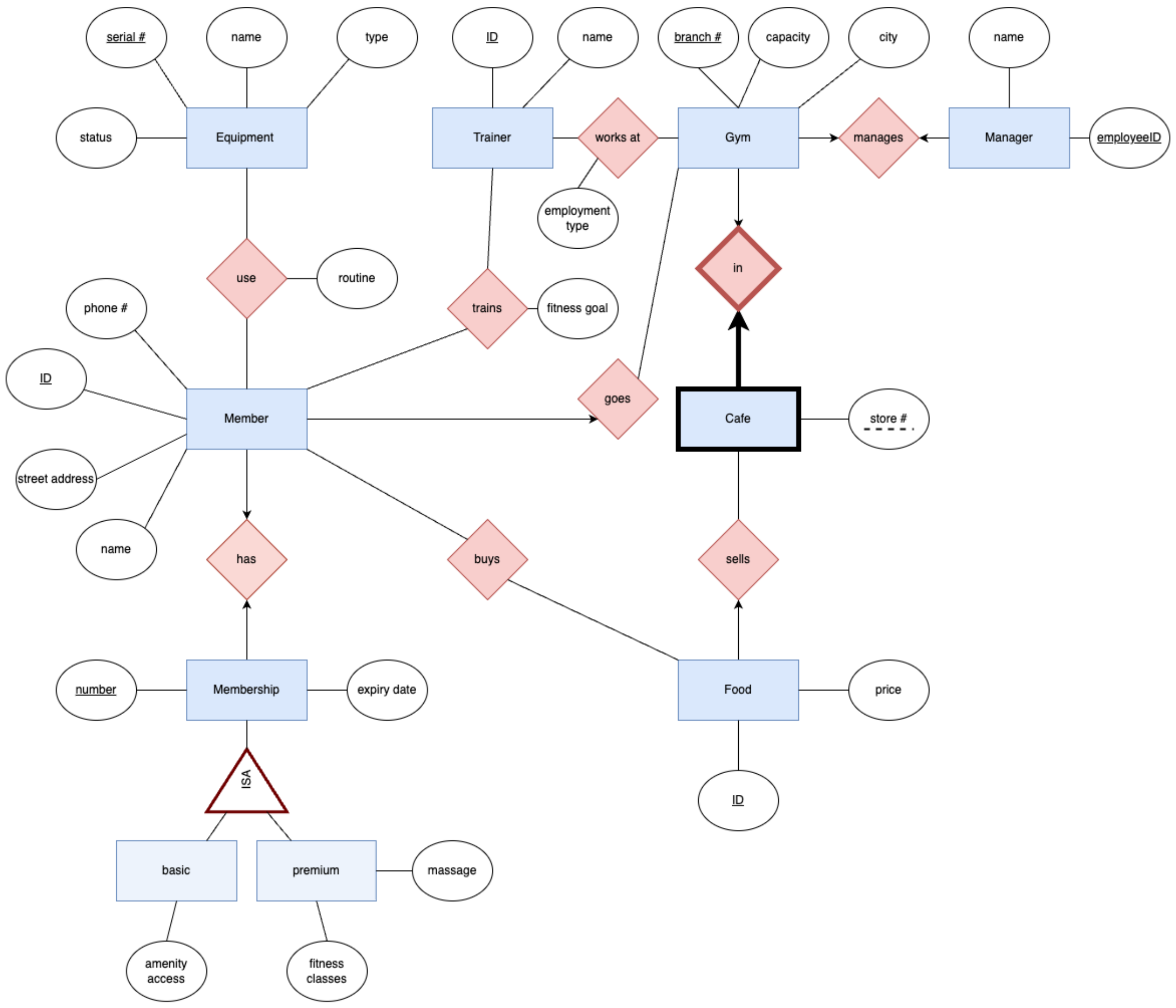
Group Number: 71

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Clara Kim	51832228	p2h3b	Kclara0302@gmail.com
Helyn Walther	44738599	Z0k4s	Helyn.walther@yahoo.com
Celina Xiong	51788099	B6a3b	Celinaxiong0321@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

1. A completed cover page (template on Canvas)
2. The ER diagram you are basing your item #3 (below) on. This ER diagram may be the same as your milestone 1 submission or it might be different. If you have made changes from the version submitted in milestone 1, attach a note indicating what changes have been made and why.



Note: we added an entity Manager because we were told in M1 that we are missing an entity.

3. The schema derived from your ER diagram (above). For the translation of the ER diagram to the relational model, follow the same instructions as in your lectures. The process should be reasonably straightforward. For each table:
    a. List the table definition (e.g., Table1(attr1: domain1, attr2: domain2, ...))
    b. Specify the primary key (PK), candidate key, (CK) foreign keys (FK), and other constraints that the table must maintain.

Gym(

BRANCHNUMBER: INTEGER **Primary Key**,
capacity: INTEGER,
city: CHAR[20],
ManagerID: INTEGER,
Foreign Key (ManagerID) **REFERENCES** Manager(EMPLOYEEID)

Member(

ID: INTEGER **Primary Key**,
phoneNumber: INTEGER,
streetAddress: CHAR[20],
name: CHAR[20],
membershipNumber: INTEGER,
branchNumber: INTEGER,
Foreign Key(membershipNumber) **REFERENCES** Membership(NUMBER),
Foreign Key(branchNumber) **REFERENCES** Gym(BRANCHNUMBER))

Trainer(

ID: INTEGER **Primary Key**,
name: CHAR[20])

Equipment(

SERIALNUMBER: INTEGER **Primary Key**,
name: CHAR[20],
type: CHAR[20],
status: CHAR[20])

Cafe(

STORENUMBER: INTEGER,
BRANCHNUMBER: INTEGER,
Primary Key(BRANCHNUMBER, STORENUMBER),
Foreign Key(BRANCHNUMBER) **REFERENCES** Gym(BRANCHNUMBER))

Food(

ID: INTEGER **Primary Key**,
price: FLOAT
storeNumber: INTEGER
branchNumber: INTEGER
Foreign Key(storeNumber) **REFERENCES** Cafe(STORENUMBER)
Foreign Key(branchNumber) **REFERENCES** Cafe(BRANCHNUMBER))

Membership(

NUMBER: INTEGER **Primary Key**,
expiryDate: CHAR[10],

memID: INTEGER,
amenityAccess: BOOLEAN,
hasMassage: BOOLEAN,
hasFitnessClass: BOOLEAN,
Foreign Key(memID) **REFERENCES** Member(ID))

Buys(
membershipID: INTEGER,
foodID: INTEGER,
Foreign Key(membershipID) **REFERENCES** Member(ID),
Foreign Key(foodID) **REFERENCES** FOOD(ID))

Basic(
<u>NUMBER:</u> INTEGER,
expiryDate: CHAR[10],
amenityAccess: BOOLEAN);

Premium(
hasMassage: BOOLEAN,
hasFitnessClass: BOOLEAN);

Works At(
branchID: INTEGER,
trainerID: INTEGER,
employmentType: CHAR[20],
Foreign Key(branchID) **REFERENCES** Gym(BRANCHNUMBER),
Foreign Key(trainerID) **REFERENCES** Trainer(ID));

Trains(
trainerID: INTEGER,
memberID: INTEGER,
Foreign Key(trainerID) **REFERENCES** Trainer(ID),
Foreign Key(memberID) **REFERENCES** Member(ID));

Use(
routine: CHAR[20],
serialNumber: INTEGER
memberID: INTEGER
Foreign Key(serialNumber) **REFERENCES** Equipment(SERIALNUMBER)
Foreign Key(memberID) **REFERENCES** Member(ID));

Manager(
<u>EmployeeID</u>: INTEGER,

Name: CHAR[20],
gymNumber: INTEGER,
Foreign Key(gymNumber) **REFERENCES** Gym(BRANCHNUMBER));

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

Note: In your list of FDs, there must be some kind of valid FD other 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.

R(BRANCHNUMBER -> capacity, city)
R(SERIALNUMBER -> status, name, type)
R(ID -> name)
R(EMPLOYEEID -> name)
R(ID -> phone number, street address, name)
R(NUMBER -> expiry date)
R(STOREID -> branch number, capacity)

Your design must go through a normalization process.

5. 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.

R1(BRANCHNUMBER -> capacity, city, managerID)
R1 is in BCNF because there are no other non-trivial relationships. The superkey determines all other attributes, CK, and FK.
Gym(
    <u>BRANCHNUMBER</u>: INTEGER **Primary Key**,
    capacity: INTEGER,
    city: CHAR[20],
    managerID: INTEGER,
    Foreign Key (managerID) **REFERENCES** Manager(EMPLOYEEID)

R1(ID -> phoneNumber, streetAddress, name, membership, branchNumber)
R1 is in BCNF because there are no other non-trivial relationships. The superkey determines all other attributes, CK, and FK.

Member(

> ID: INTEGER **Primary Key**,
> phoneNumber: INTEGER,
> streetAddress: CHAR[20],
> name: CHAR[20],
> membershipNumber: INTEGER,
> branchNumber: INTEGER,
> **Foreign Key(**membershipNumber) **REFERENCES** Membership(NUMBER),
> **Foreign Key**(branchNumber) **REFERENCES** Gym(BRANCHNUMBER))

Member:

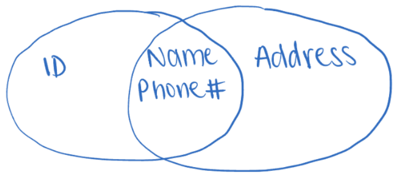
ID → Name

Name, Phone# → Address

↳ Not in BCNF

Decomp:

x → b



R_1(Name, Phone#, Address)

R_2(ID, Phone#, Name)

R1(ID -> name)
R1 is in BCNF because there are no other non-trivial relationships. The superkey determines all other attributes, CK, and FK.

Trainer(

> ID: INTEGER **Primary Key**,
> name: CHAR[20])

R1(SERIALNUMBER -> name, type, status)
R1 is in BCNF because there are no other non-trivial relationships. The superkey determines all other attributes, CK, and FK.

Equipment(
SERIALNUMBER: INTEGER **Primary Key**,
name: CHAR[20],
type: CHAR[20],
status: CHAR[20])

R1(STORENUMBER -> BRANCHNUMBER)
R1(BRANCHNUMBER -> STORENUMBER)
R1 is in BCNF because there are only two attributes that both determine one another. There are no other non-trivial relationships.
Cafe(
STORENUMBER: INTEGER,
BRANCHNUMBER: INTEGER,
Primary Key(BRANCHNUMBER, STORENUMBER),
Foreign Key(BRANCHNUMBER) **REFERENCES** Gym(BRANCHNUMBER))

R1(ID -> price, storeNumber, branchNumber)
R1 is in BCNF because there are no other non-trivial relationships. The superkey determines all other attributes, CK, and FK.
Food(
ID: INTEGER **Primary Key**,
price: FLOAT
storeNumber: INTEGER
branchNumber: INTEGER
Foreign Key(storeNumber) **REFERENCES** Cafe(STORENUMBER)
Foreign Key(branchNumber) **REFERENCES** Cafe(BRANCHNUMBER))

R1(NUMBER -> expiryDate, memID, amenityAccess, hasMassage, hasFitnessClass)
R1 is in BCNF because there are no other non-trivial relationships. The superkey determines all other attributes, CK, and FK.
Membership(
NUMBER: INTEGER **Primary Key**,
expiryDate: CHAR[10],
memID: INTEGER,
amenityAccess: BOOLEAN,
hasMassage: BOOLEAN,
hasFitnessClass: BOOLEAN,
Foreign Key(memID) **REFERENCES** Member(ID))

R1(EmployeeID -> name, gymNumber)
R1 is in BCNF because there are no other non-trivial relationships. The superkey determines all other attributes, CK, and FK.
Manager(

```
        EmployeeID: INTEGER,
        name: CHAR[20],
        gymNumber: INTEGER,
        Foreign Key(gymNumber) REFERENCES Gym(BRANCHNUMBER));
```

6. The SQL DDL statements required to create all the tables from item #5. The statements should use the appropriate foreign keys, primary keys, UNIQUE constraints, etc.

```
CREATE TABLE Gym(
        BRANCHNUMBER: INTEGER Primary Key,
        capacity: INTEGER,
        city: CHAR[20],
        ManagerID: INTEGER,
        Foreign Key (ManagerID) REFERENCES Manager(EMPLOYEEID)

CREATE TABLE Member(
        ID: INTEGER Primary Key,
        phoneNumber: INTEGER,
        streetAddress: CHAR[20],
        name: CHAR[20],
        membershipNumber: INTEGER,
        branchNumber: INTEGER,
        Foreign Key(membershipNumber) REFERENCES Membership(NUMBER),
        Foreign Key(branchNumber) REFERENCES Gym(BRANCHNUMBER))

CREATE TABLE Trainer(
        ID: INTEGER Primary Key,
        name: CHAR[20])

CREATE TABLE Equipment(
        SERIALNUMBER: INTEGER Primary Key,
        name: CHAR[20],
        type: CHAR[20],
        status: CHAR[20])

CREATE TABLE Cafe(
        STORENUMBER: INTEGER,
        BRANCHNUMBER: INTEGER,
        Primary Key(BRANCHNUMBER, STORENUMBER),
        Foreign Key(BRANCHNUMBER) REFERENCES Gym(BRANCHNUMBER))

CREATE TABLE Food(
        ID: INTEGER Primary Key,
```

price: FLOAT
storeNumber: INTEGER
branchNumber: INTEGER
Foreign Key(storeNumber) **REFERENCES** Cafe(STORENUMBER)
Foreign Key(branchNumber) **REFERENCES** Cafe(BRANCHNUMBER))

CREATE TABLE Membership(
<u>NUMBER</u>: INTEGER **Primary Key**,
expiryDate: CHAR[10],
memID: INTEGER,
amenityAccess: BOOLEAN,
hasMassage: BOOLEAN,
hasFitnessClass: BOOLEAN,
Foreign Key(memID) **REFERENCES** Member(ID))

CREATE TABLE Manager(
<u>EmployeeID</u>: INTEGER,
Name: CHAR[20],
gymNumber: INTEGER,
Foreign Key(gymNumber) **REFERENCES** Gym(BRANCHNUMBER));

7. 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 on.

INSERT INTO Member
(ID, phoneNumber, streetAddress, name)
VALUES
(1, 888-123-4567, 123 Some Street, Jane Doe),
(2, 888-111-2222, 223 Test Avenue, John Dee),
(3, 888-123-1212, 990 Fake Street, Phil Frank),
(4, 888-199-8823, 102 Random Boulevard, Sara Jones),
(5, 888-222-2277, 98 Hello World, Meredith Grey)

INSERT INTO Manager
(employeeID, name)
VALUES
(100, Jim Gym),
(101, Heather Gray),
(102, Mark Sloan),
(103, Steve Jobs),
(104, Bailey Graham)

INSERT INTO Gym

```
(branchnumber, capacity, city)
VALUES
(111, 100, Vancouver),
(222, 80, Victoria),
(333, 200, Toronto),
(444, 50, Medicine Hat),
(555, 95, Calgary)

INSERT INTO Equipment (SerialNumber, Name, Type, Status)
VALUES
          (12345, Dumbbell, Weight, Available),
          (13131, Kettlebell, Weight, In Use),
          (23232, Treadmill, Cardio, Available),
          (45677, Stairmaster, Cardio, Available),
          (23432, Rowing Machine, Cardio, In Use);

INSERT INTO Food (ID, Price)
VALUES
          (183645, 3),
          (93716, 3),
          (9374, 15),
          (13645, 8),
          (532, 7);

INSERT INTO Trainer (ID, name)
VALUES
          (9274659, Jeff Gates),
          (9274660, Bill Jobs),
          (9274699, Jeff Bezoz),
          (9274640, Steve Jobs),
          (9274629, Bill Gates);

INSERT INTO Cafe(StoreID)
VALUES
          (329857),
          (398470),
          (329857),
          (398470),
          (273463);

INSERT INTO Basic (AmenityAccess)
VALUES
          (true),
          (false),
        (false),
        (false),
        (true);

INSERT INTO Premium (FitnessClasses, Massage)
VALUES
        (true, true),
        (true, true),
        (true, false),
        (false, false),
        (false, true);
```