# Name of Your Company

Fetch my Data

# Project Title

Students Clubs Management

# Team

* Yan Fung Yenny Hou (MySQL Expert)
* Hector Onato (MS SQL Server Expert)

# Weekly Meeting Hours

We will meet and work on the project every Wednesday from 2:30pm to 4:30pm.

# Project Description

There is a database which holds data about student clubs. Examples of clubs are sport clubs, religious clubs, programming club, computer networks club, database club, music club. Students based on their interest join those clubs. Each club can have several groups. Students join groups and not clubs. We store start date and end date of membership in each group that a student joins.

Members of the group can be the students currently registered at the school or they can be an alumnus. Each member is identified by the student ID and store their name in the database. One member can be part of many groups of a club or member of many groups in many clubs. A member may lead several clubs and may also be the head of the group of the club.

If a member is alumnus, we need to store his/her work history information (name of company, position, start date, end date). A person may join a company in different time, i.e., a person may join a company, leave it, and join in again several years later.

Each club is identified by club ID, we store its name and its lead. And each group is identified by group ID, we store its name and its head. Each group organizes some events. Each event is organized by a group, and is identified by the event ID. An event has a subject, date, time, location (room number, floor, building) and registration fee.

Groups may work on some funded projects. A project has a project code, name and how much money (fund or budget) a project receives. The fund is divided and paid to the members who contributed on the project, and store the portion of money that each member receives due to working on that project.

# Assumptions about Cardinality and Participations

You can write all the assumptions about Cardinality and Participations (total/partial) here.

* A member can join at least one or more groups, and a group can be joined by one or more members.
* A member may lead more than one club, and a club must only be led by one member.
* A member may be the head of many groups, and a group must have only one head.
* A group can organize zero or many events, and an event is organized by only one group.
* A group may work on zero or many funded projects, and a project must be handled by only one group.
* A student may work on zero or many funded projects, and a project can be handled by one or more students.

# EER Modeling Diagram

In the following drawing canvas, EER Modeling shapes have been provided. You can copy and replicate them (Ctrl+C to copy and Ctrl+V to paste. You can also select a shape, then press Ctrl button and drag and drop to copy a shape) and edit them to build your diagram.



Event

Club

M

1

organizes

1

M

joins

Member

N

M

Alumnus

Student

M

1

N

M

Project

works on

manages

Group

has

heads

leads

1

M

M

1

# ER-Model Mapping to Database Relational Schema

The relational Schema is written here:

Member(MemberID, firstName, lastName, MemberType)

Project(ProjectCode, name, budget)

Club(ClubID, **MemberID,** ClubName)

Club\_Group(**ClubID**, GroupID, **MemberID,** GroupName) // weak Relationship

Event(EventID, (**ClubID**, **GroupID)**, subject, date, time, registrationFee, building, floor room)

Group\_Manages\_Project(**ProjectCode, (ClubID, GroupID))**

Member\_Joins\_Group(**MemberID, (ClubID, GroupID))**

Member\_Works\_On(**MemberID, ProjectCode**, MemberPortion)

Alumnus\_WorkHistory(**AlumnusID**, Company, Position, StartDate, EndDate)

# Normalization

All relations must be normalized up to BCNF. You must explain why you believe every relation in your database in normalized.

**Relation:** Member(MemberID, Name, MemberType)

**FD(s):** MemberID -> {Name, MemberType}

**1NF**: Each member can only have one name, so *Name* is an atomic attribute and we do not store several names in this attribute.

We assume *MemberType* is atomic too, a single-valued character to store whether the member is a student or alumnus.

So it is in 1NF.

**2NF**: The non-prime attributes, i.e., *Name*, *MemberType* are fully dependent on *MemberID*. So it is in 2NF.

**3NF**: There is no transitive dependency for non-prime attributes. So it is in 3NF.

**BCNF**: Non-prime attributes *Name and MemberType* cannot determine the key *MemberID*. So it is in BCNF.

**Relation:** Club(ClubID, Name, Lead)

**FD(s):** ClubID -> {Name, Lead}

Name -> ClubID

**1NF**: Each club can only have one name, and we assume it only has one lead. Both *Name* and *Lead* are atomic. So it is in 1NF.

**2NF**: Non-prime attributes *Name* and *Lead* are fully dependent on *ClubID*. So it is in 2NF.

**3NF**: There is no transitive dependency for non-prime attributes *Name* and *Lead*. So it is in 3NF.

**BCNF**: Non-prime attributes *Lead* cannot determine the key *ClubID*. *Name* is either a unique attribute, it is a candidate key. The FD *Name* -> *ClubID* does not violate BCNF, so it is in BCNF.

**Relation:** Club\_Group(GroupID, **ClubID**, Name, Head)

**FD(s):** {ClubID, GroupID} -> {Name, Lead}

{GroupID} -> {Name, Lead}

**1NF**: For each group of each club can only have one name and a head, and both *Name* and *Head* are atomic. So it is in 1NF.

**2NF**: Non-prime attributes *Name* and *Head* are fully dependent on the composite key {*ClubID* and *GroupID*}. However, there is another FD *GroupID* -> {*Name*, *Lead*}, which is a partial dependency to the composite key, i.e., *GroupID*, but the *GroupID* can also be chosen as the candidate key (also the primary) of this relation instead of composite key. Therefore we have two options:

Option 1: Keep the composite key {*ClubID*, *GroupID*} as primary key

If we keep the composite key, FD *GroupID* -> {*Name*, *Lead*} is a partial dependency. Therefore we need to split the relation into two separate relations in order to fulfil the definition of 2NF as below:

Club\_Group(**GroupID**, **ClubID**)

Group(GroupID, Name, Head)

Option 2: Use *GroupID* as the primary key

If we change the primary key, the non-prime attributes are fully dependent on GroupID. So it will be in 2NF and renamed as below:

Group(GroupID, **ClubID**, Name, Head)

According to our assumptions: "A club has at least one group, and a group must belong to only one club.", we choose option 2 over option 1. It is because option 1 can support a many-to-many relationship between Club and Group, which we do not need in this case. Another reason is that option 2 will only create one relation, i.e., one relation fewer than option 1, which can eliminate the unnecessary table join in data retrieval in the future. Therefore, the relation will use option 2 and be renamed to below in 2NF:

Group(GroupID, **ClubID**, Name, Head)

**3NF**: There is no transitive dependency for non-prime attributes *ClubID*, *Name* and *Head*. So it is in 3NF.

**BCNF**: Non-prime attributes *ClubID* and *Head* cannot determine *GroupID*. *Name* cannot determine the key as well since different club may have group in the same name, e.g. "Group1". So it is in BCNF.

**Result:** The relation after normalization to BCNF will become as below:

Group(GroupID, **ClubID**, Name, Head)

**Relation:** Event(EventID, Subject, Date, Time, RegistrationFee, Room, Building, Floor, **GroupID**)

**FD(s):** EventID -> {Subject, Date, Time, RegistrationFee, Room, Building, Floor, GroupID}

**1NF**: We assume all non-prime attributes, i.e., *Subject, Time, RegistrationFee, Room, Building, Floor, GroupID* are atomic and can have only one value and none of them is multivalued. So it is in 1NF.

**2NF**: Non-prime attributes are fully dependent on the key *EventID*. So it is in 2NF.

**3NF**: There is no transitive dependency for non-prime attributes. So it is in 3NF.

**BCNF**: Non-prime attributes cannot determine key *EventID*. So it is in BCNF.

**Relation:** Project(ProjectCode, Name, Budget, **GroupID**)

**FD(s):** ProjectCode -> {Name, Budget, GroupID}

**1NF**: We assume all non-prime attributes, i.e., *Name, Budget, GroupID* are atomic and can have only one value and none of them is multivalued. So it is in 1NF.

**2NF**: Non-prime attributes are fully dependent on the key *ProjectCode*. So it is in 2NF.

**3NF**: There is no transitive dependency for non-prime attributes. So it is in 3NF.

**BCNF**: Non-prime attributes cannot determine key *ProjectCode*. So it is in BCNF.

**Relation:** Member\_Joins\_Group(**MemberID, GroupID**)

**FD(s):** {MemberID, GroupID} -> {MemberID, GroupID}

**1NF**: There is no non-prime attribute, and *MemberID* and *GroupID* are atomic. So it is in 1NF.

**2NF**: No non-prime attribute and composite key *MemberID* and *GroupID* can always fully determine itself. So it is in 2NF.

**3NF**: No non-prime attribute and no transitive dependency. So it is in 3NF.

**BCNF**: No non-prime attribute. So it is in BCNF.

**Relation:** Member\_WorksOn\_Project(**MemberID, ProjectCode**, MemberPortion)

**FD(s):** {MemberID, ProjectCode} -> MemberPortion

**1NF**: We assume non-prime attribute *MemberPortion* is atomic and can have only one value. So it is in 1NF.

**2NF**: Non-prime attribute *MemberPortion* is fully dependent on composite key {*MemberID*, *ProjectCode*}. So it is in 2NF.

**3NF**: There is no transitive dependency for non-prime attribute *MemberPortion*. So it is in 3NF.

**BCNF**: Non-prime attribute *MemberPortion* cannot determine the key composite key {*MemberID*, *ProjectCode*}. *S*o it is in BCNF.

**Relation:** Alumnus\_WorkHistory(**MemberID,** Company, Position, StartDate, EndDate)

**FD(s):** MemberID -> {Company, Position, StartDate, EndDate}

**1NF**: We assume all non-prime attributes, i.e., *Company, Position, StartDate, EndDate* are atomic and can have only one value and none of them is multivalued. So it is in 1NF.

**2NF**: Non-prime attributes are fully dependent on the key *MemberID*. So it is in 2NF.

**3NF**: There is no transitive dependency for non-prime attributes. So it is in 3NF.

**BCNF**: Non-prime attributes cannot determine key *MemberID*. So it is in BCNF.

After the normalization to BCNF, the relations are adjusted to below:

Member(MemberID, Name, MemberType)

Club(ClubID, Name, Lead)

Group(GroupID, **ClubID**, Name, Head)

Event(EventID, Subject, Date, Time, RegistrationFee, Room, Building, Floor, **GroupID**)

Project(ProjectCode, Name, Budget, **GroupID**)

Member\_Joins\_Group(**MemberID, GroupID**)

Member\_WorksOn\_Project(**MemberID, ProjectCode**, MemberPortion)

Alumnus\_WorkHistory(**MemberID,** Company, Position, StartDate, EndDate)

# Determining Data Types (Domain) and Constraints

You explain why you choose a certain data type for a field and why you apply certain constraints

# Creating Database and Tables - SQL DDL

You do not need to copy SQL commands here. Save your SQL commands in a script file and just mention the name of the file here. Make sure the script file is stored besides this document within the same folder.

# Inserting Values in Tables

You do not need to copy SQL commands here. Save your SQL commands in a script file and just mention the name of the file here. Make sure the script file is stored beside this document within the same folder.

# SQL Queries

You do not need to copy SQL commands here. Save your SQL commands in a script file and just mention the name of the file here. Make sure the script file is stored beside this document within the same folder.

# Views

You do not need to copy SQL commands here. Save your SQL commands in a script file and just mention the name of the file here. Make sure the script file is stored beside this document within the same folder.