Part A: The ER Model

Project Name: College Club Management System

Team Members: Ferduse Rahman, Maryam Tawfik, Kamila Karshieva

Team ID: 5

Task A: Draw an ER diagram

The ER diagram should reflect the submitted proposal. You should start by stating the business rules (requirements) and relations you describe in your proposal, and then add (or modify) the entities if required. You need to identify the entities, their attributes and how are those related (relationships).

For this project, you are only allowed to use the Crow's Foot notation to model the entities, attributes, and relationships. Identify any weak entities. Is their any class hierarchy in the ER diagram? If yes, model those as "is-a" relations.

- Provide Cardinality, total participation, (min, max) constraints for each entity for each relation and state any assumptions.
- For drawing the ER diagrams, there are several utilities on the web, for ex: www.draw.io, creately.com, lucid charts.com

Task B: Convert the ER diagrams into relations.

Once the ER diagrams are ready, you need to convert those into a relational model and go through the process of normalization This step will require to list all of the functional dependencies.

Report should have following components.

1. A list of top relations and attributes (Think of data dictionary)
   - Indicate primary and foreign keys (if any) for each of the relations and Justify the choice.
   - Write detailed description of each attribute, it's purpose and data type
   - Indicate each attribute's default value or if the attribute can be set to "null".
2. A mapping from ER diagram to Relations.
   - For each relation, provide the correspondence between the relation and the entity and/or relationship in the ER diagram that the relation is derived from. Justify your decision.

Task A: Draw an ER diagram

Link to diagram

1. Buisiness rules and relationships
   a. A club can organize many events, but each event is organized by exactly one club.
   b. Each club belongs to exactly one category, but a category can have many clubs.
   c. A student can join many clubs, and a club can have many members.
   d. A member can attend many events, and an event can be attended by many members.
   e. Event attendance is recorded for every member for every event they attend.

    f. Each student is unique in the Members table identified by their student ID.
    g. Each membership must include the student's role within the club, with a default of Member.
    h. Clubs have budgets allocated by the administration.
2. Entities & Attributes
    a. Category
        i. category_id (PK)
        ii. name
    b. Club
        i. club_id (PK)
        ii. club_name
        iii. category_id (FK)
        iv. budget
    c. Members (Students)
        i. student_id (PK)
        ii. first_name
        iii. last_name
        iv. email
    d. Event
        i. event_id (PK)
        ii. club_id (FK)
        iii. club_event
        iv. location
        v. event_date
    e. Club Membership
        i. club_id (PK and FK)
        ii. student_id (PK and FK)
        iii. role
    f. Event Attendance
        i. student_id (PK and FK)
        ii. event_id (PK and FK)
3. Weak entities
    a. Club Membership: identified by composite key (club_id + student_id) and it cannot exist without Club and Member
    b. Event Attendance: identified by composite key (event_id + student_id) and it cannot exist without Student and Event
4. Class hierarchy
    a. There is no class hierarchy
5. Cardinality, participation, (min, max) constraints for each relation and assumptions
    a. Categories & Clubs

i. One-To-Many
ii. Category: Partial Participation. Category → Club: (0, N)
iii. Club: Total Participation. Club → Category: (1, 1)
iv. Assumption: A category can exist without any club, but a club must have exactly one category.

b. Clubs & Members
i. Many-To-Many
ii. Club: Partial Participation. Club → Member: (0, N)
iii. Member: Total Participation. Member → Club: (1, N)

c. Clubs & Club Memberships
i. One-To-Many
ii. Club: Partial Participation. Club → Club Membership: (0, N)
iii. Club Membership: Total Participation. Club Membership → Club: (1, 1)
iv. Assumption: A club may have 0 members.

d. Members & Club Memberships
i. One-To-Many
ii. Member: Total Participation. Member → Club Membership: (1, N)
iii. Club Membership: Total Participation. Club Membership → Member: (1, 1)
iv. Assumption: Members table only contains students in at least one club.

e. Clubs & Club Events
i. One-To-Many
ii. Club: Partial Participation. Club → Club Event: (0, N)
iii. Club Events: Total Participation. Club Event → Club: (1, 1)
iv. Assumption: A club can have 0 events.

f. Members & Club Event s
i. Many-To-Many
ii. Member: Partial Participation. Member → Club Event: (0, N)
iii. Club Event: Partial Participation. Club Event → Member: (0, N)

g. Club Events & Event Attendance
i. One-To-Many
ii. Club Event: Partial Participation. Club Event → Event Attendance: (0, N)
iii. Club Attendance: Total Participation. Event Attendance → Club Event: (1, 1)
iv. Assumption: An event can have 0 attendees.

h. Members & Event Attendance
i. One-To-Many
ii. Member: Partial Participation. Member → Event Attendance: (0, N)
iii. Event Attendance: Total Participation. Event Attendance → Member: (1, 1)
iv. Assumption: A student can attend 0 events.

Task B: Convert the ER diagrams into relations.

1. Convert into relational model
2. Normalization
3. List of FDs

Clubs Table: stores clubs and their budgets

Column Name	Data Type	Key	Description	Additional information
club_id	Int	PK	Unique id for each club	Not null
club_name	Varchar		Name of club	Not null
category_id	Int	FK	References which category the club belongs	Not null
budget	Int		Budget allocated to the club	Not null, Default = $0

Justification: **PK-** We use a unique numeric id because its simple and avoids possible conflicts. Names can be similar or change so using club_name is not ideal. **FK-** Every club must belong to a category and using a separate table and id instead of the actual name avoids spelling issues.

Categories Table: stores club categories for consistency

Column Name	Data Type	Key	Description	Additional information
category_id	Int	PK	Unique id for each	Not null
name	Varchar		Type of club (academic, arts, sports, etc)	Not null

Justification: **PK-** We use a unique numeric id as the PK because it's simple and avoids possible conflicts. The reason for providing a separate table for categories was to ensure that the naming is consistent for all clubs, so the name cannot be PK.

Club Events Table: stores clubs' event information

Column Name	Data Type	Key	Description	Additional information
event_id	Int	PK	Unique event id	Not null
club_id	Int	FK	References club	Not null

			organizing event	
event_name	Varchar		Name of the event	Not null
location	Varchar		Where the event is	Not null
event_date	Date		Date of event	Not null

Justification: **PK-** We use a unique numeric id as the PK because it's simple and avoids possible conflicts. Names can be similar or change so using event_name is not ideal. **FK-** Events belong to a club, so the club_id identifies to which it belongs to.

Members Table: stores student information

Column Name	Data Type	Key	Description	Additional information
student_id	Int	PK	Unique student id given by school	Not null
first_name	Varchar		Student first name	Not null
last_name	Varchar		Student last name	Not null
email	Varchar		Student's school email address	Not null, unique

Justification: We didn't use email as the key despite it being unique because a student id is assigned by the school is generally used as an identifier and is more intuitive.

Event Attendance Table: stores attendance of a member for an event

Column Name	Data Type	Key	Description
student_id	Int	PK & FK	References student id
event_id	Int	PK & FK	References event id

Justification: **PK & FK-** The primary key is a composite key because a record only exists when both FKs do. There is no need to create a separate unique id.

Club Membership Table: stores student club membership and role

Column Name	Data Type	Key	Description	Additional information
club_id	Int	PK & FK	Refrences club_id in Club table	Not null

student_id	Int	PK & FK	Refrences student_id inStudents table	Not null
role	Varchar		Role in club (President, Secretary, Member, etc)	Not null, default value of Member

Justification: **PK & FK-** The primary key is a composite key because membership only exists when both FKs do. It's one student per club and using both uniquely identifies their role for any.

1. **Correspondence Justification (ERD → Relational Model)**
   - 1.1. Entity: Club
     ERD Component: Club entity with attributes (club_id, club_name, budget).
     Corresponding Relation: Clubs table
     Reason: Each club is a single independent entity, so it becomes its own relation.
     Key Selection Justification: club_id is used as PK because club names are not guaranteed unique.
   - 1.2. Entity: Category
     ERD Component: Category entity with attributes (category_id, name).
     Corresponding Relation: Categories table
     Reason: Categories are an independent lookup entity used for consistency across clubs.
     Key Justification: category_id is simple, stable, and avoids duplication/misspellings.
   - 1.3. Relationship: Club — Category (Many-to-One)
     ERD Component: Each club belongs to exactly one category.
     Relational Conversion: A foreign key (category_id) is added to the Clubs table.
     Justification: Many-to-one relationships are implemented by placing the FK on the "many" side (club).
   - 1.4. Entity: Event
     ERD Component: Event entity with (event_id, event_name, location, event_date).
     Corresponding Relation: Club Events table
     Reason: Events are a separate object with their own descriptive attributes and a PK.
     Relationship Handling: Each event is linked to exactly one club → add club_id as FK.
   - 1.5. Entity: Member
     ERD Component: Student/member entity with (student_id, first_name, last_name, email).
     Corresponding Relation: Members table
     Key Justification: student_id is chosen as PK because it is a standard institutional identifier.

1.6. Relationship: Club — Member (Many-to-Many)
ERD Component: A member can join multiple clubs, and a club has multiple members.
Relational Conversion: Club Membership table
Reason: M:N relationships require an associative relation (junction table).
PK: Composite key (club_id, student_id) uniquely identifies each membership record.

1.7. Relationship: Event — Member Attendance (Many-to-Many)
ERD Component: Members can attend multiple events; events have many attendees.
Relational Conversion: Event Attendance table
Reason: M:N relationships become a junction table linking event_id and student_id.
PK: Composite key (event_id, student_id).

2. Functional Dependencies (FDs)

2.1. Clubs(club_id, club_name, category_id, budget)
Functional Dependencies:
- club_id → club_name, category_id, budget
- club_name → (no FD; names may repeat)
- category_id → (category properties are stored separately)

Key: club_id
Normal Form: 3NF / BCNF (all non-key attributes depend solely on PK)

2.2. Categories(category_id, name)
Functional Dependencies:
- category_id → name
- name → (no FD; names may not be unique)

Key: category_id
Normal Form: BCNF

2.3. Club_Events(event_id, club_id, event_name, location, event_date)
Functional Dependencies:
- event_id → club_id, event_name, location, event_date
- event_name → (no FD; not unique)

Key: event_id
Normal Form: 3NF / BCNF

2.4. Members(student_id, first_name, last_name, email)
Functional Dependencies:
- student_id → first_name, last_name, email
- email → student_id (email is unique → candidate key)

Keys:
- Primary Key: student_id

- Candidate Key: email

Normal Form: BCNF (no transitive or partial dependencies)

2.5. Event_Attendance(student_id, event_id)
(Composite PK, no additional attributes)
Functional Dependencies:
- {student_id, event_id} → (entire tuple)
- student_id → (nothing alone)
- event_id → (nothing alone)

Key: {student_id, event_id}
Normal Form: BCNF (only key-based dependencies)

2.6. Club_Membership(club_id, student_id, role)
(Composite PK with extra attribute)
Functional Dependencies:
- {club_id, student_id} → role
- club_id → (nothing alone)
- student_id → (nothing alone)

Key: {club_id, student_id}
Normal Form: 3NF / BCNF (role depends only on full composite key)