HW + CE 2

Sunday, October 13, 2024 1:11 PM

Explain the following key types:

- Candidate key
  - A column in a table which is potential Primary Key, there can be many candidate key in a table or relationship and it is a design decision to choose one according to the project requirements.
- PK
  - Unique identifier of an entity instance, which has 2 properties of uniqueness and minimality
- FK
  - Primary Key of another entity type, which is saved in foreign entity instance as a reference (pointer) to the given row in FK instance type table.
- Composite key
  - Primary key, consisting of multiple columns, especially needed when single column can't provide uniqueness (mostly used in relationships)
- Natural key
  - Type of candidate key, which is naturally identifier itself in the context, like person ID number, email or other data making it unique in a dataset.
- Surrogate key
  - Redundantly added extra column acting as a primary key, especially needed when Primary Key as natural or composite is too long or bad in other factors (maybe its long VARCHAR PK), it is the tool for helping entity type with minimal PK.

Strong-Weak Entity Relation

What is the difference in the mapping of a strong-weak entity relation and a 1:N relation in regard to the

1. PKs of the resulting relations?
   - In strong-weak relationship weak can't exist without strong entity as it needs a composite primary key of **weak identifier + strong PK**, also weak has mandatory total participation. So weak entity will have foreign key of strong entity creating a composite PK in a weak.
2. N and 1 Side will have their PK or Composite key independently.

3. FKs of the resulting relations?
   - Weak entity will always have strong entity key as FK to create Composite Key for its own.
   - Many side will always have its foreign key connecting to one side in 1:N relationship.

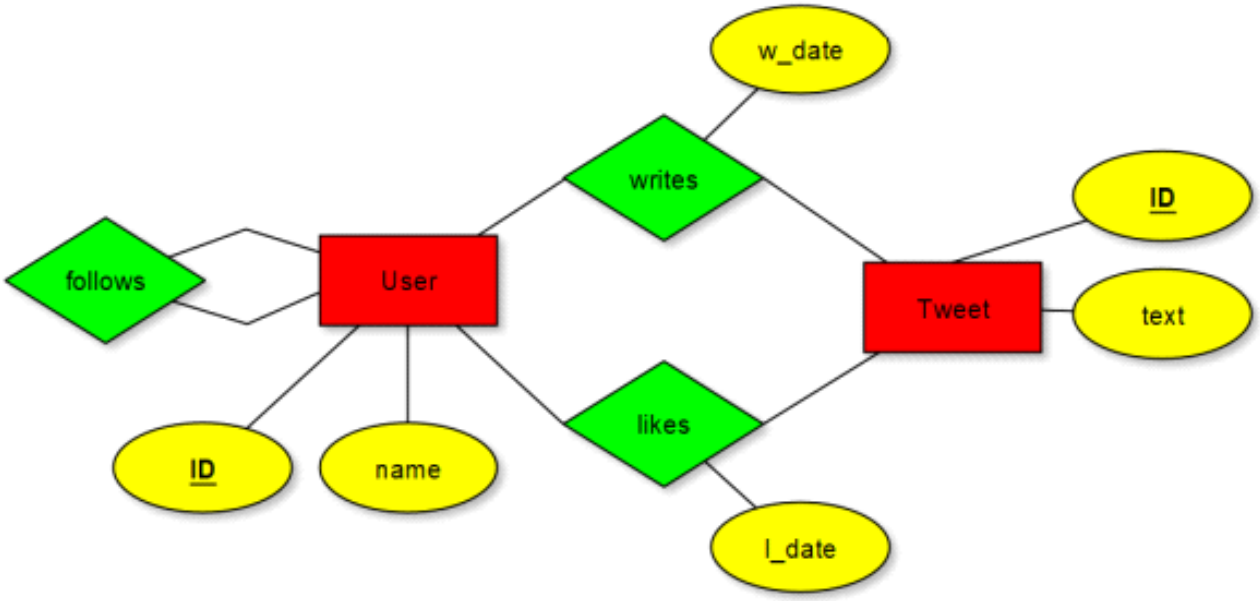
Write down the resulting relations with PKs and FK. Explain the difference!

1.

Room: {[room_number: int, building_number: char, …]}
Building: {[building_number: char, …]}

2.

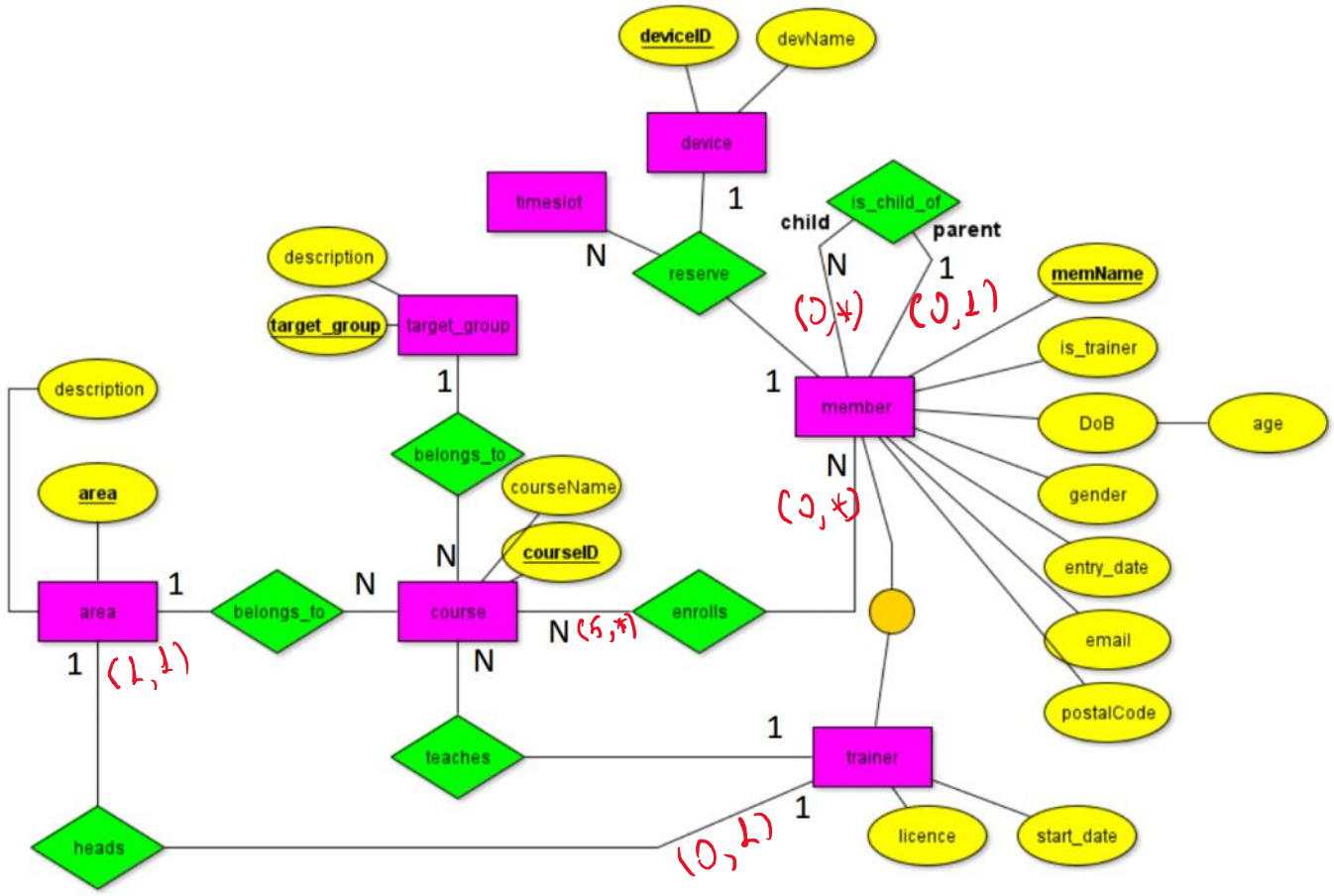
Room: {[room_number: int, building_number: char, …]} // here building_number is just a FK
Building: {[building_number: char, …]}



1. Map the ER model into a database schema.
2. How many final relations do you get?

User: {[ID: int, name: VARCHAR]}
Tweet: {[ID: int, text: VARCHAR, user_id: int, w_date: DATETIME]}
Like: {[user_id: int, tweet_id: int, l_date: DATETIME]}
Follow: {[follower_id: int, followed_id: int]}

We got 4 Relations in total, 2 for entities, two for Many-to-Many Relations.



(min,max) Cardinalities

Add the following (min, max) cadinalities to the sportsClub ER model:

1. A trainer does not need to manage an area but can at most manage one area. Each area is managed by exactly one trainer.
2. Members do not need to enroll in courses. They can enroll in as many courses as they want. A course needs to be taken by at least 5 members to take place.
3. A member can have one or more children in the club. A member does not need to have children in the club. Each child needs to have one parent in the club.

Map the SportsClub ER Model into a Relational Schema

Member: {[memName: string, is_trainer: bool, date_of_birth: date, gender: string, entry_date: date, email: string, postal_code: string, is_child_of: string]}

Device: {[deviceID: int, dev_name: string]}

Reservation: {[memName: string, deviceID: int, start_time: datetime, end_time: datetime]}
// by using member and timeslot as composite key, it naturally adds constraint to force user reserve
//only one device in one timeslot

Trainer: {[memName: string, license: string]} // Subtype relation

Area: {[area: int, description: string, headed_by: int]}

Target_group: {[target_group: string, description: string]}

Course: {[courseID: int, courseName: string, target_group: string, area: int, teacher: string]}

Enrollment: {[memName: string, courseID: int]}

1-4 done above.

5. Write down if you need to implement additional constraints.
   Constarints of min, max from prev exercise

6. Do you use vertical or horizontal partitioning when mapping the generalization? Why?
   Vertical, as it is optimal because there is subrelation in a schema, which only relates to trainer member

7. Why would it be a bad idea to map the generalization into a all-in-one relation?
   Because there would be many NULLified fields of license and start_date, but relationships would not create a problem as FKs are stored in other sides of relation (Area, Course)

Mapping of Generalizations

Given is a disjoint and total generalization.
Which mapping would you choose in the following scenarios?

- Queries often refer to common attributes.
  Vertical, as common attributes will be accesible in supertype, + maintaining inheritence clean structure
- Queries usually addres both, the common and the specific attributes of a subtype.
  All-in-one because we need to access only one table for taking data, but for design best practices, still vertical is better.
- Generalization gets constantly a lot of insert operations.
  Horizontal, as we will need to insert into only one table, if we assume that we don't need any more computation to determine which subtype it is.

What is more challenging in regard to the PK of a subtype:
- Duplicating the PKs from the supertype to the subtypes or
- Enforcing uniqueness of the PKs over the subtype relations?

Second one is more challenging, as first one only requires a constraint to protect uniqueness and correct duplication of supertype pk, as it is subtype pk too. But in second scenartio if we take all-in-one structure, we have to add additional constraints to control uniqueness and most importantly - disjointless as it will mess up, if we add one row, using 2 subtype attributes and being a mix of

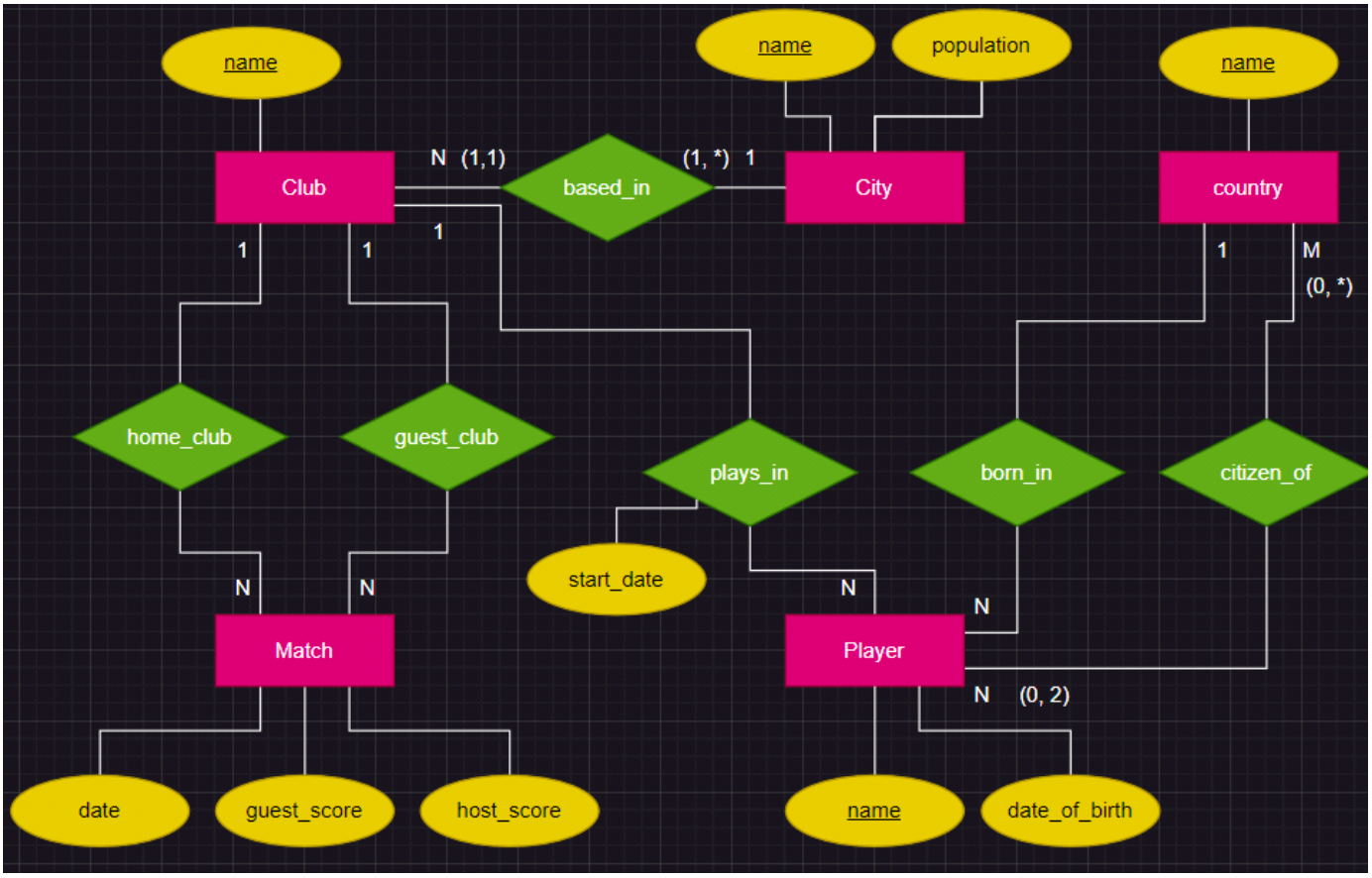
all, it will damage consistency, so it requires more backend or constraint controls to remain consistent structure according to requirements logic.

Which scenario belongs to which mapping? 1 is for Vertical and second is for All in one.

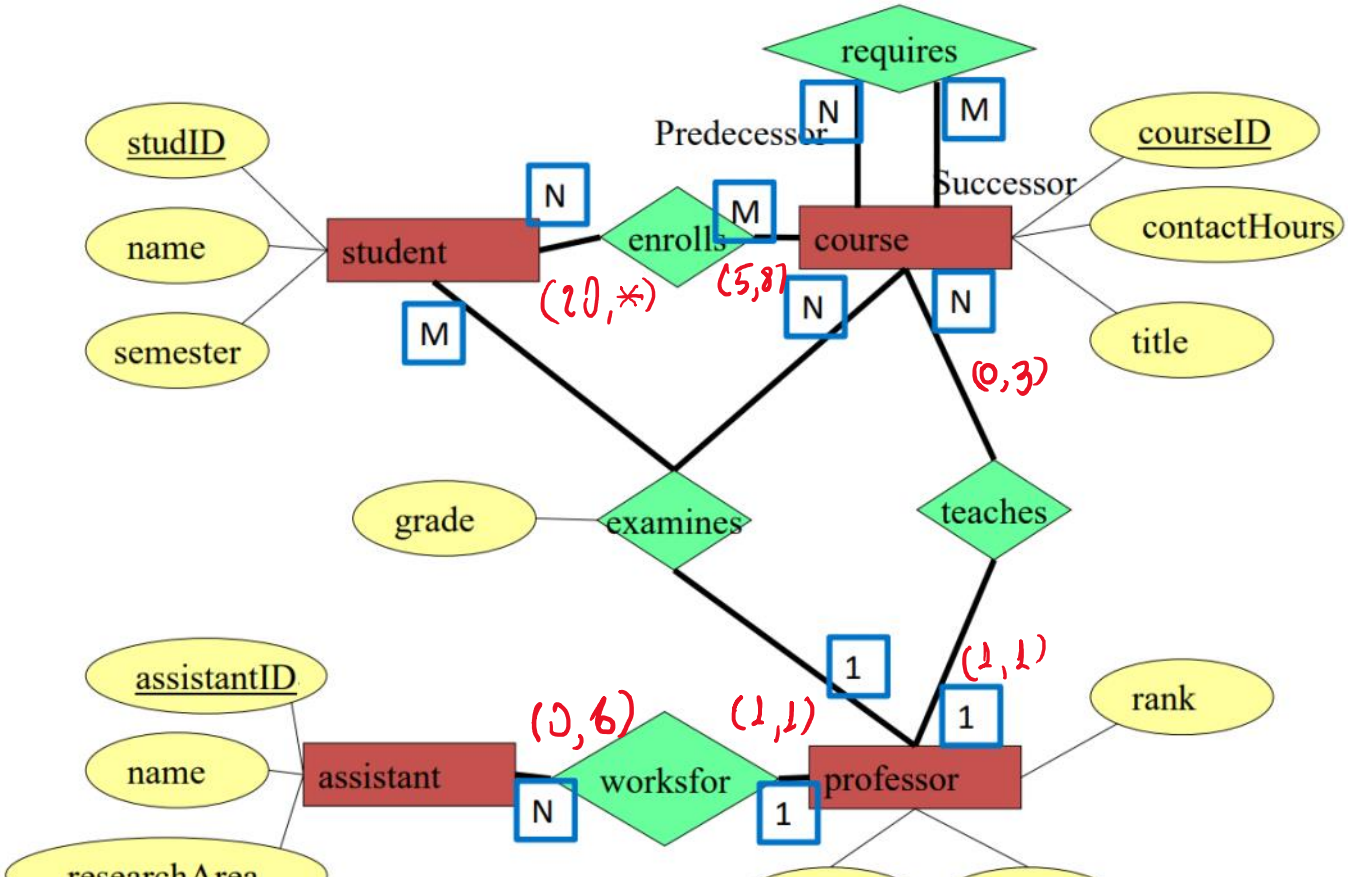
ER-Model “German Soccer Clubs” (modified midterm task)

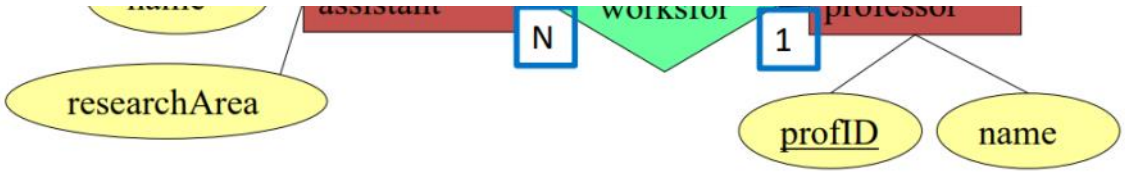
1. The database stores German soccer clubs. Each club is identified by its unique name. Each club is based in a city. In one city, there can be many clubs. (Example would be Bayern Muenchen and 1860 Muenchen, both based in Munich). City names are assumed to be unique. The city population is also stored.
2. The database stores information about all national league matches. The national league is organized such that each team plays twice against each other team, one match in each teams's hometown. E.g. Bayern Muenchen against Leverkusen once in Munich and once in Leverkusen.
   For each match is stoed:
   1. the two clubs, that played against each other (home club, guest club),
   2. the result (goals home club and goals guest club) and
   3. the date. Multiple matches are on the same date.
3. The database stores information about all players that play in a national league club. For each player the name is stored. It is assumed that there are no two players with the same name. The date of birth, the country of birth and country or countries of citizenship are stored, as well.
4. For each player it is stored, for which club he plays. Each player can of course play for only one club. The database also stores the start date of the contract.

1. Develop the corresponding ER-model, with entities, relationships and attributes.
2. Underline PKs
3. Write standard cardinalities for all relationships.
4. Add the following (min,max) cardinalities:
   a. A player can have 0, 1 or 2 citizenships. Many countries exist that no player in the German league has citizenship of.
   b. Each city is base of at least one club. Each club is based in exactly one city.



Central Exercises





Add the following (min, max) cadinalities to the university ER model:

1. Each professor can have a maximum of 6 assistants. It is possible that a professor does not have assistant, at all. Each assistant works for exactly one professor.

2. A course needs to have at least 20 students. There is no upper limit for enrollment. Each student needs to take between 5 and 8 courses.

3. A professor does not need to teach a course. A professor can teach up to 3 courses. Each course needs to be taught by exactly one professor.

Map the University ER Model into a Relational Schema

1. How many final relations (tables) do you get?
2. Map the university ER model into a database schema.
3. Name the relationship relations preferably with a noun: enrollment, examination, requirement
4. Mark PKs, FKs and PKs that are FKs at the same time.

We got 6 relationships (tables):

Student: {[studID: int, name: string, semester: int]}

Professor: {[profID: int, name: string, rank: string]}

Assistant: {[asistantID: int, name: string, researchArea: string, works_for: int]}

Course: {[courseID: int, contactHours: int, title: string, professor_id: int]}

Enrollment: {[studID: int, courseID: int]}

Examination: {[studID: int, courseID: int, profID: int, grade: decimal]}

Modify the university ER model such, that it allows students to retake courses.
Modify your relational university database schema such that it represents this change.

We need to modify enrollment and examination relations:

Enrollment: {[enrollment_id: int, studID: int, courseID: int]}
// New surrogate key is needed to let student enroll multiple times.

Examination: {[studID: int, courseID: int, attempt: int, profID: int, grade: decimal]}

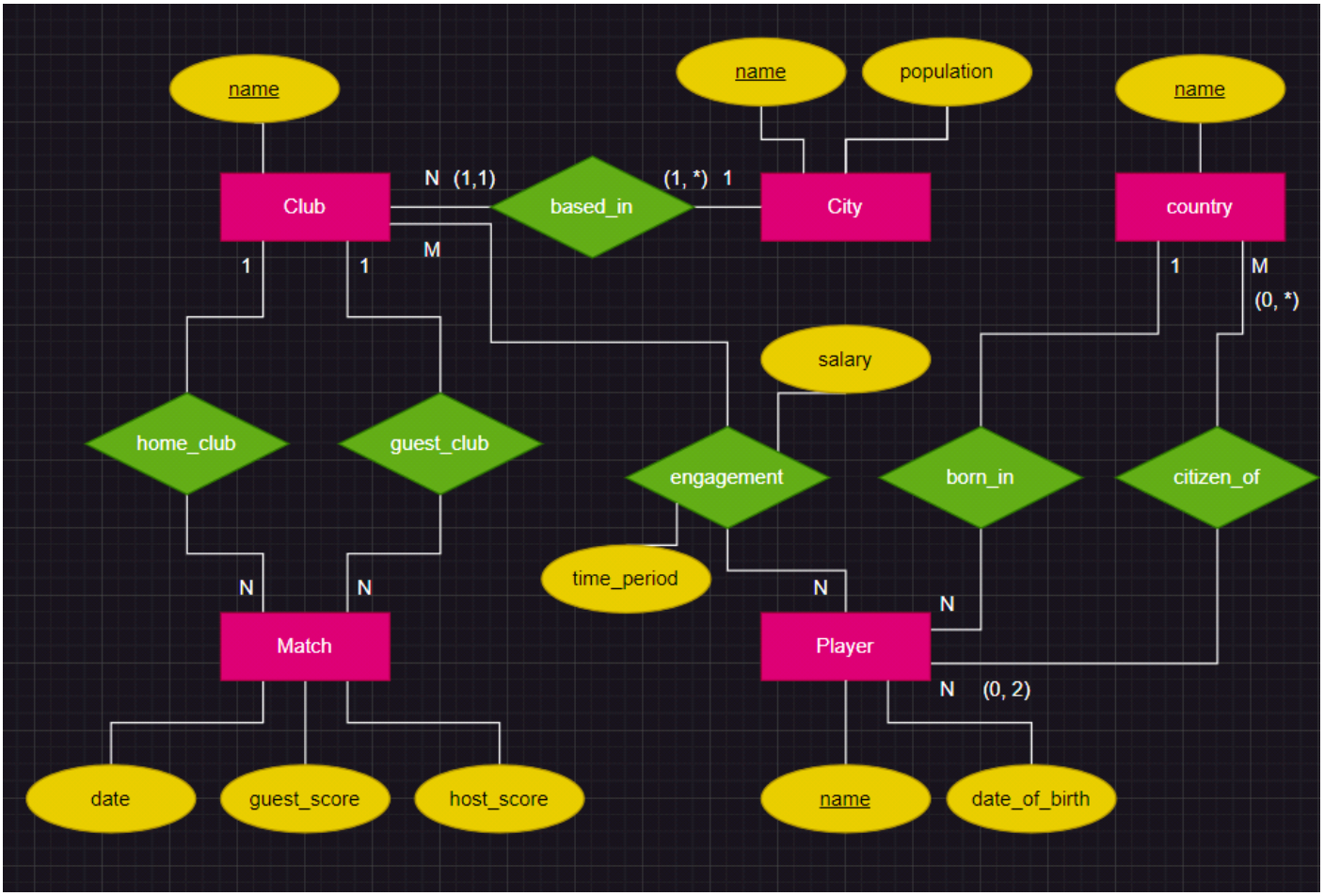
// we add attempt count to let student take exam multiple times, and primary key adds attempt number as composite component

ER-Model “German Soccer Clubs” (modified midterm task)

We want to add an engagement history for each player: When did the player play for what club.

Additional Requirement:
The database keeps track of the players’ engagements: For each player it is stored, for which club(s) he was or is under contract. For each contract the time period is given. A player can have been with the same club several times in different time periods. The salary for each engagement is stored, as well.

Change your ER model accordingly.



We can change plays_in relationship into engagement. Current team can be determined by query

Map your ER model into a relational schema.

City: {[name: string, population: int]}

Country: {[name:string]}

Club: {[name: string, based_in: string]}

Match: {[date: date, guest_club: string, host_club: string, guest_score: int, host_score: int]}

Player: {[name: string, date_of_birth: date, born_in: name]}

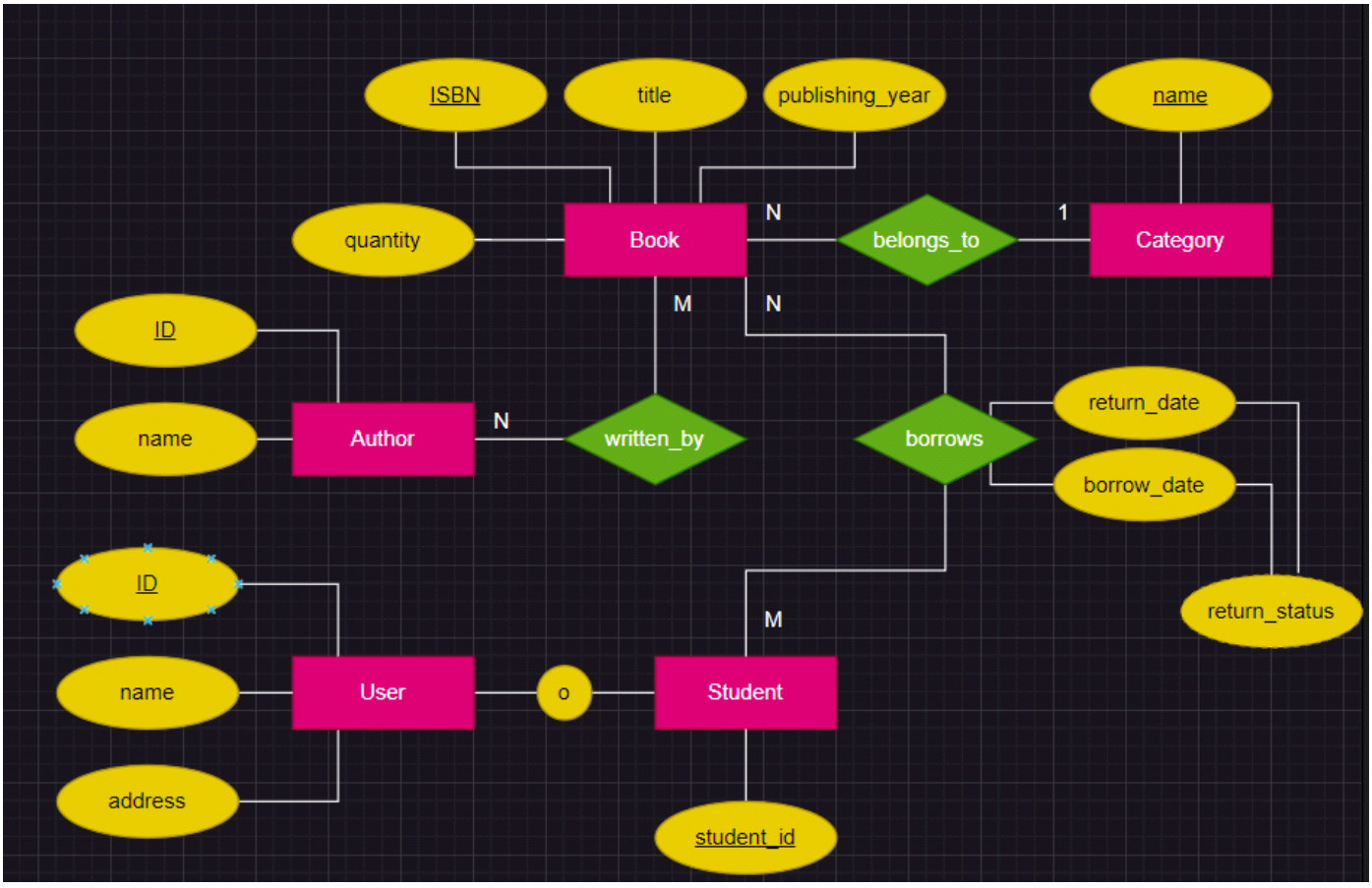
Citizenship: {[player: string, country: string]}

Engagement: {[club: string, player: string, start_date: date, end_date: date, salary: int]}
// We can take player and start date of contract as PK, because player can not play in multiple clubs at the same time.

ER-Model "University Library"

Requirement list for the university library:

1. The database stores books with ISBN (unique international book number), title and publishing year.
2. There can be multiple copies of one book ISBN in the library.
3. Each author is stored with id and name. A book can be written by obe or multiple authors. Each author can write one or multiple books.
4. The database categorizes all book ISBN in categories like e.g. math, CS, Management, etc. Each book belongs to exactly one category.
5. Everybody can be user of the library. A user needs to be registered with an id, name and address. A general user is only allowed to read the books in the library. He is not allowed to borrow out books.
6. Students can also be users of the library. They are registered with the same attributes as other users. Their id is their student id. They are allowed to borrow out books. Students can borrow multiple books but always only one copy of a book. The database also can track if they return the book(s) on time or late or not at all.


ISBN
title
publishing_year
name
quantity
Book
N
belongs_to
1
Category
M
N
ID
name
Author
N
written_by
borrows
return_date
borrow_date
return_status
ID
M
name
User
o
Student
address
student_id