Project Title
Illini CMS (Classroom Management System)

Project Track 1: Stage2

CS 411: Database Systems, Spring 2024

TEAM 108: SQLScholars
Pranav Sharma (pranav24)
Ketaki Gokhale (kag12)
Panshul Jindal (panshul2)
James Flaherty (jamespf3)

Entities and Relations

Assumptions - Entities:
We are making an Online Learning system targeted at Primary-Secondary schools. New users are added to the Users table. This would contain user profile data on the admin, the students, their parents, and their teachers. We have a separate Student table which contains details of all the students such as their class grades, addresses, and date of birth. We also have a Parents table which contains basic details of all the parents mapped along with their child and a Teachers table with details of the teachers such as their date of joining, department and home address. We have a courses table which would have the information about the courses offered at the school (eg. English, Maths, Science) and a Classrooms table which has details of each classroom (eg. Grade 8-A, Grade 8-B). The ClassroomGroup table has all combinations of courses and classrooms. (eg. Grade 8-A English, Grade 8-A Math, Grade 8-B Science, etc.) So, the ClassroomGroup is a weak entity that depends on Courses and Classrooms for its existence. Assignments and ClassGroupRecordings contain the details of assignments and recordings for all the fields in ClassroomGroup. They too are therefore weak entities, dependent on the ClassroomGroup. Each ClassroomGroup can contain multiple recordings and multiple assignments, since we can have multiple professors teaching the same subject but to different grades.
Note: We take into account the existence of multiple classrooms for the same school year (Grade) in scenarios where we have nearly 250 students enrolled in 8th Grade, with each room having a capacity of 50 students, so the 250 students are divided into 5 different houses or sections.

Why are these different entities?
Users, Students, Parents, Teachers, Classrooms, Courses, ClassroomGroup, Assignments, and ClassGroupRecordings all represent distinct concepts. By representing them as different entities we can accurately represent complex relationships and capture all relevant information about each entity. These entities can be translated as real-world objects that have a separate existence as compared to a relation or exist as attributes of a separate entity.
The following cardinality-related issues also indicate the need for a new entity:

- ClassGroupRecordings will have multiple recordings for a single ClassroomGroup
- Assignments will have multiple assignments for a single ClassroomGroup

Assumptions - Relations:
The Users table has teachers who grade the Assignments, so Grades is a relation between Users and Assignments. Each assignment will have a single grade. The Users use the ClassRoomGroup, which is defined as a relation called ClassroomUsers, which contains information about those users who are part of a Classroom(teachers and students). Users have attendance in Classrooms, so we have a relation Attendance to depict this. Each user will have multiple attendance records for different days, but only one record on one day.

Following is the assumed cardinality of all the relationships in our table:

1. ClassroomGroup - Courses -> It will have at least 1 course or can have any number of courses **(1..*)**

2. Courses - ClassroomGroup -> A newly added course will not be in the class group and multiple courses can be present in a class group **(0..*)**

3. ClassGroup - Classrooms -> A class group will have at least 1 classroom and can have many classrooms **(1..*)**

4. Classrooms - ClassGroup -> A classroom might not have been added to a ClassGroup on creation and can belong to multiple classGroup (12-a Math, 12-a Sci) **(0..*)**

5. Teachers - ClassGroup -> One teacher may not have been assigned a class and subject she will be teaching when newly onboarded and may later belong to multiple such ClassGroup (Eg. Professor Abdu teaches Grade 8-A CS, Grade 8-B CS) **(0..*)**

6. ClassGroup - Teachers -> One class group can have at least 1 teacher teaching or may have many users(12-A has 5 teachers, 12-B has 4) **(1..*)**

7. Students - ClassGroup -> One student may not have been assigned to a ClassGroup upon new enrollment and may later belong to multiple such ClassGroup (Eg. Student Pranav studies Grade 8-A CS and also studies Maths in Grade 8-A Math ClassGroup) **(0..*)**

8. ClassGroup - Students -> One class group can have at least 1 student or may have many students like 12-A has 50 users, 12-B has 49 **(1..*)**

9. Users - Teachers -> One user account will always be mapped to only one teacher **(1..1)**

10. Teachers - Users -> One teacher will always have only one account **(1..1)**

11. Users - Students -> One user account will always be mapped to only one student **(1..1)**

12. Students - Users -> One student will always have only one account **(1..1)**

13. Users - Classrooms -> A user might not belong to a classroom(parent) and might belong to multiple classrooms **(0..*)**

14. Classrooms - Users -> 1 Classroom may belong to many users (many students and teachers) **(1..*)**

15. Users - Parents -> One user account will belong to only one parent **(1..1)**

16. Parents - Users -> One parent will only have one user account **(1..1)**

17. Students - Assignments -> A student might not have an assignment (when there are no assignments created) and might have multiple assignments later (0..*)

18. Assignments - User -> One assignment can belong to multiple students (1..*)

19. ClassGroup - ClassGroupRecording -> A ClassGroup might not have a recording when the lecture is ongoing and might have multiple for multiple days (0..*)

20. ClassGroupRecording - ClassGroup -> 1 recording belongs to only 1 classGroup (1..1)

21. ClassGroup - Assignment -> A ClassGroup might not have an assignment when the group has just been created and might have multiple for multiple assignments (0..*)

22. Assignment - ClassGroup -> 1 assignment only for 1 classGroup (1..1)

Normalization of Schema

We used 3NF over BCNF to normalize our databases because we opted for dependency preservation which BCNF does not have. Additionally, 3NF is more preferred for practical applications which do not demand the use of complex architecture and modern microservices, making the schema complex. Since, 3NF can establish a balance between the complexity of our architecture and also remove redundancy, since all non-key attributes are dependent on the primary key, we choose 3NF.

Process of normalization for each entity:

- ***Users***
    - userId->firstName
    - userId->lastName
    - userId->email
    - userId->password
    - userId->isStudent
    - userId->isAdmin
    - userId->isTeacher
    - userId->isParent

1. **Get Minimal Basis**
    a. **Only Singleton in RHS**
        Every functional dependency only has a singleton in the right hand side.
    b. **Remove unnecessary attributes in the left hand side.**
        Every functional dependency has only necessary attributes in the left hand side.
    c. **Remove FDs that can be inferred from the rest**
        There are no functional dependencies that can be inferred from the rest.

2. **For each FD A->B in minimal basis, use AB as the schema for a new relation**
   Created a table with userId as primary key containing firstName, lastName, email, password, isStudent, isAdmin, isTeacher, isParent, parentEmail1, parentEmail2, and createdAt date.

3. **If none of the schemas from step 2 is a superkey then create a relation whose schema is a key for the original relation**
   The value userId is a superkey.

- ***Classroom***
  - classRoomId->className
  - classRoomId->createdAt

1. **Get Minimal Basis**
   a. **Only Singleton in RHS**
      Every functional dependency only has a singleton in the right hand side.
   b. **Remove unnecessary attributes in the left hand side.**
      Every functional dependency has only necessary attributes in the left hand side.
   c. **Remove FDs that can be inferred from the rest**
      There are no functional dependencies that can be inferred from the rest.

2. **For each FD A->B in minimal basis, use AB as the schema for a new relation**
   Created a table with classRoomId as a primary key containing className and createdAt date.

3. **If none of the schemas from step 2 is a superkey then create a relation whos schema is a key for the original relation**
   a. The value classRoomId is a superkey.

- ***Courses***
  - courseId->subjectName
  - courseId->rating

1. **Get Minimal Basis**
   a. **Only Singleton in RHS**
      Every functional dependency only has a singleton in the right hand side.
   b. **Remove unnecessary attributes in the left hand side.**
      Every functional dependency has only necessary attributes in the left hand side.

c. **Remove FDs that can be inferred from the rest**

There are no functional dependencies that can be inferred from the rest.

2. **For each FD A->B in minimal basis, use AB as the schema for a new relation**
   a. Created a table with courseId as a primary key containing subjectName and rating.

3. **If none of the schemas from step 2 is a superkey then create a relation whose schema is a key for the original relation**
   a. The value courseId is a superkey.

- ***<u>ClassroomGroup</u>***
  - classGroupId->classRoomId
  - classGroupId->courseId
  - classGroupId->zoomLink
  - classGroupId->classStartTimings

1. **Get Minimal Basis**
   a. **Only Singleton in RHS**

   Every functional dependency only has a singleton in the right hand side.

   b. **Remove unnecessary attributes in the left hand side.**

   Every functional dependency has only necessary attributes in the left hand side.

   c. **Remove FDs that can be inferred from the rest**

   There are no functional dependencies that can be inferred from the rest.

2. **For each FD A->B in minimal basis, use AB as the schema for a new relation**

   Created a table with classGroupId as a primary key containing classRoomId, courseId, zoomLink, classStartTimings and classDuration

3. **If none of the schemas from step 2 is a superkey then create a relation whose schema is a key for the original relation**

   The value classGroupId is a superkey.

- ***<u>ClassroomUsers</u>***
  - userId, classGroupId->userJoinedAt

1. **Get Minimal Basis**
   a. **Only Singleton in RHS**

   Every functional dependency only has a singleton in the right hand side.

b. **Remove unnecessary attributes in the left hand side.**
   Every functional dependency has only necessary attributes in the left hand side.
c. **Remove FDs that can be inferred from the rest**
   There are no functional dependencies that can be inferred from the rest.

2. **For each FD A->B in minimal basis, use AB as the schema for a new relation**
   Created a table with userId and classGroupId as a primary key containing userJoinedAt date.
3. **If none of the schemas from step 2 is a superkey then create a relation whos schema is a key for the original relation**
   The values of userId and classGroupId are the superkeys.

- ***Assignment***
  - assignmentId->classGroupId
  - assignmentId->googleFormLink
  - assignmentId->maximumGrade

1. **Get Minimal Basis**
   a. **Only Singleton in RHS**
      Every functional dependency only has a singleton in the right hand side.
   b. **Remove unnecessary attributes in the left hand side.**
      Every functional dependency has only necessary attributes in the left hand side.
   c. **Remove FDs that can be inferred from the rest**
      There are no functional dependencies that can be inferred from the rest.

2. **For each FD A->B in minimal basis, use AB as the schema for a new relation**
   Created a table with assignmentId as a primary key containing classGroupId, googleFormLink, and maximum grade.

3. **If none of the schemas from step 2 is a superkey then create a relation whos schema is a key for the original relation**
   The value of assignmentId is the superkey.

- ***Grades***
    - assignmentId,userId->grade
    - assignmentId,userId->remarks
    - assignmentId,userId->sentimentScore
    - assignmentId,userId->isNotificationSent

1. **Get Minimal Basis**
    a. **Only Singleton in RHS**
        Every functional dependency only has a singleton in the right hand side.
    b. **Remove unnecessary attributes in the left hand side.**
        Every functional dependency has only necessary attributes in the left hand side.
    c. **Remove FDs that can be inferred from the rest**
        There are no functional dependencies that can be inferred from the rest.
2. **For each FD A->B in minimal basis, use AB as the schema for a new relation**
    Created a table with assignmentId and userId as a primary key containing grade, remarks, sentimentScore, and isNotificationSent.

3. **If none of the schemas from step 2 is a superkey then create a relation whos schema is a key for the original relation**
    The values assignmentId and userId are the superkeys.

- ***Attendance***
    - userId,classRoomId->attendanceDate
    - userId,classRoomId->isParentsNotified
    - userId,classRoomId->IsPresent

1. **Get Minimal Basis**
    a. **Only Singleton in RHS**
        Every functional dependency only has a singleton in the right hand side.
    b. **Remove unnecessary attributes in the left hand side.**
        Every functional dependency has only necessary attributes in the left hand side.
    c. **Remove FDs that can be inferred from the rest**
        There are no functional dependencies that can be inferred from the rest.

2. **For each FD A->B in minimal basis, use AB as the schema for a new relation**
    Created a table with studentId and classRoomId as a primary key containing attendanceDate, isParentsNotified, and isPresent.

3. **If none of the schemas from step 2 is a superkey then create a relation whos schema is a key for the original relation**

   The values studentId and classRoomId are the superkeys.

- ***<u>ClassRecordings</u>***
  - recordingId->classGroupId
  - recordingId->classDate
  - recordingId->recordingLink

1. **Get Minimal Basis**
   a. **Only Singleton in RHS**

      Every functional dependency only has a singleton in the right hand side.

   b. **Remove unnecessary attributes in the left hand side.**

      Every functional dependency has only necessary attributes in the left hand side.

   c. **Remove FDs that can be inferred from the rest**

      There are no functional dependencies that can be inferred from the rest.

2. **For each FD A->B in minimal basis, use AB as the schema for a new relation**

   Created a table with recordingId as a primary key containing classGroupId, classDate, and recordingLink.

3. **If none of the schemas from step 2 is a superkey then create a relation whose schema is a key for the original relation**

   The value recordingId is the superkey.

- ***<u>Students</u>***
  - userId->dob
  - userId->class
  - userId->address

1. **1.Get Minimal Basis**

   Only Singleton in RHS

   Every functional dependency only has a singleton in the right hand side.

   Remove unnecessary attributes in the left hand side.

   Every functional dependency has only necessary attributes in the left hand side.

   Remove FDs that can be inferred from the rest

   There are no functional dependencies that can be inferred from the rest.

2. **For each FD A->B in minimal basis, use AB as the schema for a new relation**
   Created a table with userId as a primary key containing dob, class, and address.

3. **If none of the schemas from step 2 is a superkey then create a relation whos schema is a key for the original relation**
   The value of userId is the superkey.

- ***Teachers***
  - userId->dob
  - userId->department
  - userId->address

1. **Get Minimal Basis**
   Only Singleton in RHS
   Every functional dependency only has a singleton in the right hand side.
   Remove unnecessary attributes in the left hand side.
   Every functional dependency has only necessary attributes in the left hand side.
   Remove FDs that can be inferred from the rest
   There are no functional dependencies that can be inferred from the rest.

2. **For each FD A->B in minimal basis, use AB as the schema for a new relation**
   Created a table with userId as a primary key containing dob, department, and address.

3. **If none of the schemas from step 2 is a superkey then create a relation whos schema is a key for the original relation**
   The value of userId is the superkey.

- ***Parents***
  - userId->sid
  - userId->occupation
  - userId->relation

1. **Get Minimal Basis**
   Only Singleton in RHS
   Every functional dependency only has a singleton in the right hand side.
   Remove unnecessary attributes in the left hand side.
   Every functional dependency has only necessary attributes in the left hand side.
   Remove FDs that can be inferred from the rest

There are no functional dependencies that can be inferred from the rest.

2. **For each FD A->B in minimal basis, use AB as the schema for a new relation**
   Created a table with userId as a primary key containing studentID(sid), occupation, and relation with student.

3. **If none of the schemas from step 2 is a superkey then create a relation whos schema is a key for the original relation**
   The value of userId is the superkey.

Relational Schema

Users(userId: varchar(10) [PK], firstName: varchar(255), lastName: varchar(255), email: varchar(255), password: varchar(255), isStudent: Boolean, isAdmin: Boolean, isTeacher: Boolean, isParent: Boolean, createdAt: Date)

Classrooms(classroomId: varchar(10) [PK], className: varchar(20), createdAt: Date)

Courses(courseId: varchar(10) [PK], subjectName: varchar(50), rating: Float)

ClassroomGroup(classGroupId: varchar(10) [PK], classroomId: varchar(10) [FK to Classroom.classRoomId], courseId: varchar(10) [FK to Courses.courseId], zoomLink: varchar(255), classStartTimings: Date, classDuration: Float)

Students(userId: varchar(10) [FK to Users.userId], dob: Date, class: varchar(100), address: varchar(255))

Parents(userId: varchar(10) [FK to Users.userId], sid: varchar(10) [FK to Students.userId], occupation: varchar(255), relation: varchar(255))

Teachers(userId: varchar(10) [FK to Users.userId], doj: Date, occupation: varchar(255), relation: varchar(255))

ClassroomUsers(userId: varchar(10) [FK to Users.userId], classGroupId: varchar(10) [FK to ClassroomGroup.classGroupId], userJoinedAt: Date)

Assignment(assignmentId: varchar(10) [PK], classGroupId: varchar(10) [FK to ClassroomGroup.classGroupId], googleFormLink: varchar(255), maximumGrade: Float)

Grades(assignmentId: varchar(10) [FK to Assignment.assignmentId], userId: varchar(10) [FK to Users.userId], grade: Float, remarks: varchar(255), sentimentScore: INT, isNotificationSent: Boolean)

Attendance(studentId: varchar(10) [FK to Users.userId], classroomId: varchar(10) [FK to Classroom.classroomId], attendanceDate: Date, isParentsNotified: Boolean, isPresent: Boolean)

ClassGroupRecordings(recordingId: varchar(10) [PK], classGroupId: varchar(10) [FK to ClassroomGroup.classGroupId], classDate: Date, recordingLink: varchar(255))