Computer Science Department Database Project

by: Arnold Sanders

Course: CMP 420
Database Systems
Prof. Fulakeza
Date: 12/6/24

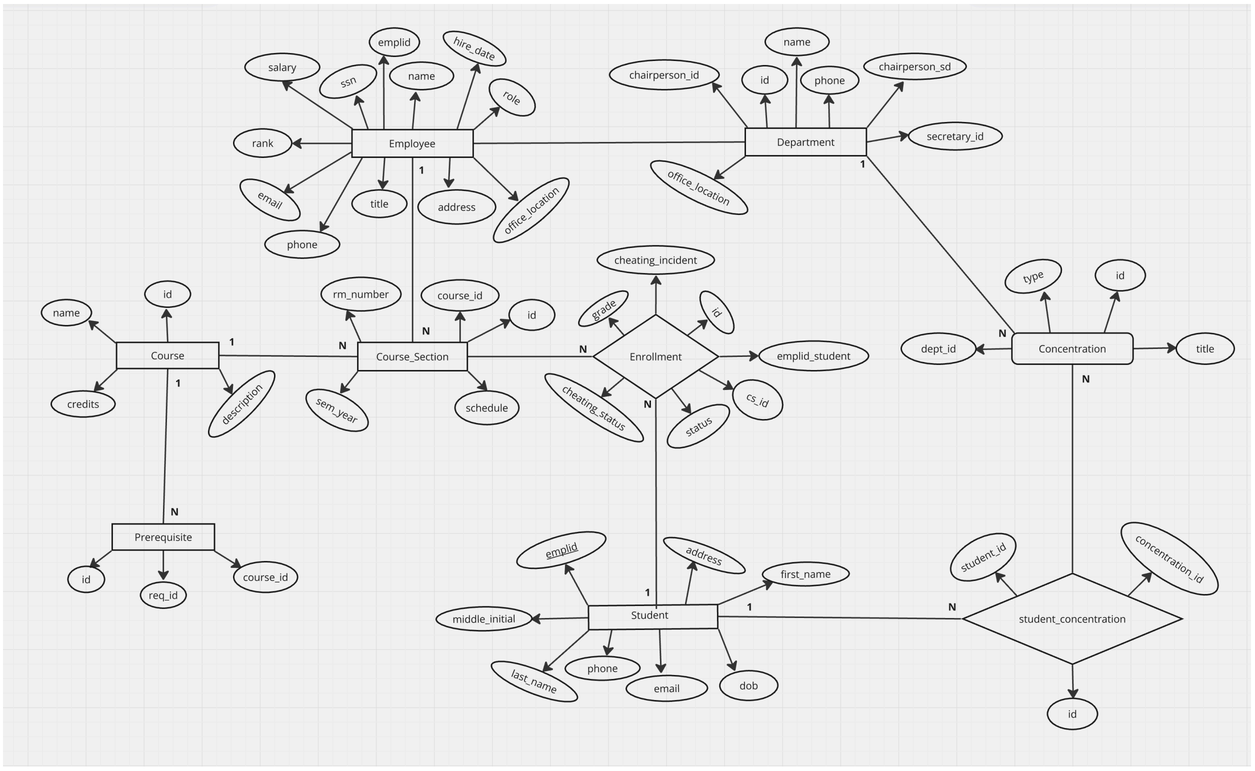
Project description:

As a database architect, you have been hired by the Department of Computer Science to design their database. The database keeps track of the employees in the department, courses offered by the department, and students majoring or minoring in programs offered by the department such as computer science, computer information systems, cybersecurity, data science and AI. The employees are categorized into faculty and staff.

- Each employee has a name, unique emplid, unique ssn, phone, email, home address, salary, office location, and date of hire, title, role, rank. Employees can either be Faculty or Staff, which can be represented in the role field.
- Faculty teaches sections of courses. A course has a unique course id, unique course name, description, course prerequisites, course_credits.
- A Course Section is a specific offering of a course in a given semester. A course section has a semester year, room Number, and schedule. Schedule is the time(s) when the section is taught (e.g., Monday 10:00 AM - 12:00 PM). Note that one course can have multiple sections in the same semester.
- Each student has a unique emplid, First Name, middle initial, Last Name, Date of Birth, email, phone, home address, and a major offered in the department or minor in the department. A student can do multiple majors in the department.
- A student can enroll in multiple course sections in a semester and each section can have multiple students. enrollment has a status that states whether the students are "active", "dropped", "completed". A student earns a grade in a course section they are taking. You also want to keep track of cheating incidents of students in the section they enrolled and their cheating incident statuses.
- A Department has a unique department id, unique name, phone, and office location, you also have a faculty who is the chairperson

of the department, the chairperson start date. You can also have an employee who is a secretary of the department.

Entity Relationship Model Diagram (ER diagram or EER). Explain what you did. And add your diagram also.



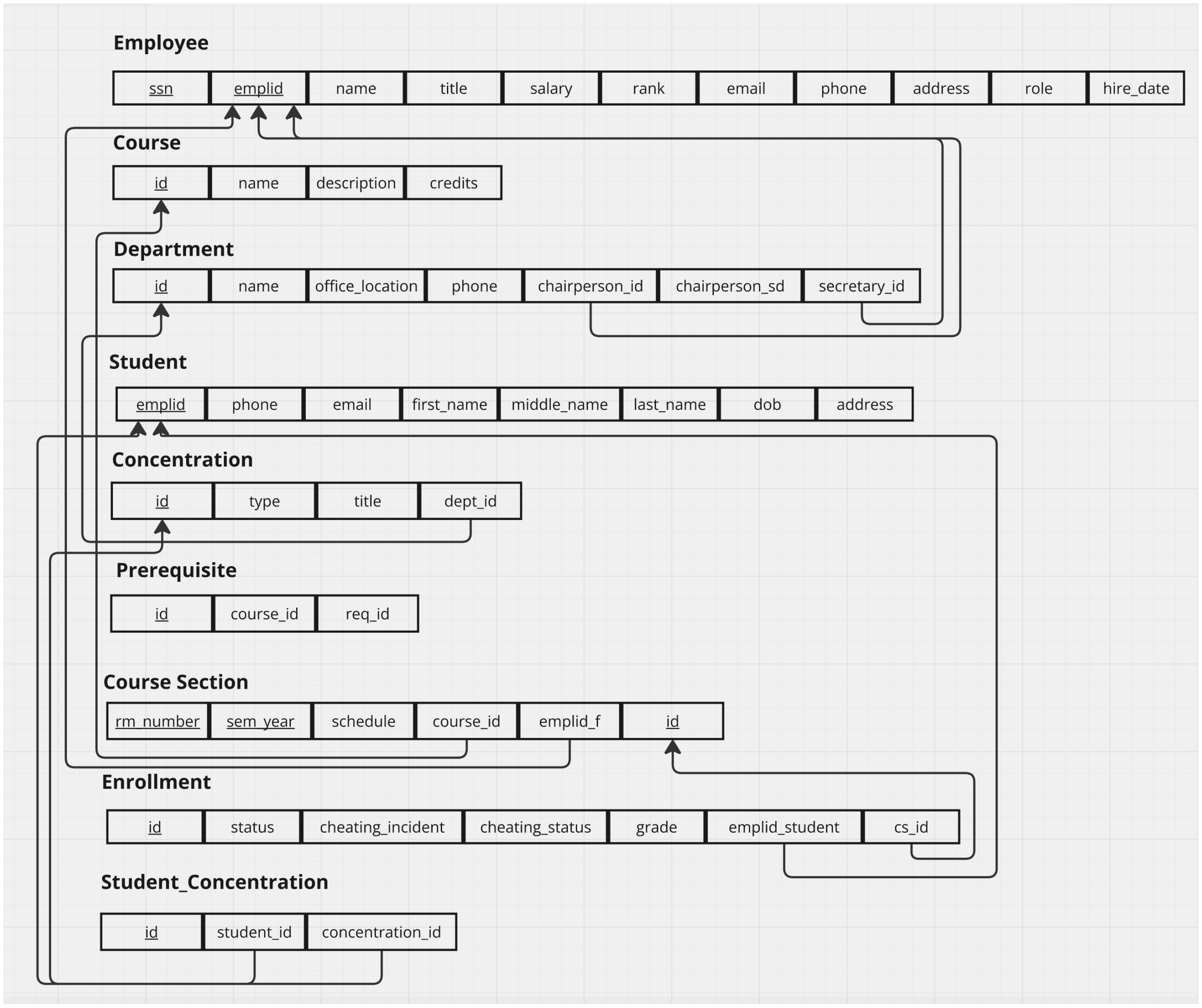
On the Department model, I decided to store foreign keys for the employee records designated as chairperson and secretary. Since the department will never have more than one chairperson or secretary I made a 1:1 association between the employee and the department directly.

I also made a table called prerequisites as a join table for the courses. My idea was that storing a list of course id's on the courses themselves would be violating 1NF, which states that no multi values should be stored in a table. The prerequisites table has rows of instances that correspond individually to the id of a desired course and the individual course that would need to be taken before you can take that course. The result is that you can search for courses that have prerequisites or prerequisites of courses. Search, in any direction is possible.

I also made a reference table for majors/minors called Concentrations. It's a table that keeps track of the available majors/minors in between Departments. The concentrations table has a foreign key column that relates back to the Department table which correlates each major/minor to the department it belongs to.

Lastly I made a third joins table that keeps track of students to concentrations called students_concentrations. This is a many:many relationship where at the moment there is no limit to how many majors or minors any one student can be assigned to.

Map (Convert) the ER (EER) model into a relational model. Explain the steps that you followed to convert your ER (EER) model to relational model.



First I mapped the regular entity types. These are entity types without foreign keys. Then I mapped the 1:1 relationships, these are the relations where the tables mapped 1 column of Table 1 to table 2, with no table in between. I then went ahead and mapped the 1:M relationships. These are the tables that depended one another but without a table in between them. I lastly mapped the M:M relationships with the join table between the two relations.

Normalization: Normalize your relations. As you are normalizing each relation, explain the process. Make sure you list all the functional dependencies.

1NF - Atomic Values
On the course table there was a multivalue attribute being suggested in the requirements description. I was able to normalize the course table by removing the prerequisite attribute and turning it into it's own table.
On every other table, all of the attributes were atomic. The tables are all 1NF normalized.

2NF - Full functional dependence on prime attributes
In my database, there are no super keys that individually can be broken down to gain access to any other attribute on the same table. The tables are all 2NF normalized.

3NF - There are no transitive functional dependencies on any of the tables in my database.

<u>Employee Closure</u>:

SSN+ : ***{*** Emplid, name, title, salary, ranking, email, phone, address, role_title, hire_date ***}***

Emplid+ : ***{*** SSN, name, title, salary, ranking, email, phone, address, role_title, hire_date ***}***

<u>Student Closure</u>:

Emplid+ : ***{*** phone, email, first_name, date_of_birth, middle_initial, last_name, address ***}***

<u>Course Closure</u>:

Id+ : ***{*** name, description, credits ***}***

Department Closure:

Id+ : { name, office_location, phone, chairperson_start_date, secretary_id, chairperson_id ***}***

Name+: { id, phone, office_location, chairperson_start_date, secretary_id, chairperson_id ***}***

Course Section Closure:

Id+ : { schedule, sem_year, rm_number, course_id, emplid_f ***}***
[sem_year, rm_number]+ : { schedule, course_id, emplid_f ***}***

Enrollment Closure:

Id+: { status, cheating_incident, cheating_status, emplid_student, cs_id, grade ***}***

Write SQL queries to create all your relations. (You take the queries from the database implementation and place them here.

```
CREATE TABLE COURSE_SECTION (
    id integer(250) NOT NULL,
    schedule date NOT NULL,
    sem_year integer(4) NOT NULL,
    rm_number varchar(4) NOT NULL,
    course_id integer(100),
    emplid_f varchar(8),
    primary key (`id`),
    foreign key (`course_id`) references `course` (`id`),
    foreign key (`emplid_f`) references `employee` (`emplid`)
)
```

```
CREATE TABLE COURSE (
    id integer(100) NOT NULL,
    name varchar(20) NOT NULL UNIQUE,
    description varchar(60),
    credits integer NOT NULL,
    primary key (`id`)
)


CREATE TABLE DEPARTMENT (
    id integer(20) NOT NULL,
    name varchar(25) NOT NULL UNIQUE,
    phone varchar(9),
    office_location varchar(50),
    chairperson_start_date date,
    secretary_id varchar(8),
    chairperson_id varchar(8),
    primary key (`id`),
    foreign key (`chairperson_id`) references `employee`
(`emplid`),
    foreign key (`secretary_id`) references `employee`
(`emplid`)
)


CREATE TABLE EMPLOYEE (
    name varchar(20),
    emplid varchar(8) NOT NULL UNIQUE,
    ssn varchar(9) NOT NULL UNIQUE,
    phone varchar(9),
    email varchar(24),
    home_address varchar(24),
    salary integer(7),
    office_location varchar(24),
    hire_date date,
    title varchar(20),
    role_title varchar(10),
    ranking varchar(20),
    primary key (`emplid`),
    CHECK (role_title = 'staff' OR role_title = 'faculty')
)



CREATE TABLE STUDENT (
    emplid varchar(8) NOT NULL UNIQUE,
    first_name varchar(15),
    middle_initial varchar(1),
    last_name varchar(15),
    date_of_birth date,
    email varchar(24),
    phone varchar(9),
    home_address varchar(50)
)



CREATE TABLE ENROLLMENT (
    id integer(255) NOT NULL,
    status varchar(9),
    cheating_incident date,
    cheating_status varchar(20),
    emplid_student varchar(8),
    cs_id integer(250),
    grade varchar(3),
    foreign key (`emplid_student`) references
`student`(`emplid`),
    foreign key (`cs_id`) references `course_section`(`id`),
    primary key (`id`),
    CHECK (status='active' OR status='dropped' OR
status='completed'),
    CHECK (grade='A' OR grade='B' OR grade='C' OR grade='D' OR
grade='F'),
    CHECK (cheating_status='open' OR cheating_status='closed')
)

CREATE TABLE PREREQUISITE (
    id integer(255) NOT NULL,
    course_id integer(100),
    req_id integer(100),
    primary key (`id`),
    foreign key (`course_id`) references `course`(`id`),
    foreign key (`req_id`) references  `course`(`id`)
)


CREATE TABLE STUDENT_CONCENTRATION (
    id integer(255) NOT NULL,
    student_id varchar(8),
    concentration_id int(100),
    primary key (`id`),
    foreign key (`student_id`) references `student`(`emplid`),
    foreign key (`concentration_id`) references
`concentration`(`id`)
)


CREATE TABLE CONCENTRATION (
    id int(100) NOT NULL,
    type varchar(9) NOT NULL,
    title varchar(45) NOT NULL,
    dept_id integer(20),
    primary key (`id`),
    foreign key (`dept_id`) references `department` (`id`),
    CHECK (type='major' OR  type='minor')
)
```

Conclusions:

It was challenging to get the constraints of the Database correct.
As I was reading through the project specification, I walked through it slowly pulling out the nouns and verbs to figure out what was an object and what was a relationship. The objects were easily represented as tables and attributes. However I needed to read a few times to figure out what was the nature of the relationships between those objects.

Also I realized while doing the Relational Diagram, some of the connections seemed to violate normal forms. It was complex to build the ER and Relational diagrams. However it made the process of building the tables and attributes very simple. Specifically, the relational diagram seemed to map 1:1 to the SQL queries I needed to write to make each table.

I used Google Gemini to speed up the process of generating the sample data. I found it time consuming to get the sample data for the database. Most specifically the join tables were quite challenging to think through.

In regards to databases, I plan to work as a full stack developer in the near future. If I am going to be building web applications, it will almost always require a database, either Relational or NonRelational. If I had more time I would add indexes on some of the columns. I would also add some views to make some of the data more readily available to see.