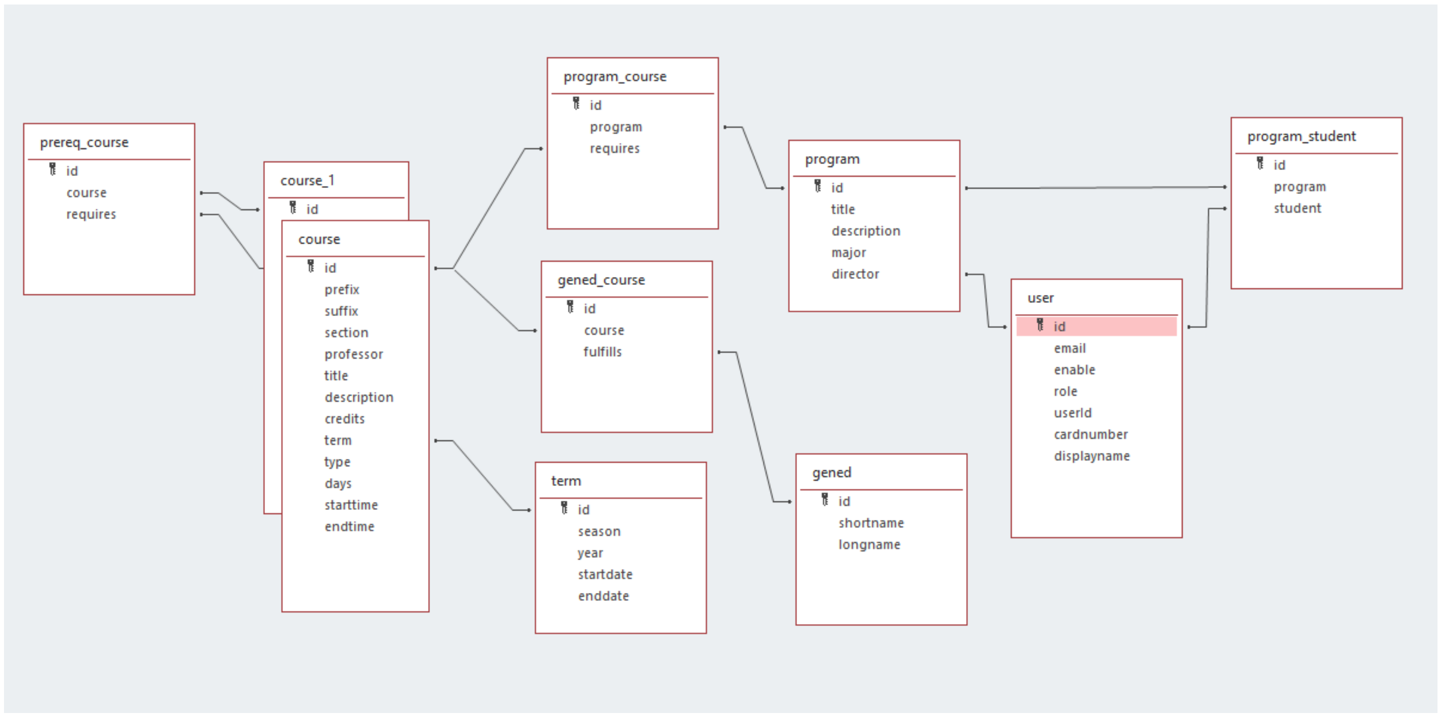
Rahmi Husein Schema Proposal March 23, 2022

Proposal

I have put forth the following schema for consideration by the API team.



This schema has been designed and implemented in a Microsoft Access database but should be readily translatable into PostgreSQL using equivalent types and validation checkers. Please note that `course` and `course_1` are the same table, but Access displays them as separate entities due to foreign keys from `prereq_course` both going to the same table.

Documentation

Each table — its fields, data types, constraints, and intended usage / rationale — are documented below. The primary keys are omitted because they are identical across all tables.

`program` — Stores an academic program.

- `title: short text` — The name of the program.
- `description: long text` — The description of the program.
- `major: boolean` — Whether this is a major.
- `director: **foreign key [user]**` — The program director.

`term` — Stores an academic term.

- `season: short text` — The season of the program.
    - `"spring" or "summer" or "fall" or "winter"`
- `year: integer` — The season of the program.
    - `year > 0`
- `startdate: date/time` — The start date of the program.
- `enddate: date/time` — The end date of the program.
    - `enddate > startdate`

`course` — Stores a course.

- `prefix: short text` — The letter code prefix (i.e., CS, MATH, BIO, …)
- `suffix: short text` — The course number, which may include "HON" or other suffixes.
- `section: integer` — The section number.
    - `section > 0`
- `professor: short text` — The preferred name of the professor.
- `title: short text` — The friendly name of the class.
- `description: long text` — The course description.
- `credits: integer` — The number of credit hours this course is worth.
    - `credits > 0`
- `term: `**`foreign key [term]`** — What term the course is currently offered in.
- `type: short text` — The meeting type for this course.
    - `"sync" or "async" or "hybrid"`
- `days: integer` — The days of the week on which this course meets. **Please see Addendum.**
- `starttime: integer` — The time from midnight (in seconds) the meeting begins
    - `starttime ≥ 0 and starttime < 86400`
    - `allow null iff type == async`
- `endtime: integer` — The time from midnight (in seconds) the meeting ends.
    - `starttime ≥ 0 and starttime < 86400 and endtime > starttime`
    - `allow null iff type == async`

`user` — Stores a user. The following are **NEW** features included in this existing schema:

- `cardnumber: integer` — The user's employee ID or student ID.
- `displayname: short text` — The user's preferred name.

`gened` — Stores a general education category.

- `shortname: short text` — The abbreviation of the category (e.g., ARHU, CISS)
- `longname: long text` — The full name of the category (e.g., Arts and Humanities)

`gened_course` — Relates a course to the general education categories it satisfies.

- `course: foreign key [course]` — The course with a general education category.
- `fulfills: foreign key [gened]` — The general education category this course satisfies.

`program_course` — Relates a program to the courses it requires.

- `program: foreign key [program]` — The program with a required course.
- `requires: foreign key [course]` — The course required by the program.

`program_student` — Relates a program to the students enrolled in it.

- `program: foreign key [program]` — The program with an enrolled student.
- `student: foreign key [user]` — The Advisor account of the enrolled student.

`prereq_course` — Relates a course to its prerequisite(s).

- `course: foreign key [course]` — The course with a prerequisite.
- `requires: foreign key [course]` — The prerequisite required by the course.

Addendum

There has been some confusion about how to store multiple weekdays. I believe I have a viable solution.

We can create an enumeration as follows:

```
const days = {
  sunday: 64,
  monday: 32,
  tuesday: 16,
  wednesday: 8,
  thursday: 4,
  friday: 2,
  saturday: 1
}
```

This can map weekdays to bits in a 7-bit integer. We can store the weekdays in a single byte. In the following example, classes are held on Mondays, Tuesdays, and Thursdays:

```
SUN MON TUE WED THR FRI SAT
 0   1   1   0   1   0   0  = 52

Validation rule: days > 0 AND days < 128
```

SQL provides bitwise operators to allow for easy lookup. For instance, select all matching

```
days & (tuesday | thursday) == (tuesday | thursday)

days & 0b00010100 == 0b00010100

days & 20 == 20
```

to see classes that **include** Tuesday and Thursday as part of their schedule. Note: this is pseudocode and may not be proper SQ L. Another example:

```
days == (monday | wednesday | friday)

days == 0b00101010

days == 42
```

to see classes that **only** run Monday, Wednesday, and Friday.

Conclusion

I hope this finds everyone well and gets ratified. I'm welcome to design changes if anyone foresees any issues with this model.