Data Model and Schema

This section aims to give an overview of the data model. It concentrates mainly on the entities and their relationships. The details of attributes are provided in the SQL schema.

Some considerations in the development of our data model:

- there are two main kinds of **people** in the system: staff and students; all people have certain basic information associated with them (e.g. name); staff have additional information related to their employment; students have additional information related to the degree that they are studying; there are also people who are neither staff nor students that UNSW wants to record (e.g. members of the University Council)
- UNSW runs a number of teaching **terms** each year (these are also called sessions or semesters)
- a **subject** is a unit of study in a particular area (e.g. introductory programming, database systems, etc.); a subject is defined primarily by its syllabus
- a **course** is a particular offering of a subject in a particular teaching term; it has a course convener (also called lecturer-in-charge), perhaps an enrolment quota, and is associated with a number of classes
- a **class** is a teaching activity at a scheduled time in a scheduled place; examples of classes are lectures, tutorials and labs; a class is associated with a course
- a **degree** is an award given to a student who completes a specified program of study
- a **program** is a named program of study leading to one or more degrees
- a **stream** specifies the precise requirements for study in a specific area; it is used to implement the notions of *major* and *minor*
- in a particular program, students choose at least one stream from a range of possible streams (in a double degree, they will choose two streams, one from each set of streams for the constituent degrees); the program will specify precisely what are the allowed/required combinations of streams
- to satisfy the requirements of the program, a student must satisfy the requirements of all the streams that they enrol in from the program; in addition, there may be requirements from the program itself (e.g. general education, total units of credit completed, etc.)
- there are several different types of requirements:
    - subject requirements (core subject, electives, limitations)
    - stream requirements (e.g. must take one from the BCom majors)
    - program requirements (e.g. must be enrolled in 3648 to take SENG1010)
    - UOC requirements (e.g. overall plan needs at least 144 UC)
    - WAM requirements (e.g. must have WAM of at least 65 for Hons)
    - stage requirements (e.g. must be in stage 2 of program to take COMP2 courses)
    - stream requirements (e.g. complete one major from a set of majors)
    - miscellaneous requirements (e.g. industrial training)
- in specifying requirements, we frequently need to deal with sets of academic objects (either programs or streams or subjects); we call these (generically) **academic groupings**
- we use **subject groups** in specifying subject requirements; each subject group has a name (e.g. "level 3/4 COMP courses") and an associated set of subjects
- similarly for **stream groups** (e.g. "set of BA majors") and **program groups** (e.g. "all programs offered by CSE")
- a **subject requirement** specifies: a subject group, a number of UOC associated with the group, whether this number is a minimum or maximum requirement; this is flexible enough to allow us to describe:
    - core requirements (group size 1, must complete 1 course from the group)

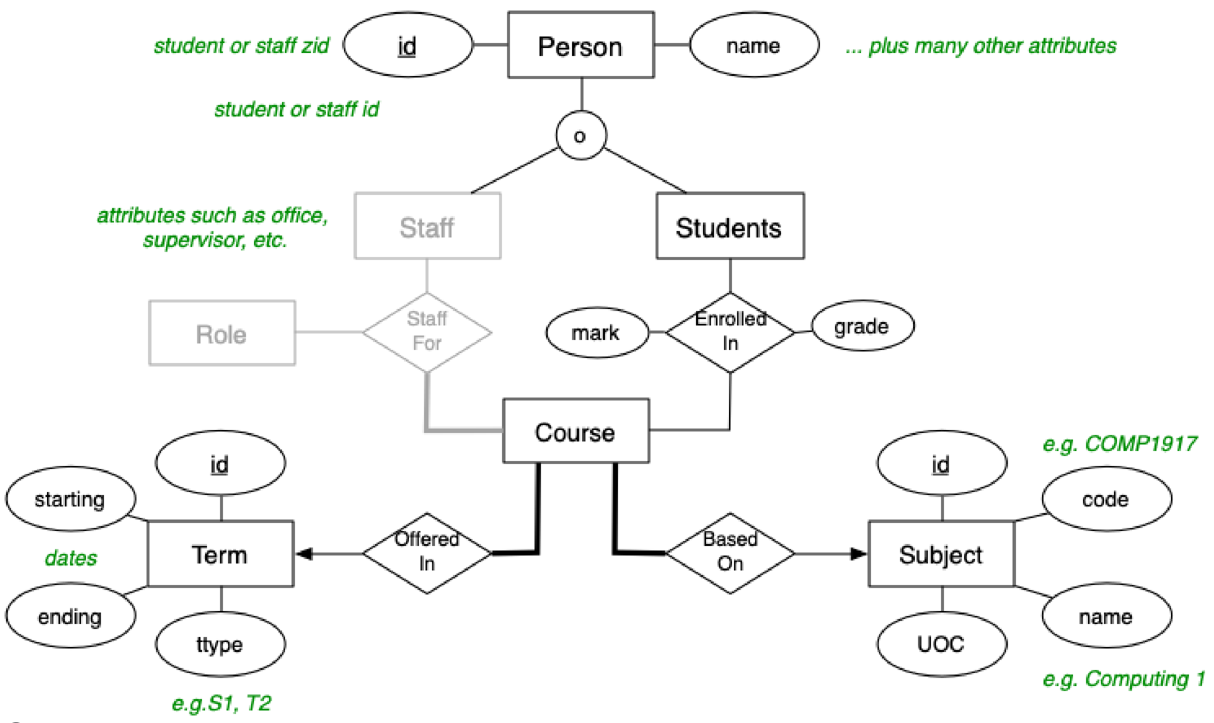
    - alternatives (several related courses, must complete 1 of them)
    - professional electives (set of courses from one area, must complete *k* of them)
    - limitations (e.g. no more than 72 UC of level 1 courses)
- in terms of the ideas above, programs and streams are defined as collections of requirements; a particular student must satisfy all of the requirements before they are regarded as having completed the program or stream
- how to determine whether a student has satisfied requirements depends on the type of requirement:
    - for UOC, use the course enrolment information
    - for course requirements, use the course enrolment information
    - for miscellaneous, must explicitly record that the student has met them (because there's no other data that will allow us to work it out)
- at any given time, each student is **enrolled** in one program, one or more streams (associated with the program), and generally several courses and classes within those courses; we need to record all four kinds of enrolment
- for enrolment in a program, it is useful to know when the student's enrolment commenced, when it ended (if it has ended), and their current status (e.g. active, on leave, etc.)
- over their lifetime, students may enrol in several programs, each with associated streams and courses
- a **schedule** describes when in a stream particular courses should be taken; in our terms, it will relate subject groups to streams and associate specific (year,semester) combinations with them
- sometimes, we wish to allow a student to **vary** from the standard requirements of their degree plans; there are three types of substitutions:
    - substitution: replace one course by another within a program
    - advanced standing: get credit for a course from elsewhere (or from a partly-completed UNSW degree) to use in place of some course in a program
    - exemption: get recognition for having studied a course elsewhere so that this can be used as a pre-requisite for further study at UNSW

Treat the ER data model description below as an overview, and consult the SQL Schema for full details on how the database is implemented. The ER diagrams below make a number of simplifications to the complete SQL schema: many attributes are omitted, some names are changed (e.g. entity names are singular here, but plural in the SQL schema).

To make the presentation clearer, the data model is presented in a number of sections.

People/Courses/Terms

Entities and relationships related to students/staff/courses/terms ...

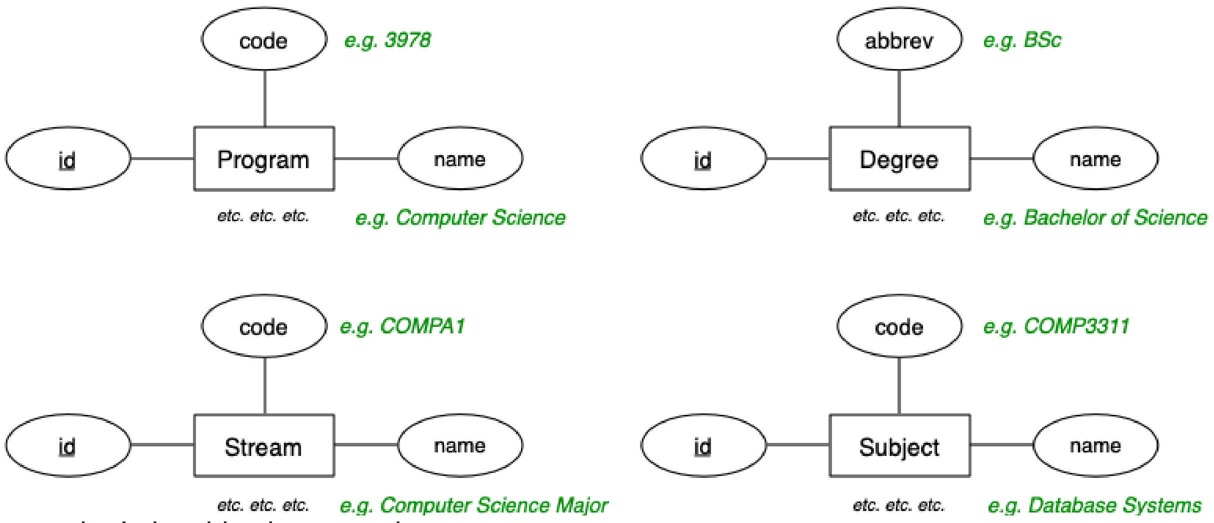


Comments:

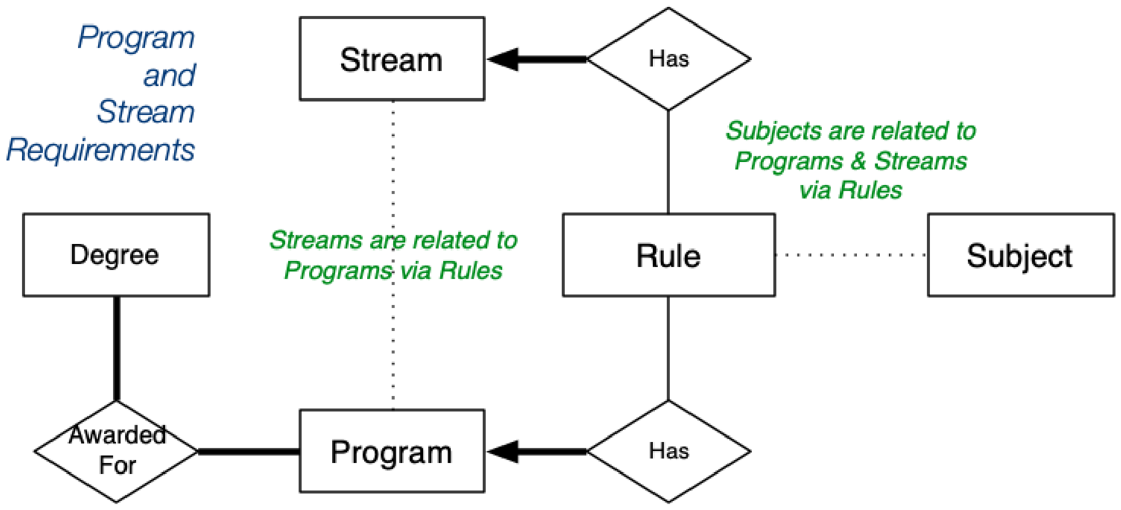
- the Person entity would clearly have much additional data associated with it in a real implementation (contact details, etc.)
- note that the use of `Subjects.id` rather than `Subjects.code` as a primary key actually allows us to implement multiple versions of a given subject; we also need to know the period over which the particular version is relevant, and this is recorded in the subject record
- in reality, a course offering has a lot of other information associated with it (e.g. enrolment quotas, assessment schemes, classes); some of these appear in the database, some don't
- the *n:m* relationship allows multiple staff to be associated with the teaching of a given course offering; one of these staff is required to be a course convener (this could have been implemented via an extra field in the `Courses` table to force the above constraint, but since it's also a Role, we decided to implement it as such and do the constraint via a trigger)

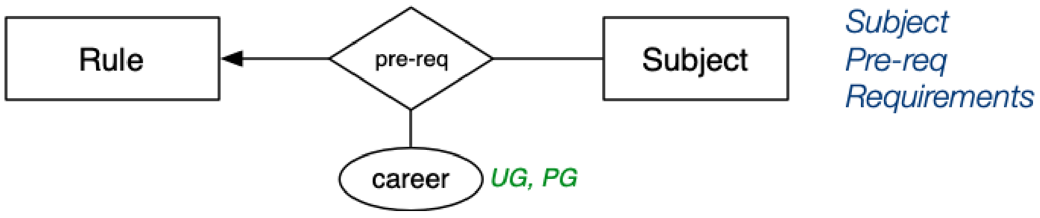
Programs/Streams/Degrees

Entities related to programs/streams/degrees ...



... and relationships between them ...


Program and Stream Requirements
Stream
Has
Subjects are related to Programs & Streams via Rules
Degree
Streams are related to Programs via Rules
Rule
Subject
Awarded For
Program
Has

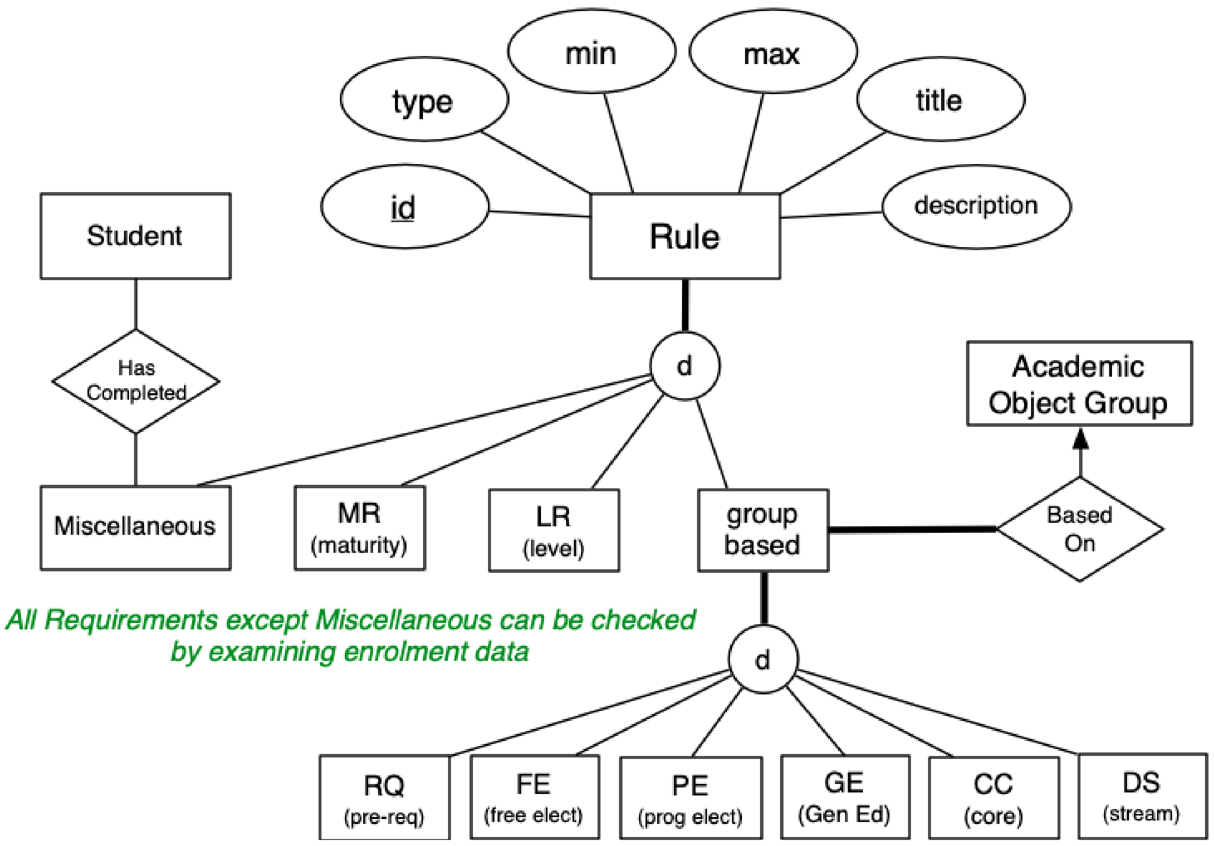


Comments:

- here, we show minimal attributes for the Program, Degree and Stream entities; see the schema for full details
- a program leads to one or more degrees (e.g. BSc, BE/BCom)
- a degree occurs in at least one program (e.g. straight BE, BE/BCom, BE/BSc)
- streams may be used in several programs (e.g. BE component of combined degrees)
- specific requirements may also be used in several plans

Requirements

Entities and relationships related to requirements ...



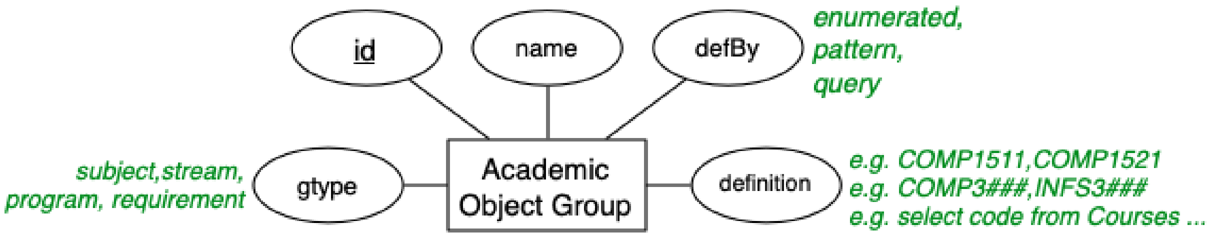
Comments:

- every requirement is either a
    - *UOC* ... student must meet minimum/maximum UOC limits
    - *stage* ... student must be in specified stage of program
    - *WAM* ... student must meet minimum/maximum WAM limits
    - *inProgram* ... student is currently enrolled in one of a set of programs
    - *inStream* ... student is currently enrolled in one of a set of streams
    - *inSubject* ... student is currently enrolled in one of a set of subjects
    - *doneProgram* ... student has completed requirements of program(s)
    - *doneStream* ... student has completed requirements of stream(s)
    - *doneSubject* ... student has successfully completed subject(s)
    - *compound* ... combines other requirements via boolean operator
    - *miscellaneous* ... other non-computable requirement (e.g. industrial training)
- many of the requirement types refer to a set of academic objects (e.g. set of subjects in an elective group); such sets are represented by the `AcadObjectGroups` table
- a requirement can be negated via a flag in the requirement record
- requirements may specify minimum and maximum values (see examples above)
- using min, max and academic object sets allows you express requirements like
    - must complete between 24 and 36 UOC from COMP3* courses (subject group)
    - must complete one major from the BA majors (stream group)
    - must be enrolled in a CSE degree (program group)
- if a requirement is really a logical combination of other requirements, it can be expressed as a compound requirement; a compound requirement combines a set of requirements via logical AND or logical OR; compound requirements can be nested
- note that the sets of requirements for Programs, Streams amd Subject pre-requisites are implicitly conjunctive (i.e. you need to satisfy all of them before you have completed the Program or Stream or met the pre-requisite requirement for the Subject)

- miscellaneous requirements pick up all of the other non-course-related requirements that exist in various degrees (e.g. industrial training for Engineers); since there are no enrolments records or other kinds of records to check that these were completed, we need an explicit link between the student and the requirement to indicate this

Academic Object Groups

Groups of academic objects that are central to the definition of requirements ...


id
name
defBy
enumerated, pattern, query
subject,stream, program, requirement
gtype
Academic Object Group
definition
e.g. COMP1511,COMP1521
e.g. COMP3###,INFS3###
e.g. select code from Courses ...

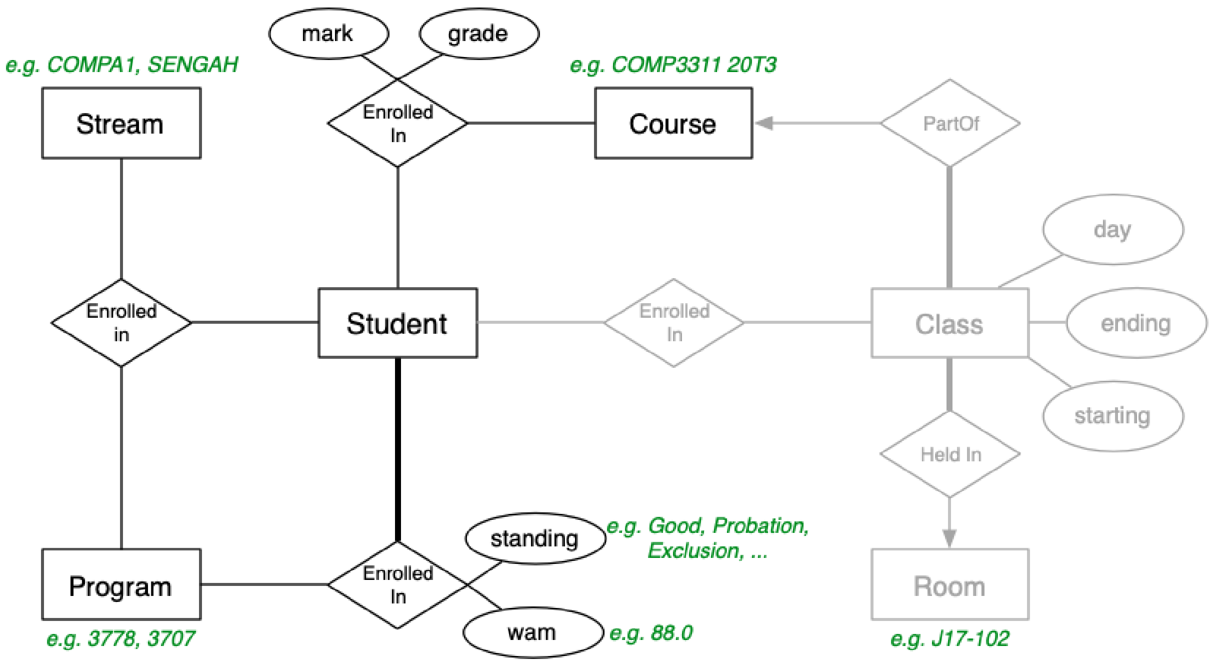
- an academic object group (*AcObjGroup*) specifies a set of academic objects of one type (e.g. a set of streams, set of courses, etc.)
- each *AcObjGroup* has a name which describes its purpose and is used for looking up groups when programs and streams are being defined
- the members of an *AcObjGroup* may be defined in three ways:
  - by enumeration (explicitly noting each member of the group)
  - by giving patterns to identify the members (e.g. `COMP3###`)
  - by giving an SQL query to lookup the members and return a set of IDs
- in the case of a set of requirements, you can also specify the logic of how the members are defined (conjunction or disjunction)

Note: in these patterns, a '#' is like a '_' in an SQL `like` pattern or a '.' in a regular expression pattern.

Both `enumerated` and `pattern` groups are defined using a definition which is comma-separated list of items. The difference is that `pattern` groups may have items like `COMP3###`, which matches any course with a code starting with "`COMP3`", while an `enumerated` group is simply a list of course or stream codes. One exception is that alternatives may be specified in an enumerated list as `{MATH1131;MATH1141}`; this means that a student can take either MATH1131 or MATH1141 to satisfy the group requirements.

Enrolment

Relationships for various kinds of enrolment ...



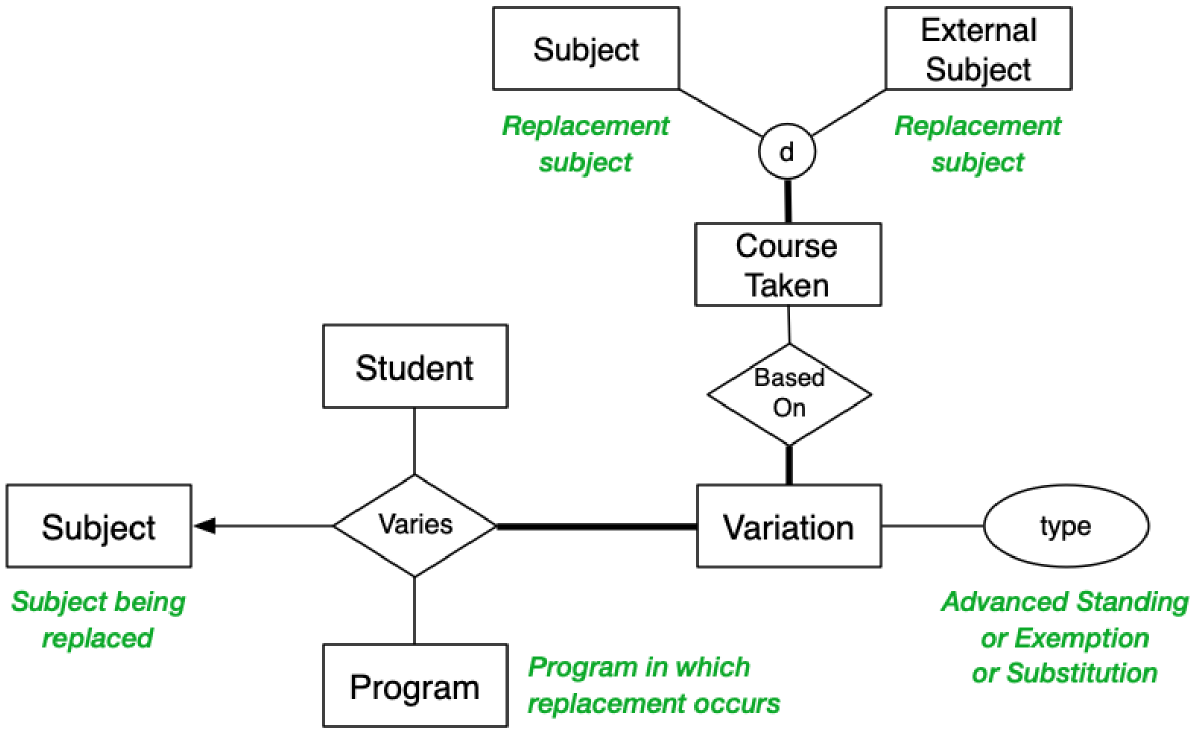
Comments:

- students initially start at UNSW by enrolling in a program
- once enrolled, they choose a specific stream (or streams) to study within that program
- this requires them to enrol in courses, and they generally enrol in one or more classes in the course (note that NSS uses enrolment in the lecture class as the way of indicating that a student is enrolled in a course; our schema does not do this)
- all these notions of enrolment have different kinds of information associated with them
- we have used total participation for Students-EnrollIn-Program to indicate that they must be enrolled in at least one program; it's an n:m relationship because, over time, they may enrol in other programs (e.g. complete their undergraduate degree and then later do a coursework masters degree)
- note that the mark alone is not sufficient to determine whether a student has successfully completed a course; they need a passing grade to ensure this (the set of grades indicating a pass includes SY, PC, PS, CR, DN, HD ... there is also a PT grade which is no longer used but is included to allow us to deal with old data)

Variations

Note: for this assignment we will be ignoring variations. We include their schema for the sake of completeness.

Variations of meeting requirements within programs ...



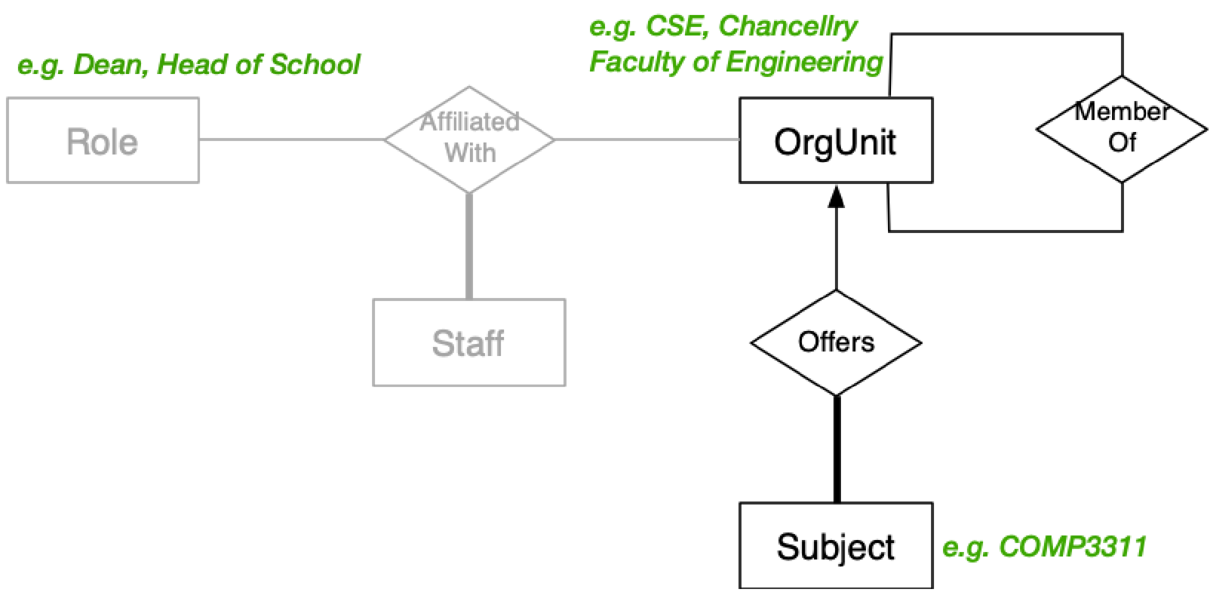
Comments:

- the purpose of a variation is to indicate that a student has effectively completed one course at UNSW, without necessarily having enrolled in and passed that course
- there are three kinds of variation: substitution, advanced standing, exemption
- a substitution might be used to replace a core course in a plan by some other course if the core course is not available
- advanced standing gives credit towards a degree, based on study towards a different degree either at UNSW or elsewhere (typically, advanced standing is only granted when the degree containing the course was not completed)
- including courses from other institutions means that we need a representation for them; we have used a very simple representation
- finally, exemptions allow us to record that a student has some specific background knowledge (to use as a pre-requisite for some other course); an exemption does not confer credit towards a degree, however

Organisational Units

Note: for this assignment, we are largely ignoring organisational units, except as owners of subjects.

Units (e.g. Faculties, Schools, Departments) within UNSW ...



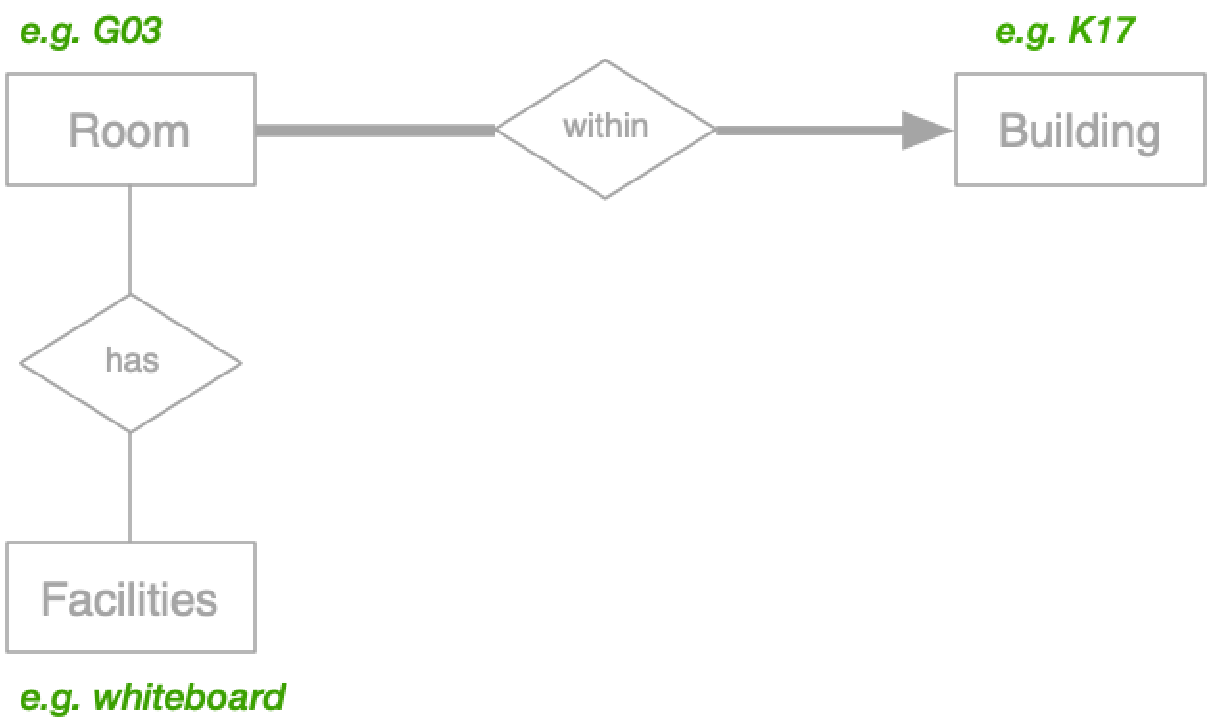
Comments:

- units are nested (e.g. the CSE school is in the Engineering Faculty which is within UNSW)
- this nesting is handled via a `UnitGroups` table
- there is a meta-orgunit `UNSW` which all of the faculties are members of
- since we don't have staff in the database, there's no need for affiliations
- the n:m relationship between staf:roles:orgunits means that a staff member could be affiliated with several orgiunits or could have multiple roles within a single orgunit

Rooms and Buildings

Note: for this assignment we will be ignoring rooms and buildings. We include their schema for the sake of completeness.

Places to hold classes, offices, etc.



SQL Schema

We have developed an SQL schema based on the above data model. With the above background, you are now in a good position to examine this schema, keeping in mind that the schema does not follow precisely what has been specified above in all cases.

Database Contents

The student/course/enrolment part of the database has been populated using real data which was transformed to make it anonymous (i.e. names, marks, enrolments changed to protect the innocent). The subject/program part of the database was populated from enrolment data. The requirements part will be populated from data in the ECLIPS database and the Handbook. The class data would have been populated using data from the UNSW timetable site, if we were including classes.

For this assignment, we have decided to focus on students enrolled in degrees offered by CSE (e.g. Computer Science, Software Engineering), and so only rules relevant to these degrees are included, and only enrolments by CSE students are included.

Note that the task of populating such a large database is extremely time-consuming and so many parts of the database are unpopulated or populated only very thinly. Some parts have data that is close to reality, while others have data that is a pale shadow of reality. In order to get a feeling for what is actually present, you must spend some time exploring the database when it is eventually populated.