COMP3311 22T1

Assignment 1
MyMyUNSW Schema

Database Systems

Last updated: **Sunday 27th February 3:07pm**

Most recent changes are shown in red ... older changes are shown in brown.

[Assignment Spec] [Database Design] **[Schema in SQL]** [check1.sql] [Examples] [Fixes+Updates]

schema.sql

```
-- COMP3311 22T1 Assignment 1 Schema
--
-- MyMyUNSW Schema
-- Original version: John Shepherd (Sept 2002)
-- Latest version: John Shepherd (August 2013)
-- Originally for PostgreSQL7.2
-- Conformed to SQL standard and ported to Oracle8: April 2003
-- Minor mods: April 2005, Sept 2006, April 2007, May 2008
-- Added extra Rule structures: March 2011
-- Adjusted some tables to fit MAPPS/AIMS data structures: August 2013
-- Simplified schema (e.g some tables removed): October 2020
--
-- Gives a standard SQL description for data to maintain information
-- about academic matters at UNSW. Options for simplifying the schema
-- by exploiting non-standard PostgreSQL features are marked with "PG:"
--
-- The notion is that this data should enable all of the functionality
-- currently provided by NSS, CATS, UNSW Staff directory, ...
--
-- To keep the schema a little shorter, I have ignored my usual
-- convention of putting foreign key definitions at the end of
-- the table definition.
--
-- Some general naming principles:
--   max 10 chars in field names
--   all entity tables are named using plural nouns
--   for tables with unique numeric identifier, always call the field "id"
--   for cases where there's a long name and a short name for something,
--      use "name" for the short version of the name (typically for display),
--      and use "longname" for the complete version of the name (which might
--      typically be used in lists of items)
--   for foreign keys referring to an "id" field in the foreign relation,
--      use the singular-noun name of the relation as the field name

--      OR use the name of the relationship being represented
--
-- Null values:
--  for each relation, a collection of fields is identified as being
--    compulsory (i.e. without them the data isn't really usable) and
--    they are all defined as NOT NULL
--  reminder: all of the primary keys (e.g. "id") are non-NULL
--  note also that fields that are allowed to be NULL will need to be
--    handled specially whenever they are displayed e.g. in a web-based
--    interface to this schema
```

```
--
-- Enum relations:
--   some relations in the schema contain little more than (id,name)
--   they were not done simply as varchar attributes:
--     for consistency (all relations referring them get common spelling, etc)
--     for efficiency (saves space in the referring relation)
--     for easier use in menus in the user interface
--   examples of such relations: Countries, Room_types, Job_classes, ...
--   you could argue that these should be replaced by PostgreSQL "enum"
--     types, but (a) enums are non-standard, and (b) if you want more
--     info than just a label (e.g. also want a description), you need
--     a table with extra fields
--
-- Meta information:
--   in a couple of cases, the data stored in the database needs to be
--     further interpreted before the actual results can be obtained
--   Examples:
--     Student groups: an SQL query is stored in the DB to extract a
--       list of students in the group
--     Rules: an expression in a simple "requirements language"
--       is stored in the DB and needs to be interpreted by a PLpgSQL
--       function to determine whether the requirements are met
--
-- Oracle port:
--   this schema was converted to standard SQL to run on Oracle in 2003
--   the PostgreSQL non-standard features have been retained as comments

-- Domains: specific kinds of values used throughout
--   In PostgreSQL, some could be defined as simple enumerated types
--   Since we're trying to be standard SQL, we use domains

-- ShortStrings are typically used for values appearing in tables in the UI
create domain ShortString as varchar(16);
create domain MediumString as varchar(64);
create domain LongString as varchar(256);
create domain TextString as varchar(16384);

-- ShortNames are typically used for values appearing in tables in the UI
create domain ShortName as varchar(16);
create domain MediumName as varchar(64);
create domain LongName as varchar(128);

-- If we could rely on having regexps, we could do a better job with these
create domain PhoneNumber as varchar(32);
create domain EmailString as varchar(64) check (value like '%@%');
create domain URLString as varchar(128) check (value like 'http://%');

create domain CareerType as char(2)
        check (value in ('UG','PG','HY','RS','NA'));

create domain GradeType as char(2)
        check (value in (
                'AF', 'AS', 'CR', 'DF', 'DN', 'EC', 'FL', 'FN',
                'GP', 'HD', 'LE', 'NA', 'NC', 'NF', 'PC', 'PE',
                'PS', 'PT', 'RC', 'RD', 'RS', 'SS', 'SY', 'UF',
                'WA', 'WC', 'WD', 'WJ', 'XE',
                'A', 'B', 'C', 'D', 'E'
        ));

create domain CampusType as char(1)
        check (value in (
                'K', -- Kensington
                'P', -- COFA/Paddington
                'Z', -- ADFA/UniCollege
                'C', -- CBD (Sydney)
                'X'  -- External
        ));


create domain CourseYearType as integer
        check (value > 1945);  -- UNSW didn't exist before 1945

create domain TermType as char(2)
        check (value in ('S1','S2','X1','X2','T0','T1','T2','T3'));


-- Countries: country codes and names

create table Countries (
        id          integer, -- PG: serial
        code        char(3) not null unique,
        name        LongName not null,
        primary key (id)
);

-- Buildings: building information
-- e.g. (1234, 'MB', 'Morven Brown Building', 'K', 'C20')
--      (5678, 'K17', 'CSE Building', 'K', 'K17')
--      (4321, 'EE', 'Electrical Engineering Building', 'K', 'G17)

create table Buildings (
        id          integer, -- PG: serial
        unswid      ShortString not null unique,
        name        LongName not null,
        campus      CampusType,
        gridref     char(4),
        primary key (id)
);



-- Room_types: different kinds of rooms on campus
-- e.g. 'Lecture Theatre', 'Tutorial Room', 'Office', ...

create table Room_types (
        id          integer, -- PG: serial
        description MediumString not null,
        primary key (id)
);
```

```
-- Rooms: room information

create table Rooms (
        id          integer, -- PG: serial
        unswid      ShortString not null unique,
        rtype       integer references Room_types(id),
        name        ShortName not null,
        longname    LongName,
        building    integer references Buildings(id),
        capacity    integer check (capacity >= 0),
        primary key (id)
);


-- Facilities: things in rooms (e.g. data projector, OHP, etc.)

create table Facilities (
        id          integer, -- PG: serial
        description MediumString not null,
        primary key (id)
);


-- Room_facilities: which facilities are available in which rooms

create table Room_facilities (
        room        integer references Rooms(id),
        facility    integer references Facilities(id),
        primary key (room,facility)
);


-- OrgUnit_types: kinds of organisational units at UNSW
-- notes:
--   examples: 'Faculty', 'School', 'Division',...
--   used so that people can invent other new units in the future

create table OrgUnit_types (
        id          integer, -- PG: serial
        name        ShortName not null,
        primary key (id)
);



-- OrgUnits: organisational units (e.g. schools, faculties, ...)
-- notes:
--   "utype" classifies the organisational unit
--

create table OrgUnits (
        id          integer, -- PG: serial
        utype       integer not null references OrgUnit_types(id),
        name        MediumString not null,
        longname    LongString,
        unswid      ShortString,
        phone       PhoneNumber,
        email       EmailString,
        website     URLString,
        starting    date, -- not null
        ending      date,
        primary key (id)
);



-- OrgUnit_groups: how organisational units are related
-- notes:
--   allows for a multi-level hierarchy of groups

create table OrgUnit_groups (
        owner       integer references OrgUnits(id),
        member      integer references OrgUnits(id),
        primary key (owner,member)
);



-- Teaching Periods (aka terms, sessions, semesters)
-- notes:
--   all dates should be not null, but we don't have access to them

create table Terms (
        id          integer, -- PG: serial
        unswid      integer not null unique,
        year        CourseYearType,
        session     char(2) not null, -- has constraint in database
        name        ShortName not null,
        longname    LongName not null,
        starting    date not null,
        ending      date not null,
        startBrk    date, -- start of mid-semester break
        endBrk      date, -- end of mid-semester break
        endWD       date, -- last date to withdraw without academic penalty
        endEnrol    date, -- last date to enrol without special permission
        census      date, -- last date to withdraw without paying for course
        primary key (id)
);



-- Public_holidays: days when regular teaching is cancelled
-- These could be done as WholeDay/OneOff Events, but they would also
--   need to generate exceptions for all of the Class Events scheduled
--   on those days
-- Notice that there's no primary key; there could be several holidays
--   (e.g. different religions) on the same date

create table Public_holidays (
        term        integer references Terms(id),
        description MediumString, -- e.g. Good Friday, Easter Day
        day         date
);


-- Staff_roles: roles for staff within the UNSW organisation
-- handles job classes under which staff are employed
-- e.g. "Associate Lecturer", "Professor", "Administrative Assistant",
--      "Computer Systems Officer", "Clerk", "Caterer"
-- and also handles specific roles for some staff members
-- e.g. "Vice Chancellor", "Dean", "Head of School",
--      "Teaching Director", "Admin Assistant to Dean",
--      "School Office Manager", ...
-- this could either describe the specific duties under the
--   job classification, or duties that are additional to the
--   basic job classification
-- notes:
--   in the real NSS, hooks to the HR system would be here
--   for example, we might have base salary for each role
--   which represent a job classification

create table Staff_role_types (
        id          char(1),
        description ShortString,
        primary key (id)
);

create table Staff_role_classes (
        id          char(1),
        description ShortString,
        primary key (id)
);

create table Staff_roles (
        id          integer, -- PG: serial
        rtype       char(1) references Staff_role_types(id),
        rclass      char(1) references Staff_role_classes(id),
        name        LongString not null,
        description LongString,
        primary key (id)
);


-- People super-class
-- contains:
--   unique id internal to database
--   personal information
--   home contact info
-- notes:
--   family,given names are displayed on transcripts
--   sortname is to handle unusual names (e.g. de Kleer as K)
--   name is what will be displayed (except on transcripts)
--        it allows preferred form of name(s) to be used
--   phone numbers are assumed to be Australian numbers
--   the phone field sizes allow for future expansion of phone #s
--   familyname is allowed to be null for people with only one name
--   the "not null" fields indicate which info is compulsory
--   nowadays, people are required to have an email address
--   the password field is used by the web interface
--   allows people in the database who are not staff or students
--     e.g. members of the University Council

create table People (
        id          integer, -- PG: serial
        unswid      integer unique, -- staff/student id (can be null)
        password    ShortString not null,
        family      LongName,
        given       LongName not null,
        title       ShortName, -- e.g. "Prof", "A/Prof", "Dr", ...
        sortname    LongName not null,
        name        LongName not null,
        street      LongString,
        city        MediumString,
        state       MediumString,
        postcode    ShortString,
        country     integer references Countries(id),
        homephone   PhoneNumber, -- should be not null
        mobphone    PhoneNumber,
        email       EmailString not null,
        homepage    URLString,
        gender      char(1) check (gender in ('m','f')),
        birthday    date,
        origin      integer references Countries(id),  -- country where born
        primary key (id)
);


-- Student (sub-class): enrolment type

create table Students (
        id          integer references People(id),
        stype       varchar(5) check (stype in ('local','intl')),
        primary key (id)
);

-- Student_groups: groups of students (used in specifying quotas)
-- uses SQL queries stored in the database to extract lists of
--   students belonging to particular classes

-- decided to use this approach rather than explicitly storing
--   lists of (student,group) pairs because these lists would
--   be very large and hard to setup and maintain
-- of course, with this approach, getting a list of students
--   in a given group requires something beyond SQL (e.g. PLpgSQL)

create table Student_groups (
        id          integer, -- PG: serial
        name        LongName unique not null,
        definition  TextString not null, -- SQL query to get student(id)'s
        primary key (id)
);



-- Staff (sub-class): employment and on-campus contact info
-- all staff have a unique staff id different to their person id
-- anyone who teaches a class has to be entered in this table
--   (they would normally be entered into the UNSW HR database)

create table Staff (
        id          integer references People(id),
        office      integer references Rooms(id),
        phone       PhoneNumber, -- full number, not just extension
        employed    date not null,
        supervisor  integer references Staff(id),
        primary key (id)
);



-- Affiliations: staff roles and association to organisational units
-- notes:
--   most staff will be attached to only one unit
--   "role" will describe things like "Professor", "Head of School", ...
--   if this is their job class for HR, isPrimary is true

create table Affiliations (
        staff       integer references Staff(id),
        orgUnit     integer references OrgUnits(id),
        role        integer references Staff_roles(id),
        isPrimary   boolean, -- is this role the basis for their employment?
        starting    date not null, -- when they commenced this role
        ending      date,  -- when they finshed; null means current
        primary key (staff,orgUnit,role,starting)
);



-- Programs: academic details of a degree program
-- notes:
--   the "code" field is used for compatability with current UNSW practice
--     e.g. 3978 is the code for the computer science degree

create table Programs (
        id          integer, -- PG: serial
        code        char(4) not null, -- e.g. 3978, 3645, 3648
        name        LongName not null,
        uoc         integer check (uoc >= 0),
        offeredBy   integer references OrgUnits(id),
        career      CareerType,
        duration    integer,  -- #months
        description TextString, -- PG: text
        firstOffer  integer references Terms(id), -- should be not null
        lastOffer   integer references Terms(id), -- null means current
        primary key (id)
);
```

```
-- Streams: academic details of a major/minor stream(s) in a degree

create table Streams (
        id          integer, -- PG: serial
        code        char(6) not null, -- e.g. COMPA1, SENGA1
        name        LongName not null,
        offeredBy   integer references OrgUnits(id),
        stype       ShortString,
        description TextString,
        firstOffer  integer references Terms(id), -- should be not null
        lastOffer   integer references Terms(id), -- null means current
        primary key (id)
);


-- Degree_types: types of awards for degrees

create table Degree_types (
        id          integer, -- PG: serial
        unswid      ShortName not null unique, -- e.g. BSc, BSc(CompSci), BE, PhD
        name        MediumString not null,  -- e.g. Bachelor of Science
        prefix      MediumString,
        career      CareerType,
        aqf_level   integer check (aqf_level > 0),
        primary key (id)
);


-- Program_degrees: degrees awarded for each program
--   a concurrent degree will have two entries for one program

create table Program_degrees (
        program     integer references Programs(id),
        degree      integer references Degree_types(id),
        name        LongString not null,
        abbrev      MediumString,
        primary key (program,degree)
);


-- Degrees_awarded: info about student being awarded a degree


create table Degrees_awarded (
        student     integer references Students(id),
        program     integer references Programs(id),
        graduated   date,
        primary key (student,program)
);


-- Academic_standing: kinds of academic standing at UNSW
-- e.g. 'good', 'probation1', 'probation2',...
-- An enumerated-type table

create table Academic_standing (
        id          integer,
        standing    ShortName not null,
        notes       TextString,
        primary key (id)
);


-- Subjects: academic details of a course (version)
-- "code" is standard UNSW course code (e.g. COMP3311)
-- "firstOffer" and "lastOffer" indicate a timespan during
--   which this subject was offered to students; if "lastOffer"
--   is null, then the subject is still running
-- Note: UNSW calls subjects "courses"

create table Subjects (
        id          integer, -- PG: serial
        code        char(8) not null,
--                    PG: check (code ~ '[A-Z]{4}[0-9]{4}'),
        name        MediumName not null,
        longname    LongName,
        uoc         integer check (uoc >= 0),
        offeredBy   integer references OrgUnits(id),
        eftsload    float,
        career      CareerType,
        syllabus    TextString, -- PG: text
        contactHPW  float, -- contact hours per week
        _excluded   text,    -- plain text from MAPPS
        excluded    integer, -- references Acad_object_groups(id),
        _equivalent text,    -- plain textfrom MAPPS
        equivalent  integer, -- references Acad_object_groups(id),
        _prereq     text,    -- plain text from MAPPS
        prereq      integer, -- references Rules(id)
        replaces    integer references Subjects(id),
        firstOffer  integer references Terms(id), -- should be not null
        lastOffer   integer references Terms(id), -- null means current
        primary key (id)
);


-- Course: info about an offering of a subject in a given term
-- we insist on knowing the lecturer because there's no point running
--   a course unless you've got someone organised to lecture it
-- Note: UNSW calls courses "course offerings"

create table Courses (
        id          integer, -- PG: serial
        subject     integer not null references Subjects(id),
        term        integer not null references Terms(id),
        homepage    URLString,
        primary key (id)
);
```

```
-- Course_staff: various staff involved in a course
-- allows one Staff to have multiple roles in a course

create table Course_staff (
        course      integer references Courses(id),
        staff       integer references Staff(id),
        role        integer references Staff_roles(id),
        primary key (course,staff,role)
);


-- Course_quotas: quotas for various classes of students in a course
-- if there's no quota, there's no entry in this table
-- alternatively, we could have allowed quota to be null
--   and used that as a mechanism for indicating "no quota"

create table Course_quotas (
        course      integer references Courses(id),
        sgroup      integer references Student_groups(id),
        quota       integer not null,
        primary key (course,sgroup)
);


-- Program_enrolments: student's enrolment in a program in one term
-- notes:
--   "standing" refers to the students academic standing
--   "wam" is computed from marks in enrolment records

create table Program_enrolments (
        id          integer,
        student     integer not null references Students(id),
        term        integer not null references Terms(id),
        program     integer not null references Programs(id),
        wam         real,
        standing    integer references Academic_standing(id),
        advisor     integer references Staff(id),
        notes       TextString,
        primary key (id)
);


-- Stream_enrolments: student's enrolment in streams in one term

create table Stream_enrolments (
        partOf      integer references Program_enrolments(id),
        stream      integer references Streams(id),
        primary key (partOf,stream)
);


-- Course_enrolments: student's enrolment in a course offering
-- null grade means "currently enrolled"
-- if course is graded SY/FL, then mark always remains null

create table Course_enrolments (
        student     integer references Students(id),
        course      integer references Courses(id),
        mark        integer check (mark >= 0 and mark <= 100),
        grade       GradeType,
        stuEval     integer check (stuEval >= 1 and stuEval <= 6),
        primary key (student,course)
);


-- Books: textbook details

create table Books (
        id          integer, -- PG: serial
        isbn        varchar(20) unique,
        title       LongString not null,
        authors     LongString not null,
        publisher   LongString not null,
        edition     integer,
        pubYear     integer not null check (pubYear > 1900),
        primary key (id)
);


-- Course_books: relates books to courses
-- books are related to a Course rather than a Subject because texts
--   may change over time, even if the syllabus remains constant

create table Course_books (
        course      integer references Courses(id),
        book        integer references Books(id),
        bktype      varchar(10) not null check (bktype in ('Text','Reference')),
        primary key (course,book)
);


-- ClassType: names for different kinds of class
-- e.g. "Lecture", "Tutorial", "Lab Class", ...

create table Class_types (
        id          integer, -- PG: serial
        unswid      ShortString not null unique,
        name        MediumName not null,
        description MediumString,
        primary key (id)
);


-- Classes: a specific regular teaching event in a course
-- we ignore streams, since they make class registration too messy
-- we don't allow day/time/place info to be null; this forces us to
--   already organise a time/place before we enter them in the system
-- weekly repetitions are handled by (repeats=1 or repeats is null)
-- we assume that all classes are multiples of 1-hour in duration
--   and cannot start before 8am or finish after 11pm)

create table Classes (
        id          integer, -- PG: serial
        course      integer not null references Courses(id),
        room        integer not null references Rooms(id),
        ctype       integer not null references Class_types(id),
        dayOfWk     integer not null check (dayOfWk >= 0 and dayOfWk <= 6),
                                          -- Sun=0 Mon=1 Tue=2 ... Sat=6
        startTime   integer not null check (startTime >= 8 and startTime <= 22),
        endTime     integer not null check (endTime >= 9 and endTime <= 23),
                                          -- time of day, between 8am and 11pm
        startDate   date not null,
        endDate     date not null,
        repeats     integer, -- every X weeks
        primary key (id)
);


-- Class_teachers: who teaches which class
-- unfortunately, no way to describe how two staff who
--   are allocated to a given class teach together
--   e.g. teach on alternating weeks

create table Class_teachers (
        class       integer references Classes(id),
        teacher     integer references Staff(id),
        primary key (class,teacher)
);


-- Class_enrolments: one student's enrolment in a class

create table Class_enrolments (
        student     integer references Students(id),
        class       integer references Classes(id),
        primary key (student,class)
);


-- External_subjects: represents courses from other institutions
-- used to ensure consistency in awarding advanced standing
-- if student X gets advanced standing based on course Y at Z,
--   then a later student who has done course Y at Z can be given
--   the same advanced standing
-- to do this properly, we'd need to set up a table of external
--   institutions and use a foreign key ... as it stands, if
--   people award credit for the same course, but spell either
--   the course name or the institution name differently, it
--   will be treated as a different course
```

```
create table External_subjects (
        id          integer,
        extsubj     LongName not null,
        institution LongName not null,
        yearOffered CourseYearType,
        equivTo     integer not null references Subjects(id),
--      creator     integer not null references Staff(id),
--      created     date not null,
        primary key (id)
);



-- Variations: replacement of one subject or another in a program
-- handles several cases (which are more or less similar):
--   advanced standing for courses studied either at UNSW or elsewhere
--   substitution of one course for another to satisfy requirements
--   exemption from one course, to use as a prerequisite
-- in the case of exemptions, no credit is granted towards a program;
--   the subject is being recorded to use as a pre-req
-- the substitution is for one subject towards the requirements
--   of one stream
-- there are two sub-cases represented in this single table:
--   the subject is an internal UNSW subject (internal equivalence)
--   the subject is from outside UNSW (external equivalence)
-- can't enter Advanced Standing without saying who you are, since
--   Advanced Standing is like awarding a pass in a UNSW course
-- if we wanted to record external subjects being used as a basis
--   for pre-requisites but not credit (i.e. exemption), we would
--   need to add a new field to indicate that no credit was involved

create domain VariationType as ShortName
        check (value in ('advstanding','substitution','exemption'));

create table Variations (
        student     integer references Students(id),
        program     integer references Programs(id),
        subject     integer references Subjects(id),
        vtype       VariationType not null,
        intEquiv    integer references Subjects(id),
        extEquiv    integer references External_subjects(id),
        yearPassed  CourseYearType,
        mark        integer check (mark > 0), -- if we know it
        approver    integer not null references Staff(id),
        approved    date not null,
        primary key (student,program,subject),
        constraint  TwoCases check
                      ((intEquiv is null and extEquiv is not null)
                      or
                       (intEquiv is not null and extEquiv is null))
);

-- Acad_object_groups: groups of different kinds of academic objects
--  academic objects = courses OR streams OR programs
-- different kinds of academic objects that can be grouped
-- each group consists of a set of objects of the same type

create domain AcadObjectGroupType as ShortName
        check (value in (
                'subject',      -- group of subjects
                'stream',       -- group of streams
                'program'       -- group of programs
        ));

-- how to interpret combinations of objects in groups

create domain AcadObjectGroupLogicType as ShortName
        check (value in ( 'and', 'or'));

-- how groups are defined

create domain AcadObjectGroupDefType as ShortName
        check (value in ('enumerated', 'pattern', 'query'));

-- there are some constraints in this table that we haven't implemented

create table Acad_object_groups (
        id          integer,
        name        LongName,
        gtype       AcadObjectGroupType not null,
        glogic      AcadObjectGroupLogicType,
        gdefBy      AcadObjectGroupDefType not null,
        negated     boolean default false,
        parent      integer, -- references Acad_object_groups(id),
        definition  TextString, -- if pattern or query-based group
        primary key (id)
);

alter table Acad_object_groups
        add foreign key (parent) references Acad_object_groups(id);

alter table Subjects
        add foreign key (excluded) references Acad_object_groups(id);

alter table Subjects
        add foreign key (equivalent) references Acad_object_groups(id);

-- Each kind of AcademicObjectGroup requires it own membership relation


create table Subject_group_members (
        subject     integer references Subjects(id),
        ao_group    integer references Acad_object_groups(id),
        primary key (subject,ao_group)
);

create table Stream_group_members (
        stream      integer references Streams(id),
        ao_group    integer references Acad_object_groups(id),
        primary key (stream,ao_group)
);

create table Program_group_members (
        program     integer references Programs(id),
        ao_group    integer references Acad_object_groups(id),
        primary key (program,ao_group)
);

-- Rules: requirements for programs and stream, pre-reqs for subjects

create domain RuleType as char(2)
        check (value in (
                'CC', -- core courses ... with min, max, subject group
                'PE', -- program electives ... with min, max, subject group
                'FE', -- free electives ... with min, max, group with FREE?###
                'GE', -- general education ... with min, max, group with GEN??###
                'RQ', -- subject pre-req ... typically with min, max, subject group
                'WM', -- WAM requirement ... typically with min WAM score
                'LR', -- limit rule ... with min or max, big subject group (####...)
                'MR', -- maturity rule ... with min UOC and (optionally) a subject gr
                'DS', -- done stream ... with min, max, stream group
                'RC', -- recommended ... with subject group, useful for suggestions
                'IR'  -- information rule ... doesn't need checking
    ));

-- Various types of rules ...
-- Some rules require reference to a group of subjects or streams
-- min/max can have different kinds of units depending on rule type
--   (frequently they are UOC, sometimes just counters)
-- Rule names don't have a standard form and are not very useful
-- Rule descriptions are slightly more useful

create table Rules (
        id          integer,
        name        MediumName,
        type        RuleType,
        min         integer check (min >= 0),
        max         integer check (min >= 0),
        ao_group    integer references Acad_object_groups(id),
        description TextString,
        primary key (id)
);


create table Subject_prereqs (
        subject     integer references Subjects(id),
        career      CareerType, -- what kind of students it applies to
        rule        integer references Rules(id),
        primary key (subject,career,rule)
);

create table Stream_rules (
        stream      integer references Streams(id),
        rule        integer references Rules(id),
        primary key (stream,rule)
);

create table Program_rules (
        program     integer references Programs(id),
        rule        integer references Rules(id),
        primary key (program,rule)
);
```