Design Doc Version 2 - Team 2

Gee, Shouse, Starbuck, Waddle

INPUT: using HTML, PHP, a CRN and the optimal NEW time and room number
(or same room number if not specified)...

OUTPUT: list of least to greatest conflicts per time slot and room number to a web page

We need:
Student schedules put into a clean MySQL database,
Student classification (we really just care about Senior yes or no),
Availability of rooms in the database (available, yes or no)

Project Areas:

- Web site:
  - git repo maps to Web directory on server
  - filesystem setup

[Adam and Ethan]

- User interface: (HTML)
  - usability
    - styling
  - Web filesystem design (how do we structure the web site as a whole; e.g. anchors/links
  - About us (We want to be the very best, and perhaps catch 'em all)
- Databases: (MySQL)
  - setting up MySQL
  - designing schema
  - student schedules
  - course information
  - building information

[Roger and Jonathan]

- Backend: (PHP)
  - PHP APIs for Database access (higher level abstractions in code to the database)
  - class libraries for each entity kind
  - PHP interface code to User interface (Web pages with embedded PHP)
  - PHP APIs for necessary computations

Project Areas (detail):

Web site filesystem setup -

```
.
|— aboutus.html
|— app
|   |— conflictor.php
|   └── index.php
|— browse
|   |— course.php
|   |— index.php
|   └── schedule.php
|— index.php
|— libs
|   |— conflicts-api.php
|   └── databases.php
```

User interface - We construct a user interface using the file structure described above

Goals: We want to view registration information (mini-Banner) through the site. Then we want to provide an interface for proposing moving course times and seeing conflicts.

- **Home Page**
    - Simple search – think "Google home page"
- **Browse course page**
  (minimize output)
    - of kind
    - in building (2D misfits approach?)
    - by professor
    - hyperlink to schedules of students enrolled in course
    - ETC.
- **Browse student schedule**
    - hyperlink to courses in which the student is enrolled
- **Course editing page**
    - select a course and change its time
    - report conflicts (minimized)
    - options: provide interface to set options to tailor the algorithm
        - raw versus tailored conflicts
        - scope of the run (check all room? check all buildings?)

Databases:
We describe database in the file design/schema.txt.

Backend -
We store PHP libraries in libs/. These are different from PHP interface files. They contain just APIs (no code executed by default; contain class hierarchies and functions) for accomplishing the tasks of our Web application.

libs/databases.php
Contains APIs for accessing database entries; provides a programming interface to objects extracted from the site's database; this interface is used by conflicts-api.php to implement the site's functionality.

libs/conflicts-api.php
Contains APIs that apply algorithms to user input data and database objects; this is where we implement the core functionality of what our application does.

We discuss these in more detail later on in this document.

Intro.

We here describe database schema for our project, including entities and their relationships.

A. Entities

```
Entity List:
------------
student
professor
course
office-hours
building
room

student
-------
id (key)          --Banner id
first-name
middle-initial
last-name
classification
hours

professor
---------
id (key)
first-name
middle-initial
last-name
```

```
course
------
crn (key)
department            --e.g. CS, IT, or MATH
number                --e.g. 120, 221, or 331
professor-id
name                  --"Design and Analysis
                      of Algorithms"
time                  --start time
duration              --integer minutes
dow                   --string: "M" "MWF"
                        "MW" "TR"
room-id               --(will map to building)

student-course          --intersection data for
                          student and course
--------------
student-id (key)        --dual primary key
course-id  (key)

office-hours
------------
id (key)
time                  --start time
duration              --integer minutes
dow                   --string: "M" "MWF"
                      "MW" "TR"
room-id                      --(maps to building)

office-hours-assignment         --intersection
data for office-hours and professor
-----------------------
professor-id
office-hours-id (key)    -- this MUST be sole
                            primary key so that
                            only one professor
                            may be assigned to an
                            office-hours entity

building
--------
id (key)
name              --full name: "Mabee Business
                    Building"
abbrv            --abbreviation: MBB

room
----
id (key)
number
building-id
has-lab           --Boolean
```

B. Relationships

- We define the relationships between entities in this section.
- 'Intersection Data' includes entities that function to relate different entities and is not a direct entity to be related in and of itself.
- The HAS MANY relationship implies 'one or more'. The HAS A relationship implies 'only one'.

Entity List:
----
student
professor
course
office-hours
building
room

Intersection Data:
----
student-course
office-hours-assignment

STUDENT has many COURSEs	COURSE has many STUDENTs
PROFESSOR has many COURSEs	COURSE has a PROFESSOR
COURSE has a ROOM	ROOM has many COURSEs
BUILDING has many ROOMs	ROOM has a BUILDING
PROFESSOR has many OFFICE-HOURs	OFFICE-HOUR has a PROFESSOR

API's

Intro

- Here we propose an API for managing database objects.
- This API will be written in PHP.
- We stay away from describing any HTML related output tasks. We design these APIs to output primitive (integer, string, and array of primitive) data types.

A. Database access

We will encapsulate database connection and querying in a class that will serve as a base to classes representing each kind of entity stored in the database. This will help streamline database access and allow for easier coding of later functionality. Each of these subclasses will support retrieving a pre-existing object from the database or creating a new one.

```
DatabaseObject
--------------
c-stor()

abstract create(/*variable argument list*/)      --create from variable input data; constructor will invoke
                                                   this potentially; can init default
abstract create_from_key($primary_key)           --create object from primary-key (in other words, make
                                                   a database query)
abstract create_from_row($row_description)       --create object from various fields (row description:
                                                   '$field=$value')
static protected connection                      --connection object shared between all instances
static protected query($query_string)            --perform database query

Student extends DatabaseObject
------------------------------
first_name()
last_name()
middle_initial()
classification()
hours()
array get_courses()

Professor extends DatabaseObject
--------------------------------
first_name()
middle_initial()
last_name()
```

```
array get_office_hours()          --might have multiple office-hour sessions

Course extends DatabaseObject
-----------------------------
department()
number()
title()
full_title()          --department() . " " . number() . " " . title()
time()
duration()
day_of_week()
room()                --retrieve actual room object, not just id
ConflictDescription compute_conflicts($course)      --maybe make this take '$changeTime' as parameter
array get_students()
```

**note: compute conflicts finds raw conflict information in the form of a 'ConflictDescription' object

```
OfficeHours extends DatabaseObject
----------------------------------
time()
duration()
day_of_week()
room()
get_professor()

Building extends DatabaseObject
-------------------------------
name()
name_abbrv()          --abbreviated name

Room extends DatabaseObject
---------------------------
number()
building()
has_lab()
--------------------------------------------------------------------------------
```

B. Conflicts API

- The conflicts API will be used to implement the algorithms needed to compute course time-change conflicts. There is a 'compute_conflicts()' method bound to any course object. It returns an object of type 'ConflictDescription' which can be used to describe course conflicts and perform computations based on conflict information.

- This object is primarily used to analyze conflicts, not find them (that has already been accomplished at this point).

- Any 'ConflictDescription' contains conflict information stored as a list of students, each with a set of conflicting courses. Most likely the set of conflicting courses will contain only 1 course, however it is possible that it contain more than one.

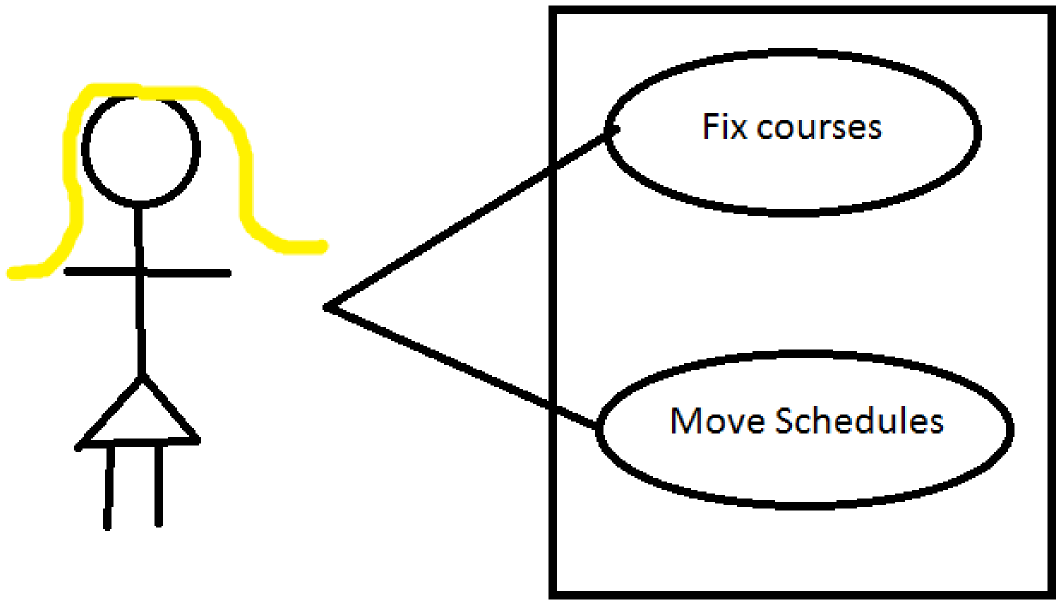
```
ConflictDescription
-------------------

[impl]
$course              --course being altered
$changeTime          --information about course time change
$changeDuration
$flicts = array(array($student,$course)[, ...])      --store students and conflicting courses; there might be
                                                        more than one conflict per instance; such
[interface]                                          --as when the move time overlaps more than one conflicting
course
count()                                              --retrieve number of conflicts
student_exists($student)   --does the specified student have a conflict (exists in the ConflictDescription)?
```

C. Optimizations

Sometimes a batch of database entities is required for some operation. In this case it is efficient to limit the number of database queries. At other times, it is efficient to make a query that extracts multiple objects at once across multiple tables. This may involve extracting a partial object (customized).
We provide APIs for accomplishing these batch/custom queries.

```
function BatchQuery($query, array $typeList); // creates row_description strings to create
DatabaseObjects
```

Use Cases


Fix courses
Move Schedules