

Relation Schemas

Our Database Design implements both foreign key constraints and foreign key tables to allow for a wide variety of data manipulation in a safe way via queries.

The foreign key tables were used on most many to many relations as they give a connection between tables over multiple rows.

users - Grants login privileges to the system. Is extended by Staff and Instructor.

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| user\_id | int(255) | Unique identifier for the user |
| username | text | For login |
| password | text | For login |
| firstname | text | Personal info |
| lastname | text | Personal info |
| email | text | Personal info |

staff - is extended from a user\_id to allow staff on a user login

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| staff\_id | int(255) | Implementation Purposes |
| user\_id | int(255) | Link to user |
| is\_manager | int(1) | Modifies privileges |

instructors - extended from a user\_id to allow instructor on a user login

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| instructor\_id | int(11) | Unique id |
| user\_id | int(11) | Foreign key |
| type | text | Type of instructor |
| title | text | Title |
| tenured | int(1) | Boolean if they are tenured |
| distribution\_preference | int(11) | Distribution preference for work load |
| date\_joined | text | Date they were entered into the system |

teaches - uses foreign keys to allow multiple professors to teach multiple classes

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| user\_id | int(255) | Foreign id |
| course\_id | int(255) | Foreign id |

preferences - makes a list of instructors preferences for their course

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| instructor\_id | int(11) | Foreign id |
| course\_id | int(11) | Foreign id |
| preference | int(1) | Preference for the course |

special\_request - list that is filled by professors

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| special\_request\_id | int(11) | Primary key |
| user\_id | int(11) | Foreign Key |
| course\_code | text | Course Code |
| title | text | Course Title |
| justification | text | The justification of making this class for the department |

courses - courses for the department

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| course\_id | int(255) | Primary Key |
| course | text | Course Title |
| section | text | Section |
| section\_capacity | int(11) | Section Capacity |
| start\_time | text | Start Time |
| end\_time | text | End Time |
| building | text | Building |
| room | text | Room |
| room\_capacity | int(11) | Capacity |
| enrollment | int(11) | Enrollment |
| ta\_grader | text | Grader Name |
| ta\_hour | int(11) | How many hours the TA will work for the class in a week |
| catalog\_year | text | What catalog |
| academic\_year | text | What year |
| course\_semester | text | What semester |

textbooks - textbooks that can be used by any course

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| textbook\_id | int(11) | Primary Key |
| title | text | Title of Book |
| author | text | Author |
| edition | text | Edition |
| isbn | text | ISBN |
| publisher | text | Publisher |

course\_textbooks - foreign keys to allow multiple textbooks for any course

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| course\_id | int(11) | Foreign Key |
| textbook\_id | int(11) | Foreign Key |

Problem sql:

1. get @instructor,

@n

SELECT courseCode, courseTitle, Semester, Enrollment, Building

FROM(

SELECT \*

FROM courses C

JOIN teaches T on C.course\_id = t.course\_id

JOIN users U on T.user\_id = U.user\_id

WHERE C.academic\_year > (DATE(get current year)-@n),

AND U.username== @instructor

ORDER BY year DESC  
)

3. get @courseNum,

@n

Select C.section, U.userName, C.enrollment, C.date

from courses C

JOIN teaches T on C.course\_id = t.course\_id

JOIN users on T.user\_id = U.user\_id

WHERE C.academic\_year > (DATE(get current year)-@n),

AND c.course = @courseNum

order by year

1. get @n

select U.userName, P.preference, C.courseName

from courses C

right join preferences P on C.course\_id = P.course\_id

left join users U on P.user\_id = U.user Id

where P.academic\_year == @n

2. get @n

select C.courseCode, U.username, C.enrollment

from courses C

JOIN teaches T on C.course\_id = T.course\_id

JOIN users on T.user\_id = U.user\_id

WHERE C.academic\_year > (DATE(get current year)-@n),

AND C.semester == 'Summer I'

OR C.semester == 'Summer II'

Our php/MySQL program was set up using some slightly unconventional methods, in comparison to the other students. We set up our program using a plugin called Smarty, which allows us to separate out the php from the html, javascript, and css. We also used Bootstrap, a liquid layout, and css plugin developed by the design team at twitter to start our styling. The page began as a php page where one signs in, then a new page is called depending on the user’s credentials, weither they be an admin, a teacher, or a business manager. These pages then have clear buttons to determine what action to perform. This buttons will call a php file, including a html template to layout the page. This page will contain a form for the user to fill in information required for their task. After this, submitting the form will call javascript to pass the information to a new php file, desined to directly interact with the sql database. These may either insert, update, delete, or retrieve the data. Depending on what has been done, information about it is then passed back via json, and displayed on the same page that the action was performed from. This allows the user to complete many actions at once, and not break workflow.

By basing our design off of bootstrap, we created a simple, yet stylized base. From this, we tried to keep every new page as similar as possible to previous pages, in order to not confuse. We also tried to keep a straight forward design, of mostly vertical, centered divs, in a very limited set of solid colors. It can’t get much simpler than that.

As the project started, Noah and Derrick began work on the uml, and nearly completed it before Brian came onto the team. The uml diagram was then finalized as a group. After another group discussion on what everyone was most comfortable with, and how we would proceed with the project, Derrick then set up the sql table, the basic folder layout, and the required php, and smarty template files. Derrick then continued to produce the php files to interact with the SQL server, while Brian and Noah wrote the html templates, and required JavaScript to interact with the php. After the administrative input was completed, Derrick then wrote the teacher input, back and front end alike, as Brian and Noah continued to work on the bm portion of the page. Before this, Noah continued to add to the database, and inserted the provided data. Brian and Noah then collaboratively completed this report, and installation instructions.