Lisa Percival CS340 Project

Outline

The database content is the back-end for a website I call ClassDoor that functions similarly to glassdoor.com, but for academia. There are mechanisms for users to write and view reviews of schools, programs, and professors. Through the website, one can create an account (add a reviewer), create a review (of one of the 3 types), add a school, add a program, add a professor, view a set of reviews filtered by criteria such as school, view information on a school, view information on a program, view information on a professor, view a reviewer's profile, edit your account (modify a reviewer), indicate that a professor works at an additional school (edit a professor), delete a review, and delete a program. This involves creating and updating a variety of data to provide the functionality of reviewing academia.

This data is especially well-suited to an interesting final project because there are a number of relationships with some complexity, which are further outlined in detail in the ER diagram, along with the details of entity attributes. The most important relationship is that a review will apply to a single school, program or professor. Since reviews will be implemented as 3 types, this means that the school/ program/ professor a SchoolReview/ ProgramReview/ ProfessorReview applies to, respectively, is an inherent attribute of the review. Similarly, all types of reviews also have a single author. This database incorporates two many-to-many relationships, which occur between schools and the reviewers who have attended them and between schools and the professors who have worked at them.

Database Outline in Words

The entities I will have in my database are:

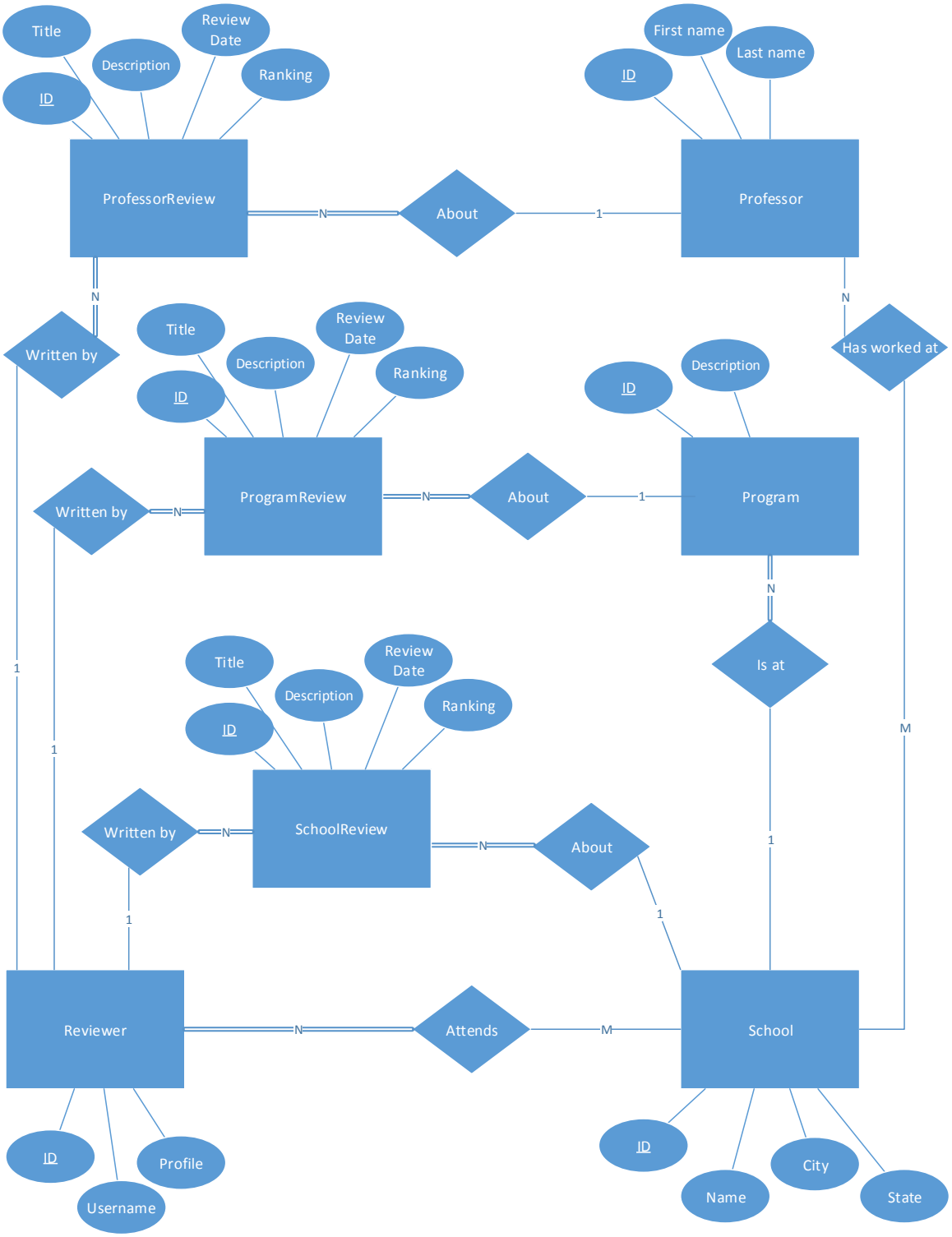
- Reviews- The focus of the database that the other entities will revolve around. They will have an author (reviewer), a title, a description, a date, a number ranking, and will apply to a school, program, or professor.
    - Note one difference from my initial project proposal: Reviews will actually be implemented as 3 distinct entity types, which are SchoolReview, ProgramReview, and ProfessorReview. All of these entities will have the original attributes of reviewer, title, description, date, and ranking. SchoolReviews will apply to a school, ProgramReviews will apply to a program, and ProfessorReviews will apply to a professor.
- Reviewers- The people who write a review. They will have a username, a brief profile, and the school(s) they attend(ed).
- Schools- These will have a name, city, state and an average ranking (calculated based on reviews, displayed on the website but not stored separately in the database).
- Programs- These will have a description and an average ranking (calculated based on reviews), and belong to a particular school.

- Professors- These will have a first name, last name, and an average ranking (calculated based on reviews), and may have worked at one or more schools.

The relationships I will have are:

- Reviews are written by Reviewers- A review must have exactly 1 author, but reviewers may write many reviews.
    - Note: Reviews refers to SchoolReviews, ProgramReviews, and ProfessorReviews.
- SchoolReviews apply to Schools- A SchoolReview must apply to exactly one school, but a school may have many SchoolReviews.
- ProgramReviews apply to Programs- A ProgramReview must apply to exactly one program, but a program may have many ProgramReviews.
- ProfessorReviews apply to Professors- A ProfessorReview must apply to exactly one professor, but a professor may have many ProfessorReviews.
- Reviewers attend Schools- Many-to-many, a reviewer must attend one or more schools and a school may have many reviewers that attend it.
- Programs are at a School- A program must be at exactly one school, but a School may have many programs.
- Professors work at Schools- Many-to-many, a professor may have worked at many schools, and each school may have many professors.

ER Diagram of Database



Database Schema

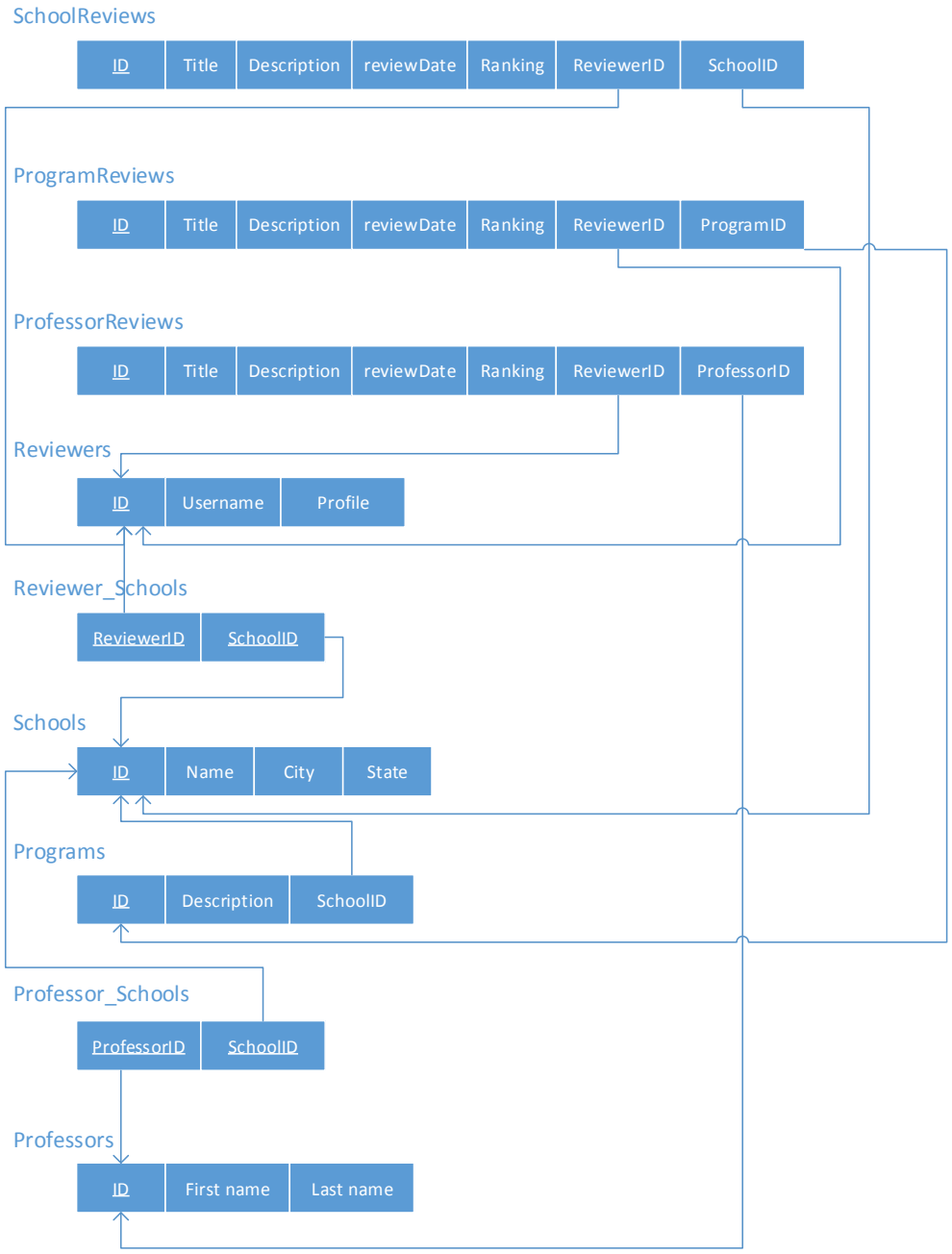


Table Creation Queries

SchoolReviews

```
CREATE TABLE schoolReviews (
id int NOT NULL AUTO_INCREMENT,
title varchar(255) NOT NULL,
description TEXT NOT NULL,
```

```
reviewDate DATE NOT NULL,
ranking int NOT NULL,
reviewerID int NOT NULL,
schoolID int NOT NULL,
PRIMARY KEY (id),
FOREIGN KEY (reviewerID) REFERENCES reviewers (id),
FOREIGN KEY (schoolID) REFERENCES schools (id)
) ENGINE = InnoDB;
```

ProgramReviews

```
CREATE TABLE programReviews (
id int NOT NULL AUTO_INCREMENT,
title varchar(255) NOT NULL,
description TEXT NOT NULL,
reviewDate DATE NOT NULL,
ranking int NOT NULL,
reviewerID int NOT NULL,
programID int NOT NULL,
PRIMARY KEY (id),
FOREIGN KEY (reviewerID) REFERENCES reviewers (id),
FOREIGN KEY (programID) REFERENCES programs (id) ON DELETE CASCADE
) ENGINE = InnoDB;
```

ProfessorReviews

```
CREATE TABLE professorReviews (
id int NOT NULL AUTO_INCREMENT,
title varchar(255) NOT NULL,
description TEXT NOT NULL,
reviewDate DATE NOT NULL,
ranking int NOT NULL,
reviewerID int NOT NULL,
professorID int NOT NULL,
PRIMARY KEY (id),
FOREIGN KEY (reviewerID) REFERENCES reviewers (id),
FOREIGN KEY (professorID) REFERENCES professors (id)
) ENGINE = InnoDB;
```

Reviewers

```
CREATE TABLE reviewers (
id int NOT NULL AUTO_INCREMENT,
username varchar(255) NOT NULL,
profile TEXT,
```

```
PRIMARY KEY (id),
UNIQUE KEY (username)
) ENGINE = InnoDB;
```

Reviewer_Schools: many-to-many relationship between reviewers & schools

```
CREATE TABLE reviewer_schools (
reviewerID int NOT NULL,
schoolID int NOT NULL,
PRIMARY KEY (reviewerID, schoolID),
FOREIGN KEY (reviewerID) REFERENCES reviewers (id),
FOREIGN KEY (schoolID) REFERENCES schools (id)
) ENGINE = InnoDB;
```

Schools

```
CREATE TABLE schools (
id int NOT NULL AUTO_INCREMENT,
name varchar(255) NOT NULL,
city varchar(255),
state varchar(255),
PRIMARY KEY (id),
UNIQUE KEY (name, city, state)
) ENGINE = InnoDB;
```

Programs

```
CREATE TABLE programs (
id int NOT NULL AUTO_INCREMENT,
description varchar(255) NOT NULL,
schoolID int NOT NULL,
PRIMARY KEY (id),
UNIQUE KEY (description, schoolID),
FOREIGN KEY (schoolID) REFERENCES schools (id)
) ENGINE = InnoDB;
```

Professor_Schools: many-to-many relationship between professors & schools

```
CREATE TABLE professor_schools (
professorID int NOT NULL,
schoolID int NOT NULL,
PRIMARY KEY (professorID, schoolID),
FOREIGN KEY (professorID) REFERENCES professors (id),
FOREIGN KEY (schoolID) REFERENCES schools (id)
) ENGINE = InnoDB;
```

Professors

```
CREATE TABLE professors (
id int NOT NULL AUTO_INCREMENT,
firstName varchar(255) NOT NULL,
lastName varchar(255) NOT NULL,
PRIMARY KEY (id)
) ENGINE = InnoDB;
```

General Use Queries

Display Reviews on Home Page

```
(SELECT sr.id, sr.title, sr.description, sr.reviewDate, sr.ranking, s.name , 'schoolReview' AS type FROM
schoolReviews sr
INNER JOIN schools s ON sr.schoolID = s.id)
UNION ALL
(SELECT pr.id, pr.title, pr.description, pr.reviewDate, pr.ranking, p.description, 'programReview' AS type
FROM programReviews pr
INNER JOIN programs p ON pr.programID = p.id)
UNION ALL
(SELECT por.id, por.title, por.description, por.reviewDate, por.ranking, CONCAT (po.firstName, " ",
po.lastName) AS name, 'professorReview' AS type FROM professorReviews por
INNER JOIN professors po ON por.professorID = po.id)
ORDER BY reviewDate DESC;
```

Add Reviewer (Make Account)

To populate dropdown:

```
SELECT id, name, city, state FROM schools ORDER BY name ASC;

INSERT INTO reviewers (username, profile) VALUES ([username], [profile]);
```

For every school selected in the multi-select dropdown:

```
INSERT INTO reviewer_schools (reviewerID, schoolID) VALUES (
[id of reviewer that was just created], [school id from dropdown]);
```

Edit Reviewer (profile and additional schools only)

To populate multi-select dropdown with only schools they're not already associated with:

```
SELECT s.id, s.name, s.city, s.state FROM schools s
WHERE s.id NOT IN
(SELECT s2.id FROM schools s2
INNER JOIN reviewer_schools rs ON s2.id = rs.schoolID
INNER JOIN reviewers r ON rs.reviewerID = r.id
WHERE r.id = [id of reviewer whose page were on])
```

```
ORDER BY s.name ASC;

UPDATE reviewers SET profile=[profile info] WHERE id=[id of reviewer whose page were on];

For every school selected in the multi-select dropdown:
INSERT INTO reviewer_schools (reviewerID, schoolID) VALUES ( [id of reviewer whose page were on],
[school id from dropdown]);
```

Display Reviewers

```
SELECT id, username FROM reviewers;

For each reviewer (loop in PHP):
SELECT s.name FROM schools s
INNER JOIN reviewer_schools rs ON s.id = rs.schoolID
WHERE rs.reviewerID = [their id];
```

Show Reviewer Profile

```
SELECT username, profile FROM reviewers
WHERE id =[id of  user viewing];

SELECT s.name FROM schools s
INNER JOIN reviewer_schools rs ON s.id = rs.schoolID
WHERE rs.reviewerID = [id of user whose profile viewing];

(SELECT sr.id, sr.title, sr.description, sr.reviewDate, sr.ranking, s.name, 'schoolReview' AS type FROM
schoolReviews sr
INNER JOIN schools s ON sr.schoolID = s.id
WHERE reviewerId = [id of user whose profile viewing])
UNION ALL
(SELECT pr.id, pr.title, pr.description, pr.reviewDate, pr.ranking, p.description, 'programReview' AS type
FROM programReviews pr
INNER JOIN programs p ON pr.programID = p.id
WHERE reviewerId = [id of user whose profile viewing])
UNION ALL
(SELECT por.id, por.title, por.description, por.reviewDate, por.ranking, CONCAT (po.firstName, " ",
po.lastName) AS name, 'professorReview' AS type FROM professorReviews por
INNER JOIN professors po ON por.professorID = po.id
WHERE reviewerId = [id of user whose profile viewing])
ORDER BY reviewDate DESC;
```

Add School Review

```
To populate dropdown:
SELECT id, name, city, state FROM schools ORDER BY name ASC;
```

INSERT INTO schoolReviews (title, description, reviewDate, ranking, reviewerID, schoolID)
VALUES ([title], [description], [date], [ranking], [id of user who came to page], [school]);

Add Program Review

To populate dropdown:
SELECT p.id, p.description, s.name FROM programs p
INNER JOIN schools s ON p.schoolID = s.id
ORDER BY description ASC;

INSERT INTO programReviews (title, description, reviewDate, ranking, reviewerID, programID)
VALUES ([title], [description], [date], [ranking], [id of user who came to page], [program]);

Add Professor Review

To populate dropdown:
SELECT id, firstName, lastName FROM professors ORDER BY lastName ASC;

INSERT INTO professorReviews (title, description, reviewDate, ranking, reviewerID, professorID)
VALUES ([title], [description], [date], [ranking], [id of user who came to page], [professor]);

Show Review Details

Based on the type (decision made in PHP) do 1 of the following 3:

SELECT sr.title, sr.description, sr.reviewDate, sr.ranking, r.username, s.name, s.city, s.state
FROM schoolReviews sr
INNER JOIN reviewers r ON sr.reviewerID = r.id
INNER JOIN schools s ON sr.schoolID = s.id
WHERE sr.id = [id of link selected];

SELECT pr.title, pr.description, pr.reviewDate, pr.ranking, r.username, p.description, s.name
FROM programReviews pr
INNER JOIN reviewers r ON pr.reviewerID = r.id
INNER JOIN programs p ON pr.programID = p.id
INNER JOIN schools s ON p.schoolID = s.id
WHERE pr.id = [id of link selected];

SELECT pr.title, pr.description, pr.reviewDate, pr.ranking, r.username, p.firstName, p.lastName
FROM professorReviews pr
INNER JOIN reviewers r ON pr.reviewerID = r.id
INNER JOIN professors p ON pr.professorID = p.id
WHERE pr.id = [id of link selected];

Delete Review

Do one of the following depending on type of review (decision made in PHP):

DELETE FROM schoolReviews WHERE id = [id of review viewing];

DELETE FROM programReviews WHERE id = [id of review viewing];

DELETE FROM professorReviews WHERE id = [id of review viewing];

Add School

INSERT INTO schools (name, city, state) VALUES ([name], [city], [state]);

Show all Schools

SELECT id, name, city, state FROM schools;

View School Profile

SELECT s.name, s.city, s.state, AVG(sr.ranking) FROM schools s
LEFT JOIN schoolReviews sr ON s.id = sr.schoolID
GROUP BY s.id
HAVING s.id = [id of school viewing];
*Note the LEFT JOIN is required because otherwise none of the data shows up if the school doesn't have any reviews.

Get list of reviews for school:
SELECT id, title, description, reviewDate, ranking, 'schoolReview' AS type FROM schoolReviews
WHERE schoolID = [id of school viewing];

Get list of programs at the school:
SELECT id, description FROM programs WHERE schoolID = [id of school viewing];

Get list of professors who have been at the school:
SELECT p.id, p.firstName, p.lastName FROM professors p
INNER JOIN professor_schools ps ON p.id = ps.professorID
WHERE ps.schoolID = [id of school viewing];

Add Program

To populate dropdown:
SELECT id, name, city, state FROM schools ORDER BY name ASC;

INSERT INTO programs (description, schoolID) VALUES ([description], [school]);

Show All Programs

SELECT p.id, p.description, s.name FROM programs p
INNER JOIN schools s ON p.schoolID = s.id;

View Program Profile

SELECT p.description, s.name, AVG(pr.ranking) FROM programs p
LEFT JOIN programReviews pr ON p.id = pr.programID
INNER JOIN schools s ON p.schoolID = s.id
GROUP BY p.id

HAVING p.id = [id of program viewing];
*Note the LEFT JOIN is required because otherwise none of the data shows up if the program doesn't have any reviews.

Get list of reviews for program:
SELECT id, title, description, reviewDate, ranking, 'programReview' AS type FROM programReviews
WHERE programID = [id of program viewing];

Delete Program

DELETE FROM programs WHERE id = [id of program viewing];

Add Professor

To populate dropdown:
SELECT id, name, city, state FROM schools ORDER BY name ASC;

INSERT INTO professors (firstName, lastName) VALUES ([first name], [last name]);

For every school selected in the dropdown:
INSERT INTO professor_schools (professorID, schoolID) VALUES
([id of professor just created], [school id from dropdown]);

Show All Professors

SELECT id, firstName, lastName FROM professors ORDER BY lastName ASC;

For every professor:
SELECT s.name FROM schools s
INNER JOIN professor_schools ps ON s.id = ps.schoolID
WHERE ps.professorID = [their id];

View Professor Profile

SELECT p.firstName, p.lastName, AVG(pr.ranking) FROM professors p
LEFT JOIN professorReviews pr ON p.id = pr.professorID
GROUP BY p.id
HAVING p.id = [id of professor viewing];
*Note the LEFT JOIN is required because otherwise none of the data shows up if the professor doesn't have any reviews.

Get list of reviews for professor:
SELECT id, title, description, reviewDate, ranking, 'professorReview' AS type FROM professorReviews
WHERE professorID = [id of professor viewing];

Additionally get list of schools they have worked at:
SELECT s.name FROM schools s
INNER JOIN professor_schools ps ON s.id = ps.schoolID
WHERE ps.professorID = [id of professor viewing];

Edit Professor (just add schools)

To populate multi-select dropdown with only schools they're not already associated with:

```
SELECT s.id, s.name, s.city, s.state FROM schools s
WHERE s.id NOT IN
(SELECT s2.id FROM schools s2
INNER JOIN professor_schools ps ON s2.id = ps.schoolID
INNER JOIN professors p ON ps.professorID = p.id
WHERE p.id = [id of professor whose page were on])
ORDER BY s.name ASC;
```

For every school selected in the multi-select dropdown:

```
INSERT INTO professor_schools (professorID, schoolID) VALUES ([id of professor whose page were on],
[school id from dropdown]);
```

HTML, PHP & Style

The source code is included in a zip file and the live site can be viewed starting at http://web.engr.oregonstate.edu/~percival/CS340/project/home.php.