# CMSC 23500 — Introduction to Database Systems
# Homeworks #7 and #8

May 15, 2008

In this homework, you will develop a web application to manage submissions of papers to conferences. This web application will be composed of multiple PHPs, all accessing the same database. Please note that these final two homeworks will have a slightly different handin schedule than previous homeworks:

- Homework #7 will be worth 100 points, and is composed of exercises 1 through 3 in this document, **due on May 21st**.
- Homework #8 will be worth 150 points and is divided into two parts:
  - Exercises 4 through 7 in this document (100 points), **due on May 30th at 7pm** (Friday, *not* the usual Wednesday). These exercises require writing web forms that have not been covered in the lectures. You will be provided with hints on how to do this on the day homework #8 would ordinarily be assigned (May 22nd). Nonetheless, you are encouraged to figure out how to write those forms before getting the hints, and are allowed to share your findings with your classmates on the mailing list.
  - Two additional exercises (worth 50 points), on material to be covered in the lectures, will be assigned on May 22nd, and **due on May 28th at 7pm**.

When submitting your solutions, make sure you do the following:

- Follow *all* naming conventions described in this handout.
- Hand in all your PHP files in a single directory. Your PHPs should use a database called `conference.db` in that directory.

## 1 The conference database

The database that the web application will use has been designed for you (see figure 1). The specifics of the schema are the following:

- The database keeps track of paper submissions for several conferences.
- Each conference has a code, a name, and a location.
- The system has users. Users can be authors, conference administrators, paper reviewers, or part of a program committee of a conference. There is a single User table, and the type of user is determined by its relations to other tables (a user can have more than one role, like being and author and also being on the program committee for several conferences).
- Users have a unique username, a name, and a contact e-mail address.
- A conference can have multiple submissions. For example, a conference may have one first submission soliciting paper drafts and due on 05/01/08. Once the accepted papers have been announced, the conference may have a second submission for final manuscripts due on 07/01/08. It could also have a third submission for slides that will be presented at the conference.

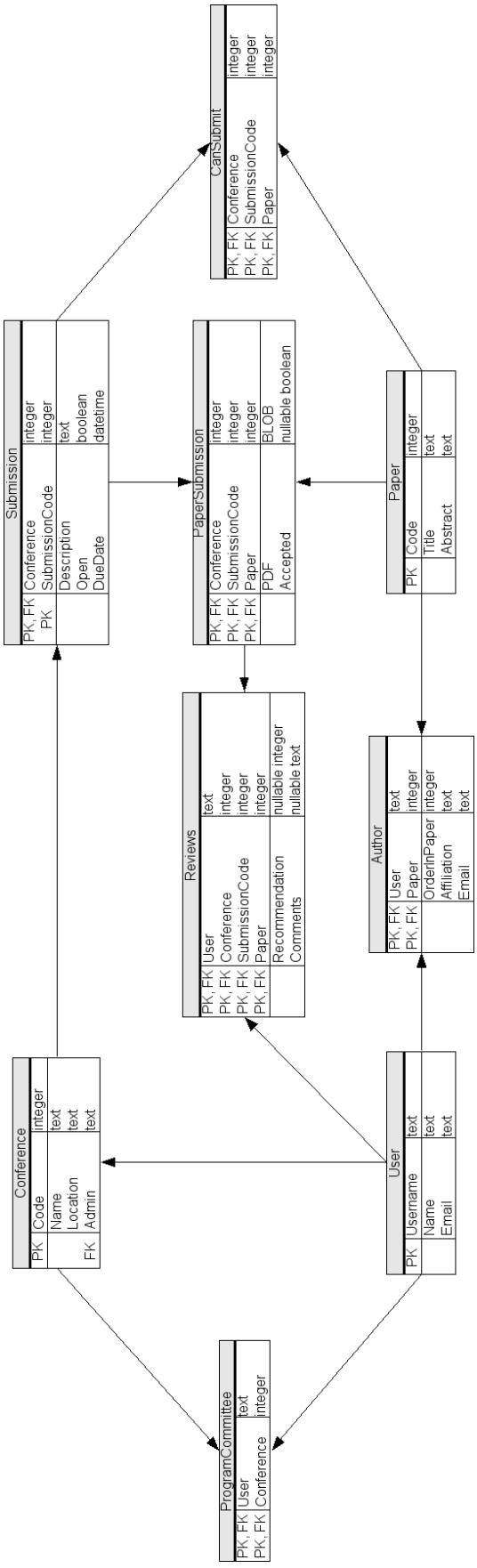


Figure 1: Relational schema for the Conference database

- A submission can be *open* or *restricted*. An open submission is listed publicly on the website, and anyone can submit to it. A restricted submission is only available to those papers flagged by the conference administrator as eligible for that submission (in the CanSubmit table).
- A paper is authored by one or more users. An author has a name (obtained from the User table), an affiliation, and an e-mail (note that this e-mail may be different from the one in the User table; the latter is a contact e-mail, and the former is the one the author wants listed on the paper and the proceedings).
- A paper has a title and an abstract.
- A paper can be submitted to one or more submissions. The "lifecycle" of a paper would be:
  - A paper is first submitted to an open submission.
  - If there are more submissions (e.g., "final manuscript"), the paper can only be submitted to them if the conference administrator adds the paper submission to the CanSubmit table. For example, the conference administrator may only allow papers that were favorably reviewed to be submitted as a final manuscript.
  - In each paper submission, a PDF is sent. This PDF can be different in each submission (we will not actually be storing anything in PaperSubmission.PDF). For simplicity, we assume that nothing else in the paper would change from one submission to another (the title, the abstract, or the author list).
  - In each submission, the paper can be accepted or rejected. A NULL value in PaperSubmission.Accepted means there has been no decision yet.
- A paper submission may have reviewers assigned to it. A reviewer must provide comments on the submission (Reviews.Comments), and make a recommendation (Reviews.Recommendation): -2 (Strong reject), -1 (Weak reject), 1 (Weak accept), 2 (Strong accept). A NULL value in these fields means that a reviewer has been assigned to the paper submission, but that said reviewer has not provided feedback yet.
- For avoidance of doubt:
  - A *submission* refers to a submission tied to a specific conference. For example, conference FOOBAR could have two submissions: "Initial submission" (open) and "Final manuscript" (restricted).
  - A *paper* refers to a scholarly article authored by one or more users.
  - A *paper submission* refers to a specific version of a paper that is presented to a conference submission. For example, paper "FOOBAR considered harmful" can have two paper submissions: first to the "Initial submission" and then to the "Final manuscript". Each paper submission would have different versions of the actual paper (i.e., the PDF that is sent to the conference organizers), but everything else about the paper (authors, title, abstract) is immutable across submissions.

You are not provided with a sample dataset. However, you are allowed (and, in fact, encouraged) to create sample datasets and share them with your classmates.

## Simplifying assumptions

As you code up the exercises, you can make the following assumptions:

- Your users are unconcerned with the appearance of your web pages. Make your web pages pretty only if you have free time after you've solved all the exercises.
- The due date of a submission is just for show. You do not need to do any validation based on it, nor do you have to filter out submissions that are past due.

- Papers will have, at most, two authors. You do not have to come up with general-purpose web forms that can accommodate any number $N$ of authors.
- Similarly, paper submissions will have exactly two reviewers.
- Security is not a concern. All your users are going to use the system exactly the way you tell them to use it. No one is trying to game the system. No one will try to modify HTTP headers in–flight. No one is trying to do SQL injection on your forms. And no, there are no ninjas in the backyard.

Finally, all exercises are based on a specific user being logged into the web application. We will not be using login forms or any form of authentication, so all your PHPs must include the following statement at the top:

```
include("user.php");
```

The `user.php` file must contain the following:

```
<?php
$user = 'foobar';
?>
```

You can replace `foobar` in that file with whatever user you want to test the application with (take into account that the database may have a variety of users: authors, reviewers, users who are simultaneously authors and reviewers, etc.). This way, you can use the variable `$user` to obtain the username of the user that is currently "logged into the application".

## Exercise 1: The user dashboard (40 points)

When users log into the system, the first page they will see is the *dashboard* page. This page contains useful information for the user, like paper submissions that are due (if the user is an author), reviews that they must write (if the user is a reviewer), etc. More specifically, you must write a PHP called `index.php` that shows the following:

- A link to `submit.php`, the paper submission PHP (this will be implemented in exercise 2).
- For each paper that the user has submitted, show the following in a table:
  - Conference name
  - Description of the submission (i.e., the Description field of the Submission table)
  - Title of the paper
  - If the paper has been accepted or rejected. If this is not yet known, then the status of the paper will either be "Submitted" (if no reviewers have been assigned yet) or "Under review" (if reviewers have been assigned).
- If the user has authored a paper, and has been flagged as being able to submit to a restricted submission (and has not already done so), then for each pending submission show the following in a table:
  - Conference name
  - Description of the submission
  - Title of the paper
  - Due date
  - A link to `submit.php?paper=`*`paper_code`*`&conference=`*`conference_code`*`&submission=`*`submission_code`* (these are GET parameters; how to process them is explained in the next exercise)

- If the user is a reviewer for a paper submission, show a table with the following for each review:
  - Conference name
  - Description of the submission
  - Title of the paper
  - If the reviewer has not completed the review, show a link to `review.php?paper=`*`paper_code`*`&conference=`*`conference_code`*`&submission=`*`submission_code`* (this will be exercise 3). Otherwise, just show the text "Review done".
- If the user is a conference administrator, show a table with the following for each conference the user administrates:
  - Name of the conference
  - Link to `create_submission.php?conference=`*`conference_code`* (Exercise 6)
  - For each submission in the conference, links to:
    * `assign_reviewers.php?conference=`*`conference_code`*`&submission=`*`submission_code`* (Exercise 4)
    * `accept.php?conference=`*`conference_code`*`&submission=`*`submission_code`* (Exercise 5)

# Exercise 2: Paper submission (40 points)

You will write two PHPs, `submit.php` and `submit-process.php`. The former is the web form used to submit a new paper, whereas the former processes the submission and updates the database accordingly.

## GET parameters

For convenience, we will be using GET parameters in some of the exercises. These are very similar to POST parameters, and the main difference is that, instead of being sent in the headers on the HTTP request, they are specified in the URL of the request. For example, if we wanted to send two parameters (`foobar` with value `42`, and `foobaz` with value `37`) to `submit.php`, the URL would look like this:

```
submit.php?foobar=42&foobaz=37
```

The list of parameters appended to the URL is typically called the *querystring*. To retrieve the value of these parameters from a PHP, we simply need to use the `$_GET` object instead of the `$_POST` object. GET parameters are usually unadvisable in practice, since it is very easy for a user to modify the value of a parameter (POST parameters are not any safer, but modifying HTTP headers is not as easy as just typing something into the URL field of your web browser). In these exercises we are using GET parameters just for the convenience of creating querystring-based URLs in the dashboard page, instead of having to create a form with POST parameters for every link in the dashboard.

### `submit.php`

If this PHP is requested without any parameters in the querystring, then the user will be able to submit to any open submission. The PHP can also be requested with the following three parameters in the querystring:

- `conference` and `submission`: The primary key (Submission.Conference and Submission.SubmissionCode) of a conference submission.
- `paper`: The primary key of a paper (Paper.Code).

If these parameters are specified, then the PHP will be used to submit a paper (which must already exist in the database) to that specific (presumably restricted) submission.

When submitting to an open submission, the PHP will be a form with the following fields:

1. **Conference submission**: Drop-down list with all the open conference submissions.
2. **Paper title**: Text field.
3. **Paper abstract**: Text field.
4. **Author #1 username**: Text field.
5. **Author #1 affiliation**: Text field.
6. **Author #1 e-mail**: Text field.
7. **Author #2 (if any) username**: Text field.
8. **Author #2 (if any) affiliation**: Text field.
9. **Author #2 (if any) e-mail**: Text field.
10. **PDF file**: Text field.

When submitting to a specified submission (via parameters), the PHP must print out the conference submission name, paper title, paper abstract, and author information (name, affiliation, e-mail). The only editable field will be the PDF field.

The form must submit the data to `submit-process.php`. The names of the form fields (i.e., the parameters that are passed to `submit-process.php`) are entirely up to you.

### `submit-process.php`

This PHP will process the paper submission, and will update the database accordingly. If the paper is being submitted to an open submission (i.e., there is no prior record of the paper in the database), this PHP must do the following:

1. Add a row in Paper.
2. For each author, add a row in Author.
3. Add a row in PaperSubmission.

If this is a submission to a restricted submission (i.e., the paper is already in the database), then the PHP just needs to create a new row in PaperSubmission.

## Exercise 3: Leaving a review (20 points)

You will write two PHPs, `review.php` and `review-process.php`. The former is the web form used to create a new review, whereas the latter processes the review and updates the database accordingly. Note that, to test this exercise before doing exercise 4, you will need to manually add rows to Reviews (with Recommendation and Comments set to NULL) so you will have users marked as reviewers for paper submissions.

`review.php` will receive the following three parameters in the querystring:

- `conference` and `submission`: The primary key (Submission.Conference and Submission.SubmissionCode) of a conference submission.
- `paper`: The primary key of a paper (Paper.Code).

The PHP will be a form with the following fields:

1. **Recommendation**: Drop-down list with values -2 (Strong reject), -1 (Weak reject), 1 (Weak accept), and 2 (Strong accept).

2. **Comments**: Text field.

The form must submit the data to `review-process.php`. The names of the form fields (i.e., the parameters that are passed to `review-process.php`) are entirely up to you. `review-process.php` must update the row in Reviews corresponding to the paper submission that is being reviewed (by the user who is logged into the web application).

## Exercise 4: Assigning reviewers to papers (25 points)

Once papers have been submitted, the conference administrator can assign reviewers to them. To do this, you will write two PHPs, `assign_reviewers.php` and `assign_reviewers-process.php`.

`assign_reviewers.php` will receive two parameters through the querystring: `conference` and `submission`: The primary key (Submission.Conference and Submission.SubmissionCode) of a conference submission. Based on those parameters, the PHP will generate a table with one row for each paper submitted to that submission. The columns of the table will be:

- **Title of the paper and authors**
- **Abstract**
- **First reviewer**: A drop down list with the names of all the users that are in the program committe of the conference.
- **Second reviewer**: idem.

All the fields in the table will be part of a *single* form which must be submitted to `assign_reviewers-process.php`. The names of the form fields (i.e., the parameters that are passed to `assign_reviewers-process.php`) are entirely up to you. `assign_reviewers-process.php` must update the database by adding a row to Reviews for each reviewer specified in the form.

## Exercise 5: Accepting/rejecting papers (25 points)

Once papers have been reviewed by the reviewers, the conference administrator must decide what papers will be accepted in a submission. To do this, you will write two PHPs, `accept.php` and `accept-process.php`.

`accept.php` will receive two parameters through the querystring: `conference` and `submission`: The primary key (Submission.Conference and Submission.SubmissionCode) of a conference submission. Based on those parameters, the PHP will first check if all the papers in the submission have been reviewed by all the reviewers assigned to each paper (remember that entries in Reviews have Recommendation and Comments initially set to NULL until the reviewer provides that information). If there are papers with pending reviews, the PHP must show a message indicating this. Otherwise, it will generate a table for each paper submission with the following columns:

- **Title of the paper and authors**
- **Abstract**
- **All the comments left by the reviewers**
- **Score**: The sum of the reviewers' recommendations (remember that the recommendation is a number ranging from -2 to 2).
- **Accepted?**: Checkbox. If checked, that means the paper will be accepted.

The table must be sorted in decreasing order of score. All the fields in the table will be part of a *single* form which must be submitted to `accept-process.php`. The names of the form fields (i.e., the parameters that are passed to `accept-process.php`) are entirely up to you. `accept-process.php` must update the PaperSubmission table to reflect what papers have been accepted and which ones have been rejected.

## Exercise 6: Creating a new submission (25 points)

A conference administrator will have to create, at the very least, an initial open submission for the conference. Once papers have been submitted to that submission, and some of the papers have been accepted, the administrator may want to create a new restricted submission seeded with the accepted papers from the previous submission. For example, a restricted "Final manuscript" submission should only allow papers accepted to a previous open "Initial submission". To do this, you will write two PHPs, `create_submission.php` and `create_submission-process.php`. `create_submission.php` will receive one parameters through the querystring: `conference`, the primary key (Conference.Code) of a conference. The PHP will show a simple form with the following fields:

- **Description of the submission**: Text field
- **Due date of the submission**: Text field
- **Open or restricted?**: Two radio buttons.
- **Restrict submission to accepted papers of**: A drop down list with the submissions for that conference (note that this field is not relevant if we are creating an open submission).

The form must be submitted to `create_submission-process.php`. The names of the form fields (i.e., the parameters that are passed to `create_submission-process.php`) are entirely up to you. `accept-process.php` must do the following:

- Create a new row in Submission.
- If this is a restricted submission, obtain the list of accepted papers from the specified previous submission, and add the pertinent rows to CanSubmit. If the previous submission has any papers where acceptance/rejection is still pending, you should abort the entire operation.

## Exercise 7: Extending the database design (25 points)

The presented schema makes some simplifying assumptions. In this exercise, you must *extend the design* of the schema. **This is not an implementation exercise**. Given an additional requirement for the database, you must discuss (in no more than 2-3 paragraphs) why the current schema does not support that requirement, and how you would alter the schema to support it.

Choose *one* of the following:

1. The title, abstract, and list of authors of a paper can sometimes vary between an initial submission and a final manuscript (or, in general, between submissions). For example, an extra author may be added to write a new section that addresses the reviewer's comments. The database must support these changes *and* keep a record of what the title, abstract, and list of authors was for a specific submission.

2. Papers are usually matched to reviewers that do research in the same field. This requires asking the authors to specify a set of categories for the paper (e.g., "Operating systems", "Scheduling algorithms", etc.), and each user (who can potentially be a reviewer) to indicate their fields of interest. The database must support this, so that a tentative list of reviewers for each paper can be generated, instead of being manually selected from scratch by the conference administrator.