Homework 5: Application Development

CS186: Introduction to Database Systems
UC Berkeley

April 4, 2003
Due: April 18, 2003 (midnight)

1 Introduction

By now you have worked extensively with Postgres and have become experts on its internals. This homework will give you a taste of using an SQL database like Postgres for application development. A common way to build interfaces to a database application is though the web. In this assignment, you will be using PHP, a popular web scripting language, to build an application using a Postgres database as a backend.

The application you will be developing for this assignment is a simplified interface to a Package Tracking System used by a fictitious delivery firm based at Berkeley called **Golden Bears Package Delivery**. This application allows the firm to manage all the packages being delivered, and allows their customers to create accounts to keep track of the status of their packages. More complete examples of web-based tracking systems include those at FedEx (`http://www.fedex.com`) and UPS (`http://www.ups.com`). In this assignment, we have greatly simplified the tracking system and you will only be required to implement a limited subset of features you see on the commercial sites.

1.1 Learning PHP

You should be able to pick up any PHP skills you need on the fly – it's really quite easy to learn. You can find an extensive PHP manual at `http://www.php.net/manual/en`. To help you get started on learning PHP, we have provided some sample source code for the group registration application which is written in PHP. The source code is available `/home/ff/cs186/sp03/Hw5/groupreg.tar.gz`. You should also step through the simple tutorial `http://www.php.net/manual/en/tutorial.php` while reading the sample source code that we provide.

We are giving you much of the PHP code and all of the HTML templates for the webpages; you only need to add the PHP code that interfaces to the database and gets the results back. We have provided some screenshots at `http://inst.eecs.berkeley.edu/~cs186/hwk5/screenshots.html` and you can use these as a guide when building your web application.

1.2 Getting Started

The distribution for this assignment can be found in `/home/ff/cs186/sp03/Hw5/hw5.tar.gz`. We have created a `~/public_html` directory for you. cd to the `~/public_html` directory and copy over the initial files using ``gtar zxvf /home/ff/cs186/sp03/Hw5/hw5.tar.gz''. This creates a subdirectory `hw5` which contains the initial scripts. To ensure that your scripts can be read and executed by the web server, cd to `~/public_html` and type ``chmod -R 755 hw5''. We have also ensured that your work cannot be seen by other students. **Make sure that you place the code in only the `~/public_html/hw5` directory or it risks being seen by other groups. Also, do not delete the `~/public_ html` directory.** View your website at `http://inst.eecs.berkeley.edu/~cs186-xx/hw5/main.php` or `http:`

//inst.eecs.berkeley.edu/~cs186-xx/hw5/admin.php, where cs186-xx is your login. We will describe individual scripts in more detail in later sections.

We have ensured that the scripts are working correctly on the instructional machines Pentagon and Rhombus, and we will continue to provide support. We will not provide support on any other platforms/machines. Before submitting, you MUST make sure your code runs on the instructional machines.

2 Database

By now, you should be familiar with how to create and start the Postgres DBMS. We will refresh your memory with some basic instructions. When you first get started, run `initdb` at the shell to initialize the database directory at $PGDATA. To reinitialize, you have to delete $PGDATA first. `pg_ctl start` and `pg_ctl stop` will start and stop Postgres respectively. After starting Postgres using `pg_ctl start`, use our provided script `loadDB.sh` to create the database 'hw5' and load in some test data. You only have to run `loadDB.sh` once unless you want to reinitialize your database. Subsequently, when you are testing your scripts, make sure that Postgres is started or the PHP scripts will not be able to make database connections.

2.1 Database Schema

In `schema.sql`, we have provided the DDL statements to create all database tables and load in the initial test data. This will be executed when you first run `loadDB.sh`. Do study the DDL carefully, especially the primary, foreign keys and value constraints. Some of the constraints are not implemented in the schema, but must be imposed by the application logic in your scripts. We call these *application-level constraints*. You will have to implement these constraints which we highlight in **bold** below. There are four relations in the schema. Columns in the primary key are underlined.

- **Customer** (login: varchar(10), name: varchar(50), password: varchar(10), street: varchar(50), city: varchar(30));
- **Package** (trackingNumber: integer, weight: integer, sourceStreet: varchar(50), sourceCity: varchar(30), destinationStreet: varchar(50), destinationCity: varchar(30), login: varchar(10));

  Each `Package` has a unique trackingNumber. This tracking number is auto-incremented using a special PostgreSQL feature called a SEQUENCE. SEQUENCEs are essentially database-persistent counters that can be atomically incremented, and are used to guarantee you get a unique numerical identifier. See the PostgreSQL command reference for `CREATE SEQUENCE` if you want all the details, e.g. at `http://www14.us.postgresql.org/users-lounge/docs/7.2/postgres/sql-createsequence.html`. See `schema.sql` for the declaration of the SEQUENCE named `trackid`, and its use in the `Package` table. Note that `trackid` starts at 10 to account for the packages we have preloaded in your initial testdata. The next package that you add to the system will start from a tracking number of 10.

  Each `Package` may have a registered customer associated with it. This is reflected by the login field which is a foreign key to `Customer.login`. If this field is null, it means that the package is not associated with any registered customer. **There is an *application-level constraint* that each package has to commence from a `sourceCity` and ends at a `destinationCity` that is listed in the `Rate` table.** All other routes not reflected in the `Rate` table are not supported by Golden Bears. Package weight is in pounds (lbs).

- **PackageStatus** (trackingNumber: integer, trackingDate: date, trackingTime: time, currentCity: varchar(30), comments: varchar(50))

  Each `Package` can be associated with multiple `PackageStatus` records, as the package is being delivered from city to city. At each city, the tracking system allows the delivery person to update the status of the `PackageStatus` based on the current time and date, and also enter comments on

the state of the package (e.g. in transit, in warehouse, etc). `trackingNumber` is a foreign key to `Package.trackingNumber`. **There is an *application-level constraint* that the first package status information for each package has to commence from its source city.** For simplicity, we will allow intermediate cities to be anything, including cities not listed on the `Rate` table below.

- **Rate** (sourceCity: varchar(30), destinationCity: varchar(30), rate: real)

  The `Rate` records the amount each package delivery cost per pound to deliver from source city to destination city. We have added a constraint that source and destination cities must be different, as Golden Bears do not deal with local deliveries. Routes whose rates are not listed are not supported by Golden Bears.

  Note: City names are not case sensitive. I.e., "SAN FRANCISCO" and "San Francisco" refers to the same city. This matters when you are comparing city names while implementing the application-level constraints. For simplicity, you should store all city names and do the comparisons in UPPER CASE.

2.2 PHP Postgres Interface

The PHP language provides a rich set of functions for several tasks; among those are functions to communicate with Postgres. For details, see `http://www.php.net/manual/en/ref.pgsql.php`. We have placed some initial database interface code inside `database.php`. Currently, this includes the code for connecting to the Postgres database and all the Postgres interface code for `show_rate.php`. To connect to the Postgres database properly, set the `USER` variable in `database.php` to your login, and the PORT variable to your assigned port (do a `echo $PGPORT` to figure out your port number). The `HOST` variable is set to the machine you are using to run your database. If you start your Postgres on Pentagon, define that variable as `pentagon.cs.berkeley.edu`. If you use Rhombus, set that to `rhombus.cs.berkeley.edu` instead.

You are free to place the database interface code in `database.php` or elsewhere, whichever suits you most. You will at least need to use the functions `pg_query`, `pg_connect`, `pg_num_rows`, `pg_affected_rows` and `pg_fetch_array`. Whenever you insert or update a row, use the `pg_affected_rows` function to make sure that the operation is successfully carried out, or else output the appropriate error message.

As we will explain in the next section, some of our scripts require you to formulate your queries with the correct ORDER BY clause as you have done in homework 3. `Please remember to put these ORDER BY clauses in!!`

3 Web Frontend

In this project, you only need to worry about the *application logic* and not the *presentation*. We have provided you with nearly all of the necessary code to generate HTML output. You only need to "fill in the blanks" and make the scripts actually communicate with the database to produce the output. To help you get started, we have provided you our solution to `show_rate.php`.

There are "begin" and "end" delimiters in the code that should guide you in modifying the appropriate places in the code you are given. Apart from these delimiters, we have also placed dummy `for` loops, which you will alter to iterate over result rows, at the locations marked in the code. You are free to place your code elsewhere or in other scripts apart from those that we recommend.

The subsequent sections give short description on each PHP script. Refer to our screenshots for some examples.

3.1 Main Customer Pages

These pages are only viewable by customers. These pages are as follows:

- **main.php**

This site has a form that allows customers to login, and it also provides links to `show_rate.php` (icon), `track.php` (icon) and `register.php`.

- **customer.php**

  After a customer logs in, this page will display all the status of all the customer's packages, including how much they paid for each package (computed from rate $\times$ package weight). The information should be displayed ordered by package `trackingNumber`, and for each package, you should output the status information ordered by `trackingDate` and `trackingTime` of `PackageStatus`.

- **register.php**

  This site allows a new user to register as a customer.

- **show_reg.php**

  Confirmation page after a user successfully registers as a customer.

- **show_rate.php**

  Display rate information for all routes provided by Golden Bears Delivery ordered by `sourceCity`, `destinationCity`.

- **track.php**

  A general page for users to enter a tracking number and retrieve the current status of the package corresponding to that tracking number. Package Status should be ordered by `trackingDate` and `trackingTime`.

3.2 Administrative Pages

These pages are viewable only by the staff at Golden Bears, and are used by them to add new packages and update their current status.

- **admin.php**

  The site that the staff first visits. This site has links to the following PHP scripts below.

- **viewAllCustomers.php**

  Shows all the customers registered with Golden Bears ordered by login.

- **newPackage.php**

  Provides a form to enter information on new packages. In our system, the assignment of packages to customers' login is done by the staff at Golden Bears through this form. The login field can be left null if it is not associated with a registered customer. The source and destination cities have to be based on routes Golden Bears provide (retrievable from `Rate` table).

- **show_Package.php**

  Confirmation page after a new package is successfully added. The tracking number of the new package is also displayed.

- **updatePackageStatus.php**

  Provides a form to enter a new status for a Package. The date and time for the PackageStatus are set based on the current system time. Updates to packages that do not exist are not allowed, and for each package, the first package status information entered has to commence from the source city.

- **show_packageStatus.php**

  Confirmation page after a new Package Status is entered successful in the system.

- **updateRateInformation.php**

  Provides a form to enter new rate information or update existing rate information. The source and destination cities entered must be different.

- **viewAllPackages.php**

  Unlike the `track.php`, this allows the display of ALL the packages ORDER BY `trackingNumber`. For each package, display the package status ORDER BY `trackingDate` and `trackingTime`.

3.3 Error Handling

In the process of generating the web page based on the PHP scripts, errors may be encountered. Some of them will be due to internal errors in the database, while the others will mostly be due to erroneous data entered by users. It is important in writing a web application to handle these errors gracefully.

To standardize error output, in each PHP script, we provide HTML code that will print out `$MSG` in red whenever there is an error. We have also provided standardize error messages that we expect you to output in each PHP script. Whenever you encounter an error, simply set `$MSG` to the appropriate error message. For example, in PHP script `register.php`, we have provided error messages if the registration form is incomplete, the two passwords entered do not match, or if the login chosen has been used by others. For general database errors (such as cannot connect to database for whatever reasons, unable to insert or delete a tuple, unable to issue a query, etc), output the generic error message `$DB_ERR_MSG` that we have predefined in `database.php`. **Do not use error messages other than those that we provide, and you must handle all the error messages that we provide at the top of each php script.**

4 What to turn in

You will work in your respective groups in this project as you have done in homework 1 and homework 2. When you are ready to submit the project, create a directory called `Project5`. Put all your PHP files into that directory. Include a README that contains you and your group members' logins and SID, and also indicate what works and what does not. `cd` into that directory and type `submit Project5` to submit the files.

At the same time, remember to submit the review for your group. Write your reviews in a file called `prj5_review` and place that in a directory called `prj5_review`. Use `submit prj5_review` to submit the review. Use the same format as you have done in homework 1 and homework 2 in the reviews.