Project B: Self-designed Project

You will propose, design, and implement a self-designed project with an intensively used database back-end. The project should:

- have a non-trivial data model with strong integrity constraints.
- involve concurrency with updates and queries happening at any time.
- use transactions to deal with concurrency.
- include non-trivial queries.
- avoid having too much application logic (this is a *database* course, after all)
- present a web front-end, although it can be very simple.

The project may be done in groups of up to three people, with expectations being commensurate with the number of people in the group. In some cases, the above requirements may be relaxed. The final implementation must run in our environment (Using Oracle, Apache, Perl, and the other tools available on the undergraduate lab server) and we must be able to talk to it via your web front-end.

We expect that your project will be placed in ~you/public_html/selfdef. You should provide a introductory web page which points to all the components of the project (see below) and to the current working project.

The following schedule and proposed projects are based on the level of work for a student for whom web application programming is new. If you're an old hand at web applications or think that these projects might not challenge you, please contact us. We have other projects that you can work on, some of which could well lead to independent study projects or research experience for undergraduates projects.

Schedule

The project will have the following stages. You may work faster than this.

- **Proposal** (Week 1): You will write a one page project proposal and hand it in. The proposal should briefly describe the project and explain why it meets the requirements described above.
- **Specification**, **Entity-relationship Diagram, and Relational Schema** (Week 2): You will meet with your customers and develop a detailed specification of their problem, including what their data looks like, its constraints, what updates will look like, what queries look like, etc. Your specification will include a formal data model in the form of an entity-relationship diagram and a relational schema based on your ER diagram. Your schema must be 3NF or better. You will hand in the specification, ER diagram, and relational schema.
- **DDL, DML, Queries, and Application Logic** (Week 3): You will realize your relational schema using Oracle SQL as your DDL. You will also write insert, update, and delete statements or blocks for your data using Oracle SQL as your

DML. You will then populate your schema with representative data using your DML code. Next, you will write the necessary queries against your schema, and develop the application logic needed in your application. Your application logic will be written in a high-level language (Perl, for example) and will talk to the database using a standard interface (DBI, for example). You will hand in your code.

- **Web front-end** (Week 4): You will now clean up your application logic and write your web interface. Remember that it is not important that this be very complex or beautiful.

Example Projects

You will have the best experience if you build something that you already want to build! I am open to any project as long as it meets the criteria outlined above and looks like it is feasible in the timeframe given your skill level. Even if it doesn't quite meet the criteria, ask!

The second best kind of project is one where there is an interested "customer" other than you. I will ask faculty and graduate students to suggest some projects that they are interested in. You should ask your friends, coop employers, etc.

The following are some examples of projects I have thought up.

Shared knowledge bases

Tools in which people can add facts to shared knowledge base. People can then pose detailed questions about these facts.

- **DealBase**: This would be a shared social bookmark database (like http://del.icio.us) but for putting information about good deals (like slickdeals.net). A user would be able to bookmark particularly good online purchases (or bad experiences with a vendor, brand, or product) and other users would be able to find them by popularity or search terms.
- **CorruptionBase**: A database that contains information about political donations to candidates (required to be publicly available), voting records in Congress (or state legislatures) (available via Thomas and otherwise), lobbyist registrations, etc. Qualified individuals would contribute such facts and anyone could use them to pose questions like "how much money does it take to elect a congressman from Texas?" or "which contributors supported sponsors of all three of these contradictory bills?"
- **Recommender**: People would anonymously rate things they have or have experienced, etc. So, associated with each person would be a list of things and their ratings. The recommender would find similar
- **Spam filter**: People would forward spam email they receive to the service. In return, they could ask the service "have you seen this before". (For this class, this tool need not scale).

- **Web filter**: Similar to the community spam filter, but here we would use reports of web ads to filter them out, like the tool AdSubtract. (For this class, this tool need not scale).
- **Textbook Exchange**: A web application to enable students to exchange and/or buy/sell textbooks.
- **Customer Care Auctions**: A web application that would match people with customer care complaints that need resolving with people who are good at getting companies to do the right thing. Each complaint would be auctioned, with each bid being either a dollar value or a percentage of the funds recovered.

Personal databases

- **Music lover's database** (detailed info about recordings – an online version of iTunes)

Communication

- **MicroFriendster** (http://www.friendster.com) or **MicroOrkut** (http://orkut.com) - a social network system.
- **Forum 4000** (see http://aardvarko.snappyanswers.com/mirrors/xalton.forum2000.org/index.html and http://conversatron.com/)
- **Chat archiver and searcher**. This tool would subscribe to a chat session on all instances and archive all messages to the database. The searcher would then allow a web user to search for messages of interest.
- **Database-based chat system**. This would be a full-fledged interactive chat system (using server-side push, ideally) in which messages would be archived to the database. On joining an instance, it would be possible to scroll back to messages sent even before joining.

Web Games

- **Chess Community**: A web site that lets individuals play chess with each other. Many games can be played simultaneously and the site keeps track of who has won against whom, suggesting competitive matches. The game states and records are kept in the database.

Exploratory Data Analysis

- Build a system to study multi-dimensional data that includes a data cube operator (http://citeseer.nj.nec.com/gray97data.html) and graphical output.
- **Climate Watch**: A web application that lets ordinary users make queries about the climate data that is used to make global warming, pollution, and other ecological claims. Graphical output (graphs of temperature versus time, for example) would be featured.

Projects Proposed By Various Folks in the CS Department

- **CIM in SQL**: Create a SQL implementation of the CIM model for computer systems management (http://www.dmtf.org/standards/cim). No web interface needed for this. Contact pdinda@cs.northwestern.edu if you are interested.
- **Critiquing Tool**: a database backend for Chris Riesbeck's critiquing tool. If you are interested, contact riesbeck@cs.northwestern.edu.
- **Advisor Helper**: a database application to present useful information to faculty advisors and student advisees based on internal record keeping.
- **GA-IDS** online. (http://ga-ids.cs.northwestern.edu) (pdinda@cs.northwestern.edu)