CPS 216 Fall 2001
Course Project

Important Dates

Milestone 1: Thursday, October 18
Milestone 2: Thursday, November 15
Project Demo and Presentation Period: December 3-7, 10-12
Final Project Report Due: Thursday, December 13

You can choose either one of the following two tracks for your course project: DBMS applications, or DBMS research. The first track focuses on using a DBMS to build applications. The second track focuses on the internal workings of a DBMS, and provides an opportunity for you to conduct research in databases.

Unless you have never worked with a DBMS before and wish to gain more experience developing database applications, I would encourage you to try the research track. In general, this track allows you to exercise more of what you learned in CPS 216. It is potentially more rewarding too—if the results look promising, you can turn your project into conference or workshop publications with some additional effort.

On the other hand, your score will not be affected by your choice of one particular track over the other. We will study your proposal and ensure that your project meets the minimum requirements of depth and scope. Once your proposal is accepted, your score will depend on whether you meet the deadlines and how much you finally deliver with respect to your proposal. You are encouraged to tackle problems of significant depth and scope, and you will be rewarded with extra credits to offset your risk. Because of limited time, it is important to stay focused and ensure that certain pieces of your project are completely done; it is difficult to judge a project where nothing works.

DBMS Application Track

If you choose this track, your task will be to build a substantial database application for a real-world scenario of your liking. You can work alone or in teams of two or three. You will design a schema for your database, create it in a DBMS, and populate it with meaningful data. You will then develop your application. For applications targeted at end users, a graphical or Web-based interface would be nice. At the end of the course, you should be able to demonstrate your database-driven application or Web site.

Try to pick an application that is relatively substantial, but not too enormous. For project ideas, talk to your peers or consult with us. Besides applications that are traditionally database-driven (you will find plenty of them on the Web and in large organizations), I would encourage you to consider also applications that are traditionally *not* database-driven. Examples include managing USENET newsgroups and messages, managing your personal data (such as emails, contacts, and expenses), storing and analyzing simulation traces or Web clickstreams, etc. Hey, you can even toy with more radical ideas such as replacing the file system altogether! Intuitively, databases may or may not make sense for some of these applications. With your project, however, you will be able to back up the intuition with real data points, or, perhaps, you might find that your intuition was wrong!

You do not have to target your application at end users. Another possibility is to build better administration tools or query interfaces. How about table and column name completion (like in a UNIX shell)? How about automatically suggesting join conditions for the WHERE clause based on the referential integrity constraints on the FROM tables? How about form-based query interface used in Microsoft Access? The list goes on.

You can use choose any DBMS and platform that your team is comfortable with. We will maintain an IBM DB2 server for you to use, but we might be unable to provide much support, especially in setting up non-trivial application development environments.

Milestone 1 deliverables:

- A list of your team members.
- A project proposal (one to two pages). Its format will vary depending on the particular application, but you may wish to address some of the following (not necessarily in this order):
    - A description of your application.
    - Why it is important, interesting, and/or useful.
    - Preliminary system architecture and a list of tools you will use. For example, Java Swing client accessing Microsoft SQL Server via JDBC, Apache with PHP and MySQL, etc.
    - Plans to acquire or generate the data to populate your database.
    - A specification of your user interface.

Milestone 2 deliverables:

- Schedule a half-hour time slot with the course staff to demonstrate your project during the project demo and presentation period (see **Important Dates**).
- A project status report (one to two pages). Again, its format may vary, but it may include:
    - Changes to your original proposal (if any).
    - System architecture and the current status of each component.
    - Major tasks to be completed.

Final deliverables:

- A demo with the course staff on the scheduled date. The demo itself should be no longer than 20 minutes; it will be followed by a question-and-answer session of no longer than 10 minutes. Feel free to invite other students to your demo.
- A final project report (four to ten pages). It may include the following (not necessarily in this order):
    - Motivation and description of your application.
    - System architecture.
    - Design, loading, and maintenance of your database.
    - Screenshots of your user interface in action.
    - Retrospection on your design choices. If you are using DBMS in a non-conventional way, you should also discuss whether it was the right choice. For example, did the use of DBMS simplify or complicate development and maintenance? Did it improve or decrease performance? Back up your discussion with data and facts.

DBMS Research Track

The goal of this track is to provide an opportunity for you to conduct research in a database-related area. You can work alone or in teams of two or three. The depth and scope of your project may vary, but I encourage you to aim high and think of it as a conference-quality paper. You need to pose a question, design a framework in which to answer the question, survey previous and related work, conduct the research, and present your results and experience in a final project report.

If you choose this track, your project can be very open-ended. See the end of this document for a list of sample ideas. You are encouraged to come up with your own ideas, and I will be happy to talk to you about them. Note that your topic may or may not require substantial programming. It is always a good idea to consult with the course staff before Milestone 1 to ensure that your topic is of appropriate depth and scope.

Milestone 1 deliverables:

- A list of your team members.
- A project proposal (no more than five pages). It should include the following components (not necessarily in this order):
    - A description of the problem.
    - Why it is important, interesting, and/or useful.
    - Survey of previous work. Discuss how they related to your problem and identify any limitations and/or flaws. It does not need to be an exhaustive literature search at this point, but make sure that you have covered the recent literature and that you are not duplicating any previous work.
    - Initial thoughts on how to approach the problem.
    - Preliminary system architecture, experimental setup, and required resources (if applicable).
    - Format of the conclusion that you wish to draw at the end of this project. Although you obviously do not know the results yet, you should have some idea about the format of the results, e.g., "for typical so-and-so query loads, we will show that our approach outperforms the previous ones by so-and-so in terms of the number of I/Os." Think of it this way: What would you write in your abstract and conclusion to get the attention of program committee members? Having a clear goal helps you stay focused.

Milestone 2 deliverables:

- Schedule a half-hour time slot with the course staff to present your results during the project demo and presentation period (see **Important Dates**).
- A project status report (no more than two pages). It should include the following:
    - Changes to your original proposal (if any).
    - Summary of any results obtained and/or tasks completed so far.
    - Tasks to be completed before the final due date.

Final deliverables:

- A presentation (or demo, if applicable) to the course staff on the scheduled date. The presentation itself should be no longer than 20 minutes; it will be followed by a question-and-answer session of no longer than 10 minutes. Feel free to invite other students to your presentation.

- A final project report (eight to twenty pages). Its format should be that of a conference paper, complete with abstract and references. Be honest about the weaknesses in your work, and turn them into something positive—such as a "future work" section. I hope we will be able to produce several conference-quality reports from this class, and I will be happy to work with you to turn them into real submissions.

Below are some project ideas for the research track. Some are very open-ended and definitely require you to narrow them down further. You are welcome to talk to me if you are interested in a topic and would like some more pointers.

- *Downsizing DBMS*. Many databases in the future will not reside in large data centers, but instead on devices such as PDAs, cell phones, refrigerators, etc. What are the data management and communication requirements of such devices? A full-fledged DBMS is obviously an overkill and simply will not fit on these devices. Many DBMS vendors thus have started from scratch. Sybase iAnywhere took an interesting approach of "compiling" a mini-DBMS for PDA applications. The compiled mini-DBMS has a very small footprint, because it only supports the necessary features required by the specific applications. How would you improve the flexibility of this approach?

- *Consistency criteria for materialized views*. Materialized views are widely used to improve the performance of complex queries. The idea is to pre-compute and store the results of frequently asked queries as materialized views. Then, we can use them to answer subsequent queries efficiently. The trade-off is that we now need to refresh the contents of the materialized views when base tables are updated. To get better update performance, can we relax the consistency requirement? Previous work considered relaxing the "freshness" requirement—applications can access materialized views that are not up-to-date (although they still accurately reflect an old state of the base tables). Does it make sense to relax the "accuracy" requirement as well? In other words, the contents of the materialized views may not reflect any accurate state of the base tables, but they are guaranteed to be within some error bound. Could you use this approach to make it less expensive to maintain materialized views?

- *Web query caching and prefetching*. In particular, we are interested in techniques for caching and prefetching Web pages that are dynamically generated by querying a backend DBMS. Where would you place the cache? Does it make sense to do it at the DBMS rather than at the Web servers or proxies? And who would initiate prefetching? What prediction algorithm would you use for prefetching? Could data mining over clickstreams help? Which one is the bottleneck, network or DBMS? How would you design a flexible scheme that can adapt to the changes in system load?

- *Synopsis data structures*. Generalizing the idea of materialized views, synopses are small data structures that capture the essential characteristics of a database for providing fast, approximate answers to certain queries. As with materialized views, we must maintain synopses up-to-date. The key difference between synopses and materialized views is that synopses provide approximate answers, while materialized views traditionally provide exact answers. An example of synopsis is a histogram that captures the distribution of data in a table. How about applying the idea to more complex queries or to non-relational data, e.g., XML?

- *Scalable real-time stream processing*. Suppose that data is streaming to us continuously (e.g., from sensors, access logs), and we need to provide up-to-date answers to queries over the data that we have seen. With limited resources for query processing, it is possible that we cannot keep up with an increase in the rate of input. The naïve approach is to buffer the input data and continue processing them in order, hoping that we would catch up later. A more graceful approach is to start trading accuracy for time, so that our answers are always "current" in some sense. Later, when input slows down, we can go back to the buffered data and refine our answers. Moreover, it should be easy to add and remove query-processing resources dynamically. How would you design such a system and develop algorithms to support it?

- *DBMS with active disks*. Active disks are "smart" disks with processors that can execute application-specific code. Traditional DBMS query optimizers assume "dumb" disks that only know how to transfer entire pages to and from memory. Active disks allow much processing to be push down. How about some new query processing algorithms using active disks or an optimizer that knows how to exploit them? Or the possibilities of encryption, compression, and re-mapping of disk pages for better memory and cache performance?

- *Personal information integration*. Almost every single activity of ours is tracked by someone else's database—banks, credit card companies, stores, utility companies, Web sites, etc. Amazingly, we have no convenient means of tracking and managing such information ourselves. We could conduct our transactions through a personal device with data recording capability (such as PDA or smartcard), but that device still has to deal with different data exchange interfaces and formats used by other parties (say, Kroger versus Harris Teeter). In the area of information integration, "wrappers" and "mediators" are often used to mask the heterogeneity of information sources. Would a similar architecture make sense for personal information integration? What are the new issues?

- *Online and/or incremental data mining.* As databases grow to hundreds of terabytes, data mining operations take excessively long time to run. One solution is online data mining, which provides immediate feedback to users with approximate answers. Users also can control the continuous refinement of answers. Another technique is incremental data mining. The idea is to keep the result of a mining operation around, perhaps with some additional information. When the database is updated, we compute only the incremental changes to the result, instead of running the mining operation all over again on the new state of the database. Can you develop an online and/or incremental version of a traditional mining algorithm?

- *Web search on a relational database*. Many Web search engines make heavy use of the hyperlink structure to improve search. What are the "hyperlinks" in relational databases? How about referential integrity constraints? Does it make sense to apply tricks such as Google's PageRank to relational databases?