MINI-DB: Demystifying the Inner Workings of a Database Systems

Hossein Hakimzadeh, Robert Batzinger, Susan Gordon
Department of Computer and Information Sciences
Indiana University - South Bend
South Bend, IN 46615

[hhakimza, rbatzing, sgordon] @ iusb.edu

ABSTRACT
As computer science continues to move toward a more pragmatic market driven discipline, some of the traditional core topics in computing have become less popular. At-risk topics include compilers, file organizations, and operating systems. Viewed collectively, it appears that courses that either deal with the internal working of computers or courses that require system programming are being systematically removed from the undergraduate curriculum. This trend can be seen in the 2001 and 2005 ACM/IEEE Computing Curricula recommendations. This paper advocates a project-based approach to reintroduce system development topics into the computer science curriculum. The authors describe the structure of a database system development course, and provide additional ideas for similarly structured courses.

Categories and Subject Descriptors
K.3.2 [**Computer and Information Science Education**]

General Terms
Algorithms, Design.

Keywords
System Development, Advanced Databases, Algorithms, Database Engine, Project-based Approach

1. INTRODUCTION
Most contemporary computer science programs base their curriculum on Computing Curricula recommendations put forth by the ACM /AIS / IEEE joint task force for computing curricula [1, 2]. With each new report, these recommendations move away from core computer science topics and toward contemporary topics such as cyber security, distributed computing, bioinformatics, and game programming.

Core topics such as compilers, file organizations, and operating systems appear to be at risk. The trend suggests that courses dealing with the internal working of computers or system software are being systematically removed from the curriculum. Although most computer science graduates will not be expected to develop file systems, compilers, or operating systems at their jobs, the knowledge and skills they obtain in these core topics create an intuitive understanding of "system software" which fine tunes their ability to design more efficient and more reliable "application software".

This paper proposes increasing student exposure to system level topics by incorporating a series of project-based system development courses into the undergraduate curriculum. It will discuss a course in "*Advanced Database Systems*" as a case study for this approach. One objective of this course is to design and implement a database engine as the vehicle for fostering system development skills.

2. BACKGROUND
In an article published in 2002, Mary Shaw [3] advocates constructing the computer science curriculum around abstract themes that cut across the discipline. She further criticizes the teaching of "compiler construction" and "operating systems" as teaching "system-artifact dinosaurs". It is clear that members of the CC2001 Task Force [2] were inspired by this article and consequently, the Computing Curricula recommendations [1, 2] clearly place less emphasis on system development topics such as compiler construction and file organizations. Fortunately, the study of operating systems still remains prominent in the computer science curriculum; however, there are pressures to create a hybrid operating system course which integrates topics such as networking and security.

Naturally, the CS curriculum at Indiana University South Bend has mirrored the CC2001 and CC2005 recommendations. The department stopped offering *"Assemblers and Compilers"* and *"Information Organization and Retrieval"* in the late 1990's. In their place, more courses such as graphics, networking, computer vision, game programming, software engineering, artificial intelligence, bioinformatics, and computer security are offered. Although an excellent case can be made for teaching any of the above courses, the fact remains that students leave our program with significantly less exposure to system programming skills.

The central questions remain: **Can computer science graduates thrive without these core skills? How will**

the loss of these skills affect the next generation of system designers? Although, the concept of creating courses that cut across the discipline is appealing, we must be careful not to deprive the future computer scientists from an important body of knowledge in computing.

Since 2000, the department has attempted to increase course and degree offerings in the area of "interdisciplinary computing" [1][4, 5]. At the same time, it has introduced courses that increase the coverage of "system programming" concepts and practices.
This paper profiles the implementation of a course in "*Advanced Database Systems*". The primary focus of this course is to study the inner working of system software, specifically how database management systems work. The course systematically leads the students through the design and implementation of a database engine called MiniDB[6].

Similar project-based courses in advanced database design were introduced by David DeWitt [7] and later extended by Mike Carey and Raghu Ramakrishnan [8] at the University of Wisconsin. Another system inspired by the DeWitt has been implemented by Albano *et al* [9].

In the remainder of this paper, we will discuss the structure of this course, future directions, and lessons learned.

3. THE MiniDB SYSTEM

The MiniDB system is developed as a semester long project to introduce the underlying theories, principles and practices to implement a simple but flexible database engine. The prerequisite for the course is an undergraduate database course that introduces the students to data modeling, the relational model, relational algebra, SQL, and additional topics such as transaction management, concurrency control, and data mining.

Conceptually, the course and its project (MiniDB) are divided into five phases as shown in (Figure 1):

1. Preparation
2. Design and implementation of core algorithms (implementation of MiniDB Engine)
3. Researching advanced algorithms
4. Implementation of advanced algorithms
5. Presentation of final project

[1] In 2002 the department introduced a new degree program in Informatics which emphasizes applications of information technology to other disciplines such as sciences, arts, and health care.

The above phases are planned such that in the first ten weeks of the semester, students construct a fully functional mini database engine. In the last five weeks, each student chooses to research one or more advanced topics in databases and integrates their research findings into their MiniDB implementation. Sections 3.1 to 3.4 below describe each phase of the course.

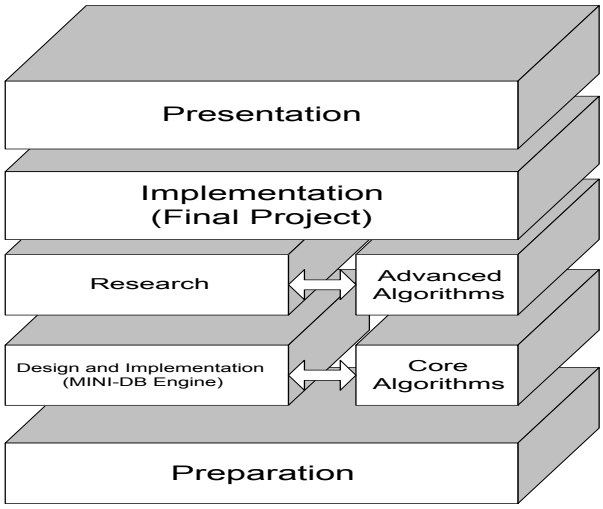

Figure 1. MiniDB Conceptual Model

3.1 Preparation

The preparation phase (Figure 1) includes an introduction to I/O devices, file organizations and basic I/O facilities. Each student selects a programming language and investigates its file manipulation API. The result of their investigation is compiled into a survey paper. Most students select C++ or Java for this purpose, although languages such as C# and Ruby have also been selected.

The goal of this phase is three-fold. First, it provides the students with extensive exposure to file I/O, a topic which is often lightly-covered in earlier programming and data structure courses. It also allows them to refine their research skills and provides an opportunity to collect and organize a comprehensive paper with useful examples of I/O facilities in their chosen language. The paper serves as a quick reference guide as they work toward the development of the MiniDB.

3.2 Design and Implementation of MiniDB

The design and implementation of the MiniDB engine is performed in three stages, each corresponding to an assignment. During the first stage, the students construct classes for performing sequential, random, and index sequential file access. These classes create the underlying infrastructure for constructing the ***data***, ***meta-data***, and ***index*** files [10] which are necessary for creating a database table.

During the second stage, the meta-data class is extended to allow the storage of meta-data in XML format. In addition during this stage, students begin the development of a minimal set of relational algebra operators such as *select*, *project*, and *cartesian product*[11].

The third stage refines the functionality and extends the relational algebra class to include additional operators such as *join*, *union*, *intersection* and *difference*. It also extends the index class to include hashing [12]. At this point, each student has implemented a simple yet functioning database engine based on relational algebra. (Figure 2)

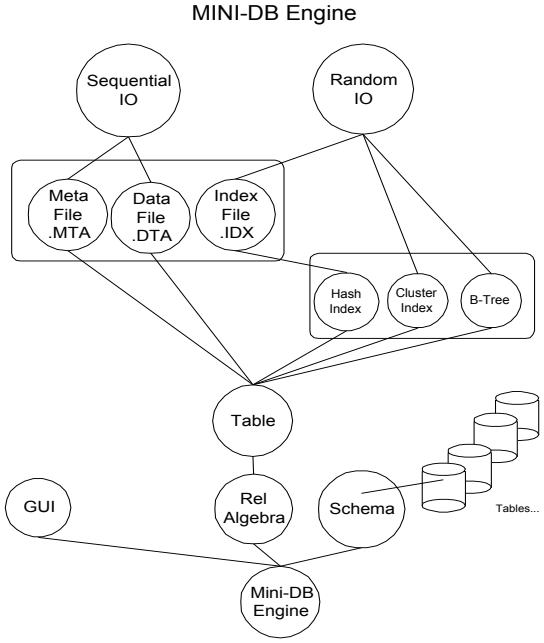

Figure 2. MiniDB Implementation Model

3.3 Research in Advanced Algorithms

During the implementation of the MiniDB, while the students are engaged in constructing the engine, approximately 50% of the lectures are devoted to the discussion of advanced database concepts. This is done to prepare the students for the research phase of the course. Topics include query optimization, security, concurrency control, replication, distributed databases, and deductive databases.

At this point, the students are asked to select a topic, review the related literature and write a research paper. As part of their paper, they are asked to propose an "Implementation Plan" to incorporate one or more of the techniques discussed in their paper into their MiniDB engine. During the next several sessions, class lectures turn into class discussion and brainstorming of the above mentioned proposals.

3.4 Presentation and Demonstration

In the final phase of the project, students follow through with their *implementation plan*, develop a *test plan* and *present* their results to their peers. In the discussion that typically follows the presentation, students share their memorable experiences.

4. DESIGN CONSIDERATIONS

Initially, most students will find it difficult to envision creating a database engine from scratch. In order to guide the design and implementation process, the task is broken down into a series of deliverables which are due at two weeks intervals. Each deliverable serves two purposes. First, it seeks to incrementally construct new building blocks that move the project toward the goal of a working database engine. Second, it allows the students to systematically refine the previously constructed components. This ability to incrementally refine one's work is an essential yet underdeveloped skill among many students.

The remainder of this section discusses the project deliverables. These include the creation of access mechanisms, data definition language (DDL), data manipulation language (DML), relational algebra operators, meta-data, XML, and index structures.

4.1 Access Mechanisms

The goal of the first deliverable [10] is to construct a series of classes for creating and manipulating simple data, index and meta-data files. These three classes provide the basis for creating a database table.

The ***data file*** is a sequentially organized but randomly accessed file. The sequential organization provides for better space efficiency. At the same time, the direct access improves speed. The data file has a simple format which separates the fields and records using delimiters such as the "^" and "~" characters respectively.

The ***index file*** is a direct access file. Records in this file are fixed size and have the structure shown below, allowing for records with numeric primary keys.

```
unsigned long Key;     // Key to search for
unsigned long Address; // Physical file location
char Flag;             // ACTIVE / DELETED
```

The ***meta-data file*** is a sequential file that will maintain schema information about the database. Meta-data files are central to creating the initial database engine as well as future advanced optimization algorithms. Meta-data files are used at many levels; first, they are associated with each data file created by the user. Later, meta-files can be

used to maintain schema information such as user access and authorization, log information, or query optimization statistics. Initially, the meta-data file format is quite simple, with each record having the following format:

Tag Name=^ Field information[^Field information…]~.

Figure 3 below provides an example of meta-data file.

```
DATABASE_NM=^University~
TABLE_NM=^Student~
NUM_FIELDS=^2~
FN=^StudentID~
FS=^5~
FT=^String~
FN=^StudentName~
FS=^25~
FT=^String~
PK=^StudentID~
```

Figure 3. Sample meta-data file

4.2 Creating the Data Definition and Data Manipulation Language

Once the initial data access objects are implemented, the next goal is to construct a simple data definition and data manipulation language for the MiniDB engine [8]. Relational algebra is chosen for this purpose. The basic relational algebra operations include select, project, join, union, intersection,

```
Class Mini_Rel_Algebra {
        bool create(relation_name, schema);
        bool insert(relation_name, attribute_list, value_list);
        bool delete(relation_name, attribute_name, condition,
                        attribute_value);
        bool modify(relation_name, search_attribute_name,
                        condition, search_attribute_value,
                        modify_attribute_list,
                        modify_value_list);
        result_rel select(relation_name, attribute_name,
                        condition, attribute value);
        result_rel project(relation_name, attribute_list);
        result_rel cartesian_product(relation_1, relation_2);
}
```

Figure 4. Relational Algebra Operations

difference, cartesian product and divide. However, we split the implementation of these operators with the select, project and cartesian product implemented first (Figure 4).

In addition to creating a new class for relational algebra operators, this assignment also incrementally refines the meta-data class. During this phase, we replace the initial meta-data file format with a simple XML format (Figure 5). In addition students create a new XML parser.

```
<SCHEMA_NAME>
        Database Name
</SCHEMA_NAME>
<TABLE_NAME>
        Table Name
</TABLE_NAME>
<NUM_FIELDS>
        Number_of_Fields_In_Table
</NUM_FIELDS>
<FIELD>
        <FIELD_NAME>
                Field Name
        </FIELD_NAME>
        <FIELD_SIZE>
                Field Size
        </FIELD_SIZE>
        <FIELD_TYPE>
                Field Type
        </FIELD_TYPE>
</FIELD>
::
<PRIMARY_KEY>
        Field Name
</PRIMARY_KEY>
<FOREIGN_KEY>
        Field Name
        <REFERENCES_FOREIGN_TABLE>
                Table Name
        </REFERENCES_FOREIGN_TABLE>
</FOREIGN_KEY>
```

Figure 5. XML definition of the meta-data file

4.3 Extending the Relational Algebra Class and Refining the Indexing Mechanism

In this phase [9] students extend the set of relational algebra operators to include join, union, intersection and difference (Figure 6).

```
Class Mini_Rel_Algebra {
        …
        result_rel join(relation_1, relation_2, condition_list);
        result_rel union(relation_1, relation_2);
        result_rel intersect(relation_1, relation_2);
        result_rel difference(relation_1, relation_2);
}
```

Figure 6. Relational Algebra Operations (Extended)

The index class is refined and optimized using hashing techniques (Figure 7). This Hash_Index class can inherit the Index class and override its *find()* method. As an option, some will also develop a cluster index class to handle indexing based on non-key attributes (Figure 7). The Cluster_Index can be based on the Index or Hash_Index class, or it can be a separate class.

```
Class Hash_Index {
        long insert(char *key);
        long find(char *key);
}
Class Cluster_Index {
        long build_index(relataion_name, char *nonKeyArtribute);
        long find(char *nonkey);  // return the pointer to cluster
}
```

Figure 7. Refined Index Classes

The Cluster_Index class is often implemented using one of the existing primary index classes and extends its functionality to accommodate cluster indexes for non-key attributes (Figure 8).

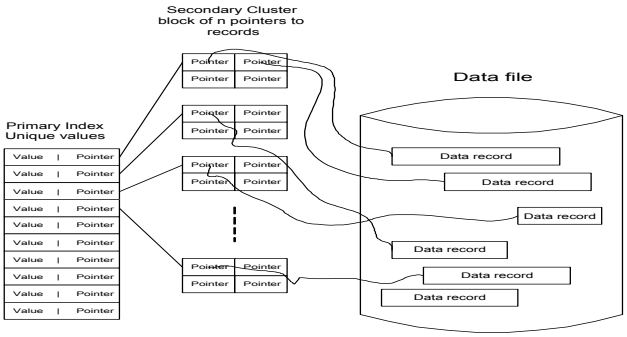

Figure 8. MiniDB Cluster-Index

4.4 Final Phase (Advanced Algorithms)

The final phase of the course involves the creation and integration of advanced components on top of the basic MiniDB engine. During past offerings of this course, students have been able to develop algorithms for concurrency control, database security, access control, database integrity [14], external sorting, data mining [15], join optimization [16], distributed databases [17], and deductive databases.

5. FUTURE DIRECTIONS

Our plan for the future is two-fold. First, our goal is to make the course more manageable for undergraduate students. Second, we plan to make MiniDB an open source, teaching and research platform.

Toward this end, we have developed an open source API for the first phase of the project which will provide the students the proper infrastructure to begin the course. Students will use the first few weeks of the semester to study and master the MiniDB concepts, learn its API, and begin to customize the API for future use. The current open source API, a set of assignments, and suggestions for research ideas are available through the MiniDB web site www.cs.iusb.edu/minidb. Interested faculty are encouraged to contact the first author to obtain the source code for the MiniDB project.

In addition to aiding our pedagogical goals, MiniDB will continue to be used as a tool for database research. Both undergraduate and graduate students who finish this project are well positioned to conduct research in database systems. In the recent past, two graduate students have used their project as a stepping stone in developing their graduate thesis proposals. Having source-level access to the MiniDB platform will also allow students to propose their own algorithms and benchmark their implementations against previously published algorithms.

CONCLUSION

Computer science is an evolving and growing discipline. The computer science curriculum is under constant pressure for change. This pressure comes from many constituencies, including the ACM / AIS / IEEE-CS report on Computing Curricula [1, 2]; ABET / CAC / EAC [9] accreditation guidelines; business and industry demands; and the general globalization of information technology. Although these forces are not always aligned, the combined trajectory appears to be toward contemporary topics such as cyber security, distributed computing, data mining, bioinformatics, and game programming and away from traditional core topics. The cumulative and compound effect is a reduced understanding and appreciation of system software among computer science graduates.

This paper describes the design and implementation of a database engine as a vehicle for the reintroduction of system development topics into the computer science curriculum. Further, this paper describes the use of a project-based approach to systematically develop and refine code components for the database engine known as MiniDB.

The above project-based, system development approach can be applied to other courses such as computer networks, computer graphics, and computer security. For example, a computer networking course can be augmented such that, through a series of assignments, students finish the course with their own MiniNetwork API based on the OSI model. Students in computer graphics could build their own MiniGameEngine. Finally, students in computer security could build a simple vulnerability scanner. Each of the mentioned projects help to improve the students' system development capabilities. As the computer science curriculum continues to refine and redefine itself, we hope that system development will regain more prominence in the curriculum.

REFERENCES

[1] IEEE/ACM-CS Computing Curricula 2001,Computer Science, Final Report, The Joint Task Force on Computing Curricula, IEEE Computer Society, Association for Computing Machinery, December 15, 2001.

[2] ACM/AIS/IEEE-CS, Computing Curricula 2005, by The Joint Task Force for Computing Curricula 2005, 30 September 2005.

[3] Shaw, M., "We can teach software better." Computing Research News 4(4):2-12, September 1992.

[4] Wolfer, J., Schwartz, R. B., Hakimzadeh, H., "Informatics and the Diversification of the Computing Curriculum", International Conference on Engineering and Computer Education (IEEE/ICECE'05). Nov. 13-16, 2005, Madrid, Spain.

[5] Schwartz, R. B., Hakimzadeh, H., Wolfer, J., "Meeting Computing Curriculum Challenges: A Profile of the Indiana University South Bend Informatics Program", The 9th Annual IJME / INTERTECH Joint International Conference on Engineering and Technology. (Oct. 19-21 2006), New York.
[6] Hakimzadeh, H., Batzinger, R., Gordon, S. "System Development: A Project Based Approach", ACM-SIGCSE 2009 Conference, Chattanooga, Tennessee, March 4-7, 2009
[7] DeWitt, D., the Minirel project. A database course project that involved building a small relational DBMS.
[8] Carey, M., Ramakrishnan, R., the Minibase project. Extension and redesign of Minirel project. www.cs.wisc.edu/coral/mini_doc/project.html
[9] Albano, A., JRS (Java Relational System) is a relational DBMS implemented in Java and designed for educational use. Accessed on the web on March 2008, www.di.unipi.it/~albano/JRS/project.html
[10] http://www.cs.iusb.edu/minidb/assignments/assign2.pdf
[11] http://www.cs.iusb.edu/minidb/assignments/assign3.pdf
[12] http://www.cs.iusb.edu/minidb/assignments/assign4.pdf
[13] Computer Science Accreditation Criteria (CAC), and Computer Engineering Accreditation Criteria (EAC), accessed on web on Dec. 2007 http://www.abet.org/
[14] Gordon, S., "Database Integrity: Security, Reliability, and Performance Considerations", Technical Report: TR-20071226-1, accessed on web on Dec. 2007. www.cs.iusb.edu/technical_reports/TR-20071226-1.pdf
[15] Batzinger, R., "Calling R from Ruby", Technical Report: TR-20080109-1, accessed on web on Jan. 2008. www.cs.iusb.edu/technical_reports/TR-20080109-1.pdf
[16] Rupley, Michael, Jr., "Introduction to Query Processing and Optimization", Technical Report: TR-20080105-1, accessed on web on Jan. 2008. www.cs.iusb.edu/technical_reports/TR-20080105-1.pdf
[17] Rababaah, H., "Distributed Databases Fundamentals and Research", Technical Report: TR-20050525-1, accessed on web on Dec. 2007. www.cs.iusb.edu/technical_reports/TR-20050525-1.pdf
[18] Robergé, James, Carlson, C. R., "Broadening the computer science curriculum", Proceedings of the twenty-eighth SIGCSE technical symposium on computer science education, San Jose, California, ISSN:0097-8418, 1997, pp. 320 – 324.
[19] Dahlbom, B., Mathiassen, L., "The future of our profession", Communications of the ACM, Volume 40 , Issue 6 (June 1997) Pages: 80 – 89, 1997, ISSN:0001-0782