MINI-DB:
A Pedagogical tool for Teaching Advanced Database Systems

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

ABSTRACT

It seems that designing and implementing database engines may have become a lost art. Although most standard database text books [1] [2] [3] include ample coverage of algorithms for design and implementation database engines, many computer science programs seem to provide minimal coverage of file organizations, theoretical foundations, and algorithms necessary to build a database engine.

The systematic removal of "file organizations and information retrieval" as a topic of study coupled with greater emphasis on the so called "practical applications" of databases, have joined hands to eliminate the coverage of theory and implementation of the underlying database engine.

1. INTRODUCTION

At IU South Bend, the computer science program offers a number of database courses including a one at the 100 level, another at 400 level, and a third course at the 500 level. The 100 level is for non-majors and targets novice practitioners. Our 400 level database course targets juniors and seniors in our Computer Science and Informatics programs and its goal is to survey the basic concepts and theories behind the modeling and implementation of small to medium scale database management system. The 500 level course targets seniors and graduate students who have already completed the 400 level database course and moves toward the internal design and implementation of database engines.

In this paper, we will discuss a step by step process by which students in our advanced database course design and construct a simple, yet fully functional database engine. We will also explore some lessons learned and future directions.

2. THE MiniDB SYSTEM

The goal of the advanced database course is to introduce the students to the underlying theories, principles and practices for implementing a simple and flexible database engine. The prerequisite for the course is an undergraduate database course which introduces the students to data modeling, relational model, relational algebra, SQL, and some additional topics such as transaction management, concurrency control, and data mining.

Conceptually, the advanced database course is divided into five phases 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

Below we will describe each phase.

2.1 Preparation

During the preparation phase (bottom layer of Figure 1), students are provided with a quick introduction to I/O devices, file organizations and basic I/O facilities. Each student selects a language and researches the file manipulation API for that language. The result of their research is the compilation of a survey paper. Typically, most students select C++ or JAVA for this purpose, however, languages such as C, C#, and Ruby have also been selected.

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

2.2 Design and Implementation MiniDB

The design and implementation of MiniDB engine is performed in three stages and each stage corresponds 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 [6], which are necessary for creating database tables.

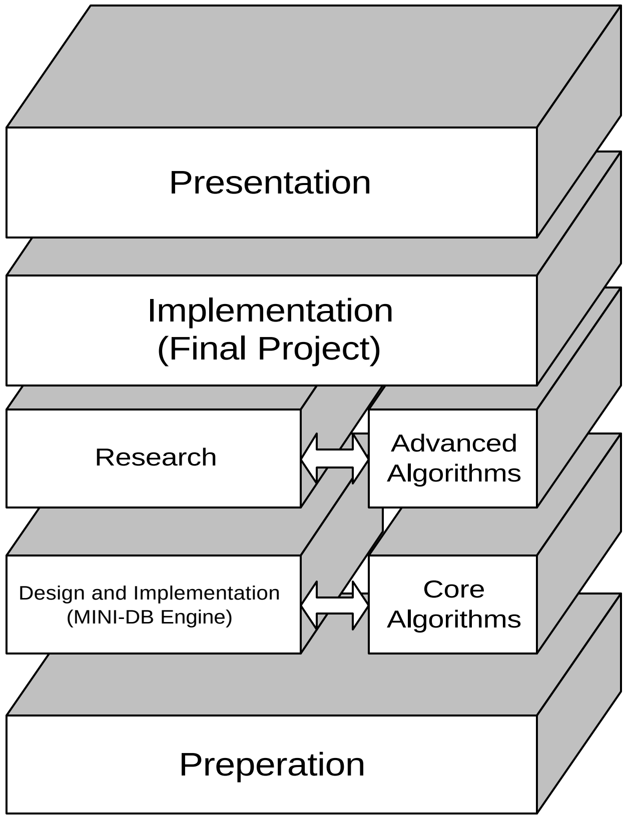
Presentation
Implementation (Final Project)
Research
Advanced Algorithms
Design and Implementation (MINI-DB Engine)
Core Algorithms
Preperation

Figure 1. MiniDB Conceptual Model

During the second stage, the meta-data class is extended to include XML capabilities, and builds a new class for implementing a minimal set of relational algebra operators such as select, project, and cartesian product [7].

The third stage extends and refines the relational algebra class to include additional operators such as join, union, intersect and difference. It also extends the index class to include hashing [8].

Approximately ten weeks into the semester, after the completion of the third stage, each student has a simple yet functioning database engine which is based on relational algebra.

2.3 Research in Advanced Algorithms

During the Implementation of MiniDB engine, while the students are engaged in constructing the engine, approximately 50% of the lectures are dedicated to advanced database concepts. Such as query optimization, security, concurrency control, replication, distributed databases, and deductive databases. The above coverage is meant to prepare the student for the next phase of the course.

At this point, the students are asked to select a topic that most interest them, review the related literature and write a research paper on that topic. As part of their paper, they are asked to propose an "***Implementation Plan***", as to how they would implement into their MiniDB, one or more of the techniques discussed in their paper.

During the next several session, class lectures turn into class discussion and brainstorming of the above proposals.

2.4 Presentation and Demonstration of Final Project

In the final phase of the project students follow through with their ***implementation plan***, develop a ***test plan*** and ***present*** their results in class.

In the following section, we will highlight the design considerations for the construction of the MiniDB system.

3. DESIGN CONSIDERATIONS

Initially, most students will find it difficult to envision creating a database engine such as one shown in Figure 2, from scratch. In order to guide the design and implementation process of the MiniDB engine, a series of two week long deliverables have been created. Each deliverable serves two purposes. First, it seeks to incrementally construct new building blocks that move the project toward the goal of constructing a database engine. The second purpose is to systematically refine the previously constructed code components.

The remainder of this section discusses the project deliverables. These include the creation of: access mechanism, data definition language (DDL), data manipulation language (DML), relational algebra operations, meta-data, XML and the ability to create primary and clustered index structures.

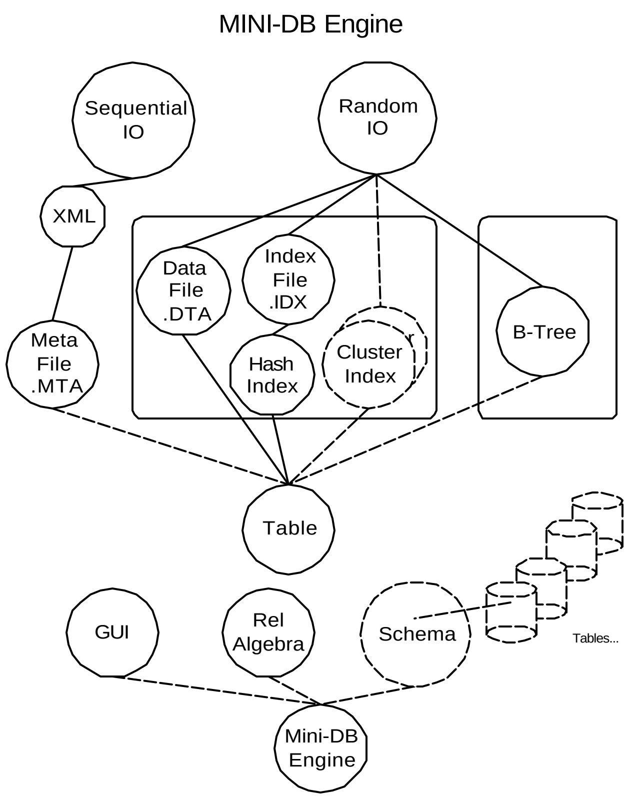

Figure 2. MiniDB Implementation Model

3.1 Access Mechanism

The goal of the of the first deliverable [6] is to construct a series of classes for creating and manipulating simple data (.dta), index (.idx) and meta-data (.mta) files. These three classes provide the basis for creating a database table.

The ***data file*** (.dta) is a sequentially organized but directly (randomly) accessed file. The file is sequentially organized in order to provide for better space efficiency. At the same time the data files are directly accessed to improve access speed. The data file has a simple format which separates the fields and records using delimiters such as "^" and "~" characters respectively.

The ***index file*** (.idx) is direct access (random) file. Records in this file are fixed size and have the following format:

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

The ***meta-data file*** (.mta) is a sequential file which will maintain schema information about the database. Meta-data files are quite central to creating a database engine. Meta-data files are used at many levels; first, they are associated with each data files created by the user. Meta-data files are also used to maintain other schema information such as user access and authorization, log information, query optimization information, and other internal schema information. Initially, the meta-data file format is quite simple; however at later stages (described later in the paper) this class will be enhanced to accommodate the XML file format. Initially, the records in this file have 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=^Student Name~
FS=^25~
FT=^String~
PK=^StudentID~
```

Figure 3. Sample meta-data file

3.2 Creating the Data Definition and Data Manipulation Language

Once the initial data access objects are implemented, we are ready to tackle the next phase. The goal of this phase is to construct a simple data definition and data manipulation language for our MiniDB engine [7]. Relational Algebra is chosen for this purpose. The basic relational algebra operations include select, project, join, union, intersection, difference, cartesian product and divide. However, we split the implementation of these operators in to two assignments. During the first assignment, the select, project and Cartesian_product are implemented (Figure 4), and during the next assignment, the relational algebra class is extended to include join, union, intersect and difference (Figure 6).

```
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

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 XLM format. In addition we will create a new XML parser. The format of the XML meta-data file is shown in Figure 5.

```
<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

3.3 Extending the Relational Algebra Class and Refining the Indexing Mechanism

The goal of this phase [8] is to first, extend and complete the set of relational algebra operators (Figure 6) second, to refine and optimize the index class using hashing techniques (Figure 7), and finally to develop a cluster index class to handle indexing based on non-key attributes (Figure 8).

```
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);

      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 Hash-Index class can inherit the Index class and override its *find()* method. This will allow for much better space utilization of the index file as well as ability to accept multiple key type such as long or character strings.

```
Class Hash_Index {
      long insert(char *key);
      long find(char *key);
}
```

Figure 7. Hash Index

The Cluster_Index class is primarily designed to accommodate indexing of non-key attributes. The Cluster_Index can be either based on the Index, class, Hash_Index class, or it can be separate class.

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

Figure 8. Cluster Index

The Cluster_Index class will use one of the existing primary index classes and extends it's functionality to accommodate cluster indexes on non-key attributes (Figure 8, 9). Cluster indexes can be used to optimize a number of different operations, such as sort, select, and join.

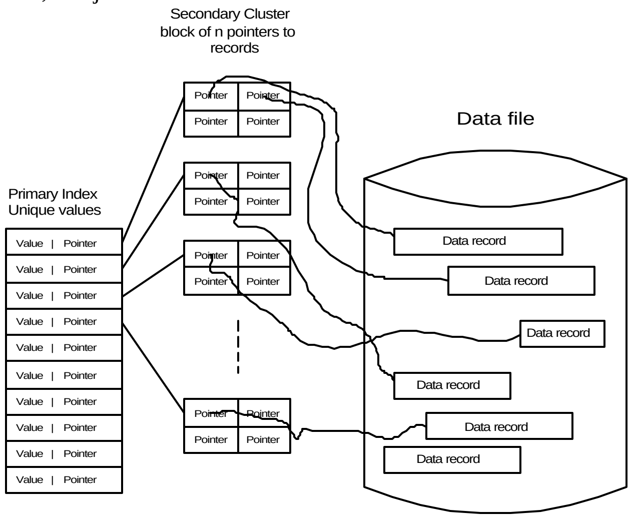
Secondary Cluster block of n pointers to records
Primary Index Unique values
Value | Pointer
Pointer
Data file
Data record

Figure 9. MiniDB Cluster-Index

3.4 Final Phase (Advance Algorithms)

The final phase of the course involves the creation and integration of advanced components on top of the basic MiniDB engine (Described in section 3.3 and 3.4 above). During past offering of this course, student have been able to develop algorithms for concurrency control [Aarti Khaire], database security, access control, database integrity [Sue Gordon], external sorting [Hung Truong Quoc], data mining [Bob Batzinger], query optimization[Mike Rupley], distributed databases [10], and deductive databases [Tom Perez].

Student who complete the final phase of the project are well positioned to continue their research in the area of databases. In the recent past, two graduate students have used their project as a stepping stone in developing and proposing their graduate thesis. Similar research opportunities are available for undergraduate students who are interested in further study in the area of database systems.

5. FUTURE DIRECTIONS

Our plan for the future is two folds. First, our goal is to make the course more reachable to undergraduate students. Second, we would like to make this project an open source, research and teaching platform.

At the present, MiniDb is implemented in an advanced database course which targets our graduate students. In order to make the course more reachable to undergraduate students, we will develop a robust open source API for the first two phases of the project. This API provides our undergraduates with the proper infrastructure to begin the course. We expect that undergraduates will spend the first couple of weeks of the semester to read and master the MiniDB concepts and learn its API.

Aside from our pedagogical goals, we will continue to use the MiniDB as a tool for database research. Undergraduates and graduate students who finish the project will be positioned to conduct research in database systems. Having source-level access to the MiniDB platform will allow them to implement existing state of the art algorithms or propose their own, and then implement and benchmark their algorithm against the state of the art.

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 ACM / AIS / IEEE-CS report on Computing Curricula [1,2]; ABET / CAC / EAC [7] 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 more toward contemporary topics such as cyber security, distributed computing, bioinformatics, and game programming, and slightly away from traditionally core topics such as compilers, file organizations, and operating systems. The cumulative and compound effect can result in reduced understanding and appreciation of systems software among our graduates. This paper discusses the design and implementation of a database engine as the vehicle for reintroduction of system development topics back into the computer science curriculum.

REFERENCES

[1] Elmasri, R., Navathe, S., “Fundamentals of Database Systems,” Fifth Edition, , Addison-Wesley

[2] Silberschatz, A., Korth, H., Sudarshan, S., “Database System Concepts” , 5th edition, McGraw Hill, 2005.

[3] Date, C. J., “An Introduction to Database Systems”, Eighth Edition, Addison Wesley, 2004.

[4] http://mypage.iusb.edu/~hhakimza/561/assign2.pdf

[5] http://mypage.iusb.edu/~hhakimza/561/assign3.pdf

[6] http://mypage.iusb.edu/~hhakimza/561/assign4.pdf

[7] Computer Science Accreditation Criteria (CAC), and Computer Engineering Accreditation Criteria (EAC), accessed on web on Dec. 2007 http://www.abet.org/

[8] 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

Appendix A

Design specification for advanced algorithms typically implemented during the last Phase of the course.

Query Optimization:

```
Class QueryOptimizer {
        Tree *QueryTree;     // original query tree
        Tree *OptimizedTree;          // optimized query tree.
        Tree *Insert( Relational_algebra_operator,
                relation1, [relation2], [conditions] );
        Tree *OptimizeQueryTree();
        PrintQueryTree(Tree *);
etc.
}
```

2PL Concurrency Control:

```
Class ConcurrencyControl {
    bool XLock( T-id, relation);
    bool UnLock( T-id, relation);
    bool XLock( T-id, relation, record );
    bool UnLock( T-id, relation, record );
    bool PrintLockTable();
    bool DetectDeadlock();
    T-id ResolveDeadlock();
    Bool Abort(T-id);
    etc.
}
```

Distributed Database

```
Class DistributedDB {

    bool AddRelationToGlobalSchema(Relation,
                                   OriginatingNode);
    bool RemoveRelationFromGlobalSchema(Relation,
                                   OriginatingNode);
    bool DisplayGlobalSchema( );
    bool XLockRelation(T-id, relation );
    bool UnlockRelation(T-id, relation );
    relation ReadRemoteRelation(T-id, relation );
    bool WriteRemoteRelation(T-id, relation );
    etc.
}
```

Deductive Database

```
Class InferenceEngine{

    bool Assert(Fact);   // Fact =Headless horn clause
    bool Assert(Fact_Relation);

    bool AssertRule(Rule);  //Rule Horn clause
    bool AssertRule(Rule_Relation);

    bool DeductiveQuery(Fact);
    relation DeductiveQuery(Rule);

    etc.
}
```

Simple SQL Interface based on EBNF Grammar.

```
Class SQL{
    bool Lex (“SQL Query”);   // Lexical Analysis
    bool Parse(“SQL Query”);
    // Translate SQL to Relational Algebra

    string Translate2RelAlgebra(“SQL Query”);
    etc.
}
```

```
sql_statement = insert | delete | update | select.

insert= INSERT INTO table_name
    [ '(' field_name { ',' field_name } ')' ]
    ( VALUES '(' value_litteral { ',' value_litteral } ')'
    | select_expression ) .

delete= DELETE FROM table_name
   [ WHERE search_condition ].

update= UPDATE table_or_view_name
    SET column_name '=' value_litteral
    { ',' column_name '=' value_litteral }
    [ WHERE search_condition ] .

select= SELECT [ DISTINCT | ALL ]
   ( '*' |   [ '(' field_name { ',' field_name } ')' ] )
   FROM from_table_name { ',' from_table_name }
   [ WHERE search_condition ]
   [ ORDER BY order_list ] .
```