Bharat K. Bhargava

CS541 Course Project, Fall 2008

Designing and Implementing a Mini Relational DBNS

Credit: 20 points

Due Date: Midnight of November 14, 2008 without any penalties.
The last day to submit the program is November 21, 2008 with 2 points penalty per class period the program is late (maximum loss of 10 points).

Project Description:

The mini relational database management system (DBMS) that you are going to design and implement should provide the following functions:

- *An SQL interpreter*:
  The SQL interpreter should be able to parse and execute basic SQL commands (a subset of SQL command set), provide integrity checking, and print out appropriate error messages in case errors are involved in the SQL commands.

- *Persistent data management*:
  Persistent data management requires that data as well as schemas in databases be retained in disks after the program terminates. Thus, any changes to data and databases schemas must eventually be committed to disks.

More specifically, the system should:

1. Allow the creation and the deletion of schemas for the database relations. The schema will contain the name of the relation, the attributes of the relation, and the allowable range of domains for attributes (constraints), as well as information about the ownership and access authorities of the relation, etc.
2. Allow the insertion and the deletion of tuples.
3. Allow the update of values of the attributes of tuples.
4. Allow the retrieval of data that may involve one or more relations.
5. Deny or abort any operations that would cause the schema integrity constraints to be violated.
6. Report lexical, syntactic, or semantic errors involved in SQL commands.
7. Provide access control mechanism that allows data sharing or monopoly (exclusive access) among multiple users. (EXTRA CREDIT).

Project Details

A database, for this project, is defined as a collection of relations or tables and their corresponding schemas. Some commercial DBMSs store the whole database as a single huge

file, which involve complex physical data management. In this project, you can store each relation in a UNIX file for simplicity.

A schema of a relation contains information about this relation, such as the name of the relation, names and types of attributes belonging to this relation, as described earlier. Besides, a schema can contain more dynamic information such as number of tuples in the relation, making the implementation easier. Schemas themselves in a database form a relation and can also be stored in a separate UNIX file. Although this relation can be handled (queried) as any other relations, care has to be taken for doing so. Users are not allowed to update this relation, nor can they access it under certain circumstances. So the system has to provide only limited access to this relation or simply prevent users from accessing it for security and privacy purposes.

Since relations are stored in UNIX files, you must commit every transaction to disk, except "read-only'' transactions. It is also required that each transaction be committed separately. For example, after every transaction that alters the state of the database, you shouldn't simply write out the entire database on the disk since this would be grossly inefficient on the existing hardware. For certain operations, you are encouraged to find a more efficient way to implement them. A smarter way of dealing with tuple insertion, for example, is to reuse the space of previously deleted tuples.

For simplicity, the DBMS system needs to handle only one user at a time and you can assume that you will never have more than you can store in main memory. Also, this project does not require the DBMS system to maintain a log, and you need not worry about recovery in case of hardware or system software errors.

Schemas, Subschemas, and Access Control

A schema (relation table definition) will have a fixed structure with the following information: relation name, attribute name, attribute type, and constraint. An example of the schema could be:

Relation	Attribute Name	Attribute Type	Constraint
test	student_name	char(15)	student_name != ""
test	student_id	iInt	student_id !=0
test	student_age	int	(student_age > 10) AND (student_age < 70)

The possible types for an attribute field are:

int	Integer
char(str_len)	Character string of length str_len

decimal	Real number

An attribute name must be alphanumeric (starting with an alphabetic character) has a length of at most 256 characters, and must accept the underscore character (_).

The field constraint is a Boolean predicate consisting of valid attribute names, the logical operators (AND, OR), the relational operators (=, !=, >, <. >=, <=), the arithmetic operators (*, /, +, –), and constants of equivalent types. It is used mainly for integrity checking.

For example, the following definitions are valid:

Relation	Attribute Name	Attribute Type	Constraint
test	s1	char(15)	S1 != “”
test	s2	decimal	(s2 * 1.0) > 10.2
test	s3	int	(s3 > 10) OR s3 = 0 and s2 < 5)
test	s4	char(20)	(s4 != “TST”) and (s4 != “”) and (s4 != “xyzzy”
test	s5	char(3)	(s5 = “T1”) OR (s5 = “T2”) or (s5 = “T3”)

Your DBMS should be able to handle the concept of users and subschemas. For example, if the following relation is defined:

Relation	Attribute Name	Attribute Type	Constraint
students	name	char(20)	Name != “”
students	ssn	char(9)	(ssn != “”) and (ssn != “000000000”
students	phone	char(7)	
students	gpa	decimal	(gpa >= 0.0) and (gpa <= 4.0)

then, an employee of the registrar’s office should be able to access all fields. However, a student using the database would have the following view of the relation:

Relation	Attribute Name	Attribute Type	Constraint
students	name	char(20)	Name != “”

students	phone	char(7)

Hence, only limited access to the database is granted to that user. Also, for this project, you are not required to handle passwords for the users.

There is no default way in designing access control mechanism in this project. You can design you own as long as it makes sense. One alternative way could be the following:

> you can have a DBA in your system, whose is responsible for creating new users, deleting old users, granting or cancelling access authorities, etc. Each ordinary user can have his own right, of course, to enable or disable access permissions to data owned by him

which is similar to that in a UNIX system.

Data Definition and Data Manipulation Commands

You are required to process the following data definition and data manipulation commands:

Create Table:	A description can be found in section 7.1.1 of the textbook.
Drop Table:	A description of this command can be found in section 7.1.2 of the textbook.
Select:	A description of this command can be found in sections 7.2.1 to 7.2.3 of the textbook.
Insert:	A description of this command can be found in section 7.3.1 of the textbook. Note however, that NULL is not permitted for any attribute. Also, your program needs not support the nested SQL statement shown on example U3B on section 7.3.1 required.
Delete:	A description of this command can be found in section 7.3.2 of the textbook. Again, the nested SQL statements are not required, as illustrated by example U4C.
Update:	A description of this command can be found in section 7.3.3 of the textbook. Again, the nested SQL statements illustrated by example U6 are not required.

Besides the above mentioned commands, a *help* command, with the syntax shown below, should give the user the syntax of all the commands supported by the database, a list of all relations defined in the database and their schemas.

HELP TABLES -- Prints out the list of tables defined.
HELP DESCRIBE T_NAME -- Describes the schema of table T_NAME.
HELP CMD -- Describes the built-in command CMD.

If you decide to use LEX and YACC for this project, which we strongly encourage, you should first construct a set of BNF rules describing the syntactic structure of the SQL commands that must be supported by your DBMS system.

Remember that you do not need to support nested statements and that you are working with a subset of the language.

Program Education

There are two execution modes: interactive mode and batch mode. When running in the interactive mode, your program should always generate prompt for the next command; when running in batch mode, your program will read sequences of SQL commands from a UNIX file. To achieve this, your SQL interpreter must allow inputs only from the standard input so that UNIX file redirection can be used to process SQL command sequences from files.

Appropriate error messages should be displayed at all times in a human readable form (meaning if you can't tell immediately why you got the error, it's not human readable). Points will be deducted for errors that are not handled correctly.

Development Guideline

This is a project requiring considerable effort in time and learning. You should start early. We recommend that you develop your software as follows:

- Learn about LEX and YACC and design your EBNF rules (SQL grammar rules). You are free to use FLEX and BISON, as long as compatibility is guaranteed. There are on-line manual pages and documents on mentor.cc for these tools. Some of them are listed here:

- For LEX, see file /usr/doc/lex;
- For YAC, see files /usr/doc/yacc/*;
- For FLEX, see its manual pages;
- For BISON, see files /usr/local/gnu/info/bison.*
- Build your command line interpreter.
- Design and implement utilities for managing schemas. Leave out the integrity constraints but plan on including them later.
- Design and implement utilities that will allow you to manipulate the data in the main memory.
- Completely finish your project in main memory.
- Use the expression evaluator (for logical and arithmetic expressions) developed for WHERE clause, to enforce integrity constraints.
- Worry about committing to disk. Read UNIX manual pages regarding file I/O operations in C if you are not familiar with them.

However, the above is just a guideline. For some, persistent data management can be considered and implemented at the same time as SQL command interpreter.

Grading System

Partial grades will be given for completion of the following tasks:

Description:		Share of Credits
- Creation and deletion of schemas and subschemas, access control are - divided as follows:		5 pts
-- Correct handling of “Create Table” command	2 pts	
-- Correct handling of “Drop Table” command	1 pt	
-- Integrity constraints checking	1 pt	
-- Access Control	1 pt	
- Database manipulation is divided as follows: -		10 pts
-- Correct handling of “Select” command	4 pts	
-- Correct handling of “Insert” command	2 pts	
-- Correct handling of “Delete” command	2 pts	
-- Correct handling of “Update” command	2 pts	
- Appropriate handling of errors, unexpected input and incorrect queries -		1 pt
- Modularity, documentation, and programming style, including: -		3 pts
-- Correct handling of “help” commands	1 pt	

20 pts

TOTAL PROGRAM GRADE (as a % of your final grade)

Points will be deducted under the following circumstances:

- SQL commands are not implemented or correctly handled.
- Main memory databases fail to commit to disk.
- Databases do not support incremental commitment (i.e., the entire database is committed for every transaction).
- Integrity constraints of schemas are not enforced.
- Memories are not deallocated after they are no longer in use.
- Submission is late (10% taken off for each class).

More details about grading will be announced in the first week of November 2008.

Submitting your project

Once you are done with your project, you must submit the following:

1. A design document that clearly addresses the design considerations of your project. In this design document, you must discuss your choice of data structures, commitment methods, etc.
2. A Make file that will compile and assemble your project. The Make files must not use absolute path names.
3. All your source files.
4. An executable version of your program. To facilitate grading, let's make a convention: choose the ultimate executable file name to be *dbrun*.
5. A test file demonstrating the features of your program (including how to handle errors).

You must submit these electronically with the command:

```
turnin –c cs541 –p SQL f1.c f2.c …
```

Also, a hardcopy of your design document must be turned in to the class.

GOOD LUCK WITH YOUR PROJECT!