OCAML RDBMS
Final Project Report
CS 421 - Summer 2006

Sal Razzaq
netid: razzaq

Version 1.0: July 30, 2006

Contents

1 Abstract

This project models an In-Memory Relational Database Management System using facilities of a Functional Language. In particular, we use a flavor of ML (Meta Language) called OCAML (Objective Caml - http://caml.inria.fr/) to implement a working Relational Database Management System.

2 Description

Relational Database Management Systems (RDBMS) store data as a bag of tuples. A bag of tuples is called a *Table* in RDBMS and each tuple in the table is called a *Row*.

Relational Database Management Systems use SQL, a query language, as an interface to the data. By issuing declarative statements written in SQL, users can retrieve data they desire. One of the key SQL phrases is the WHERE clause in a SELECT statement. The WHERE clause allows a user to filter rows of data contained in a table based on a predicate.

OCAML supports user-defined data types, tuples as well as lists of tuples. These features of OCAML can be used as building blocks for a RDBMS. OCAML also provides elegant ways of creating and applying High Order Functions (HOFs) for performing tasks such as filtering or folding based on a predicate.

For an implementation of a RDBMS in OCAML, we also need to interpret SQL. OCAML provides rich functionality to lex (ocamllex tool) and parse (ocamlyacc tool) statements.

3 System Components

1. A lexer to tokenize SQL statements
2. A parser to synthesize tokens into an Abstract Syntax Tree

3. An interpreter to execute the SQL Commands. We call this THE SQL INTERFACE.

4. A library of OCAML functions to support execution of SQL Commands. We call this THE NATIVE API.

4 Initial Scope

The project set out to implement the following subset of SQL commands:

- CREATE DATABASE
- CREATE TABLE
- DROP TABLE
- INSERT
- DELETE
- UPDATE
- SELECT

In order to cover all the basic SQL statements needed for an RDBMS to be functional, limited number of options were planned to be implemented for each of the above mentioned SQL constructs. For example, the initial implementation will support aggregation (SUM, AVG) to demonstrate folding recursion but SELECT will be limited to operations on a single table in a given SQL statement.

5 Completed Functionality

We were able to successfully implement the above features in the Native API layer including predicates to filter rows and aggregate functions such as AVG and SUM (see `mldbmstestfn.ml`).

We were also able to implement the SQL Interface for the following SQL Statements:

- CREATE DATABASE
- CREATE TABLE
- DROP TABLE
- SELECT

6 Lessons learned

We have implemented an end to end model of a relational database using a functional language as follows:

- We defined OCAML data types to store relational data
- We implemented OCAML functions to manipulate these data types following the semantics of operations in a typical relational database
- We implemented a lexer and parser to accept declarative SQL Statements

7 Areas for further development

This project can be further enhanced by adding support for additional relational database operations such as:

- JOIN of multiple tables
- Nested SQL Statements
- Support of additional SQL Statements in the SQL Interface

8 Demonstration

1. Place all files in the code package in a folder of your choice
2. Change directory to the folder you chose in step 1
3. Start OCAML
4. Execute the following command in OCAML: `#use "mldbms.ml";;`

You are now ready to use the OCAML-RDBMS!

Code in `mldbmstestfn.ml` creates a `SampleDatabase` that is bound to `mldbms` OCAML variable. Code in `mldbmstestfn.ml` also creates `Student` and `Grade` tables and adds rows to these tables using the Native API implemented in `mldbmsdm.ml`.

You may now try the following SQL Statements to query these tables. Note the pretty-printed table that appears right after the prompt.

- `mlsql dbms "SELECT * FROM Student";;`
- `mlsql dbms "SELECT * FROM Grade";;`

You may also try the following SQL Management Statements:

- `let newdbms = mlsqldmo "CREATE DATABASE newdatabase";;`
- `let newdbms = mlsql newdbms "CREATE TABLE MyTable";;`
- `mlsql newdbms "SELECT * FROM MyTable";;`
- `let newdbms = mlsql newdbms "DROP TABLE MyTable";;`
- `mlsql newdbms "SELECT * FROM MyTable";;`

`mldbmstestfn.ml` in the code exercises the the Native API layer implemented in `mldbmsdm.ml`.

9 Tests

We have implemented tests for the Native API layer. The tests can be found in `mldbmstestfn.ml`.

10 Members

Sal Razzaq (netid: razzaq)

11 Website

http://ml-dbms.sourceforge.net/

12 Code Listing

12.1 File List

File	Description
mldbmstypes.ml	Defines types and records to hold DBMS user data
mldbmsdm.ml	The Native API - OCAML code for DBMS manipulation
mldbmstestfn.ml	Tests for the Native API
sqllex.mll	Lex definition
sqllex.ml	Generated Lex file
sqlparse.mly	Grammar definition
sqlparse.ml	Generated Parser
sqlexpressions.ml	Types for AST created by the parser
mlsql.ml	The SQL Interface - Interpreter for the AST
mldbms.ml	Main file
readme.txt	Summary of code package

12.2 mldbmstypes.ml

```
(*** DATA TYPE DEFINITIONS **)

(* Supported SQL Column Types *)
type mltablecoltypes = INT | CHAR of int;;
```

```
(* Record to store a column definition *)
type mltablecoldef = {defcolname : string; coltype :
   mltablecoltypes};;

(* Variant type to store a value *)
type mltablecoldatavalue = Int of int
                         | Char of string;;

(* Record type to store a value for a given column in
   table *)
type mltablecoldata = {datacolname : string; value :
   mltablecoldatavalue};;

(* Record type to model a row in a table *)
type mltablerowdata = {data : mltablecoldata list};;

(* Record type to maintain a list of tables in dbms *)
type mltable = {tablename : string; coldeflist :
   mltablecoldef list; rowdatalist : mltablerowdata
   list};;

(* Record type to represent the entire database *)
type mldbms = { dbmsname : string; tablelist : mltable
   list};;

(* Relational (comparison) operators *)
type relationalop = OpEq | OpNe ;;

(* Data Aggregation Types *)
type aggregationTypes = Avg | Sum;;

(* Predicate structure *)
type predicate = { preddata : mltablecoldata; predoper
   : relationalop };;
```

12.3 mldbmsdm.ml

```
(*** DATABASE MANAGEMENT FUNCTIONS **)

(* HELPER FUNCTIONS *)
let rec substituteTable tablelist table acc = match
   tablelist with
| [] -> acc
| t::ts -> if (t.tablename = table.tablename) then
             substituteTable ts table (table :: acc)
           else
             substituteTable ts table (t :: acc);;

let rec getTableAux tablelist tablename = match
   tablelist with
| [] -> None
| t::ts -> if (t.tablename = tablename) then
             Some t
           else
             getTableAux ts tablename;;

let rec getTable tablelist tablename =
    let t = getTableAux tablelist tablename in
    match t with
    | None -> failwith ("table not found:" ^ tablename)
    | Some table -> table;;


let substituteRowDatalist (table : mltable)
   newrowdatalist =
{
  table with rowdatalist = newrowdatalist
};;

let rec length lst = match lst with
| [] -> 0
| x::xs -> 1 + (length xs);;


let int_of_coldatavalue v = match v with
```

```
| Int n -> n
| _ -> 0;;

let padStringRight s =
      Printf.sprintf "%-25s" s ;;

(* FILTER/PREDICATE FUN *)
let compareData d1 d2 op = match op with
| OpEq -> d1 = d2
| OpNe -> d1 <> d2;;

let rec evalPred p rowdata = match rowdata with
| [] -> false   (* No matching columns in row *)
| d::ds -> if (d.datacolname = p.preddata.datacolname)
   then
            compareData d.value p.preddata.value p.
               predoper
          else
            evalPred p ds;;

let rec filterrowdatalistaux pfn pred rowdatalist acc =
     match rowdatalist with
| [] -> acc
| r::rs -> if (pfn pred r.data) then (
   filterrowdatalistaux pfn pred rs (acc @ [r]))
             else (filterrowdatalistaux pfn pred rs acc)
                ;;

let filterrowdatalist pred rowdatalist =
   filterrowdatalistaux evalPred pred rowdatalist [];;

let filtertable pred table = pred table.rowdatalist;;

(* DELETE *)

let negatepred pred = match pred.predoper with
| OpEq -> {pred with predoper=OpNe}
| OpNe -> {pred with predoper=OpEq};;
```

```
let deleterowsaux pred table = filterrowdatalist pred
   table.rowdatalist;;
let deleterow dbms pred tablename =
  let table = getTable dbms.tablelist tablename in
  let table = substituteRowDatalist table (
     deleterowsaux (negatepred pred) table) in
  let newtablelist = substituteTable dbms.tablelist
     table [] in
  {dbms with tablelist = newtablelist};;

(* UPDATE *)

let rec substitutevalueaux2 coldata values = match
   values with
| [] -> coldata
| v::vs -> if (v.datacolname = coldata.datacolname)
   then
              {coldata with value = v.value}
          else
              substitutevalueaux2 coldata vs;;

let rec substitutevalueaux1 rowdata values  = match
   rowdata with
| [] -> []
| rd::rds -> (substitutevalueaux2 rd values) ::
   substitutevalueaux1 rds values;;

let substitutevalueaux row values  = {row with data = (
   substitutevalueaux1 row.data values)};;

let substitutevaluespred pred row values  =
  if (evalPred pred row.data) then
    substitutevalueaux row values
   else
    row;;
```

```
let rec updaterowsaux rowdatalist pred values =
match rowdatalist with
| [] -> []
| r::rs -> (substitutevaluespred pred r values)::
   updaterowsaux rs pred values;;

let updaterows dbms tablename pred values =
  let table = getTable dbms.tablelist tablename in
  let table = substituteRowDatalist table (
     updaterowsaux table.rowdatalist pred values) in
  let newtablelist = substituteTable dbms.tablelist
     table [] in
  {dbms with tablelist = newtablelist};;


(* ROW DATA RETRIEVAL *)
let str_of_mltablecoldatavalue v = match v with
| Int i -> string_of_int i
| Char s -> s;;

let getRowColValue coldef rowdatalistdata =
  let rec getRowColValueAux coldef rowdatalistdata acc
     = match rowdatalistdata with
  | [] -> acc
  | r::rs -> if (r.datacolname = coldef.defcolname)
     then r.value else (getRowColValueAux coldef rs acc
     )
  in
  getRowColValueAux coldef rowdatalistdata (Char "NULL
     ");;


(* DISPLAY FUNCTIONS *)

(* COLUMNS *)
let rec printColDeflist coldeflist  = match coldeflist
   with
| [] -> print_newline()
```

```
| c::cs -> print_string (padStringRight c.defcolname);
   printColDeflist cs;;


(* ROWS *)
let rec printRowDetail coldeflist rowData = match
   coldeflist with
| []-> print_newline()
| c::cs ->  print_string ( padStringRight (
   str_of_mltablecoldatavalue (getRowColValue c rowData
   )));
    printRowDetail cs rowData;;

let rec printRowlist coldeflist rowdatalist = match
   rowdatalist with
| [] -> ()
| r::rs -> printRowDetail coldeflist r.data;
   printRowlist coldeflist rs;;

let printRowCount rowdatalist =
  print_string ( (string_of_int (length rowdatalist)) ^
      " rows displayed\n");;

let printRowslist coldeflist rowdatalist =
  printColDeflist coldeflist;
  print_string
     ("-------------------------------------------------------------------------")
     ; print_newline();
  printRowlist coldeflist rowdatalist; print_newline();
  printRowCount rowdatalist;;

let printTable dbms tablename =
  let table = getTable dbms.tablelist tablename in
  print_newline();
  print_string ("TABLE: " ^ tablename); print_newline()
     ;
  printRowslist table.coldeflist table.rowdatalist;
  print_newline(); dbms;;
```

```
(* DATABASE CREATION *)
let createDB databasename =
  { dbmsname = databasename; tablelist = [] };;



(* TABLE CREATION *)
let createTable dbms tablename =
  {
    dbms with tablelist = dbms.tablelist @ [{tablename
       = tablename; coldeflist = []; rowdatalist = []
       }]
  };;

let rec dropTableAux tablelist tablename acc = match
   tablelist with
| [] -> acc
| x::xs -> if (x.tablename = tablename) then
              acc @ xs
           else
             x::(dropTableAux xs tablename acc);;

let dropTable dbms tablename =
       let newtablelist = dropTableAux dbms.tablelist
          tablename []
       in
       {dbms with tablelist = newtablelist};;



(* COLUMN ADDITION TO TABLE *)
let addColumn table coldef =
    {table with coldeflist = table.coldeflist @ [coldef
       ]};;
```

```
let addColumnToTable dbms tablename colDef =
{
      dbms with tablelist = (substituteTable dbms.
         tablelist (addColumn (getTable dbms.tablelist
         tablename) colDef) [])
};;



(* INSERT OF A DATAROW TO A TABLE *)
let addDataRowColData table rowData =
    {table with rowdatalist = table.rowdatalist @ [
       rowData]};;

let addDataRowToTable dbms tablename rowData =
{
      dbms with tablelist = (substituteTable dbms.
         tablelist (addDataRowColData (getTable dbms.
         tablelist tablename) rowData) [])
};;



(* AGGREGATES *)
let rec getIntValue rowdata colname = match rowdata
   with
| [] -> 0
| rc::rcs -> if (rc.datacolname = colname) then (
   int_of_coldatavalue rc.value) else getIntValue rcs
   colname
;;

let printaggrtitle aggr =
  match aggr with
  | Avg -> print_string ("AVERAGE ")
  | Sum -> print_string ("SUM ");;

let rec aggrsum rowdatalist colname aggr = match
   rowdatalist with
```

```
| [] -> 0
| rd::rds -> (getIntValue rd.data colname) + (aggrsum
   rds colname aggr)
    ;;

let rec aggrmain rowdatalist colname aggr = match aggr
   with
  | Avg -> (aggrsum rowdatalist colname aggr) / (length
      rowdatalist)
  | Sum -> aggrsum rowdatalist colname aggr;;

let aggregateaux rowdatalist colname aggr =
  print_string(">>>>>  ");
  printaggrtitle aggr;
  print_string(colname); print_string(": ");
  print_string(string_of_int(aggrmain rowdatalist
     colname aggr));
  print_string("   <<<<<");
  print_newline(); print_newline();;


let aggregate dbms tablename colname aggr =
  let table = getTable dbms.tablelist tablename in
  aggregateaux table.rowdatalist colname aggr;
  dbms;;
```

12.4 mldbmstestfn.ml

```
let dbms = createDB "SampleDatabase";;
let dbms = createTable dbms "Student";;
let dbms = createTable dbms "Grade";;
let dbms = createTable dbms "DumpIt";;
let dbms = dropTable dbms "DumpIt";;
let dbms = addColumnToTable dbms "Student" {defcolname
   ="LastName"; coltype= CHAR 30};;
let dbms = addColumnToTable dbms "Student" {defcolname
   ="FirstName"; coltype= CHAR 30};;
```

```
let dbms = addColumnToTable dbms "Grade" {defcolname="
   LastName"; coltype= CHAR 30};;
let dbms = addColumnToTable dbms "Grade" {defcolname="
   Score"; coltype= INT};;

let dbms = addDataRowToTable dbms "Student"  { data = [
          {datacolname = "LastName"; value = Char "
             Smith"};
          {datacolname = "FirstName"; value = Char "Joe
             "}
          ]};;

let dbms = addDataRowToTable dbms "Student"  { data = [
          {datacolname = "LastName"; value = Char "
             Smith"};
          {datacolname = "FirstName"; value = Char "
             Tubby"}
          ]};;


let dbms = addDataRowToTable dbms "Student"  { data = [
          {datacolname = "LastName"; value = Char "Doe
             "};
          {datacolname = "FirstName"; value = Char "
             Jane"}
          ]};;


let dbms = addDataRowToTable dbms "Grade"  { data = [
          {datacolname = "LastName"; value = Char "
             Smith"};
          {datacolname = "Score"; value = Int 88}
          ]};;

let dbms = addDataRowToTable dbms "Grade"  { data = [
          {datacolname = "LastName"; value = Char "Doe
             "};
          {datacolname = "Score"; value = Int 94}
```

```
            ]};;

printTable dbms "Student";;
printTable dbms "Grade";;

(* Is Smith *)
let avalue = {datacolname = "LastName"; value = Char "
   Smith"};;
let predicate = { preddata = avalue; predoper = OpEq
   };;
let filteredRowlist = filterrowdatalist predicate (
   getTable dbms.tablelist "Student").rowdatalist;;
printRowslist  (getTable dbms.tablelist "Student").
   coldeflist filteredRowlist;;


(* Is NOT Smith *)
let avalue = {datacolname = "LastName"; value = Char "
   Smith"};;
let predicate = { preddata = avalue; predoper = OpNe
   };;
let filteredRowlist = filterrowdatalist predicate (
   getTable dbms.tablelist "Student").rowdatalist;;
printRowslist  (getTable dbms.tablelist "Student").
   coldeflist filteredRowlist;;

(* Is NOT Doe *)
let avalue = {datacolname = "LastName"; value = Char "
   Doe"};;
let predicate = { preddata = avalue; predoper = OpNe
   };;
let filteredRowlist = filterrowdatalist predicate (
   getTable dbms.tablelist "Student").rowdatalist;;
printRowslist  (getTable dbms.tablelist "Student").
   coldeflist filteredRowlist;;


(* DELETE - Tubby *)
```

```
let avalue = {datacolname = "FirstName"; value = Char "
   Tubby"};;
(* NOTE: we want everyone except Tubby *)
let predicate = { preddata = avalue; predoper = OpEq
   };;
let dbms = deleterow dbms predicate "Student";;
printTable dbms "Student";;


(* UPDATE *)
let predvalue = {datacolname = "FirstName"; value =
   Char "Jane"};;
let predicate = { preddata = predvalue; predoper = OpEq
    };;
let values = [{datacolname = "LastName"; value = Char "
   Dog"}];;
let dbms = updaterows dbms "Student" predicate values;;
printTable dbms "Student";;


(* SUM *)
let dbms = aggregate dbms "Grade" "Score" Sum;;
let dbms = aggregate dbms "Grade" "Score" Avg;;
```

12.5 sqllex.mll

```
{

}

let numeric = ['0' - '9']
let letter =['a' - 'z' 'A' - 'Z']

rule sqltoken = parse
  | "SELECT"                {SELECT_token }
  | "UPDATE"                {UPDATE_token }
  | "DELETE"                {DELETE_token }
  | "INSERT"                {INSERT_token }
```

```
  | "DROP"                    {DROP_token }
  | "CREATE"                  {CREATE_token }
  | "TABLE"                   {TABLE_token }
  | "DATABASE"                {DATABASE_token }
  | "FROM"                    {FROM_token }
  | "WHERE"                   {WHERE_token }

  | "VALUES"                  {VALUES_token }
  | "SUM"                     {SUM_token }
  | "AVG"                     {AVG_token }
  | "<>"                      {NOTEQUAL_token }
  | "="                       {EQUAL_token }
  | "("                       {LParen_token }
  | ")"                       {RParen_token }
  | ","                       {Comma_token }
  | "*"                       {Star_token }
  | letter (letter|numeric|"_")* as id
                              {Id_token id }
  | (numeric)* as i           {Integer_token (int_of_string
     i) }
  | ''' (([^'"''\\'] | '\\'_)* as s) ''' {
    StringLiteral_token (s) }
  | eof                       {EOF_token }

  | [' ' '\t']                {sqltoken lexbuf}

{ let lextest s = sqltoken (Lexing.from_string s)}
```

12.6 sqlparse.mly

```
%{
  #use "sqlexpressions.ml";;
%}

%token SELECT_token
%token UPDATE_token
%token DELETE_token
%token INSERT_token
```

```
%token DROP_token
%token CREATE_token
%token TABLE_token
%token DATABASE_token

%token FROM_token
%token WHERE_token
%token VALUES_token
%token SUM_token
%token AVG_token
%token NOTEQUAL_token
%token EQUAL_token
%token LParen_token
%token RParen_token
%token Comma_token
%token Star_token
%token <string> Id_token
%token <int> Integer_token
%token <string> StringLiteral_token


%token EOF_token
%start main
%type <stmt> main
%%

main:
| sqlstmt EOF_token                { $1 }

sqlstmt:
| CREATE_token DATABASE_token Id_token          {
   CreateDB $3 }
| CREATE_token TABLE_token Id_token             {
   CreateTable $3 }
| DROP_token TABLE_token Id_token               {
   DropTable $3 }
| SELECT_token objectlist FROM_token Id_token   {
   Select ($2, $4)}
```

```
objectlist:
| Star_token                                  { [] }
| Id_token                                    { [$1] }
| Id_token Comma_token  objectlist            { [$1] @ $3 }
```

12.7 sqlexpressions.ml

```
type stmt =
    CreateDB of string
    |
    Select of string list * string
    |
    CreateTable of string
    |
    DropTable of string
    ;;
```

12.8 mlsql.ml

```
let sqlexpr s = main sqltoken (Lexing.from_string s);;

let mlsqldmo s = match (sqlexpr s) with
| CreateDB dbName -> createDB dbName
| _ -> createDB "default";;


let mlsql dbms s = match (sqlexpr s) with
| CreateTable tablename -> createTable dbms tablename
| DropTable tablename -> dropTable dbms tablename
| Select (collist, tablename) -> printTable dbms
   tablename
| _ -> createDB "temp";;
```

12.9 mldbms.ml

```
#use "mldbmstypes.ml"
#use "mldbmsdm.ml"
```

```
#use "mldbmstestfn.ml";;
#use "sqlparse.ml";;
#use "sqllex.ml";;
#use "mlsql.ml";;
```

12.10 readme.txt

```
====== FILELIST ======

mldbmstypes.ml: Defines types and records to hold DBMS
    user data

mldbmsdm.ml: The Native API - OCAML functions to
    manipulate the DBMS user data

mldbmstestfn.ml: Tests for the Native API

sqllex.mll: Lex definition

sqllex.ml:  Generated Lex file

sqlparse.mly: Grammar definition

sqlparse.ml: Generated Parser

sqlexpressions.ml: Types for AST created by the parser

mlsql.ml: The SQL Interface - Interpreter for the AST

mldbms.ml: Main file

readme.txt: this file


====== HOW TO EXECUTE ======

1. Place all the above files in a folder
```

2. Change directory to the folder you chose in step 1

3. Start OCAML

4. Execute the following command in OCAML:
#use "mldbms.ml";;

You are now ready to use the OCAML-RDBMS

====== SAMPLE EXECUTION ======

- Code in "mldbmstestfn.ml" creates a "SampleDatabase" that
is bound to "mldbms" variable

- Code in "mldbmstestfn.ml" also creates some tables, adds
rows using the Native API in mldbmsdm.ml

- You may now try the following SQL Statements to query these tables:

```
mlsql dbms "SELECT * FROM Student";;
mlsql dbms "SELECT * FROM Grade";;
```

Note: the pretty-printed table that appears right after the prompt

- You may also try the following SQL Management Statements:

```
let newdbms = mlsqldmo "CREATE DATABASE newdatabase";;
let newdbms = mlsql newdbms "CREATE TABLE MyTable";;
mlsql newdbms "SELECT * FROM MyTable";;
let newdbms = mlsql newdbms "DROP TABLE MyTable";;
mlsql newdbms "SELECT * FROM MyTable";;
```

Sal Razzaq
7/30/2006
netid: razzaq
CS421