Department of Electrical and Computer Engineering
University of Puerto Rico
Mayaguez Campus

ICOM 6005 - Database Management Systems
Project 1 - Implementation of Selection and Projection Operators

Due: 12:00PM, October 12, 2001

1 Overview

In this project, you will work with several of the modules that typically handle the execution of single-table selection and projection operations in a DBMS. Specifically, you will implement the code that:

- reads schema information from a file.
- generates a structure to hold a tuple that is based on the schema information.
- reads tuples from a data file and stores them in an associated structure for that type of tuple.
- evaluates a selection condition on a tuple.
- projects one or more columns from a tuple, after a selection operation (if any) has been passed by that tuple.
- performs some maintenance and clean up tasks related with the execution of the relational operations.

Rather than building all this code from scratch, you will be given several routines that you will use as the building blocks to implement your solution. These routines will typically perform low-level tasks such a memory allocation, file allocation, low-level data access, and several others. It is your job to figure out how and where to use these routines in your implementation.

2 Reading the Relational Schema

The schema for a given relation will be stored in a text file whose name will consist of the relation name and the extension `.sf`. For example, if the relation is called `students`, then the schema file will be called `students.sf`. In this project, all the schema files will be stored in a sub-directory called `schemas`. The convention for how a schema will be stored in a schema file as follows:

1. The name of the relation.
2. The number of tuples in the relation.

3. The name of the data file in which the tuples for the relation are stored.

4. The number of attributes in the relation.

5. A sequence of entries with information about each attribute:

   - The name of the attribute.
   - A numeric code indicating the data type of the attribute.
   - The size in bytes of the data type of the attribute.

An example of this configuration for the file `students.sf` is a follows:

```
students 5 databases/students.sto 4 rowid 2 4 name 1 10 age 2 4 addr 1 10
```

As part of your tasks you will write functions that take care of reading the schema information from the file, and storing it in a structure called the the `relation_info`. This structure is declared in file `schema.h`, and has the following organization:

```
// Maximum size in bytes of an attribute or relation name
#define MAX_NAME 128

// Attribute information structure
typedef struct attr_info{
  char attr_name[MAX_NAME]; // variable-sized attribute name string
  int  attr_type; // numeric code for the attribute data type
  int  type_size; // size in bytes of the attribute data type
}attr_info;

// Relation information structure
typedef struct relation_info {
  char rel_name[MAX_NAME]; // name of the relation
  int  rel_card; // cardinality of the relation
  char rel_file[MAX_NAME]; // file in which the relation is stored
  int  num_attrs; // number of attributes in the relation
  attr_info *attributes; // array for the information about attributes
}relation_info;
```

You have to implement the function `get_relation_info()` in file `schema.c`, which should take care of reading the schema information from the data file, and storing it in an instance of `relation_info`. The code in `schema.c` provides some of the basic functionality to manipulate the data in the schema file. Also, file `schema.h` contains the declaration of several other types, constants and function prototypes that you should be aware of at the moment of implementing your solution.

Using, the schema information you then need to create tuples that follow that schema. In addition, if you are given a list of projections to be performed, you need to create tuples that follow the schema of these projection. The structure of a tuple, defined in file `tuple.h`, is as follows:

```
// Attribute in the tuple
typedef struct atrribute{
  // type of this attribute
  int attr_type;
  // the value of this attribute
  // use attr_type to figure out which of these field is the one to access
  union value {
    int_type int_val; // integer value
    double_type double_val; // double value
    char_type char_val; // character string value
  }attr_val;
}attribute;

// Tuple
typedef struct tuple{
  int num_attrs; // number of attributes in the tuple
  attribute *attr_list;  // array for the attributes
}tuple;
```

To create the proper instance of a tuple for the given relation or set of projections, you need to implement two versions of the function `make_tuple()`, which should be defined in file `tuple.c`. One version of the function should build a tuple out of a `relation_info()` structure, and the second one should use a `proj_attr_list`, which is also defined in file `schema.h`. The corresponding interface for each of these functions is as follow:

```
tuple *make_tuple(relation_info *rel_info);

tuple *make_tuple(proj_attr_list *proj_list);
```

3 Scan Iterator

The major module that you need to develop is the `scan_iterator`. This module contains the implementation of a scan iterator, which takes cares of:

- reading tuples from a data file.
- applying a selection condition on the tuples.
- projecting one or more columns.
- returning the resulting tuple

The interface for the scan iterator is as follows:

```
scan_iterator *make_scan_iterator(relation_info *rel_info,
  proj_attr_list  *proj_list,
  bool_expr *expr, tuple *cons);

int open_scan_iterator(scan_iterator *s_iter);

tuple *next_scan_iterator(scan_iterator *s_iter, int& status);

int close_scan_iterator(scan_iterator *s_iter);
```

In this interface, each function has a well-defined task that you must implement. The task of each function is:

1. `make_scan_iterator()` - used to create a new scan iterator from a relation schema information, a list of projections, a selection condition, and set of constants (stored in a tuple) that appear in the expressions of a query.

2. `open_scan_iterator()` - used to open the data file from which the tuples for a given relation are drawn.

3. `next_scan_iterator()` - used to return the next tuple that is result of evaluating the selection condition and projections associated with the iterator.

4. `close_scan_iterator()` - used to close the data file from which the tuples were being read.

You only need to worry about implementing the functions 2-4. In this project, the data file where tuples for a given relation are stored will always be stored in a subdirectory called `databases`. The name of the data file for a given relation should be obtained from the schema information.

4 Distribution Files

You will be given a tar file containing the following modules:

- `Makefile` - the make file to compile the project.
- `error.h` - codes for the various return values of the functions used in this project.
- `schema.h` - declaration of schema structures and several other data types.
- `schema.c` - implementation of the schema related functions. YOU NEED TO ADD CODE HERE.
- `tuple.h` - declaration of the attribute and tuple structures.
- `tuple.c` - implementation of the operations related with the creating and reading tuples. YOU NEED TO ADD CODE HERE.

- `predicate.h` - declaration of logical operators used to evaluate a selection condition.
- `predicate.c` - implementation of the logical operators.
- `iterator.h` - declaration of the iterator types and related constants.
- `scan_iterator.h` - declaration of the interface for the scan iterator.
- `scan_iterator.c` - implementation of the scan_iterator interface. YOU NEED TO ADD CODE HERE.
- `schema_gen.c` - program that generates schema files interactively.
- `test_schema.c` - program used to test the schema files generated by `schema_gen.c`.
- `data_gen.c` - program that generates a data file for a given schema file, based on the user input.
- `data_rdr.c` - program that reads a data file to print the tuples stored in it.
- `query1.c` - Test query program number 1. **Your program must successfully pass this file for it to be considered a running program**.
- `query2.c` - Test query program number 2.
- `query3.c` - Test query program number 3.

You should use for account on the machine `icarus.ece.uprm.edu` to implement and test your code. You code must successfully run when tested with program `query1` for it to be considered a running program. If your code also runs well with programs `query2` and `query3` you will get at least 70 points.

5 Due Date

Project is due by 12:00 PM on October 12, 2001. You will get the instructions on how to submit your program at a later time.