

CSI 402 – Spring 2011
Programming Assignment III

Administrative Information

- **Deadline:** 11 PM, Wednesday, Mar. 23, 2011.
Cutoff: 11 PM, Friday, Mar. 25, 2011.
- The program must have two or more C source files.
- All the files (C source files, header files and the `makefile`) must be submitted together using the `turnin-csi402` command.
- README file
 `~csi402/public/prog3/prog3.README`
will be available by 10 PM on Tuesday, Mar. 8, 2011.
- The README file will contain information regarding `turnin-csi402` and additional specifications for the `makefile`.

Project Description

Goal: To implement a relational database system that supports simple retrieval queries.

Weightage: 10%

Total Points: 100 (Correctness: 85, Str. & doc: 15).

Relational Databases:

- Data stored as relations (tables).
- The database may have many relations.

Relational Databases (continued)

Example: Relation Roster.

Name	Major	Totcr	Majcr
----	-----	-----	-----
Smith, Bob	PSY	57	30
Woods, Jane	CSI	64	39
Baker, Norma	PSY	57	48

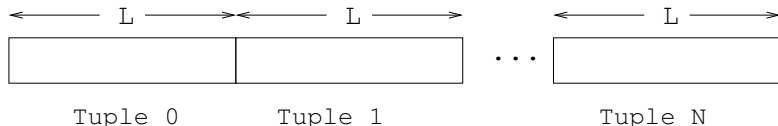
Notes:

- Name, Major, Totcr, Majcr: Attributes.
- Each row of data: Tuple.

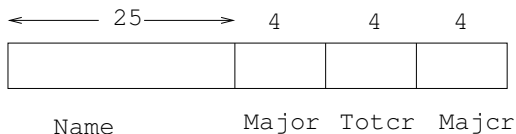
- The **schema** for each relation gives:
 - Name of each attribute.
 - Type of each attribute ('I' or 'S').
 - Length (in bytes) for each attribute.
- Length of each tuple (in bytes) = Sum of the lengths of the attributes.

How tuples are stored:

- In a binary file.
- Each tuple is of the same length (which can be determined from the schema information).



Example – Tuple for the Roster Relation:



Length of each tuple = 37 bytes.

Notes:

- Length of each integer attribute = 4 bytes.
- Length of each string attribute includes the byte for '\0'.

Operations on Relations

(a) Counting the number of tuples:

Example Query:

```
count Roster
```

Result:

3

(b) Projection on an attribute:

- Outputs the values found in the specified column.
- Duplicates must be removed.

Operations on Relations (continued)

Example I:

project	Roster	Major
---------	--------	-------

Result:

PSY
CSI

Example II:

project	Roster	Totcr
---------	--------	-------

Result:

57
64

Operations on Relations (continued)

(c) Selection under a condition:

- Outputs each tuple that satisfies the condition.

Example I:

```
select Roster Major == "PSY"
```

Result:

Smith, Bob	PSY	57	30
Baker, Norma	PSY	57	48

Selection under a condition (continued)

Example II:

```
select  Roster  Totcr  >  60
```

Result:

```
Woods, Jane  CSI  64  39
```

Unix Command Line Details

```
% p3 configfile queryfile
```

- p3: Executable version of your program.
- *configfile*:
 - Specifies the name of a text file (called the **configuration file**).
 - First line: No. of relations in the database.
 - Each subsequent line: Name of one relation.
 - Implicitly specifies the name of the schema file and the data file for each relation.

Note: For an example of a configuration file, see page 4 of the handout.

- *queryfile*:
 - Also a text file.
 - Each line has exactly one command. (In addition to `count`, `project` and `select`, there is also the `quit` command.)

Assumptions:

- The configuration, schema and data files won't contain any errors.
- Each command in the query file will be one of `count`, `project`, `select` or `quit`.
- Each command will have the correct number of arguments.

Errors to be Detected

- Errors to be detected for each command in the query file are discussed in the handout.
- When a command in the query file contains an error, your program should print an appropriate error message to `stdout` and process the next command. **The program should stop only when the quit command is seen.**
- Unix command line errors (e.g. wrong number of command line arguments, one of the specified files can't be opened) should be written to `stderr` and the program should stop right away.

Important Notes

- All responses to queries (including errors detected in queries) are written to `stdout`. Be sure to use `fflush(stdout)` after each call to `printf`.
- Binary (data) files made available on `itsunix` may **not** work on other machines.

Watch Out!!

Your program processes a sequence of commands. If it exits abnormally (due to an error such as a segmentation fault) in the middle of the command sequence, you won't get any credit for the remaining commands in that sequence.

Additional Suggestions

- Implement one command at a time.
- For processing each command (except quit) have a separate C source file that contains all the functions needed to process that command.
- Review the use of standard C functions such as `fgets`, `strcmp`, `strtok` and `fseek`. (You will also be using `fread`.)