Prof. Dr. Stefan Brass
Institut für Informatik
MLU Halle-Wittenberg

October 25, 2017

Databases IIB: DBMS-Implementation — Exercise Sheet 3 —

Part a) to i) will be discussed in class, you only have to submit Part j) and k). Please send your solutions to the instructor via EMail: `brass@informatik.uni-halle.de` (with "`dbi17`" in the subject line). The official deadline is November 1, 12:00.

Some Feedback on Homework 1

- Please make sure that the program can be compiled on Linux as well as on Windows. E.g. include files like `stdafx.h` and `conio.h` do not exist under Linux.
- If one executes a console program under Windows not inside a shell, a console is created for the program, but immediately closed when the program finishes. So one has to wait for the user pressing "Return" (only under Windows). My program contains the following code which reads characters until a line end:

  ```
  #ifdef VER_PRESS_RETURN
          cout << "\n(Please press Return/Enter to finish)\n";
          for(char c = ' '; !cin.fail() && !cin.eof() && c!='\n'; )
                  cin.get(c);
  #endif
  ```

  If one wants this version, the symbol `VER_PRESS_RETURN` must be defined (e.g. at the top, or in an include file `ver.h`):

  ```
  #define VER_PRESS_RETURN
  ```

  One can also do this on the command line with the option `-DVER_PRESS_RETURN` when calling the `g++` compiler. If one wants code that is automatically chosen on Windows, one can try

  ```
  #if defined(_MSC_VER) || defined(_WINDOWS_) || defined(WIN32)
  ```

  Some check also for `_WIN32`.

  Note that `system("pause");` is *not* a good solution: It creates a new process with a shell (command prompt) to execute this command, which is only available under Windows, and the shell might even find a different command than you expected! The function `system` should be used if great care (or better not at all) if you want to write safe programs.

- Please avoid lines longer than 80 characters. In my editor, the tabulator size is 8 characters (classical UNIX standard).
- Note that in the loop searching for divisors `i` of the possible prime number `n`, the condition `i < n/2` does stop too early if one checks 4. You can use the condition `i * i <= n`, tough. I have also seen `i < n/2+1`, which is correct, but can be written simpler as `i <= n/2`.
- Note that it is ok if a loop body is executed 0 times. Some students had a special condition for the input 2, which was not necessary.
- An advantage of `for` loops over `while` loops is that intialization, loop condition, and the update of the loop variable are all together in one place. In my view, a loop with a single loop variable that has all of these parts should be written as a `for`-loop, not as a `while`-loop. (Personally, I use `for`-loops also when some of the control parts are missing.) If the loop variable is not used outside the loop, you can declare it in the `for`. This reduction of the scope of the variable is not possible with `while`.
- Your code looks more professionally if you avoid comparisons with boolean constants, e.g. if `isPrim` is a variable declared as `bool`, instead of `if(isPrim == true)` simply write `if(isPrim)`.
- Note that if a function/method is declared as returning `int`, then every possible execution path should end a `return`-statement specifying a value. Otherwise, there is no error message (it is formally legal in C++), but the returned value is undefined (not initialized) otherwise. Switch on warnings (`-Wall`) to be informed in this case.

  However, there is one exception for the function `main`: It may terminate without a `return` and will return `0` in this case.

Repetition Questions

a) What would you answer to the following questions in an oral exam?

- Sketch data dictionary tables for the database schema, i.e. tables, columns, and constraints (keys, foreign keys). If you know the basic structure of the Oracle tables, you could use that (of course, you should concentrate on the most important columns). But you could also design your own tables (the concepts are important, you should not learn the Oracle tables by heart).
- What do the Oracle data types `NUMBER` and `VARCHAR2` mean? These are not part of the SQL standard.
- Table and column names are not case-sensitive in most SQL systems (sometimes this depends on DBMS options or settings specified at database creation time). Explain the situation when accessing the Oracle data dictionary.
- Why is it useful to specify constraint names in the SQL `CREATE TABLE` statement?

- Is it possible to store a short explanation about tables and columns inside the database in Oracle? If you use a system that has no special command for this, what could you do?
- Name some features that make constraints in real systems like Oracle more complicated than the simple view of the first database course.
- Is it possible to see the queries that define the contents of views in Oracle?
- Explain "Object Privileges" (the standard SQL access rights). Give an example for a `GRANT` command in SQL. Explain the clause `WITH GRANT OPTION`. What should be the effect of the `REVOKE` command?
- How could a data dictionary table for managing object privileges look like? Again, if you know the Oracle table, you could use that. However, only the main information content is important.
- What problem is solved with "system privileges" in Oracle? What would be an alternative solution, used in other systems?
- What is the purpose of "roles" in Oracle? How can they be used?

b) Check your knowledge of C++ by answering these questions (knowledge of C++ is not directly checked in the exam, but you will need it for getting enough homework points):

- You use a static variable in the class ("class attribute") to count the number of objects of the class. This variable is incremented in the constructor and decremented in the destructor. After running some tests, you find that the number has become negative. What happened? How can this problem be avoided?
- Suppose you define a class without constructor. What does the constructor do that the C++ compiler generates in this case? Which attributes will be initialized?
- Suppose you declare a class `C` in file `c.h` that has an attribute of class `D` (declared in `d.h`). How can you simplify the usage of your class, by making sure that the user does not have include `d.h`? Would the situation change if the attribute is only a pointer to class `D`?
- Suppose you wrote `d.h` yourself and want to make sure that there is no problem if `d.h` is included two times. How is this done?
- Suppose that C is a class. What is the difference between the following functions?
  - `void f(C x)`
  - `void f(C* x)`
  - `void f(const C* x)`
  - `void f(C& x)`
  - `void f(const C& x)`

  How would one call these functions for an object declared as "`C obj;`"?

In-Class Exercises

c) Write an SQL query to the data dictionary that shows the names of the tablespaces (use e.g. `DBA_TABLESPACES`). Where are the data files stored for these tablespaces (use `DBA_DATA_FILES`)?

d) Create a normal user account (not a DBA) for yourself (please choose your last name, maybe abbreviated). Define default tablespace `USERS` and temporary tablespace `TEMP`. Define a quota for the default tablespace. Note that quotas for temporary tablespaces are no longer supported. Assign the role "`CONNECT`" to the new user.

e) Log in as the new user. You can change your account in SQL*Plus with

```
connect username
```

Try to access data in some table, e.g. select tables from `DICT` where the table name contains the substring `OBJECT`. Then try to create a table. Does this work? If not, switch back to account `SYSTEM` and grant the `RESOURCE` role.

f) Please download the following file which creates the Oracle example tables:

[http://www.informatik.uni-halle.de/~brass/dbi17/empdept.sql]

Execute it while you are logged into SQL*Plus as your own account with the following command

```
@empdept
```

g) Now print name and type of all database objects you own (not only tables, but also indexes).

h) Create a small test table with one column and one row. Grant read access to this test table to your neighbour in class. Ask him/her to use an SQL query to access the data item in your table. Conversely, access his/her table.

Note that if you do not have read access to the table, Oracle will tell you that the table does not exist (even if it actually does exist). In this way, no information about the existence of tables can be gained from the error message.

i) Which system privileges does the role `CONNECT` contain? Note that the corresponding data dictionary view shows only the information about roles you have, even if you are logged in as DBA (e.g., `SYSTEM`).

Homework Exercises

j) Write a query to the Oracle data dictionary that shows the keys of all tables that you own (primary keys and alternative/unique keys). The output should have four columns:

- Table name
- Name of the key
- Number of the column within the key
- Column name

Sort the output in a good way.

k) Download the file

[http://www.informatik.uni-halle.de/˜brass/dbi17/homeworks.txt]

It contains the data of the example table:

HOMEWORKS			
FIRST_NAME	LAST_NAME	EXERCISE_NO	POINTS
Ann	Smith	1	10
Ann	Smith	2	8
Michael	Jones	1	9
Michael	Jones	2	9
Richard	Turner	1	5

The file contains one line per table row, and the columns are separated with a colon "`:`", e.g. the first line is

```
Ann:Smith:1:10
```

Please change your `main` program from Homework 2 such that it

- opens the file,
- reads all lines in the file,
- parses each line and creates an object of your homeworks class,
- stores the object in an array (store at most 100 objects).
- Then write a small query program that prints students (first and last name) with the maximal number of points for Homework 1.

Of course, you can define auxiliary functions or even a class (e.g. for the relation). Note that if you use a class only in one source file, you do not need separate `.h` and `.cpp`-files for the class. Simply define the class in the source file where it is needed.

Working with Files in C++

Here is some information about accessing files:

- There are different ways to access files. The high level C++ class for file input streams is `std::ifstream`. You must include `<iostream>` and `<fstream>` for the declaration of this class. You can find the documentation of this class e.g. here:

  [http://www.cplusplus.com/reference/fstream/ifstream/]

  An alternative documentation source is

  [http://en.cppreference.com/w/cpp/io/basic_ifstream]

  (This really documents a template, of which `ifstream` is an instance.)

- You open the file by calling the `open` method of the `ifstream` object.

  [http://www.cplusplus.com/reference/fstream/ifstream/open/]

  Its first parameter is the file name, e.g. `homeworks.txt`. The optional second parameter are mode bits. If the file is a binary file, i.e. the system should not replace Windows line ends (CR-LF: `"\r\n"`) by the UNIX version (LF: `'\n'`), one can open it as follows (with an object `File` of class `ifstream`):

  ```
  File.open(filename, std::ios::in|std::ios::binary);
  ```

  However, this is a text file. Note that if you use an IDE, it might execute the program in a subdirectory of your project directory (such as `Debug` for the debug configuration in Visual C++). One solution would be to specify the full path name (do not forget to escape "\" as "\\" in the string constant).

- The method `open` has no return value and does not throw exceptions (unless specifically requested). Actually, one can program in C++ without using exceptions at all. One can check `File.fail()` or `File.is_open()` after the call to `open` in order to check whether the file was successfully opened.

  If you want to access the error message of the operating system, the numeric code of the last error is stored in the global variable `errno` declared by including `<errno.h>`. This can be translated to a string with the function `strerror` declared in `<string.h>`, i.e. `strerror(errno)` should work (at least under Linux). If there are problems, you can also look for the functions `strerror_s` and `perror`.

- One method to read a line from the file into a character array is

  `getline(char *buf, std::streamsize bufsize)`.

The type std::streamsize is system-dependent, it can e.g. be int or a type for 64-bit integers (long long or int64_t defined in stdint.h). An int value for the second parameter will be ok (it is automatically converted to a larger type if needed). E.g. one declares the array "char buf[80];" and calls "File.getline(buf, 80);".

The method reads characters from the input stream until (1) the end of file was reached, or (2) the end of line was reached, or (3) no more characters fit into the buffer, i.e. bufsize-1 characters were read. In each case, the string in the buffer will be terminated with a null byte. The newline '\n' is not stored in the buffer.

The function returns the stream object itself (which might be helpful for sequences of getline-operations). The number of characters read in the last such operation can be accessed with the method gcount(). If the end of file was reached during the operation, the method eof() will return true. If the line was too long for the buffer, fail() will return true. If one wants to continue reading, one must reset the state bits of the stream object with the method clear(). (If bad() should be true, the stream is corrupted, and one cannot expect that one can continue reading.)

There is also a version of getline with an additional argument for the delimiter character (actually, there is only one version, but '\n' is the default value for the delimiter). If you prefer, you can use this with the delimiter ':' for the first fields. The delimiter will be read from the file, but not stored in the array.

- Streams can be automatically converted to boolean values, and return the negation of the fail() function in this case. I.e. "while(File.getline(buf, 80))" is possible, although the function returns a reference to the stream object File. C++ permits that the programmer defines type conversion functions for own classes. (In this case, the conversion was actually to void *, but the language treats a null pointer as false, and all other pointers as true.) Also the operator ! is defined for streams, thus if(!File.getline(buf, 80)) can be used to print an error message.
- If you want to convert a string to an integer, you can e.g. use the library functions strtol ("string to long integer") for standard character pointers or stoi for string objects. Other alternatives include sscanf (formatted input in C) and streams reading from strings: istringstream.

  The function strtol declared in <cstdlib> has three parameters: The buffer containing the string to be converted (const char*), an address of a pointer which will be set to point to the next character after the number (char **), and the base of the number representation (10 for decimal numbers). If you declare a variable "char *end;" you can write &end for the second parameter. By using this pointer, one can continue reading further data in the string after the number. The function returns the converted value as a long. It might be necessary to explicitly write a cast to int. The function returns 0 if the input did not start with a digit (possibly after skipping whitespace). Since this is a valid value, one really should check whether *end == '\0' afterwards (i.e. nothing remains in the string buffer that was not converted). It is also possible to check for overflows, see e.g.

  [https://stackoverflow.com/questions/14176123/correct-usage-of-strtol]

  The function sscanf (declared in <cstdio>) can convert several data items based on a format string. If one wants to read a single integer, the call would be

```
int success = sscanf(buf, "%d", &n);
```

  Here buf contains the input, it has type const char *, and one can pass e.g. the name of an char[] array. The variable n must have type int, and is set to the conversion result. The function returns the number of successfully converted format elements, i.e. success will be 1 in the positive case and 0 if the buffer did not contain a valid number. Note that the conversion is successfull if the buffer starts with a number, there might be additional characters not used, e.g. "123abc" as first argument would set n to 123 and return 1.

  If you want to check whether a character c is a digit, you can use isdigit(c) defined in <cctype>.

- At the end, you should close the file with close(). This would give you the option to check for errors (the method returns no value, but one can call fail() afterwards). At least for output files, it would be possible that there is still output in the internal buffer of the stream object, and that an error occurs while this is written to the file.

  However, you do not have to close the file, because the destructor of the ifstream class automatically closes the file if it is still open. This technique to manage the allocation of a resource with a local variable is called RAII ("resource acquisition is initialization").

- It is also possible to use lower level interfaces for working with files. C had file pointers (type FILE*) defined in <cstdio>.
- There is also a direct interface to the UNIX system calls open, read, write, and close. These functions work with file descriptors (small integers) to identify files. Similar functions are available under Windows (there the numbers are called file handles), but there might be differences in the details and the necessary include files are different.

  On Linux/UNIX, read and write are declared in in unistd.h:

  [http://pubs.opengroup.org/onlinepubs/7908799/xsh/unistd.h.html]

  open is declared in fcntl.h:

  [http://pubs.opengroup.org/onlinepubs/7908799/xsh/fcntl.h.html]

  It might be necessary to include also sys/types.h, sys/stat.h and sys/uio.h.

  For Visual Studio on Windows, use the include file io.h and read e.g.

  [https://msdn.microsoft.com/en-us/library/z0kc8e3z(v=vs.140).aspx]