Paper 3F6: Software Engineering and Design

Relational Databases, UI Design and Software Management

Examples Paper 4

Straightforward questions are marked †
Tripos standard (but not necessarily Tripos length) questions are marked ∗

Relational Databases

1. ∗ The Engineering department has decided to use a database to keep track of which practicals students have completed. When a practical has been signed off, the student is entered into the database along with the title of the practical completed. The design of the table 'completed=(student,lab)' is:

completed	
student	lab
Murphy	Networking over RS232
Murphy	Wireless networking
Murphy	Image processing on FPGAs
Libby	Wireless networking
Libby	Image processing on FPGAs
Libby	FPGA design in verilog
Cook	FPGA design in verilog
Cook	Welded steel structures

The course is very modular and students can choose any modules. In order to pass a module, students must complete all labs in that module. If only some labs within a module are taken, then those labs will not contribute to the student's final mark.

Quite a number of students end up taking labs which never contribute to the final mark. Therefore a column has been added to the database to indicate whether or not the module associated with a particular lab has been completed. The new design of the table is as follows:

completed		
student	lab	module_completed
Murphy	Networking over RS232	yes
Murphy	Wireless networking	yes
Murphy	Image processing on FPGAs	no
Libby	Wireless networking	no
Libby	Image processing on FPGAs	yes
Libby	FPGA design in verilog	yes
Cook	FPGA design in verilog	no
Cook	Welded steel structures	no

The director of studies can now find which students have labs which do not contribute to their final mark with a query such as:

$\Pi_{\text{student}}(\sigma_{\text{module\_completed}=\text{“no”}}(\text{completed}))$

When a student completes all labs in a module, the “module_completed” attribute is changed for that student. The C++ code for signing off a student is below. Assume that the `do_sql()` function performs SQL queries on the relevant database and returns a `std::vector` of tuples.

```
void insert_completed_lab(string student, string lab)
{
  //Do query like INSERT INTO students VALUES ('Gibson', 'Welded steel structures', 'no')
  do_sql("INSERT INTO students VALUES ('" + student + "', '" + lab + "', 'no')");

  vector<tuple> r;  //Returned tuples from an SQL query

  //Search for the two networking modules for student
  //The two modules are "Networking over RS232" and "Wireless Networking"
  r = do_sql("SELECT * FROM students WHERE student='" + student + "' AND
              lab IN ('Networking over RS232', 'Wireless Networking')");

  //If the student has done all of them, then update the module_completed attribute
  if(r.size() == 2)
    do_sql("UPDATE course SET module_completed='yes' WHERE student='" + student + "' AND
              lab IN ('Networking over RS232', 'Wireless Networking')");


  //Do the same for the FPGA course, "Image processing on FPGAs" and "FPGA design in verilog"
  r = do_sql("SELECT * FROM students WHERE student='" + student + "' AND
              lab IN ('Image processing on FPGAs', 'FPGA design in verilog')");
  if(r.size() == 2)
    do_sql("UPDATE course SET module_completed='yes' WHERE student='" + student + "' AND
              lab IN ('Image processing on FPGAs', 'FPGA design in verilog')");

  //Do the same for the structures course, "Welded steel structures" and "Earth banks"
  r = do_sql("SELECT * FROM students WHERE student='" + student + "' AND
              lab IN ('Welded steel structures', 'Earth banks')");
  if(r.size() == 2)
    do_sql("UPDATE course SET module_completed='yes' WHERE student='" + student + "' AND
              lab IN ('Welded steel structures', 'Earth banks')");
}
```

(a) Is the ‘completed’ relation well normalized? You may wish to consider whether any data is duplicated and if it is possible to insert inconsistent information into the database. Also discuss the design in terms of how easy it is to extract useful information from the database and how easy it is to alter modules without introducing bugs.

(b) Design a well normalized database schema for the task which does not rely on a complex function for inserting entries.

(c) Write down in relational algebra and SQL the query to find which modules students are enrolled in.

(d) Hence write down in relational algebra and SQL the query to find the list of labs students will need to have done in order to complete their enrolled modules. You will need to look up how to use brackets in SQL.

(e) Hence write down in relational algebra and SQL the query to find which students have labs which do not contribute to their final mark. The query should also list the offending labs.

(f) Databases tend to deal with selects of selects rather inefficiently. Manipulate the expression in the previous answer so that it only needs one select statement.

Note that if you join a table with itself the result has ambiguous attribute names. In relational algebra, the $\rho_{\text{newname}}(\text{relation})$ operator can be used to temporarily rename tables in order to avoid ambiguity. In SQL, the AS clause is used, for example, to get all possible pairings of students including self pairings:

```
SELECT DISTINCT completed.student, tmp.student FROM completed, completed AS tmp;
```

(g) Write a query to find all students who have not done a lab on the networking course.

(h) Write a query to find all students who have completed the FPGA course.

User Interface Design

2. You are a member of a team designing a web interface for an internet banking service where each client has two accounts: a current account and a savings account. Assuming that the user has logged-in and passed all security checks, write a specification for each of the following use cases:

   (a) transfer £100 from the current account to the savings account

   (b) pay a bill of £36.50 to BT plc, customer number EA3482828

   (c) close all accounts and transfer any outstanding balance to an account at another bank.

3. A prototype of the internet banking system described in Q2 has been implemented:

   (a) describe how you would conduct a usability test?

   (b) what metrics would you use and how would you measure them?

Software Management

4. $*$ As a member of a code review team you are supplied with the information shown in Fig. 1.

   (a) What is the difference between a *code inspection* and a *walk-through*?

   (b) List four distinct cases that should be considered when reviewing the operation of the `insert` function.

(c) Perform a walk-through of the `insert` function and write a brief report describing your findings.

(d) If the integer numbers i were known to lie in the range $0 < i < N$, suggest how the `insert` function could be simplified?

"The supplied C code fragment is intended to implement a linked list of integers stored in ascending order. Each element of the list is a `struct` of type `Item` holding the integer value, a pointer `pred` to the previous element, if any, and a pointer `succ` to the succeeding element, if any. The variable `head` points to the first element in the list, and `tail` points to the last element. Initially, both `head` and `tail` are `NULL`. The function `insert` is intended to insert its argument `x` into the list. If `x` is already in the list, `insert` should do nothing."

```
struct Item {
   int value;          // the integer value
   Item *succ;         // succeeding value
   Item *pred;         // preceding value
};
Item *head, *tail;     // Head and Tail of List

// Create a new item and return a pointer to it
Item* NewItem(int value, Item *succ, Item *pred);

// Insert x into linked list
void insert(int x)
{
   Item *p = head , *q;
   if(!p) {
      head = tail = NewItem(x,NULL,NULL);
   }else if (x>tail->value){
      tail->succ = NewItem(x,tail,NULL);
      tail = tail->succ;
   }else{
      while (x > p->value) p++;
      if (x==p->value) return;
      q = NewItem(x,p,p->pred);
      p->pred = q;
   }
}
```

Figure 1: The insert function

5. The C++ function below claims to efficiently locate a specific integer value in a large array of integers, provided that the array elements are stored in ascending order.

```
// Binary search routine:
// the elements of a[i] i=1..N are stored in sort order.
//   return index i of array element a[i] == x
//   return 0 if not found
int binary_search(int *a, int x, int N)
{
   int mid,low = 1, high = N;
   a[0] = x;
   do {
      mid = (low+high) / 2;
      if (low>high)
         mid = 0;
      else if (a[mid]<x)
         low = mid + 1;
      else
         high = mid - 1;
   } while (a[mid] != x);
   return mid;
}
```

Assuming that N=100000, list the specific cases that should be covered when testing this routine and design a test harness suitable for use as a regression test.

Ed Rosten
March 2011
Tom Drummond
February 2010