Database Your Way, LLC

"Databases by Design"

databaseyourway.com

Case Study #3

A moderately sized, moderately complex logistics database

Situation:

The organization concerned is an Army National Guard Regional Training Institute (RTI). This is a self-contained facility. All personnel, operational and logistical support, as well as the instructor staff are integrated into one organization, all in support of training soldiers.

Scope:

This RTI trains over 4200 Soldiers a year, through conduct of 21 different courses of instruction. Most of the courses conduct multiple classes, for a total of, on average, 97 classes per year.

The Dining Facility utilized by the students and staff is run by a civilian contractor. The Soldiers enrolled in the various courses are in different pay categories, basically either on active duty, or not. Active duty Soldiers (who get a monthly food allowance) pay as they go. Reserve component Soldiers (who don't get a monthly food allowance) are not required to pay as they go. And there exceptions to both rules (of course), and RTI staff may fall into either category, depending on their status.

The organization (the RTI) is required to make payment to the Dining Facility for all personnel in its charge who do not pay as they go. This requires a system of checks and balances to ensure the count of these personnel is accurate. This is currently accomplished by having a staff member at the entry to the Dining Facility, with a paper form. Non-paying diners present a 'meal card' as proof of their status, sign a line on the form, and tell the staff member (quaintly known as the 'headcount') the last four digits of their social security number. The Headcount then writes that number down next to the diner's name. A separate form is used for each course and for staff.

The RTI wanted to develop a more automated system, with the goal of reducing the chance of error at the 'Headcount' station and provide a more accurate accounting of the number of non-paying diners. A need exists for a method of identifying in which category each student and staff member belong as they enter the dining facility, verifying the diner is indeed a member (staff or student) of the RTI, and printing the results on the official form.

mark@databaseyourway.com 803-438-0372

Case Study #3, *cont.*

Using the basic principles of good database design, we looked at who needed data, what data they needed, and what form that data would take, getting input from all who would be impacted by this project. Additionally, we needed to know where the data would come from, and how the data would be input.

In reading this case study, keep this in mind. On the face of it, this database seems very simple. It not only appears that way, it is very simple to operate, requiring very little effort on the operator's part. However, to achieve that simplicity required a great deal of customized coding running in the background.

The organization's logistics section had the primary interest in tracking this data. The basic data that needed to be tracked was:

- Soldier's name
- Soldier's social security number (at least the last four digits)
- The Soldier's status (staff or student, and if student, which course)
- Soldier's pay category (as pertains to dining status – pay-as-you-go, or not)

The primary reports identified were:

- The 'headcount' form (one for each course, by meal)
- Count of all customers, subdivided into various categories
  (e.g., RTI paying, RTI non-paying, non-RTI paying, total diners)

Knowing what data needed to be tracked, we built tables to hold the related information for the staff. This would be the only somewhat static table in the database. Since a database previously created for this organization contained most of the needed student information, we used linked tables to eliminate redundancy and reduce the quantity of new data that would otherwise have to be entered manually at the start of each new class.

Aside from the basic personnel information tables, we needed a table to contain the data of the Soldiers who actually came through the dining facility during a given meal time. Knowing what product would be needed as an end result, primarily the 'headcount' form (by course, by meal), guided the design of the rest of the database.

During the established dining time, students of multiple courses could conceivably go through the line, which further complicates things. This led to constructing three separate tables for each course, one for each meal. The information for each meal could have been contained in a single table. So why build so many duplicates, differentiated only by course and meal? Good question – glad you asked.

Case Study #3, *cont.*

The answer is simplicity. Not in database design, but in operator input. Ease of use is always (well, *should* always be) a consideration. In this case, data input would be limited to swiping a card through a bar code reader. The operator (in this case, the 'headcount') would only need to monitor the screen as Soldiers swiped their card, and at the end of the meal period, click on a button to print out the appropriate 'headcount' form.

This concept also results in impartiality. The 'headcount' is not required to rely on the Soldier for any information as to their status, and is not allowed any input as to the Soldier's status. We need to make it very clear, here, that no aspersions are being cast on the honesty or integrity of either party. The goal is to improve the integrity of the system, and relieve either party of any responsibility. Remember, the amount of money being paid by the government to a civilian contractor is determined by the number and status of the Soldiers going through the chow line. It's all about checks and balances.

Now, let's diverge to output. The 'headcount' form is the primary record of non-pay-as-you-go Soldiers. The official form used for this purpose was reconstructed as a report in Access. Generally, in Access a report is designed with integrated data fields based on a specific record in a specific table in the database. A report will print out for each record that meets the qualifying criteria. In this case, we needed a report that would contain multiple records listed on a single report. While this is not really a problem for lists or rosters, it is not quite so simple to get this effect on a pre-formatted form. We don't need to get technical here, but the point is, a series of subreports had to be created as well. This is a good example of how end-requirements help determine the basic database design, which sometimes requires thinking outside the box.

We've identified the goals and the end product, and designed the tables and reports to meet the goal. To get data from one table to another, to get data to populate a report, to change (update) data, we need queries. Queries are the engines of the database, and can vary tremendously in complexity, depending on what you're trying to do. Some of the things we used queries for were to select specific data, update specific data, and perform calculations. Some queries you want to run singly or independently, some you may want to run in sequence through the use of macros.

We made use of a number of macros to simplify the operator's task. In this vein, we also developed a number of modules to further automate some of the processes. And as mentioned earlier, there is also a great deal of customized, behind-the-scenes coding in this database, designed to run as various Event Procedures, requiring no operator action.

Case Study #3, *cont.*

With the basic database designed, forms were designed to facilitate local data entry. In maintaining the concept of "user friendly", we kept the interface to simply two forms. Only one is an actual data entry form – the interface for accepting the bar code swipe. Some of the customized coding underlying this form provides pop-up messages to the operator each time a card is swiped, identifying the status of the Soldier. The second form is to accommodate the printing of reports, utilizing command buttons to automate the process. These command buttons actually run a series of queries and macros, resulting in printing of the required report, all of which is completely transparent to the operator.

The last thing developed was the switchboard interface. The main switchboard uses buttons to list the basic function of the database. Pick a function, click it, and you either go to that function or you're taken to another switchboard page to refine your options. This makes the database easy to navigate. A user who is only concerned with one aspect can navigate directly to that function without wading through all possible functions. The main switchboard page also incorporates the organization's logo.

After running a trial of the program, we experienced problems with maintaining seamless network connectivity. As mentioned earlier, the student data tables were linked to a separate database. At the time, the laptop used for data entry was connected to the network wirelessly.

Each time a card was swiped, the Unique Identifier from the bar code was compared to data pulled by a query which combined data from all the student data tables. Though the database worked as required, by dropping in and out of the network (if only for a second or two) the intermittent lack of data continually interrupted the process. Our conclusion was that the interference was caused by considerable amounts of RF interference in the Dining Facility. One option was to request a hardwired connection be installed. Another option was to get away from using linked tables.

It was decided to limit the requirement for network access as much as possible. To this end, new tables were designed to hold the minimum student data required by the database to function. The back-end of the student information database we had previously linked to is actually housed on a SQL Server. To avoid duplication of effort (entering data already in another database), we built an export package to update this database with information from the actual student database. This required a one-time network connection at the beginning of the class to refresh the data in the Dining Facility database. With all data resident on the single laptop in the Dining Facility, network connectivity became a non-issue, and the program worked fluidly.

Case Study #3, *cont.*

Summary

Database design starts with communication. The people who know the nature of the company and what data needs to be tracked, the people who will (or might) need some output from the data, and the people who will be putting the data into the finished database need to be involved with the database designer in the process.

We identified the nature of the organization and the needs of the organization (consulting all the players involved). We vetted the data we needed to capture by looking at the end products desired, and made any necessary adjustments to the data needs. We designed tables to hold that data, remembering the principles of efficiency, isolation, and relationship (that's where the primary keys were identified, allowing this to be a relational database). We considered use of linked tables to reduce redundancy and duplication of effort, but settled on use of an export package to reduce reliance on the network.

We designed reports that gave the end users a product that was useful to them, and developed the queries that would not only populate those reports with data, but make data transfer and modification relatively simple.

We designed macros to automate as many processes as possible, and used them extensively to generate the required reports.

We then designed forms as a user interface, not only making the database user-friendly, but attractive as well. The final form designed was the switchboard, allowing simple navigation to the desired function, with a resultant ease of use.

Case Study #3, *cont.*

A summary of the organization's situational statistics are as follows:

• Courses of instruction	21 (This database is currently designed to Accommodate 15 of those, with plans for expansion.)
• Classes per year (ave.)	97
• Students per year	4200
• Staff	140

The statistics for the database itself are as follows:

• Tables	34
• Reports/subreports	92
• Forms	3
• Queries	61
• Macros	68
• Modules	18
• Switchboard pages	1

In conclusion, a database should be designed with functionality in mind, not complexity. This database was designed accordingly. In this case, however, to meet the organization's need, a fair amount of complexity was required. The end result is a moderate-sized database, serving the organization's requirements in facilitating logistical and fiscal accountability, while keeping the user interface as simple as possible.