

## Re: Simple DBMS in VB

---

*Source:*

<http://www.tech-archive.net/Archive/VB/microsoft.public.vb.general.discussion/2007-02/msg00791.html>

---

- *From:* "Steve" <[sredmyer@xxxxxxxxxx](mailto:sredmyer@xxxxxxxxxx)>
  - *Date:* 9 Feb 2007 12:25:00 -0800
- 

On Feb 9, 2:36 pm, "Larry Serflaten" <[serfla...@xxxxxxxxxxxxxxxxxx](mailto:serfla...@xxxxxxxxxxxxxxxxxx)> wrote:

"Steve" <[sredm...@xxxxxxxxxx](mailto:sredm...@xxxxxxxxxx)> wrote

Here is the current Access DB structure used to hold the data in my app

<...>

Again although this specific bit of data management could be fairly simply implemented with flat files and sequential file access, I think I would like to try to create a more generic and useful data management system for use in this project and possibly future projects where a full-blown DB might be overkill.

Because you want to store variable length data, you leave yourself two options on how to manage the individual items. Either you store the character count and then the data of every item, or you create an index of offsets into the data, or, possibly some combination of those two techniques. A third technique, which for all intents and purposes is similar to the first, is to delimit each item using a delimiter character.

To decide which to use, you'd have to weigh the pros and cons of each method. Run length (or delimited) methods are (comparitively) easy to create and update, but require that you read each item to find any specific item of interest (slow access). Indexing allows faster access, but complicates updating when changes in the data necessitate a re-indexing of the data.

You could pick one method and apply that to your tables and rows, or you could partition out which items would not require updating and index those, while using the run length or delimiting method on data that would be changed often.

## Re: Simple DBMS in VB

You have tables that use foreign keys. They could be eliminated if you use indexing, because their position in the index could be used as their key. Another thing to consider; as soon as you add one character to any data in the file, the whole file has to be re-written to include the new data. So, you have to make the decision, do you want to force the user to have enough memory to keep the entire file in memory (for updating) or (very probably) are you going to provide a means to update the file using only disk accessing (creating a new file, copying in the new data, and erasing the old file).

As an exercise in picking which method(s) to use, you might first determine those tables (and data) that would change (relatively) often and which tables (and data) would not. At first glance it would seem the Genre table and the Album's song lists would not need updating all that often because you could supply the entire list of genres at the start and of course a published album is not very likely to get songs added to it. Everything else would be added as the user builds the database. On the other hand, when you read in an album, you will usually read the entire album's list of songs and not try to search through them on disk.

Since most of your data is being added as the user builds the database, you might make the tables using the run length method. In that case a generic table would take the form:

Table X  
Table size (Long)  
Table Name size (Long)  
Table Name (String)  
Data (Any)

The table size serves as an offset to the next table, so when loading data, you can skip through the file to find any table (similar to using the run length method). If, for example this table held the complete list of genre, you might save that list as a delimited string where upon reading the data you can use the Split command to break them out into an array, which gives you index values you can use in your other tables. Similarly for a table of Artist names.

If this was a table of albums, the data might be further broken out into another generic sort of table:

Album  
Album Entry size (Long)  
Artist Index (Long)  
Title size (Long)  
Title (String)  
Cover Image (Byte array)

Similarly for the other tables, you break the rows out into the fields you need and store them one after another in the file.

## Re: Simple DBMS in VB

Another thing to consider, is the overall size of the file. If you try to store cover images within your file, it may quickly grow in size making updating slower and slower as the new data is copied from file to file. A possible solution to that, is to keep the images in their original file and just store the path to that image, or even, keep all the images in a separate file (for easy transport when needed) and just store the offset and size so you can extract the data, write it to a temp file, and load it in as a picture file.

As was mentioned before, you have lots of decisions to make. Do a bit of planning on paper before you begin such that you build the file and then think about what you would have to do to provide all of the features you want. That may help to sway your decisions on how to store the data, once you see the work involved in updates and retrieval. Good luck, and have fun!

LFS

This sounds very interesting and is very close to what I have been cooking up in my own mind. A quick question comes to mind though, you mention the foreign keys and that I won't need them because the index will take care of it. Can you explain this a bit further. This has been my point of confusion from the beginning. How is it that I no longer need them? How are they handled by the indexes. In short how do I represent these relationships to aid in later queries of the data?

Thanks for taking the time  
Steve

.