Why Don't Scientists Use Databases?

Executive Summary

Most scientists seem to avoid using databases. In this paper we examine twelve of the principal reasons given by scientists for preferring flat files and/or more structured files (such as NetCDF or HDF5). We then explain what database extension tools we have available for addressing some of these concerns, and what further changes are necessary in database management technology and practices in order to address the remaining concerns.

执行总结

许多研究人员避免使用数据库。本文我们将来细查研究人员倾向于用简单文件或者比较有结构的文件（如NetCDF，HDF5）的十二个主要原因，然后我们来解释哪些问题可以用我们已有的数据库扩展工具来解决，以及如何来进一步地改进数据库管理技术来解决剩余的一些问题。

Introduction

Database management systems (DBMSs) are ubiquitous – in the world of business at least. However, we have seen much resistance by scientists to the use of DBMSs. Several authors (see for example [Maier and Vance, 1993], [Gray, 2005], [Buneman, 2002]) have given reasons for this resistance, but we've listened to even more arguments than those stated in these references.

This 11-page document first lays out a dozen opinions that we have heard from scientists who shun databases in favor of traditional files, be they flat files or more structured files such as NetCDF or HDF5. Then, beginning on page 5, we address each of these concerns, criticizing the ones that we feel are unfounded and, for the ones that are legitimate, explaining either what tools we can bring to bear right now or what changes in database management technology and practices are still required before DBMSs can become more scientist-friendly.

Twelve Reasons We Hear From Scientists For Not Using Databases

1) There are no benefits to using a database for the sort of work I do.

The perceived benefits to using a database – point-in-time recovery, transactional editing, concurrent editing, etc. – do not apply in my work environment. I collect data, but usually don't edit it. When I do edit it I am the only one doing so, so sophisticated concurrency control mechanisms just aren't needed.

2) File systems are familiar to me; databases aren't.

I am familiar with file systems and with programs that open, read, and write files, and analyze data. I know how to write these programs in C, Fortran, and Java, but I don't know SQL and I know nothing about database tables, indexes, and stored procedures. Learning a whole new way of dealing with my data will take too long.

3) The data that I have collected in the past is all in files.

I have thousands of files; it would be too much work to load them all into databases; I don't have the time, and there is no apparent payoff for doing it.

4) The new data I work with comes to me in the form of files.

Most of the data that I see in my day-to-day work is already in the form of files. I download logging files from instruments, I generate files by running my simulation / modeling / computation programs, and I download files from data servers such as USGODAE and OCCAM EMODS.

Files are the natural organizational unit for science data. As [Mattman, 2009] points out,

> **Organization of science data** -- science data is organized as file granules, that encapsulate a particular series of observations over space and time into a data unit -- science data is **not** organized in the relational model, as tables, with rows and columns for each row.

5) The programs that I use work with files, not databases.

I already have the programs (and there are a lot of them) that I need to work with the data I have; moving the data to a database will require me to rewrite all my programs. Some of these programs are ones that I did not write – they are "black boxes" to me; all I know is that they do what I want and they work.

6) Files are easy to keep track of.

The file system of an OS is very transparent. It's often easier to do an `ls`, `find`, or `grep` command than craft and execute some arcane SQL SELECT command. Similarly, it is easier to see what's in a file using `cat` or `vi` than to issue a bunch of queries to see what is in a database.

7) Files make handling schema evolution, versioning, and history easier.

If I change my programs, get new instruments that measure things differently, or the site I'm downloading from changes their feeds, then the file formats might change. This is not a problem if I stick with files. However, if I'm putting my data in databases then I have to go to a lot of work in restructuring my database schemas and restructuring all the data in the databases. This "schema evolution" can involve very lengthy unload / reload cycles that I cannot afford.

Tracking the history and versioning of files is easy. If I run a simulation with a different set of parameters, or QA an old file and create a new one, I just give the new file an appropriately different name. If I'm just sticking my files in a database then I can accomplish the same thing by using another attribute to distinguish one file from another. However, if my database design is such that each file gets mapped to multiple tables and columns (which is the norm), then keeping track of this "data evolution" is complicated.

8) Database data types are business - not science - oriented.

Conventional DBMS data types (character, packed decimal) are oriented to business, not science, applications. These data types tend to be very simple, and scalar. I need to do floating point arithmetic and I need to be able to deal with vectors and matrices and complex structures. Maier and Vance [1993] hypothesize that the failure of most DBMS systems to include native support for ordered data collections (i.e., arrays) is a leading reason for scientists not embracing database technology.

9) Database Management Systems are too expensive.

File systems come for free with my computer's operating system. DBMSs, on the other hand, can be quite expensive to purchase. Even if I could afford one now, my ongoing budgets could not afford the high annual support maintenance fees.

10) Databases are too complicated.

Database Management Systems add extra levels of complication to what I do. To use one I need to learn about logical data modeling, learn new languages and interfaces (e.g., SQL, ODBC, JDBC), and hire a database administrator to keep the DBMS running.

11) Complex interrelationships in a file (hierarchies, etc.) may be more difficult to represent in a database.

In his presentation, Peter Buneman [Buneman, 2002] cites an example involving the Swiss-Prot biological library of protein sequences. To convert a file, extracted and downloaded from Swiss-Prot into database tables (without losing any information) required more than 20 tables and a huge query to reconstruct the original export file. The reason for this is that even simple Ascii text files can contain a large amount of implicit structure information. Consider, for example, the following very simple two-line file:

```
A
C-AB
```

Just from the arrangement of characters of this file we might deduce the following:

i) it has two lines,
ii) the first line has one character in it; the second has four,
iii) "C" appears before "A" on the second line,
iv) a character on line 2 ("A") refers to a character seen on a previous line, and
v) the "−" on line 2 might indicate that "A" and "B" may have some relationship to "C" (e.g., they might be children of some sort).

In a good database design these sorts of implicit relationships should be made explicit and no significance can be inferred from the ordering of rows in a database table. Hence the equivalent database representation of the simple file above might be quite complex.

12) DBMS performance is poor.

For the sorts of data that I deal with, database performance can be quite poor. Adding rows to a file is always going to be much faster than adding rows to a database table, due to the extra overhead that DBMSs entail (e.g., logging, database integrity checking, and the extra levels of abstraction [view → table → dbspace → physical file chunk]). Similarly, reading rows from a single file will be faster than doing the complex join that might be needed to reconstruct the file from the corresponding set of database tables (See point 11).

Our Responses …

1) There are no benefits to using a database for the sort of work I do.

It is true that many of the benefits of using a database relate to transactional processing and concurrency control, and for many day-to-day scientific data analysis tasks such benefits just aren't relevant. However, several other benefits might be relevant and hence should not be overlooked:

- DBMS's provide built-in features for aggregating data (e.g., min, max, average, etc.).
- With a DBMS's indexes and the declarative (as opposed to procedural) nature of SQL it is usually much easier and faster to find an item of data in a database than in a file system.
- The organization of data into files imposes physical barriers between data elements; it is harder to operate on data in multiple files than in a single seamless repository such as that provided by a DBMS.

2) File systems are familiar to me; databases aren't.

A fundamental difference between accessing data from files/file systems and accessing data from a database is that *procedural* logic is used for accessing file data whereas *declarative* logic is used for selecting data from a database. In other words, rather than providing a procedural "how to" definition of how to select data from a file, the DBMS programmer just declares the data to be retrieved and leaves the how-to details up to the DBMS server. That being said, however, the DBMS programmer cannot avoid the cost of learning at least some SQL. Depending on the application and the scientific work environment, this cost might not be justified. Some factors that might mitigate the cost are:

- SQL can be quite simple, especially when compared to the procedural code that must be used to accomplish the same thing.
- Database views can be used to simplify SQL code further (although someone with SQL skills must still define the views).
- Depending on the application, the portion of code that has to be rewritten can be relatively small. It's not the case that entire C programs have to be replaced with SQL scripts. Rather, it may be the case that just the small portions that deal with file handling need to be replaced with embedded SQL calls using a 3GL interface such as (Oracle) Pro*C.

3) The data that I have collected in the past is all in files.

This is a legitimate concern. If scientists have thousands of files, then it will take some time to load them all into a database. However, there are still benefits to loading the files into a database – e.g., being able to aggregate data across files, being able to join *data-that-was-in-files* with *other-corporate-data-that-was-in-databases*.

Barrodale Computing Services (BCS) has developed some tools to greatly ease the effort involved in this regard:

i) DaL, a simple, easy-to-use graphical tool, can be used to define a mapping between files and databases. The mapping is defined just once for each type and structure of file and can be used over and over, interactively or in batch, to load all your files of that type and structure.

ii) The Universal File Interface, currently under development, can be used to access data in files as if it were in a database. The data in files appears as if it is in tables, and hence can be joined with other tables and queried using SQL. By making data from possibly several files appear as if they are in a single table, the physical barriers described earlier are removed.

4) The new data I work with comes to me in the form of files.

DaL, described in point 1, can be run in batch mode, making it very easy to integrate database loading into existing processing paths. With regard to the claim made earlier:

> **Organization of science data** -- science data is organized as file granules, that encapsulate a particular series of observations over space and time into a data unit -- science data is **not** organized in the relational model, as tables, with rows and columns for each row.

While much science data may be organized as "file granules," we claim that this is often not the most natural organization. The basic unit of much science data is the "measurement" or "single observation". By grouping such measurements or observations into files one is introducing artificial barriers between measurements – the logic involved in comparing two measurements depends on whether the two measurements are in the same file or different ones. With a database, however, all measurements and observations are treated the same way – as a row in a database table; processing logic is therefore much simpler.

5) The programs that I use work with files, not databases.

Scientists should be able to continue using the programs that they have invested their time in writing and/or using. Moving data from files to a database should not mean that scientists need to abandon their old programs and write or acquire completely new ones. But moving data from files to a database does open up many new processing options. For example, suppose that scientist A has a program, which expects data in file structure X, and he or she wishes to share data with scientist B who has another program, which expects data in file structure Y. Due to the declarative and set-oriented nature of SQL, it is generally much easier for A or B to use SQL to export data from a database in each of the two formats X and Y than it is to write a program to read a structure X file, record by record, and write it out again in structure Y.

There are basically two approaches to dealing with data in a database with programs that work with files:

i) export the data from the database to files, or
ii) provide the illusion to the program that the data, which is in the database, is actually in a file.

As stated above, writing export SQL is one way of performing approach 1. In addition, some database extensions, such as the BCS Grid DataBlade, include features for exporting data in particular file formats (the Grid DataBlade will export data in NetCDF format).

Exporting data to files may not be a viable work-around in some cases, so BCS is currently exploring the second approach as well. In particular, a "Virtual File Interface", currently being designed, will allow data in a database table, view, or row to appear to programs as if it were in a file object (e.g., a NetCDF variable or Ascii file line). With this interface, existing programs will be able to continue to run, with minimal changes, once the data that they need has been moved from files to a database.

6) Files are easy to keep track of.

The claims made earlier

> The file system of an OS is very transparent. It's often easier to do an `ls`, `find`, or `grep` command than craft and execute some arcane SQL SELECT command. Similarly it is easier to see what's in a file using `cat` or `vi` than to issue a bunch of queries to see what is in a database.

are certainly true for small data collections (a *small number* of *small files*), but once the number or size of files becomes large, the situation becomes unwieldy for file tools such as `vi` and `grep`. On the other hand, DBMSs, with their built-in indexing and parallelization, scale much better.

7) Files make handling schema evolution, versioning, and history easier.

The argument made earlier, that using files makes handling schema evolution easier, is specious. Using files allows one to temporarily ignore schema evolution, but the file differences will still need to be addressed later when the older and newer files need to be used together in an analysis. And doing this comparison at a later date may actually be quite hard if metadata describing the old files cannot be found. With databases the pain of schema evolution does need to be addressed up front, but the pain will be less up front than later.

One way of lessening the pain of schema evolution in a database is to support the use of "jagged rows", i.e., where different rows in a table are allowed to have different numbers of columns and/or different column definitions. Many DBMS's offer support for jagged rows through composite row types, with some row types inheriting part of their structure from other row types.

The BCS DBXten DataBlade also supports jagged rows. With DBXten, some blocks of tuples may have one schema while other blocks might have a different schema. Hence with DBXten a single table can support both the old and new schemas; no table unload/reload is necessary.

As stated earlier, keeping track of data evolution in a database is complicated, but not impossible. Foreign keys can be used to represent parent-child relationships between rows of the same or different tables, and "start date" / "end date" columns can be used to distinguish between different versions of the same record.

8) Database data types are business - not science - oriented.

It is true that conventional DBMS data types (character, packed decimal scalar types) are not oriented to science applications. In recent years, however, most DBMSs have added floating point support as well as more complex general data types (e.g., arrays, sets, composite types) and specific data types (e.g., XML). Furthermore, object relational databases allow for the creation of arbitrarily complex data types through the use of database extensions (e.g., Oracle Cartridges, Informix DataBlades, DB2 Extenders, etc.). Some popular examples of database extensions are:

- Spatial extensions such as Oracle Spatial, the DB2 Spatial and Geodetic Extenders, PostGIS for PostgreSQL, and the Informix Spatial and Geodetic DataBlades.
- Text search extensions such as TSearch2 for PostgreSQL and the Informix Excalibur Text Search DataBlade.
- Chemical Oracle cartridges such as DayCart and JChem.

BCS has written two database extensions: the BCS Grid DataBlade, which introduces the "Grid" data type, and DBXten, which introduces the "Chip" (a tuple-block) data type[1]. See our Whitepapers section for more information on these products.

9) Database Management Systems are too expensive.

Some DBMSs are very expensive, but others, such as PostgreSQL and MySQL, are free. Many scientific organizations already have site licenses for DBMS products, so for these organizations the incremental cost will not be an issue. Some DBMSs (e.g., Microsoft Access) are included for free in larger, widely-used products (e.g., Microsoft Office). In addition, Oracle, IBM, and Microsoft all offer free-but-size-limited versions of their DBMSs and extensions.

10) Databases are too complicated.

Earlier we cited the following claim with respect to databases being too complicated:

> Database Management Systems add extra levels of complication to what I do. To use one I need to learn about logical data modeling, learn new languages and interfaces (e.g., SQL, ODBC, JDBC), and hire a database administrator to keep the DBMS running.

Anyone who is the custodian of data can benefit from logical data modeling skills, whether or not their data is stored in a database. But it is true that to use a database one must develop new skills. SQL is a very powerful language, but its declarative, set-oriented nature can seem unnatural to someone used to dealing with data items sequentially using a procedure-oriented language. Note that the Virtual File Interface, described earlier, will reduce the need for someone to learn SQL, even after their data has been moved from files to a DBMS.

With regard to the need to hire a database administrator, different DBMSs need different amounts of ongoing attention. While some need constant "supervision", others (such as Informix, PostgreSQL, and MySQL) generally take care of themselves. The other role of a database administrator is to ensure that good data organization and sharing standards and practices are developed and followed. But, as with logical data modeling, this should be done regardless of where the data is kept.

[1] While large multi-dimensional arrays can, at present, easily be implemented as an ordered set of Chips, an upcoming version of DBXten will include *explicit* support for large multi-dimensional arrays.

11) Complex interrelationships in a file (hierarchies, etc.) may be more difficult to represent in a database.

The problem with this argument is that it is comparing a scheme where relationships can *be implicit* with one where, by its design, relationships must be *stated explicitly*. The relationships are complex, whether the data is stored in a file or in a database. And by not stating them explicitly one runs the risk of making the data useless if the knowledge of the relationships is ever lost.

12) DBMS performance is poor.

We have addressed this concern in one of our other white papers. Indexes (single or multi-dimensional) and the options for parallelism available in most DBMSs can allow a DBMS to significantly outperform applications that deal with files sequentially. And the BCS database extension products, in particular DBXten, can increase performance even more.

Conclusions

Database management systems do provide features that scientists should find appealing, in particular:

- the ability to index data for faster retrieval,
- the ability to store data in a single place, thereby removing the artificial walls that separate data when it is stored in multiple files,
- the ability to easily add and remove attributes as they start to become, or cease being, interesting,
- built in parallelism, which is sometimes difficult to program yourself.

In spite of these benefits, many scientists still shun databases. At first glance, their concerns seem legitimate. In particular, database management systems sometimes offer very poor performance for scientific data, and transitioning from a file-based culture to a database-based culture can be costly and seem unnatural. Indeed, for some applications a database management system just isn't justified:

- if all the datasets are small,
- if data storage and access requirements aren't going to change (flexibility isn't needed).

In the end, the individual scientist will have to decide whether the benefits of switching to databases are worth the cost, but this paper has described how many new database features, including the BCS database extension products, can increase the benefits and decrease the costs.

References

[Maier and Vance, 1993] A Call to Order in *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1-16. ACM Press, 1993

[Buneman, 2002] Why Scientists Don't Use Databases, Microsoft PowerPoint presentation to National e-Science Centre, April 25, 2002

[Mattman, 2009] To Database or Map/Reduce It?, "Rocket Science" blog entry, August 29, 2009

[Gray *et al*, 2005] Scientific Data Management in the Coming Decade, Technical Report MSR-TR-2005-10, Microsoft Research, 2005