



DBMS Technology for the AVO

Clive Page, University of Leicester

2004 January 14

Abstract

Database management technology is central to the development of the Virtual Observatory (VOs), since its eventual aim is that the world's astronomical data collections should look like a single large distributed database. Relational database management systems (DBMS) are good at searching through large tabular datasets such as source catalogues, but the important operation of cross-matching sources in different catalogues is supported only with difficulty. Two algorithms for doing this were examined: that based on a true spatial index was found to be superior. Since relational DBMS are not all that well suited to the requirements of the VOs, some alternative structures were also examined. An object-relational open-source package, PostgreSQL, appeared to be the most suitable of the DBMS examined.

1. Introduction

The principal aims of the Virtual Observatory (VOs) are to make the world's collection of astronomical data archives uniformly accessible, and to make it easy for astronomers to explore and mine these large data collections for new information. Database technology obviously has a vital role in building the Virtual Observatory, indeed the eventual objective is that the VOs should appear to be one huge, distributed, database.

This report describes activities under AVO work area 3.3 up to the end of 2003. Much of this was carried out as part of the AstroGrid project, which has database needs very similar to those of the AVO. Some related activities in other work areas are also described briefly to put the work into context, for example progress on standards for database queries and results.

Astronomical data centres contain data in many different formats, for example: images, spectra, time-series, complex visibilities (in radio astronomy), photon-event lists (at high energies), and source catalogues. Images and most of the other forms of data require specialised data processing software, so the role of the database management system (DBMS) is essentially just that of keeping track of each dataset and providing search facilities based on the associated metadata. A modern DBMS can store images and similar datasets within a record as a binary large object (BLOB), but one cannot do much with a BLOB except import it or export it on demand. In practice most astronomical archives keep images and similar datasets as external files,

with the DBMS just storing the metadata and a link to each file via its path or URL. In either case such use of database technology is fairly straight-forward and presents no problems worth further discussion.

There is one very important exception to this: the data product resulting from astronomical data analysis is often a list of objects detected in some region of the sky, usually known as a source catalogue. These are simple tabular datasets, much like the tables which relational DBMS are designed to process. Source catalogues vary in size from tens to billions of rows and from a few columns to a few hundred. Many of the operations needed by astronomers are similar to ones which RDBMS support, for example joining one catalogue with another. It is only in recent years, however, that DBMS have become powerful and cheap enough to store and process actual scientific data, rather than just the metadata, but experience in using them in this way is still in its infancy. Although relational DBMS are very powerful and stable packages, they are designed for use in commerce; their use in scientific research is not entirely problem-free. This document describes the main areas in which problems arise, and reports on the progress in solving them.

2. Scientific Requirements

In order to make a start on developing the VObs infrastructure, the AstroGrid project adopted the Unified Process for its software development; this depends heavily on use-cases. The first step was to collect a range of research problems which could make good use of the facilities of the eventual VObs. From this rather large set ten representative problems were selected for detailed analysis. This set of ten, however, included four problems in the field of solar physics or solar-terrestrial physics, leaving only six relevant to the AVO. Full details of the "top ten" are given in [1]. An outline, concentrating on database requirements, is given here:

- **Brown Dwarf Selection:** these may be found by looking for stars which are faint, red, and have high proper motions. The information required appears partly in astrometric catalogues and partly in deep optical or infra-red catalogues such as 2MASS, APMCAT, and SDSS (with WFCAM and VISTA in future). This requires extensive cross-matching between large catalogues, and then selections on complex criteria. The detection of high proper-motion objects may also require cross-matching catalogues of positions measured at different epochs to find sources which have moved. The science case notes that the necessary datasets are often located at different sites.
- **Deep Field Surveys:** the analysis depends large on extraction of source lists from images, but also needs a cross-matching facility. The source catalogues involved are likely to be of modest size, since they arise from small areas of sky such as the Hubble Deep Field.
- **Galaxy Clustering:** the general procedure is to use multicolour optical survey data sets (such as DPOSS), and from there select sources marked as galaxies which have particular properties, e.g. are in a particular locus of the $(g-r)$ vs $(i-r)$ colour space. Then one creates density maps.
- **High Redshift Quasars:** searches will require comparison of optical (i, z bands) and near-IR data over thousands of square degrees; current work uses SDSS results, in future WFCAM and VISTA catalogues will be used. Radio and X-ray surveys may also be used to select hi- z quasars. This certainly requires scanning large catalogues using complex selection criteria, and is likely to involve cross-matching between catalogues from different wavebands.
- **Low Surface-brightness Galaxy Discovery:** this requires extensive image-processing which is outside the scope of this document, but the initial step is to search in large catalogues such as 2MASS for all objects with non-thermal colours. The science case also notes that Radio data (e.g. Arecibo HI 21cm) can also be used: radio selected LSB's are compared with optically selected sets.
- **Supernova-Galaxy Environment:** the main activities in this science case involve searching archives for relevant image and spectroscopic data, but some database work is involved e.g. in cross-referencing the position of a SNR candidate with other objects in the vicinity such as galaxies or clusters of them with known red-shifts.

Besides solving specific problems such as these, the VObs must have various more general capabilities. Firstly, we must cope with the astronomical data explosion: the volume of data collected by observatories seems to be increasing at an exponential rate, especially because of the availability of ever larger and faster electronic photon–detectors: the consequences of this are outlined in section 3.

As more datasets from more observatories come on–line, it becomes more difficult to find all the right data to solve a given research problem, and even experts are likely find a resource location service useful. All the VObs projects now seem agreed that some form of resource registry is needed: this is of course a form of database, and is briefly covered in section 4.

Many research programmes, including most of those listed above, depend on making use of data from several different sites. It is vital therefore that the VObs adopts standards for queries and results, so that data can be interchanged or combined readily. Good progress has been made in this area, as noted in section 5. Several of the scientific problems listed above depend upon cross–matching source catalogues: algorithms for doing this on large datasets held within a DBMS have been developed under the AVO programme, and are covered further in section 6.

Data exploration and data mining facilities are important but we need to determine to what extent these can be carried out within a DBMS. If not the data will have to be exported to external packages, as explained in section 7. Database packages are designed to access data held on local discs, but VObs users will want to access data held on many remote sites: the problems of providing distributed database facilities are outlined in section 8.

Finally, it is clear that many new astronomical data centres will be coming on–line over the next few years, some large, some small. Many existing sites are also using rather basic DBMS which cannot support all the functionality desirable in the virtual observatory. We therefore carried out an assessment of various database technologies, and a more detailed evaluation of a number of the more suitable DBMS packages. Some alternatives to relational DBMS are examined in section 9, and results of our evaluations are reported in section 10 and in the Appendix.

3. The Astronomical Data Explosion

The VObs must be capable of handling the "data explosion" (otherwise known as the data avalanche or data tsunami, choose your metaphor) faced by astronomers. Moore's Law suggests that computer power and storage capacity double about every 18 months; this seems to have been true for over two decades, and seems likely to continue for at least another few years. Astronomers gain benefit from Moore's Law by using ever larger and faster CCD detectors in their telescopes, so that volumes of raw data have been expanding at much the same rate as processing capacity. Derived data such as source catalogues are growing in proportion, or perhaps more rapidly, as large systematic sky surveys are occupying a higher proportion of telescope time. The longest source catalogue at present, USNO–B, has over a billion rows, but several others are nearly as large. It is not uncommon for these tables to be vary wide as well: the widest encountered so far, 1XMM, has around 400 columns. We know, however, that instruments coming on stream over the next few years (WFCAM, VISTA, e–MERLIN, ALMA) will produce much larger volumes of data than those in use at present.

Fortunately, relational DBMS seem to able to handle tables of this size, if enough money is spent on the hardware. A recent survey by WinterCorp [2] showed that the largest known table in the commercial world had around 250 billion rows and occupied some 10 TB of disc space; secret applications may, for all we know, be even larger. This is around two orders of magnitude larger than the biggest astronomical tables in current use. We obviously need to develop systems and algorithms which are capable of being scaled up to

appropriate levels.

Telecommunications costs, unfortunately, have not been falling as rapidly, and network bandwidth available to us has not kept up with the growth in data volumes. Currently the largest source catalogues occupy around 100 GB of disc: transporting such datasets over national networks takes a few hours; over international links somewhat longer. If current trends continue, so these catalogues grow with Moore's Law but network bandwidth grows more slowly, it will get increasingly difficult to transport such datasets over the Internet. Another factor to note is that some database operations, including joins, require frequent interactions between servers, so they demand a network with low latency. Here the speed of light presents an insurmountable barrier. Because of high telecommunications costs data from mountain-top observatories are being routinely shipped back on tapes or hard discs. The option of carrying out at least the earlier stages of data reduction at the observatory and just shipping back the reduced data is also looking attractive: for those who actually need the raw or semi-processed data, the observatory will become another data centre.

3.1 Grid Computing

The Grid Computing model is currently fashionable, but different models suit different problems. Indeed it is now clear that there are at least three types of grid:

- **Computational Grid:** this provides remote processing power for big compute-intensive problems.
- **Data Grid:** this allows data resources to be accessed all over the network. This corresponds quite well to what the VObs is trying to do, but we need to bear in mind the network bottleneck.
- **Service Grid:** this provides specialised services on particular network nodes, which can be accessed from anywhere. There may be some use for this in the VObs if we need to install costly licensed software on just a few nodes, but allow remote access.

As noted by Jim Gray in his paper on Distributed Computing Economics [3], with current prices of processing power and bandwidth, a computing grid only makes sense for problems where one needs to crunch at least 100,000 instructions per byte. In astronomy a few theoretical studies and simulations may reach this level, but it is far above that of most data reduction and analysis programs. The computational grid model therefore seems mostly irrelevant for the VObs.

The data grid model, however, is very close to that of the Virtual Observatory. In the UK the OGSA-DAI Project is trying to make data grids more of a reality by integrating database access within the grid computing paradigm. AstroGrid has installed the OGSA-DAI software on nodes in Cambridge and Leicester, and will shortly be testing the facilities available and their performance in practice. It uses JDBC to access the DBMS, and has so far been tested on MySQL, Postgres, and DB2. For more information see the report on Grid Technology from Guy Rixon.

3.2 Data Centres

Data Centres cover a wide range in size: there are a few large ones (such as CDS), a somewhat larger number of medium-sized ones often with specialist holdings (for example LEDAS in Leicester), and even more small centres, many of them with rather limited facilities on the web. Not many centres have more than a few terabytes of data on-line at present, even though such storage can be purchased for just a few thousand Euro. Processing power is also cheap, as shown by the fact that most of our computers are idle for most of the time. The most scarce resources are bandwidth (as noted above) and the manpower needed to set up the databases and web-sites, and to manage the systems properly.

I expect that as well as an expansion of the established data centres, there are likely to be more of them,

mostly at the lower end of the size range. Some, as noted above, will be specialised centres at observatories, others will arise from new observatories or space missions. In addition it is notable that large sensitive CCD chips are now quite cheap, and useful work can be done by telescopes of fairly modest cost. There has also been a rise in the number of sky survey projects and of remotely operated observatories. Current examples include projects such as SuperWASP, the Faulkes Telescopes, and the Liverpool Robotic Telescopes. It is also becoming clear that a wealth of new short time-scale phenomena can be found by surveying large areas of the sky at frequent intervals; even amateur sky-watch programmes are producing valuable results and are starting to set up web sites. The Virtual Observatory needs to accommodate them all.

The smaller data centres are likely to have quite modest funding, and given the very high prices charged for the main commercial DBMS, it seems clear that the VObs ought to choose free or very inexpensive products if at all possible. Of these only Open Source packages give one the assurance that they will always be free. Fortunately most of the world's VObs projects appear to have a bias in favour of Open Source software; database management packages may be an exception even though they are some of the most expensive software packages on the market. The larger data centres often use an expensive commercial DBMS, but many data centre managers have told me that their choice of DBMS was strongly influenced by what was already licenced at their site (because it was used for teaching or administration). There seem to be very few cases in which a particular commercial DBMS has been chosen on scientific grounds.

3.3 Architectural Implications

The obvious answer to the bandwidth bottleneck is to site the processing power next to the data storage facilities: the mantra has been "ship the results, not the data", since the results are usually much smaller in volume. Remote processing of data will require a new model for managing data archive centres, which up to now have been designed just for simple searches and modest data retrievals. To provide a full range of facilities, the VObs will need to encourage data centres to provide:

- Significant processing power so that more complex searches, selections, and joins can be performed in situ.
- Ample disc store for the temporary of intermediate results.
- Appropriate software to run on their data holdings, for example a suitable DBMS with perhaps statistical and graphical packages.
- The necessary authentication and authorization infrastructure so that external users can be let loose on the system without risk.

What this requires, obviously, is that the software to perform the processing be present at each site where large datasets are located. Many of the larger data archive sites will already be using DBMS, and it may well be feasible to provide additional middleware to support the main functionality of the VObs on top of these DBMS. But a significant proportion of important astronomical data are located at smaller sites, where there may be no suitable DBMS in place, and the funding situation may not allow them to purchase an expensive commercial product. The use of cheap or free software will obviously make it possible for a much wider range of data archives to participate fully in the VObs, and thus enhance the value of the system as a whole. There are at least three Open Source relational DBMS, some commercial DBMS are also free for academic or non-profit use: for example Sybase ASE has a Linux version which can be downloaded free; IBM's DB2 is free to all academic users (including research institutes, apparently) in most countries, under the IBM Scholars' Program; and quite cheap academic licences may be obtained for Microsoft's SQL Server.

There are good reasons for the VObs to be based on Open Source software apart from the cost arguments: open source software avoids all the complexity and time that normally has to be spent on licence issues, and it essentially reduces the risk that the product will die or be taken off the market. As the story of O2 (see section 9.2 below) shows, this can be a serious risk.

4. Resource Registry

The need for a resource registry arose originally from a realisation that even the expert user cannot know the location of all the useful sources of astronomical data on the network. The registry of astronomical metadata is now an accepted part of the VObs infrastructure, and it is pleasing to report that all the VObs projects have agreed on its basic properties. There is likely to be at least one registry per VObs project, though perhaps with differing levels of detailed information. These registries will have a number of special properties: they will update their contents from one another, perhaps once a day, it is also likely that they will eventually be able to harvest their information from data archive sites automatically, much in the way that search-engine robots do. The protocols for data retrieval and harvesting are the subject of discussion, but it is likely that agreement will be reached soon.

The registry is, therefore, a rather specialised type of database, but the total volume of data is rather small, perhaps only a few megabytes. The exact functionality of the registry is still being defined but it seems likely that the search criteria will be fairly simple, and given the small data volume. The requirements can probably be satisfied with a basic relational DBMS, or an XML-DBMS, since much of the data input and output will be in the form of XML documents. It seems unlikely that the registry will present any specialised database challenges, but the AVO database team are obviously willing to give advice if required.

5. Standards for Interoperability

Considerable progress has been made on standards for interoperability after a series of meetings of the International Virtual Observatory Alliance (IVOA). AVO members have been active in all the forums of discussion. There has been general agreement that the VObs should adopt the Web Services paradigm (use of SOAP messages and WSDL descriptions) and data interchange standards based on XML. This does not render current systems obsolete since WSDL supports continued use of CGI-based interfaces, including those based on the existing ad-hoc standards such as GLU and ASU, devised by CDS some years ago.

5.1 VOTable

The first standard to be agreed was that for the results of a query sent to a database: the VOTable is an XML format with metadata based on the headers in the well-established FITS file format. There are three different ways of encoding tabular data within a VOTable:

1. TABLEDATA – which is pure XML and therefore rather verbose, but fully compatible with existing XML software.
2. BINARY – using the standard XML way of encoding binary data within a larger XML document: this is much more efficient but not easily decodable by generic XML applications.
3. FITS – the XML file stores essentially just (a copy of) the metadata, the actual data is accessed by following a link (URL) to a separate FITS file containing a binary table. This is even more efficient, and allows existing FITS files to be enclosed in an XML wrapper.

Several useful applications are already available for handling VOTable data, including TOPCAT from the UK's Starlink Project, and VOPLLOT from the VObs project of India. At present only the TABLEDATA

option is widely supported and tested; it will be essential to support at least one of the more efficient options if large chunks of tabular data are to be shipped from one site to another.

5.2 Query Language

Agreement on a query language has taken longer to reach, but current drafts envisage that three levels of query language will be needed eventually. A draft specification for the base layer, ADQL, is now agreed, and implementations in the US and in Europe have appeared. The syntax is based on the standards for SQL92 and JDBC, with both a simple SQL-like string and an XML format defined. The next layer up will be responsible for handling syntax for higher-level functionality including cross-matches between catalogues. It seems likely that a number of further iterations will be needed before this layer can be fully defined.

In the long term we need to give the remote user of a database the same facilities that local users have at present. This means allowing users to import their own datasets, export subsets that they create, to create and define their own indices, write their own user-defined functions, getting estimates of query duration (using EXPLAIN or similar), scripting sequences of queries, giving time limits to queries, and allowing run-away queries to be aborted. The need for several of these facilities has already become evident from the Skyserver system at Johns Hopkins University which allows SQL queries to the SDSS and associated datasets. Considerable further work is needed before all these facilities can be incorporated into IVOA standards.

6. Cone Searches and Catalogue Cross Matching

One of the most productive ways of extracting new information from existing source catalogues is to locate the same source in two or more of them: if the catalogues arise from observations in different wavebands this gives valuable spectral information, if they were taken at different epochs it may reveal variability in flux or motion in space. The most basic such search is to perform it one source at a time: since the position is always slightly uncertain (and some objects are extended or may have moved) it is normal to search for a small circular region about the position of interest in the sky – in three dimensions this is a cone from the observer out to infinity or the edge of the visible universe, whichever comes first. More generally one will want to cross-match all the sources in one catalogue with the corresponding sources in another: in database terms this is a join between tables, but a special type of join, because an overlap of spatial error regions is the match criterion.

6.1 Cone Search

Cone-searches are supported by quite a number of astronomical data archive sites, and are popular with users, although perhaps the fact that they are just about all that only search facility provided by most sites has some influence. It is notable that cone searches are not mentioned explicitly in any of the top six science problems analysed above. This is probably because it is a rather basic facility, often used in small investigations, whereas these science problems describe much larger programmes of work designed by experienced astronomers.

The names and units used by the various on-line cone searches vary considerably, but there have been two attempts to bring some order to the chaos. The Astronomical Server URL (ASU) was defined nearly 10 years ago by CDS; more recently the US-NVO has defined a specification for a simple cone search, which is, for reasons unknown to me, quite different in detail. The GLU/AstroGLU system, also defined by CDS, is used at a number of sites to transform a single cone search query into the actual set of CGI parameters required by a number of well-known sites. A few web sites have extended their cone-search to allow the user to upload a list of positions of interest and then perform a separate cone search on each one. The results, however, are generally left as a set of separate tables; it is a non-trivial problem to merge meaningfully results coming

from different catalogues: this is a subject of current study by Patricio Ortiz of AstroGrid.

It is perhaps best to see the cone-search as a special case of a spatial join, when the first input catalogue consists of just one position. Any system which supports a spatial join can perform a cone-search efficiently, but not vice-versa, because simple search methods do not scale up adequately.

Many current cone-search services are based on WCSTOOLS (described below in section 9.1) which essentially rely on a one-dimensional index of the sky (or a crude slicing along one axis, and an index or sequence of values along the other axis). This works well enough for small tables and small areas of sky, but is inherently un-scalable. This is shown in the following table of results, obtained on the whole 2MASS catalogue of 470,992,970 rows and 60 columns, ingested into Postgres. Cone searches covering an area of 0.01 and 2.0 square degrees were carried out, when the table was accessed by either a B-tree on declination, or an R-tree on both RA and declination. The elapsed times are in seconds.

Indexing method for cone search	0.01 square degs	2.0 square degrees
Declination index, B-tree	156.7 secs	1639.7 secs
RA+declination index, R-tree	1.2 secs	2.2 secs

The superiority of the R-tree is overwhelming on a table of this size. The R-tree access times reduce further, to a millisecond or so, when the same (or a similar) query is repeated, as the required index blocks are cached in memory, whereas repeated B-tree accesses require scanning rather a substantial fraction of the original dataset, and these hardly speed up at all.

6.2 Cross-Matching

The main criterion for matching celestial sources in two or more catalogues is normally the coincidence of celestial positions, usually expressed as a pair of spherical-polar coordinates (right ascension, declination). In practice, of course, these positions are always somewhat uncertain, and the match condition actually needs to be the overlap of the respective error regions. These regions are often circular, but some may be elliptical or more complicated in shape. The draft plan for the demonstration AVO Geometric cross-matching facility appears to suggest that only the nearest neighbour should be selected in each case, but this is hard to justify scientifically: there may be no nearest neighbour within the error-region if the source has no counterpart, or there may be several, and the next nearest may be only slightly farther away but be a better match on other grounds. Where there are several matching sources in the second catalogue then other properties of the sources may have to be used to find the actual counterpart, such as spectral type or magnitude, but this is a matter for scientific judgement and can be carried out after the list of apparent matches has been produced by the database.

The basic cross-matching operation is what in database terms is called a JOIN, and RDBMS are highly optimised to perform joins between tables. There are, however, a number of complications which the design of the cross-matching software needs to take into account:

- Most RDBMS are designed to perform efficiently only an equi-join, i.e. one where there is an *exact match* between two integers or strings and cannot use an index when floating-point numbers are compared within error limits.
- In some catalogues all error regions will be the same, in others it will depend on other columns in the table, such as an error-radius, which further complicates the join.
- One cannot use a simple cartesian distance for coordinates on the spherical surface of the sky, but the great-circle distance (and the simplest formula lacks accuracy when small angles are involved, the haversine formula is much better).

- Spherical–polar coordinate systems have problems with wrap–around of right ascension at zero hours, and singularities at the poles.
- Catalogues may have positions measured using different equinoxes (B1950 or J2000 for example): if so precession needs to be applied to one of them before coordinates can be compared.
- Nearby sources may have significant proper motions, so an increased matching distance may be needed for catalogues corresponding to widely different epochs.
- It is sometimes of great scientific interest to know whether a source in one catalogue has no counterpart in another: this corresponds to a LEFT OUTER JOIN in database terms.

These obstacles have led many previous astronomical projects (such as ROSAT and XMM–Newton) to write procedural code, in C or Fortran, to perform their cross–matching.

6.3 Spatial Join Algorithms

There are two algorithms known to be suitable for performing a spatial join using a relational DBMS:

1. Join using a spatial index (such as an R–trees) which is available with some DBMS.
2. The pixel–code algorithm, described more fully in [4]

The idea of the pixel–code algorithm is to cover the sky with a grid of pixels, numbered with integers, and build a standard B–tree index on this set of integers. A simple grid along lines of right ascension and declination is not very satisfactory as the pixel areas vary down to near zero at the poles. Astronomers have developed a number of roughly equal–area pixelations of the sky, including HTM (from Johns Hopkins University), HEALPix (from ESO), and Qbox (from CDS) and all of these solve the problem well. But whatever grid is chosen some error regions are bound to overlap two or more pixels, so a simple equijoin of pixel numbers will lose information. The pixel–code algorithm solves this problem but requires additional tables, and two simple equi–joins and one three–way join. Its great advantage is that can be used with even the most basic relational DBMS, since it needs only simple B–tree indices on the various tables. But it has some drawbacks. Firstly it is substantially more complicated. Secondly the final selection requires a SELECT DISTINCT operation to remove duplicates; this implicitly involves sorting the output table, which is an operation which does not scale linearly with table size. Thirdly it is more difficult to change the match criterion, say from two sigma to three sigma, than with an R–tree index. Because of the doubts about speed and scalability, the two algorithms were compared on some datasets of increasing size.

These tests compared a spatial join between a section of the 2MASS catalogue with the corresponding area of USNO–B using both the pixel–code algorithm and the R–tree indexing. Because Postgres had the best R–tree indexing, it was used for both algorithms. The comparison was used on a number of areas of sky of increasing size.

Sky area (sq.degs)	Pixel–code (secs)	R–tree (secs)
36	88	54
108	406	161
324	1150	553
972	4623	1964

It is clear that the R–tree join is faster than the pixel–code method, and that the speed advantage increases with dataset size. Since the R–tree join is also simpler, more scalable, and more flexible, it seems better on all

grounds.

7. Data Exploration and Data Mining

The term "data mining" is usually defined as the search for useful information hidden in large datasets. In the commercial world data mining generally takes place in the data warehouse: a large static collection of tables, often a snapshot of the highly dynamic transaction-oriented databases used to run the company. The data types are nearly all enumerable types such as integers and character strings. Floating-point numbers are rare: money is best represented in fixed point form as this avoids embarrassing problems with rounding errors. It is notable that Oracle, the market leader in DBMS, did not have any support for floating-point data types until quite recently; all numbers were stored as strings of decimal digits. A simple data mining query might be to find out whether a particular special offer has generated increased profits, for example by comparing before and after sales figures, or comparing sales from different areas of the country. The most common operations seem to be summation, grouping and sorting, but some packages support more advanced operations include clustering, classification, regression analysis and outlier detection. The last of these is used, for example, to identify fraudulent use of a credit card.

7.1 Data Mining Packages

Some of the operations supported by data mining packages are potentially useful in astronomical data analysis, so several of them were examined briefly. Nearly all of those examined were designed for what, in our terms, are very small datasets, and the examples of usage rarely used more than 3000 rows. One package was said to be capable of handling "very substantial" datasets when this meant 100,000 rows. Such limits arise, I think, from holding data entirely in memory; many packages were also interfaced to spreadsheets such as Excel, which have their own limits. Most packages were rather specialised, so that a number of them, all with very different data formats and user interfaces, would be needed to get a reasonable range of functionality. In summary: none of those examined so far seemed a good match to typical astronomical requirements. On the other hand there is a wide range of products on the market, and some astronomical datasets are fairly modest in size. Most packages were rather expensive, but the cost might be acceptable if it were possible to use the service grid model to mount the package on a single computer which could be used by anyone in the VObs community. Of course we do not really know how to do this yet, and the package licence might not permit such use. Despite these reservations, I think that a more serious evaluation of some of these products for astronomical use would be worth while.

What astronomers need rather more, I think, is what is perhaps better described as a data exploration facility: after all one has to explore the data for nuggets or even rich seams of ore before any mining can be considered.

7.2 Data Exploration

Support for interactive data exploration is essential, as users are unlikely to get their queries right first time. They will want to try an initial selection, and then examine the results, perhaps by browsing through the first or last rows on the screen, or computing simple statistics on them (range, mean, etc.), perhaps sorting them in order so that the extreme values can be inspected, or perhaps displaying the results in some simple graphical format. When the first selection stage seems satisfactory, the astronomer goes on to perform further database operations on the intermediate results. If the input dataset is large, so that sequential scan operations are slow, it may be desirable to select a representative subset randomly. Although this incremental or iterative method of working will be familiar to all those who have performed astronomical data analysis, it seems

alien to the database world. This and other difficulties are discussed further below.

7.3 Problem Areas

The relational DBMS and the SQL language have come to dominate the commercial database market, but those doing scientific data reduction and analysis have rather different needs. Some of the obvious problem areas are analysed below, with suggested workarounds or solutions.

Metadata are routinely attached to astronomical tables and their columns: the careful storage and propagation of metadata is what makes FITS files so useful and popular: columns come complete with physical units, formats for display, and free-text comments; a table can have almost any amount of additional metadata attached, applying to the table as a whole. The relational DBMS usually supports none of this, not even essential attributes like the physical units of columns. It is possible for the programmer to create additional tables to hold metadata, but this is hardly a solution as the common DBMS operations such as selection, projection, and joining will not make use of them or propagate them.

Normalisation: Relational operations only produce valid results if all tables are *normalised* according to the rules of Codd and Date (this involves basically the removal of all duplicate information and ensuring that nothing depends on the order of rows or columns). Source catalogues and other astronomical tables may look like relational tables but they are rarely normalised, except perhaps by accident. The complex collections of data generated by telescopes tend to contain one-to-many and many-to-many relationships which can be converted into a set of properly normalised tables only after a lot of manipulation using entity-relationship modelling. Although in theory the lack of normalisation can result in incorrect results from certain sequences of relational operations, in practice this is not a problem likely to affect astronomical database use very often.

Data types: All RDBMS are designed for commerce where nearly all fields are simple integers or character strings; floating-point data types, bytes, bit-strings, and data structures tend to be poorly supported. It is usually hard to display floating-point numbers in sensible formats, and reading or writing angles in sexagesimal format requires appropriate user-defined functions to be created.

Storage order: Astronomical tables, especially source lists, tend to have many columns (up to 400 in a few cases): the normal storage order is row-major, which means the entire table has to be scanned even if a selection is made using the properties of just a few columns. This means that sequential performance can be relatively poor.

Mathematical functions: VObs users will often want to use functions such as LOG or TAN in selections. The SQL92 Standard specifies very few of these functions, although in practice most DBMS support a reasonable range. Unfortunately the names are far from standardised, thus LOG may mean take a logarithm to base e or base 10, depending on the product. To fill in the gaps it is essential for a DBMS to support the definition and use of user-defined functions (UDFs). We are likely to want to define additional UDFs for functions needed in astronomy such as great-circle distance between points.

Null values representing missing information occur in most astronomical tables. The SQL Standard supports the necessary 3-valued logic (true/false/unknown) but the way in which null values are handled when sorting a column, or creating an index is not standardised.

Multi-dimensional indexing is built in to very few products; several more have it as an add-on feature, often at extra cost.

SQL is a non-procedural language, that is the user specifies the result needed, and the query analyser and optimiser do the rest. This is all very well in theory, but practice is often different. All DBMS expect you to

create the indices you need to speed up your queries, and nearly all provide an EXPLAIN command which (though the details vary) will tell you how a given query will be executed. The requirements of exploratory data analysis, as noted in the previous section, depend on being able to execute a complex sequence of operations one step at a time, if necessary, so that you want to be able to select into some temporary table. SQL makes this difficult. There is no standard way, for example, of sending the output from one query into a new table, it is expected that all you even need from a query is plain text sent to your terminal. In practice most DBMS will allow you to select into another table, but it usually involves non-standard syntax, such as SELECT INTO or perhaps CREATE TABLE ... SELECT, or it may require two separate SQL statements.

Statistical facilities are very basic in SQL: the functions min, max, mean are always provided, but standard-deviation or variance are non-standard. The calculation of a median or any other quantile is possible, but complicated. To compute more advanced statistics it is usually necessary to export the data from the DBMS and ingest them into another package. This is likely to have to use a simple CSV format, in which all column names get lost.

Graphical facilities are absent (though one can just about construct a simple bar graph in SQL, that is about the limit). If any plotting or visualisation facilities are needed, the data again have to be exported and then imported into some other package.

Packages designed for the VObs such as VOPlot and TOPCAT can import VOTable files, which preserve metadata, but without additional software, normal DBMS cannot produce them.

These problems all have solutions or workarounds, but they go some way to explain the fact that astronomers, and indeed scientists in general, have not been keen users of database technology.

8. Distributed Databases

8.1 Distributed Join

Up to now we have examined the case of the spatial join when both tables are resident within the same database. In practice the two catalogues to be cross-matched may be located at different data archive sites. How to proceed? The short answer is: copy the smaller one to the site containing the larger one using FTP or similar. A longer analysis is given below.

If it is important to identify unmatched sources in the first table, i.e. to perform a LEFT OUTER JOIN, then every row in table 1 is used in producing the output table; if all of its columns are also needed in the output, then it is surely simpler and faster to copy the entire table to the location of table 2 before performing the join within the DBMS. If it is a very wide table, and only a few columns are needed in the output, in principle it might be sufficient to project a new table with just the required columns and copy this across.

If, however, we are not interested in unmatched sources, so want an INNER JOIN, there might be a way of passing a smaller quantity of information across the link: we could project a new table with just the positional information (ra, dec, error-radius) say, and first check whether there is any match in table 2: if there is then any other columns of table 1 have to be transferred, but if the proportion of matches is small, then this would be cheaper. Since any join operation requires a lot of information passing to and fro, it is not just the bandwidth of a link but also the latency. These time delays may be a serious problem on wide-area links, and are fundamentally limited by the speed of light.

Another question arises: even if it is inefficient, can a distributed join actually be performed using current DBMS? Some mainframe products such as DB2 and Oracle have optional support for distributed databases, but as far as I can ascertain, these are intended for use over local area networks, not long distance lines. When

I have asked database experts from these companies, they say that it might be possible, but would not be very practical. One serious problem is that the connections between client and server require a number of IP ports to be open: these requirements are not at all well documented, and I suspect that there might be serious firewall problems in practice.

The Postgres DBMS has a limited ability to connect to more than one server, using a contributed package called dblink. This was very easy to set up and use, and I managed to write a script to join one table with another over our local area network. A spatial join between two small catalogues which took 8 seconds when done within one DBMS, took about 60 seconds when carried out over the local area network (100 Mb/s Ethernet). This was partly because it could only make limited use of the join functionality of the DBMS, as it was essentially a sequence of unrelated cone searches. Nevertheless it is interesting that it was possible. Postgres documentation suggested that only one port needed to be open in the firewalls, and a search of the source code (not possible with commercial products) suggested that no other port was indeed required. I have now got the necessary ports opened on request between Leicester and Cambridge, and plan to test a wide-area join shortly.

There is another interesting development: the Distributed Query Processing (DQP) project of OGSA-DAI has been set up to handle distributed joins over the wide area network. At present it requires queries to be specified in OQL not SQL, and cannot handle spatial joins at all, but development is continuing and we are keeping in touch with the group involved.

8.2 Data Warehouse Concept

In cases in which it is not feasible to perform a join over the network, and in which there are no suitable facilities at the sites holding either of the tables that one wants to join, the best alternative is, surely, to copy both of them to some third location where the necessary disc space, software, and cpu power is available. This is the idea behind the Astronomical Data Warehouse. There is no reason why any data centre should not provide ADW facilities, but not all of them may choose to do so.

9. Alternatives Database Structures

Given the limitations of the standard relational DBMS for the handling of astronomical data, this section examines a number of other structures which are used in astronomy or other fields.

9.1 Flat File Systems

WCSTools

WCSTOOLS, a package from the Harvard-Smithsonian Center for Astrophysics, is used by many astronomical archive sites. This is a wonderfully pragmatic piece of software which supports access to a number of well-known source catalogues in their original distribution format (or at least an uncompressed version of this): supported formats include those of USNO, GSC-II, 2MASS, Tycho, ACT, SAO, IRAS, and PPM catalogues. Essentially the WCSTOOLS scat program has a back-end matching the schema and structure of each of these, as well as its own (even more gloriously ad-hoc) formats for textual and binary data. The only accelerated access method used in WCSTOOLS is slicing and sorting. For example 2MASS is sliced into separate files each covering a strip of declination 1.67° high, and USNO-B1.0 is sliced into strips 0.1° high. Within each strip the rows are sorted by Right Ascension. This simple structure permits fairly rapid access provided the region of interest is small. Having been written by astronomers WCSTools has good facilities for things like precession, but there are no indexing facilities. It is not designed to execute spatial joins or complex selections on large tables.

BROWSE

This package was designed at ESOC around 1980 to access the archives of the Exosat Observatory. Software development continued at ESTEC and later GSFC/HEASARC, and the package has been used at several sites hosting X-ray data, although it is now almost obsolete. BROWSE also relies on catalogues being sorted on one coordinate: this allows it to support cone-search and other simple selections using its own command language or a subset of SQL. One notable feature is that it has good support for exploratory data reduction: the results of a selection can be saved for future use, and there are some basic graphical display facilities.

FITS-based files

The layout of a FITS binary table is actually very similar to that of a DBF file (a format introduced by dBASE-II and still in use by many PC products such as Paradox), and to the internal format of the files of MySQL, except that it can hold a greater variety of metadata. The collection of FITS utilities called FTOOLS (from HEASARC) can already perform some DBMS-like operations, such as selection and sorting, and adequate FITS browsers and graphical tools now exist in the form of FV and TOPCAT. It would not require a great deal of additional software to provide read-only database access to tables stored in FITS format. It would need an SQL parser and interpreter (but one was provided in later versions of BROWSE) and B-tree and R-tree indexing (which could make use of the open-source Berkeley DB and GiST libraries), but not much more. It would be very easy to access remote FITS files, as the cFITSIO library has access via FTP and HTTP protocols built in.

One great advantage of this would be that every FITS table that one could access would automatically become part of one's database: it would not be necessary to ingest it explicitly. One of the most frustrating features of regular DBMS is that every table has to be loaded, usually after converting it to some simple text format such as CSV. The data loading stage is needed because these DBMS store their limited metadata (column names, data types) internally in system tables. With a FITS-based system, these metadata are an integral part of the system, and so the import/export steps can be abolished. Access could, obviously, be extended to VOTables, but the advantages of efficient access would be lost for VOTables which used the XML TABLEDATA format, as there is no regular stride to the data values.

9.2 Object-oriented DBMS

The data structures which astronomers want to store are often intrinsically hierarchical: for example the XMM-Newton results database is divided into units of observations, each consisting of several exposures, these have data from six instruments, each having a number of separate data files. Storage of such structures in a fully normalised set of relational tables is much less natural. This led the XMM Survey Science Team to consider the attractions of OO-DBMS, which support a much wider variety of data structures. Another advantage of the OO-DBMS is that connections between related items in tables can be made simply by following pointers, whereas in a relational DBMS they require explicit *join* operations, which are computationally intensive. The schema of an OO database can also be designed with any required collection of astronomical metadata.

There are also some drawbacks to OO-DBMS:

- They have a very small market penetration, which means that they are expensive, and not as robust or mature as their relational equivalents.
- They come with a rather limited set of utilities such as graphical front-ends and software development tools.
- Queries use OQL, a poorly-standardised language which somewhat resembles SQL, but requires an additional learning effort.

- A working system requires a considerable amount of custom code to be produced, usually in an object-oriented language such as Smalltalk or C++.
- The schema of the database is embedded into the structure of the database, and compiled into the access code. Subsequent changes to the schema can be very disruptive, whereas the corresponding change to a relational system, adding a new table or column, would have little or no impact on a running system.

L'histoire d'O₂

The XMM-SSC team, led by CDS, evaluated a number of relational and OO DBMS and finally chose for project's internal use a pure object-oriented database product called O₂ which was a spin-off from the French informatics institute INRIA. This had the advantage of allowing programming in a version of C called O2C, and promised easy schema modification. It was in fact a good choice on technical grounds, though we encountered a few serious bugs. Otherwise things were much less rosy. The O₂ company was taken over by Unidata, which in turn became part of Ardent Software who decided to make O₂ a "mature product", and promptly discontinued development. They continued to take our money for "maintenance", however, but were soon taken over by Informix, who were then purchased by IBM. This gave IBM more DBMS than they wanted, and O₂ was one of those hived off into a separate company called Ascential. Maintenance, in the form of vital bug fixes, was eventually provided by a separate company called Oxymel although at a much higher price. The long-term future of the product is very uncertain, and the XMM-SSC project is seeking ways of extracting itself from the mess. The moral of the story is that reliance on a commercial product does not guarantee its future; an open-source product in this respect might have enjoyed cheaper and better support, with more of a long-term future.

Objectivity/DB

The Sloan Digital Sky Survey data processing system, led by Johns Hopkins University, was originally based on the use of Objectivity/DB, which they chose for similar reasons. This too was not a success: the performance was poor, they found serious bugs which were not fixed quickly enough, and had to provide their own query language parser to overcome defects in that provided. Eventually the problems became too great, and they switched to a relational DBMS instead, choosing Microsoft SQL Server. The transition seems not to have been too painful, and they claim to have experienced a very substantial performance improvement for many types of query. Further details are given in [5] and [6].

Conclusion

Although object-oriented DBMS still look attractive for certain applications, they have failed to gain a significant market presence, and the VObs is not an area where they would be a sensible choice.

9.3 XML DBMS

The use of XML as a glue between disparate applications makes good sense, and the VObs has taken advantage of the expected wealth of tools available for handling XML datasets by devising VOTable. A large number of "XML databases" is now on the market, but the term covers a wide range of products. Only a few are true XML database managers, others have a thin XML-speaking layer on top of a standard relational system. Efficient storage of large tables is unlikely to be one of the design aims in either case, and these do not merit further examination for the general needs of the VObs. There is one area in which an XML-DB may be useful, that is in the Resource Registry, which has to contain a relatively small amount of data, but where the structure is potentially quite complex.

9.4 Object–relational DBMS

The term object–relational has no generally accepted definition, but was coined back in the 20th century when the term "object oriented" was still popular. It is generally claimed by relational DBMS which can also handle user–defined data types, i.e. data structures. Most of the leading brands such as Oracle and DB2 now claim to be fully object–related, and so is PostgreSQL, the leading open–source DBMS which was designed from the outset to be object–relational. Since astronomers sometimes do want to store simple data structures, such as arrays of data, or coordinate pairs, these are to be welcomed.

9.5 Column–oriented Storage

If tables were stored with all the values in one column being contiguous it would obviously speed up selections that are not index–assisted but which only reference values of a few columns. The reason why nearly all DBMS use row–wise storage is that it is extremely expensive to perform inserts or deletes on column–oriented storage, but that is not a problem with a read–only application such as source–catalogue access.

Sybase IQ

The leading commercial product which does this is Sybase–IQ. This is a product of the Sybase company, and understands SQL, but is otherwise unrelated to their relational DBMS. It is designed for the commercial data warehouse market, and besides column–based storage, it implements a number of advanced indexing and data compression options. We have been involved in an informal evaluation of the product recently, led by Jim Lewis of IoA Cambridge: initial results are good for many types of query. Unfortunately there is at present no version for Linux (it was tested on Sparc/Solaris), it has no spatial data support. As one expects from a product of this type, licences are extremely expensive even with academic discounts. If it turned out that there was sufficient demand for the execution of the types of query that it accelerates so well, then perhaps there might be a case for installing it on a single node, accessible to VObs users over the network.

The advantages of column–oriented storage can be realised other ways: some experiments suggest that one can obtain a useful speedup if each column in a catalogue is stored as a separate one–column table in a simple DBMS such as MySQL. Of course the SQL for all queries becomes much more complicated as a result.

Given the performance advantages of column–based storage, I realised that the FITS Standard was broad enough to encompass something equally efficient: a table can be stored as a single row of a binary table, with each column being a vector of the same length (the cardinality of the original row–based table). It was easy to produce a large table in this format, together with a program to access it, using the cFITSIO library. The performance was indeed very much better for simple one or two–column selections.

The test query was to find the mean and standard deviation of a single column of a table, which was a sample of 3.5 million rows of USNO–B. This was tested last year on a 450 MHz PC, so the results are not directly comparable to others reported later.

DBMS	Time, seconds
DB2	9.0
MySQL	7.3
cFITSIO – row–oriented	2.0
cFITSIO – column–orientated	0.7

There are other ways of implementing column-oriented storage in a FITS file, for example putting each column in a separate header-data unit. This would be more compatible with existing software. It would also be possible to take advantage of data compression options with each data unit holding only a single data type, which has the potential for significantly reducing the number of I/O operations. The cFITSIO library already has good support for data compression. Current DBMS do not seem to have made use of data compression, perhaps because they were designed in the days when processors were too slow to do decompression on the fly.

10. DBMS Evaluations

The initial list of DBMS that we considered for evaluation is given below, with some initial assessment of their potential value in astronomy based on published documentation and in many cases discussions with users.

Commercial DBMS

Ingres: used at MPE and Leicester for some years; no outstanding features; no spatial indexing; poor market share.

Sybase ASE: used by CDS, Leicester, CfA, and many other sites; above average feature set for scientific data, but spatial data support is in an add-on product. There is a free version for Linux (but less up-to-date and without spatial indexing).

Oracle: current market leader, object-relational, very expensive list prices for licences. Used mainly at sites where a site licence is available already. Spatial indexing in an add-on product. Some missing functions in the SQL dialect, e.g. degrees, radians, pi.

SQL Server: easy to install and use, but no spatial indexing, and can be run under Windows only.

Informix: good spatial indexing built-in, based on R-trees, but product now owned by IBM and has an uncertain future as it competes with IBM's own DB2.

DB2: IBM's heavyweight object-relational product; spatial extender based on multi-level grid files is an add-on; free for use by registered academics under the Scholars' Programme.

Open Source DBMS

Firebird: the open source version of Borland's Interbase, now withdrawn from the market. Reputed to be a good stable product, but no outstanding features, and no known users in astronomy.

MySQL: lightweight but reputed to be fast, used by many astronomical sites; latest release has R-tree indexing built in.

PostgreSQL: full-featured object-relational DBMS originally from UC Berkeley where Ingres was invented; has R-tree indexing built-in, and a growing number of enthusiasts in the astronomical world.

SAP-DB: another former commercial product now released into the wild, but likely to be merged with MySQL soon.

10.1 Packages Evaluated

In 2002 we selected for basic testing the following: Sybase ASE, Oracle, SQL Server, DB2, MySQL, and PostgreSQL. The results of our tests are available on the AstroGrid wiki. In 2003 we selected for further evaluation with larger datasets the two most promising open source DBMS: MySQL and PostgreSQL, and for comparison one commercial product, DB2, since it had spatial indexing available and was available free of charge.

10.2 Desirable Features

Firstly we wanted to compare spatial joins using R-tree (or other spatial) indexing with the pixel-code algorithm. The results of this might determine whether a DBMS with spatial indexing was desirable or not.

Secondly we wanted to evaluate the short-listed DBMS for the following (in no particular order of importance):

- Ease of installation and use
- Ease and speed of data loading and export
- Performance on simple selection queries
- Performance on spatial joins and cone-searches
- Suitability for scientific queries, e.g. good selection of maths functions, easy to add user-defined functions

The detailed results are given in Appendix A below.

10.3 Summary

The spatial join is clearly performed better by an R-tree index than the pixel-code method; it has the advantages of speed, simplicity, and flexibility.

The R-tree indexing in MySQL is still in its infancy, and not a serious competitor to Postgres, and it has a number of other minor defects compared to Postgres. DB2 is designed for an environment where an experienced team is available to set it up and tune it. For data centres with limited manpower, Postgres is very much easier to install and use; it is a well-supported powerful DBMS with adequate performance, and no really serious faults.

11. Summary of Recommendations

Structure of the Virtual Observatory

Given the current relative prices of processing power, data storage, and network bandwidth, a computational grid is cost-effective only for problems requiring over 100,000 operations per byte, a level that astronomical data reduction and analysis rarely reaches. The data grid model is, however, very appropriate for the Vobs.

There is likely to be a significant increase in the total number of data centres: most of the new ones will be small. Commercial DBMS tend to be very expensive, so that cheap or free DBMS are likely to be given preference.

Because of limited bandwidth, large datasets need to be processed in situ if possible, with only the results sent back to the user. Large data centres should be encouraged to provide processing power, software, and temporary data storage for external astronomers to make the best use of their holdings.

The IVOA members have made excellent progress on the development of common standards such as VOTable and ADQL, but a lot remains to be done, particularly in developing advanced query languages and defining and supporting metadata.

Indexing and Cross-matching

Many current data centres use DBMS with just one-dimensional indexing for their source catalogues: this is very inefficient, but acceptable for simple services such as cone-searches. Cross-matching is much more data intensive, and a spatial indexing system is highly desirable to support datasets of more than modest size. The pixel-code algorithm, which can be supported by almost any DBMS, is satisfactory on medium-sized datasets, but scales poorly. Spatial joins based on two-dimensional indexing are simpler, faster, and more flexible. Several modern DBMS support spatial indexing, mostly using R-trees.

The power of distributed databases is inherently limited by the available bandwidth over wide-area networks, and the latency of these links. In many cases a distributed cross-match could be performed most efficiently by copying the whole of the smaller dataset across the network to the DBMS holding the larger dataset and performing the cross-match there.

Data Exploring and Mining

Many data mining packages are available for commerce, although hardly any are designed for the scientific user, who wants to be able to explore interactively large disc-based datasets. Some further effort on evaluating the most promising packages is likely to be worth-while.

The SQL language, being results-based rather than procedural, is a particular obstacle to the use by scientists of database technology. The lack of graphical and statistical facilities in DBMS is a serious shortcoming: external packages will have to be used, but data exported from DBMS lose valuable metadata. Methods of integrating such packages with DBMS still need to be explored.

Column-oriented storage has advantages for data-intensive querying, and products such as Sybase-IQ are promising for some specialised applications. Object-oriented databases look attractive in theory for some applications, but have been found wanting in practice.

DBMS Packages

IBM's DB2 package comes from the mainframe world and needs more expertise for its installation and tuning than many sites will have available. Of the packages examined, PostgreSQL seems most suitable: it is reliable, well-supported, powerful, and easy to install and use. It is a bonus that it is an open source package, and therefore usable without cost or licence restrictions.

Appendix A: Evaluation of Postgres, MySQL, and DB2

A.1 Documentation

Postgres

Postgres has four main on-line manuals: tutorial, administration guide, reference manual, and programmers guide. These can be downloaded in PDF or HTML forms, or searched on-line. There is also an attractive interactive version, somewhat wiki-like, in which one's own comments or questions can be appended to the official contents. I posted a question which resulted in several interesting responses by email. There are also two good textbooks on-line. Many Postgres books are also on sale in bookshops; after browsing a number of them in Foyles, I purchased *PostgreSQL Developers Handbook* by Geschwinde and Schönig, as it was just about the only one to even mention advanced topics such as R-tree indexing. I found this quite useful. The on-line help is useful but rather basic. Software support comes from a number of newsgroups, which are active and helpful. But because these newsgroups have not been properly registered in the Usenet hierarchy the Janet Usenet service has refused to make them available. Actually one group, comp.dbms.postgres.hackers, is provided, probably by accident, but I had to use an alternative news server to see the others. The reference manual is particularly good in telling you not only the exact syntax and facilities of each SQL command, but the differences, if any, between it and Standard SQL92. Documentation on how to use R-tree indexing was somewhat brief, and I had to do some browsing of help files and experimenting to work out which operators on spatial objects were optimised by the SQL engine. The documentation has been improved since then.

Mysql

There is one large reference manual which can be downloaded from the website of mysql.com. I also purchased two books out of the many on sale: *MySQL Reference Manual* by Widenius and Axmark (the principal authors of the package), and *Core MySQL* by Atkinson. These were both quite useful. The documentation on geometric data types and R-tree indexing only appeared on line just before I started my tests, and was somewhat sketchy, but just enough to get me started. The on-line help facilities of the client are pretty basic. Software support, for those using the free version, comes from groups on the mysql.com website. When I had problems loading spatial data I asked about external file formats; my question was not answered satisfactorily, perhaps because there are as yet very few users (but nobody from the MySQL company replied either).

DB2

The problem with DB2 is the sheer volume of documentation, which often defeated my attempt to find a particular item of information that I was sure must be present somewhere. One volume contains the index, but reading an index as a PDF file is very tedious. There are 49 main manuals, supplied as PDF files, several with over 700 pages. I found it hard to search the PDFs on line, and sacrificed a small tree to print out four of them and parts of a few others. Even so these make a large pile on my desk. I found that a user called Graeme Birchall had published on-line a book called *DB2 UDB V8.1 SQL Cookbook*, which contained a lot of useful information on advanced SQL techniques. Also on-line are some useful IBM "Redbooks": I printed out *DB2 UDB Evaluation Guide for Linux and Windows* and *Up and Running with DB2 for Linux*, not much over 300 pages each. Between them they gave me basic installation instructions, though these proved confusing and inadequate. I could find very little in the way of introductory or tutorial material. I kept coming across phrases like "ask your administration team to install this additional product for you". If only I actually had an experienced DB2 administration team on hand. Jeff Lusted, who has some experience of using DB2, said that I should try to get hold of the HTML version of the documentation, as the cross-links make it much easier to search. I could find manuals in HTML format only for versions of DB2 earlier than V8. The various DB2 clients provide some on-line help, but nothing that I actually found useful. Because these documents were so difficult to use, I decided to buy a book published by IBM called *DB2 Universal Database V8 Handbook for*

Windows, Unix, and Linux. This was quite useful in itself, but also for its pointers on where to look in the huge manual collection. We had earlier purchased an O'Reilly handbook called *SQL in a Nutshell* which is a compilation of SQL dialect variations. It is a pity that it covers the quirks only of Oracle, Microsoft SQL Server, MySQL, and Postgres, ignoring DB2. Perhaps this reflects the relatively low profile of DB2 in the market.

It was not clear to me whether, as someone not paying for the product, I qualified for direct support from IBM. There are, however, some Usenet groups covering DB2, including one specifically for the DB2 Spatial Extender. When I asked a question, I got helpful answers from two members of IBM staff, and one person who appeared to be just a helpful user. The very low traffic on this group suggests either that the Spatial Extender has very few users, or that nobody has problems with it.

A.2 Installation, Configuration, and Tuning

Postgres

I found the RPM for Redhat Linux, downloaded it, and installed it without problems. A few minutes work with the manual showed me how to disable all the access restrictions and set the server to use our RAID array for all its files. I read a number of documents on tuning Postgres, and made a few changes when I found it was using only a rather small proportion of our one gigabyte of memory, but never managed to detect any significant speed improvement. Unless I missed something important, it looks to me as if Postgres comes reasonably well tuned.

MySQL

I found the RPM for Linux, downloaded it, and installed it without problems. But configuring it took a lot longer: the RPM was set up to put all its database files on the /var partition which would not have enough space. The server appeared to have a command-line option to change this, and also an environment variable. I tried both of these in various combinations without effect. Eventually I discovered in the manual that the only way to change the disc usage setting was to download a source-code version of MySQL and compile it myself with an altered configuration file. This seemed daunting, and I searched for help on the newsgroups. As nearly always, the problems I encountered with all three DBMS were common ones, and very frequently asked on the newsgroups. Why the software producers take no notice of these common problems I cannot understand. From answers in the newsgroup I discovered that there was in fact a work-around without re-compiling: create a database, move the entire directory to the location I really wanted (this needed root and lots of file permission changes), and then insert a soft-link from the old location to the new one. This worked. I did not find any mention of tuning in the MySQL documentation, and did not do any. It seems to come ready to use.

DB2

Downloading DB2 RPMs requires logging in to the IBM Scholars web-site and entering one's user-name and password several times over. Eventually you find a huge list of DB2 products, and I downloaded the Personal Workstation kit for V8.1. This was a huge RPM, around 400 MB, and it took at least five attempts to get a complete download.

Installation was then something of a nightmare. For a start it overwrote some files from an earlier installation which I wanted to keep. It requires JDK v1.3 whereas we already had v1.4.1 installed, and I kept having to change environment variables to get the old version picked up. DB2 also requires three specific user-names to be set up: db2inst1, db2fenc1, and dasusr1. The installation wizard claimed it would create these users, but it did not: I discovered afterwards that on systems running NIS this has to be done by hand. I spend several

hours trying to get my own user-name granted sufficient privileges to be able to use DB2, but failed, but in the process messed up some of my own shell's settings. The new users were set to use Korn shell, which was unfriendly. I eventually carried out all my tests of DB2 while logged in as db2inst1, and did cuts/pastes from/to my own account for logging etc. It would be excessively tedious to report all the problems I had, no doubt some due to my own lack of experience, but I am sure that it always takes a great deal longer to install DB2 than either of the other packages.

The relatively slim IBM book on DB2 contained three whole chapters on tuning, so it is obviously an important topic. On quickly browsing these, it appeared that transaction processing was the main area needing to be tuned. Since I ran out of time and had no interest in transactions, I made no attempt to tune or optimise my DB2 queries.

A.3 General Usability

Postgres

It was easy to start and stop the server, and I could do this without logging in as root. The client was equally easy to start and stop. A long-running query could be cancelled just by typing ctrl/C – this caused the current query to be rolled back if it was doing an update, or otherwise stop quickly and cleanly. There seemed to be no ill effects from aborting queries. The command-line interface, psql, made it easy to recall and edit commands. Multi-line commands could be entered fairly easily. The error messages were good, telling me what was wrong and which part of the SQL statement had caused it. There was a command \timing to toggle timing information given when each operation concluded. The user-interface allowed command-line recall and editing, and external SQL scripts. Although I found Postgres to be generally satisfactory, there were a few quirks:

- After creating an index it is necessary to perform an ANALYZE operation, otherwise the index is not used in subsequent queries. In dynamic environments the operation has to be repeated regularly, but for a static table once is enough. Index creation takes a bit longer because of this, the times were included in those reported here.
- Floating-point constants are taken as double-precision by default with no automatic casting to other floating-point types. This causes problems when using an index on a real column. Thus the syntax `SELECT * FROM table WHERE vmag BETWEEN 12.5 AND 12.6 ;` would not use an index on the (real) column vmag, it needed to be re-phrased thus: `SELECT * FROM table WHERE vmag BETWEEN 12.5::FLOAT4 AND 12.6::FLOAT4 ;`
- In a few cases I found that queries took longer when an index was available than when it was absent, or when two indices were provided than when only one was provided. It is possible to influence the query plan, but I did not have time to explore this fully.

It is notable that the most common problem posted to all DBMS newsgroups appears to be a complaint that the DBMS is not using an index when the user expected it to be. The EXPLAIN command in Postgres at least gives some information on how it thinks any given query is best evaluated, and some tools to influence its optimiser.

Postgres allows user-defined functions to be specified in a number of languages, including its own procedural version of SQL called PL/PGSQL, and used easily in SQL queries.

MySQL

It was necessary to log in as a privileged user to start the server, but, oddly, not to stop the server, nor start/stop the client. Aborting a query did not seem to be possible without crashing the client, but it was usually possible to re-start it without serious ill-effects. The SQL implemented by MySQL is less complete and less standard than for the other DBMS, though every release brings improvements. Some sub-selects now seem to be supported. There are various back-ends: the one I used was the default one which does not fully support transactions with roll-back, since that was not necessary for our requirements. Data types were a problem: on Postgres and DB2 REAL is a 4-byte type, but FLOAT an 8-byte type; with MySQL the reverse was true. I found that the only way to get a 4-byte and 8-byte floating type for the same columns in MySQL and Postgres was to use non-standard types called FLOAT4 and FLOAT8. MySQL provides the elapsed time of each query by default. The user-interface allowed the usual command-line editing and recall, and use of external script files. MySQL also provides an EXPLAIN command, although the output is not quite as easy to interpret. There does not seem to be any way of influencing the query optimiser, should the explanation be unsatisfactory.

MySQL only supports user-defined functions if written in C, and a privileged account is then needed to link the compiled version with the MySQL executable.

DB2

There are three different user-interfaces, all of them with their own limitations:

- The db2 command provides a simple way of entering SQL, here the lines need no semi-colon at the end, so therefore long lines need a continuation marker. I could not find this documented anywhere, but found by trial and error that "\" worked. The db2 interface provided no execution or elapsed time information. There was no command-line recall or editing, no doubt designed for the user who always manages to get multi-line SQL commands perfect every time: I wonder if such users exist?
- The db2batch command seemed similar, but had the bonus of providing elapsed time information. This could be augmented by cpu-time if db2batch was started with the right command switch. The cpu-time was listed anyway, but was always zero without this switch: why it was not the default is hard to understand. The other switches for db2 and db2batch were quite different for identical functionality, presumably written by different teams who did not bother to talk to one another. Db2batch required a semi-colon, but no continuation marker. It also omitted to provide command-line recall and editing. I would have used db2batch all the time, but it did not support all SQL commands, only those in some (undocumented) core set. The data loading commands had to be issued from db2, not db2batch.
- The db2cc command invoked something called the "control centre" – a GUI which allowed other GUIs to be started, including the "command centre". This also allowed SQL commands to be entered. These also required a semi-colon, so needed no continuation marker. This would have been a good interface, had it not been programmed incompetently in out-of-date Java. I found that it did not support cut/paste to/from other windows on my screen. From critical comments on various newsgroups I guess that db2cc is not widely used.

Other problems I had with DB2 include the following:

- There seems no way in a single SQL statement to select from one table and have it generate a new table as the result. This is not required by SQL Standards, but is possible in Postgres (using SELECT INTO table) and in MySQL (using INSERT INTO table SELECT). In DB2 it required two statements, one to create a new table (with all the right column names and data types) and then a second one to do the INSERT INTO. This is because, I guess, the normal way SQL is used is just to

produce text; the idea that one might want to make a selection and then work on it further is alien to the SQL way of thinking, though this is just what scientists want to do all the time. The lack of such a facility in DB2 is unfortunate (and meant that all my SQL scripts had to be altered yet again).

- I could not find any way of stopping a run-away query. Control/C and the like had no effect. If I killed the process it seemed to prevent me using database and when loading it left it in an inconsistent state. I generally had to stop and re-start the server (using a different user-name, naturally). In one case this did not work, and I had to re-boot the machine.
- The transaction log filled up unexpectedly and stopped progress. I could not find out how to eliminate these logs nor how to disable logging, but eventually discovered how to make more space for these logs, and I increased it by a factor of one thousand. The documentation explains that several of these parameters are set by default to values too small for modern computers. This is a pity.

There is no EXPLAIN command in DB2, but there is a separate product called Visual Explain. I did not have time to try this out.

DB2 allows user-defined functions in several languages, including a procedural version of SQL.

A.4 Data Loading

Postgres

Loading data was simple, especially since the 2MASS data files were designed for use with Postgres. I saw in the manual that a data load operation is performed within a single transaction, so it is all rolled back if anything goes wrong, but I saw no errors. A typical load command is:

```
COPY mytable FROM '/home/cgp/db/data2/psc_baa.csv' WITH DELIMITER '|' ;
```

Nulls were represented by \N in the 2MASS files, the Postgres default, and I used this in the data files I prepared myself from the raw USNO-B distribution.

MySQL

Loading non-spatial data was just as easy, since MySQL seems to use the same defaults as Postgres. The load syntax is not something that the designers of SQL have ever bothered to standardize, and MySQL requires a slightly more verbose syntax:

```
LOAD DATA INFILE '/home/cgp/db/data2/psc_baa.csv' INTO TABLE mytable FIELDS TERMINATED BY " | " ;
```

DB2

Here the serious problems began. I searched the two volumes of SQL commands for a long time without finding a data load command, and the Redbooks were no help. I suppose DB2 is almost always used in a transaction-processing environment where data is captured live, and rarely needs to be loaded from external files. I finally discovered a separate PDF manual called *Data Movement Utilities Guide and Reference* a mere 429 pages long. It eventually yielded up the secrets of the copy command. I then found that it could not be used from the db2batch environment, or the db2 gui, only from the db2 command-line interpreter. The typical syntax is:

```
LOAD FROM '/home/cgp/db/data2/psc_baa.csv' OF DEL MODIFIED BY COLDEL| INSERT INTO mytable ;
```

But data loads failed repeatedly. At first I thought that it was the date/time fields which were causing problems: with 60 fields per line and hopeless error messages, it took me some time to eliminate each possibility. Tests with samples of a few hundred or thousand rows sometimes worked fine. Eventually it seemed that the nulls were not accepted. The manual suggested (without being definitive on the subject) that there was no external representation of nulls, except for the absence of data between two field delimiters. Unfortunately every such null-by-default gave rise to a warning message. I left one data load running when I went home at night; the messages were still scrolling off my terminal screen at the rate of hundreds per second when I arrived the following morning. I had to abort this run, which left the database in a mess, and eventually I had to reboot the machine. Since all the nulls were in numerical columns (magnitudes) I found a work-around by using the Unix tr utility to convert each null into the value of "-1" which was out of the normal range. By this means I finally managed to load some strips of the 2MASS data. I did not get around to converting these values back into nulls, though I suppose that an UPDATE statement would have been able to do that. Even so each load operation generated lots of incomprehensible messages, of which this is a typical example:

```
SQL3501W  The table space(s) in which the table resides will not be placed in backup pending st
since forward recovery is disabled for the database.
```

Although I could not get timing information as with other DB2 commands since the db2batch facility would not load data, the LOAD command reported the clock times at the start and end, so the performance could be measured. The only good point of DB2's data loading was that it was fast, and I noticed that it used the cpu time of both processors. The times taken to load 5.1 million rows of 2MASS were as follows:

Postgres	MySQL	DB2
116 secs	205 secs	44 secs

Given that nulls are an essential element of nearly all astronomical tables, and that finding a suitable out-of-band value to represent them is a non-trivial undertaking, I think that if it is true that DB2 has no satisfactory way of representing nulls, this omission forms a very serious obstacle to its continued use. But the loading performance if DB2 is very impressive.

A.5 Spatial Indexing

Postgres

Loading spatial data from a text file required a slightly odd format with extra parentheses, but I had no trouble in loading the required spatial column, nor in writing a program to convert USNO-B data to the required form. The basic spatial object is a rectangle, specified by the coordinate pair (x,y) of two opposite corners, and one can create an R-tree index on these bounding boxes. The SELECT statement can use operators (such as &&) to test for overlap of the rectangles. It was easy enough to read the rectangle coordinates from an external text file, or to generate the rectangles on-line using an UPDATE statement (which is obviously simpler). There is little to report here: spatial indexing worked as expected, and was reasonably fast.

The simple R-tree provided by Postgres indexes a rectangle, the dimensions of which have to be rather stretched when encompassing an error region at high declination. A better way of doing this is promised by two developments from the Sternberg Astronomical Institute in Moscow, pgSphere, which will provide a 2-d index on a surface covered by spherical-polar coordinates. A related project, pgSphere, adds certain

astronomical functionality to this. Nothing comparable to this is available for any other DBMS, as far as I am aware.

MySQL

MySQL has only added spatial data support in its latest (alpha) release, and it shows many omissions which will no doubt be rectified over in future releases. The software is aimed at the GIS market, and uses R-trees for indexing. The documentation was very sketchy, and some trial and error was needed to discover what actually worked. The first problem was to load the data, as no external format for the spatial data type was documented. I asked in a support newsgroup, but got no answer. I think this may be a feature still missing. The only alternative was to use INSERT statements to load the spatial data, but loading one row per INSERT would have taken far too long. I found by experiment that I could not load a large table in a single INSERT statement, but that 1000 rows/INSERT appeared to work. I therefore write a new utility to convert the USNO and 2MASS data files into the appropriate SQL INSERT statements. These were *very* verbose. Finding an intersection or overlap was also much more verbose, as there were no operators handy, only functions. This verbosity might have been acceptable had the performance been reasonable, but as noted below, the MySQL spatial index was many times slower than that of Postgres.

DB2

DB2 spatial indexing comes in an additional package, called the Spatial Extender. I downloaded it with difficulty (another 300 MB of software and flaky FTP connection) but it eventually installed correctly. But it would not work. I eventually discovered, via the control centre, that it was not licenced. I asked the IBM Scholars Programme administration how I got a licence and got an answer which was very helpful (but only after a week). It turns out that the licence can be downloaded from their web-site (indeed I had already done this) but it needs to be copied to a new location and then a licence installation procedure is necessary. Unfortunately by the time I got all the necessary details, I had insufficient time to complete testing this feature. The manual is fairly comprehensive, but at 574 pages, as intimidating as the others.

One immediate difference is that IBM's Spatial Extender does not use R-tree indexing, but a multi-level grid file. These are not self-adjusting like B-trees and R-trees, but require the user to consider the range and granularity of up to three levels of the grid-file. Although the data are expected to be floating-point values (geographical longitude and latitude are expected), they are scaled and then converted to 32-bit integers. These scaling factors also have to be chosen by the user.

Another problem then presented itself, just as for MySQL, the lack of an external format for spatial data. I posted a question about this to the dedicated newsgroup `ibm.software.db2.udb.spatial`, and got useful replies, including a suggestion for populating the spatial object from the (ra,dec,radius) triplet. One has to convert all the numerical values into character strings because the polygon function will only take string arguments, which is very awkward.

It would be nice if the R-tree implementation in Informix were to be retro-fitted to DB2.

A.6 Performance: Joins using R-trees

This section used only MySQL and Postgres, as I did not have time to get DB2's spatial extender working. The main test joined a section of 2MASS with an overlapping section of USBN-B.

- 2mass: 5 million rows of data, file `psc_baa` covered 0 to 360 degs RA, and 0.0 to 1.67 degs Dec, but with just these columns selected: RA, DEC, position-error, three magnitudes, error-box.

- USNO–B: 3.6 million rows of data, covering 0 to 360 degs RA, 0.0 to 0.6 degs Dec., with these columns selected: RA, DEC, position–error, five magnitudes, error–box.

The tests were carried out on the main node of hydra, which has dual–Xeon 2.2 GHz processors, and 1 GB of memory, the times are in seconds.

Elapsed times in seconds	Mysql	Postgres
Load 2mass, 5 million rows	205	116
Load USNOB, 2.6 million rows	153	93
Create spatial index on 2mass	488	617
ANALYZE (needed for Postgres)		67
Create index on USNOB	347	442
ANALYZE		27
Join tables on 1" circle	4030	283
<i>total</i>	5223	1645

The speed advantage of Mysql at all the earlier stages was lost in carrying out the join. The support for spatial data and R–trees has only just been introduced, so it is perhaps not surprising that it has not yet been optimised. On these figures, Postgres is the clear winner.

A.7 Simple Query Performance

It also seemed worth comparing the performance of Mysql and Postgres with a more substantial chunk of data. These tests used ten files of 2mass data, totalling around 50 million rows, and the times are in seconds.

Operation	Mysql	Postgres	DB2
Load data	1660	4030	2154
Sequential scan for H_M between 12.010 and 12.011	192	264	1327
Create B–tree on column H_M (+ANALYZE for Postgres)	2207	1265	1709
Indexed access for H_M between 12.010 and 12.011	0.15	419	0.05

This Postgres result was so odd that it was repeated: subsequent indexed selects took 3.9 seconds, and 0.7 seconds. Presumably some information needed to be stored in the cache before the query was executed efficiently. It was not clear why DB2's sequential scan was so slow – the cpu seemed to be very lightly loaded while the query was running. Given the amount of documentation devoted to tuning, it is likely that tuning by an expert would produce a substantial speed increase.

Inner and Outer Join

The outer join (which copies unmatched rows from one table into the results) is important in astronomy, since unmatched sources are often of scientific interest. This test was only carried out using Postgres and R–trees: the inner and outer joins took about the same times, within limits of measurement.

A.8 Summary

IBM's DB2 has the reputation as a reliable product with good performance and it competes head–on with Oracle in the commercial market, but its sheer size seems to make it unsuitable for installation by the inexperienced and unaided astronomer. There are also a number of deficiencies in the friendliness of its user

interfaces. Although the spatial indexing was not tested, the use of a multi-level grid file of scaled integers seems much less attractive than the R-tree, which is self adjusting to one's data range.

At present MySQL is used by many more astronomical sites than Postgres, but the latter appears to be gaining ground, particularly for the handling of large datasets such as 2MASS. The figures presented above suggest that MySQL is faster than Postgres only in a few operations, and the speed advantage is with Postgres for complex operations or ones on very large tables. Postgres has somewhat better documentation and support for established standards. The spatial data handling of MySQL is not yet ready for serious use, but this feature was only introduced very recently. The R-tree indexing of Postgres is reliable and performed well. In summary: Postgres seems to have few shortcomings, and can be warmly recommended.

References

- [1] <http://wiki.astrogrid.org/bin/view/Astrogrid/ScienceProblems>
- [2] <http://www.wintercorp.com>
- [3] http://research.microsoft.com/research/pubs/view.aspx?tr_id=655
- [4] <http://wiki.astrogrid.org/bin/view/Astrogrid/SkyIndexing>
- [5] http://research.microsoft.com/~gray/papers/SDSSArchive_OO_to_SQL.pdf
- [6] <http://adass.org/adass/proceedings/adass02/O7-3/>