Exploiting Relational Database Technology in GIS

Peter Batty

All systems for managing data face common problems such as backup, recovery, auditing, security, data integrity and concurrent update. Other challenges include the ability to share data easily between applications and to distribute data across several computers, while continuing to manage the problems already mentioned. Geographic information systems are no exception, and need to tackle all these issues.

Standard relational database management systems provide many features to help solve the issues mentioned so far. This article describes how the IBM geoManager product approaches these issues by storing all its geographic data in a standard relational database product in order to take advantage of such features. Areas in which standard relational database functions need to be extended are highlighted, and the way in which geoManager does this is explained. The performance implications of storing all data in the relational database are discussed.

An important distinction is made between the storage and management of geographic data and the manipulation and analysis of geographic data, which needs to be made when considering the applicability of relational database technology to GIS.

Introduction

THIS ARTICLE looks at some of the issues involved in implementing a large corporate database containing geographic information. First a brief categorisation of 'non-database approaches' to storing geographic data is given. A description follows of the way that the IBM geoManager product approaches geographic data management using a standard relational database management system.

The article then examines a number of database issues, discussing the ways in which standard relational database functions can help a GIS address these, and also highlighting where extra functions are required for a GIS. In each case the approach taken by geoManager is compared with a non-database approach. Finally the performance implications of taking the geoManager approach are discussed, and the arguments presented are summarised in a conclusion.

Throughout this article the examples given refer to geoManager and DB2, IBM's relational database product which runs under the MVS operating system. The first release of geoManager also runs under the VM operating system using the SQL/DS database product. The arguments given are by no means specific to these database products but apply in general to any relational database management systems used in the way described in this article.

Alternative approaches to storing geographic data

Non-database approaches

Most Geographic Information Systems store graphics and some or all geographic information in specialised data structures, and store alphanumeric attribute information in a standard database management system, often a relational one.

Various reasons are cited for this approach, the most important of which is generally agreed to be performance. Relational databases are designed primarily for managing alphanumeric data, not for rapid display of graphical information or certain kinds of complex processing such as some types of geographical analysis, so there are issues to be addressed in this area, which will be discussed later.

Clearly these generalised statements cover a very wide range of approaches to storage of geographic informa-

tion. However, for the purposes of this article the important fact which distinguishes these approaches is that they do not store all aspects of their geographic data in a single database management system. In the rest of this article, these will be referred to as non-database approaches.

The GFIS database approach

IBM's Geographic Facilities Information System (GFIS) takes an object-based approach to storing geographic information, rather than a map-based approach. (Note that the term 'object-based', as used in this article, is not synonymous with the term 'object-oriented', which has a quite specific meaning in computer science. With an object-oriented system relationships between objects are stored as an integral part of the database. With the object-based approach referred to in this article, relationships between objects are maintained by applications using software tools which are provided.) All the information for an object, including alphanumeric attributes, geographical location, network relationships and graphical representation, is stored together as a single logical object in the relational database. The GFIS database has no system of tiling or pre-defined maps, just a continuous set of objects. The appropriate tables are created and managed by the geoManager product. Attribute fields stored in the database for each object include the following:

- Spatial index key – used to cluster data efficiently and optimise area retrievals (discussed in more detail in the section on performance).
- Object extents (min x, min y, max x, max y) – used for area retrieval.
- Absolute points and point connectors – network nodes which are used for network retrieval.
- Detailed graphics (for example the co-ordinates of a multi-point line) are stored in a compact binary format in a variable length character string.
- Alphanumeric attributes.

Geographic analysis and graphical display is not done directly against the geoManager data structure. An important distinction is drawn in this article between the *storage and management* of geographic data and the *manipulation and analysis* of geographic data. When looking at the manipulation and analysis of geographic data, different data structures are appropriate for different kinds of application. For example, the data structure used by IBM's Graphics Program Generator product (GPG) is extremely efficient for complex network analysis, while a raster or quadtree data structure such as that used by Tydac Technologies' SPANS™ [1] is much more efficient than a vector data structure for overlaying several sets of polygons. However, it is highly desirable to have some means of storing and managing geographic data in a way which allows it to be accessed by multiple geographic and non-geographic applications within a consistent environment. The geoManager approach is intended to provide an integrated corporate database which fulfils this role.

When a user wants to work with some geographic data, he makes a request to geoManager specifying the geographic area of interest and the objects required within that area (for example buildings, roads and sewers). It is possible for geoManager to extract the requested objects from the database in one of the following three formats:

[1] SPANS is a trademark of Tydac Technologies Corporation.

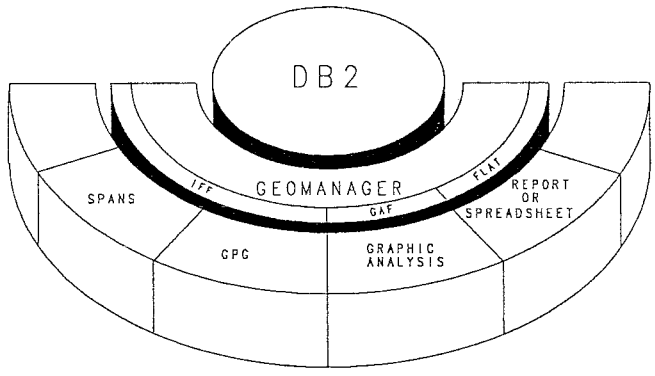


Figure 1. GFIS Architecture, including SPANS as an example of a third party GIS.

1. The Interface Format File (IFF), which contains graphical, alphanumeric and relationship information, and is used to pass data to GPG or other GIS packages for full function GIS applications.
2. The Open Format File, which contains just selected alphanumeric data, and can be used to pass data to other applications such as report generators or spreadsheets.
3. The geoManager Graphic Analysis Format, which contains just graphical data, together with a database key for each object which refers to the relevant attributes in the database. This is used by the geoManager Graphic Analysis application which provides functions to display, pan and zoom these graphics on any standard business graphics terminal, as well as the capability of viewing and updating alphanumeric data and performing various types of simple analysis and reporting.

When an IFF extract is requested, the user must specify whether any of the retrieved objects may be updated. If so then geoManager puts locks on the appropriate objects so that they cannot be updated by other users. This is discussed in more detail in the section on concurrent update. Any changes made in the GIS application are passed back to geoManager in an IFF, and this updates the database and releases the locks.

Database issues

Backup and recovery

Regular backups need to be made of any database system in order to be able to restore the database in the event of a major system failure. The larger the database, the longer it will take to back up, and therefore there may be a reasonable length of time between backups. In order to provide the capability of restoring a system to a state which is more recent than that of most recent full backup, database systems such as DB2 provide the ability to log all transactions against the database. This transaction log can be saved on tape much more regularly than a full backup can be done, since the data volumes involved are much smaller. When restoring a database from the back-up tapes, the most recent full backup of the database is restored, and the transaction log can then be applied to this to bring it to the most up to date state possible. DB2 also provides facilities to back up individual tables, or parts of large tables, so a full backup can be done more quickly by running concurrent jobs to back up different parts of the database.

Since geoManager stores all GFIS data in DB2, it can exploit all these capabilities automatically. However, a system which stored its graphics in a separate database would have to write a specific logging mechanism for graphical transactions (which included a mechanism for synchronising this log with the log for alphanumeric transactions), in order to provide the same recovery capability.

Security

In a corporate GIS with many users, it is vital to be able to control access to the GIS data. The three main types of access to data are update, read only, or no access. These categories may be subdivided further. For example, geoManager provides three types of update privilege: the ability to create entirely new data; the ability to make structural updates to existing data (i.e. change graphics or network connectivity); and the ability to update attribute data only. It is necessary to be able to grant these privileges at an object level, since some users may be permitted to update or view certain objects but not others. This requirement applies equally to both graphical and alphanumeric aspects of the data.

DB2 provides standard facilities to provide read, update, insert and delete authority on any table for any user. Since the three types of update required by geoManager do not correspond exactly to these DB2 update authorities, the geoManager application implements its own security mechanism at an object level. This security mechanism is complemented by the standard DB2 access control mechanism, which controls access to the data by non-geoManager users. In this way privileges for all users can be controlled in the appropriate way.

It is generally easier to provide a flexible approach to security using an object-based system rather than a map-based system. If the basic unit of information being handled is a map, or a layer of a map, then it is likely to be more difficult to restrict access to certain objects within a layer than it is if objects are the basic unit of information being handled.

Data integrity

There are many data integrity issues which need to be considered in a corporate GIS. This section looks at some of these issues, and the following section looks at concurrent update, which is one particular aspect of data integrity which is particularly complicated in a GIS compared to most applications.

A fundamental issue is ensuring that graphics and alphanumeric data are maintained in synchronisation with each other. At the very simplest level this means ensuring that when a graphic record is deleted the corresponding alphanumeric record is deleted, for example. This is not an issue in GFIS because of its integrated object-based approach. However, in a system where alphanumeric and graphical data are stored in separate databases, the system must ensure that this sort of synchronisation is maintained.

Synchronisation at this simple level should be straightforward to achieve, but there are far more subtle issues in the same area. Suppose that an operation has been carried out which changes both alphanumeric and graphical data for an object, and that in the process of saving the results of the operation in the database(s), the transaction fails. If at this stage the graphical change has been saved but the alphanumeric change has not, then

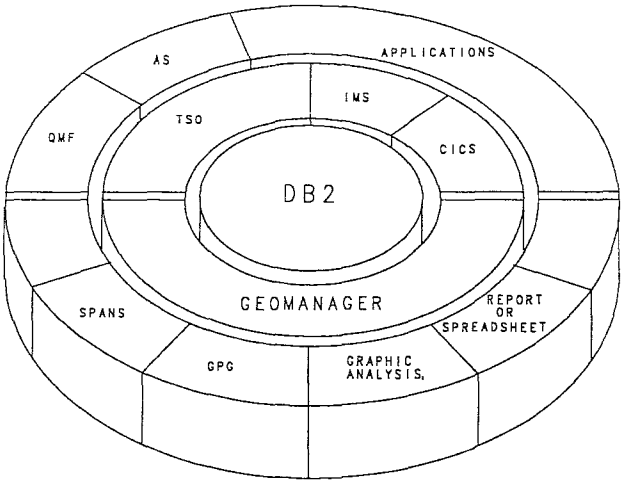


Figure 2. GFIS Architecture showing examples of integration with other applications.

the two are out of synchronisation. The system must be capable of recognising this, and either rolling back the graphical change which has been made or ensuring the alphanumeric change is also made. It would be extremely difficult to provide this level of data integrity in a system where the graphics are stored in their own data structure. However, DB2 provides function which geoManager exploits to provide exactly this sort of capability. When a group of related transactions are being carried out on the database, they will not be committed until they have all been completed successfully. If the update process fails at any stage then all the updates which have been made will be rolled back so that everything remains synchronised.

Data integrity issues also arise if relationships exist between objects which have been extracted from the database for update and other objects which have not been extracted. The geoManager system handles this by marking such objects as 'partially retrieved', which restricts the type of update operations which can be done on these objects in GPG. For example, if a pipe was extracted which crossed the selected area, and this was connected to pipes which were outside the selected area and therefore not extracted, then that pipe would be marked as partially retrieved. GPG would then not allow the ends of that pipe to be moved, for example, since that would cause the integrity of the network connectivity to be lost.

Concurrent update

A key issue in a large database with many users is the management of concurrent update problems. This is an area where GIS poses some more complex problems than simple alphanumeric applications. The native locking mechanism in standard relational database systems is based on the 'short transaction'. This means that if an application tries to access a record which is locked, the database management system will wait until the lock is released and then return the data to the application. The underlying assumption is that the application which has locked the data will only do so for a short time, a few seconds at most. However, a record which is extracted from the GFIS database for update may be checked out for hours or even days, so this approach is not appropriate.

Issues relating to such 'long transactions' are discussed by Newell and Easterfield (1990). They propose a solu-

tion based on version management, which is essentially an 'optimistic approach' which allows multiple users to concurrently update their own copies of a set of data and store both these versions in the database, on the assumption that in the majority of cases there will probably not be any conflict between versions. This approach allows great flexibility, but also has some risks. The main risk, as they acknowledge, is that the amount of work submitted before a conflict is discovered could be large. One of the implicit assumptions underlying optimistic approaches to concurrency control is that it should be relatively easy to modify and re-submit an update job if a conflict occurs. In GIS this is not generally the case, and weeks' work could be wasted. In the worst case it could be that important decisions had been made on the basis of inconsistent information. This may be an unacceptable risk in many organisations.

The geoManager locking mechanism

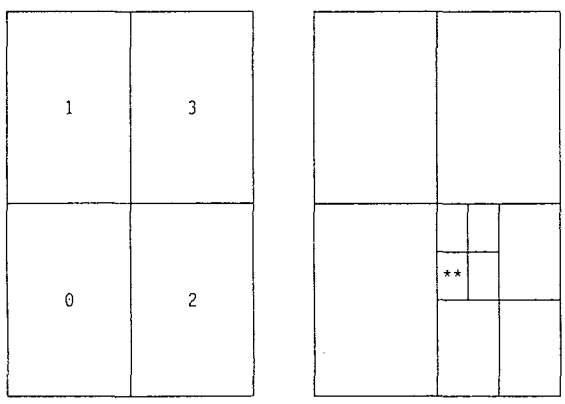
A different approach is taken by geoManager, which provides some of the features of version management but which does not incur the risks just mentioned. Whenever a user requests some data from geoManager to be put into an IFF, he is asked whether he wishes to update any of this data. If this is the case, then he is asked to indicate which objects he wishes to update. The specified objects within the extraction area are then marked by geoManager as checked out for update (a special status attribute in each object, which is described in more detail below, is used for this purpose). The user is asked to give details of the nature of his or her update for the benefit of other users. Other users are still free to extract this data in read only mode, but are not allowed to extract any objects for update which are already checked out. If an attempt is made to extract an object which is already checked out for update, then the requestor can find out who has checked the object out and obtain any information which was entered about the nature of the update. The requestor can then contact the person who has checked out the objects if necessary.

The fact that the database is object-based means that the minimum possible amount of data can be locked in order to try to minimise contention. In order to further reduce the possibility of contention, geoManager provides the capability of carrying out short geographic transactions using geoManager Graphic Analysis. When an extract is made to Graphic Analysis Format, only graphics are extracted, together with a key for each object which refers to the appropriate set of attributes in the database. No objects are locked at this stage. Using Graphic Analysis the user is restricted to performing attribute update operations, which can be handled as short transactions. A user can point to an object on the screen whose attributes are to be updated. The attributes for this object are then fetched from the database and the object is marked as checked out (assuming of course that it is not already checked out by someone else, in which case the user can view the attributes but not update them). The user can then make any updates and these are returned to the database and the object is checked in again. This ability to handle both long and short transactions is extremely useful in minimising contention. Furthermore, the same checking out mechanism can also be used by other applications to enable them to directly update data stored in geoManager, while ensuring that data integrity is maintained (this is discussed in more detail in the section on application integration).

Version management

The GFIS approach also allows a degree of version management at two levels. First, a GPG user who has extracted an IFF for update can save a copy of this data in GPG workspace format in his own personal storage. He can then create various alternative designs and save each of those in separate workspaces. He could go on and create further alternatives from any of these alternatives, creating a set of workspaces which correspond conceptually to the 'version tree' described by Newell and Easterfield. Each extra workspace duplicates data, but this will only be temporary. These workspaces are all accessible only by this user, or by any other user he chooses to pass them to. The only information the database has about this transaction at this stage is the set of objects which were originally checked out by the user. When the user decides which alternative should be passed back to the database he creates an IFF from the appropriate workspace and passes it back to geoManager.

It is at the stage of updating the database that the second level of version management can be used. When



code	level	code	level
000	0	030	2
000	1	030	3
000	2	031	3
000	3	032	3
001	3	033	3
002	3	100	1
003	3	100	2
010	2	100	3
010	3	101	3
011	3	102	3
012	3	103	3
013	3	.	
020	2	.	
020	3	.	
021	3		
022	3		
023	3		

Numbering rule for quadrants. Quad marked with (**) is 2-1-0. Order of keys when sorted (to 3 levels of division).

Figure 3. Method of assigning spatial index keys.

database updates are made they can either be done immediately, or they can be denoted as 'pending'. For example, if a new housing estate had been designed but would not be completed for a year then the updates showing the layout of the estate could be put into the database as pending. When pending work is put into the database, new objects and any modifications to old objects are stored in addition to all the old objects in that area. When a user extracts data from geoManager he can choose to see only current objects, only pending objects, or current and pending objects together (in which case any current objects which were updated by the pending job will be extracted in their new form). The user who put the pending work into the database can subsequently either cancel the pending work, in which case all the original objects are left unchanged, or commit the pending work, in which case the pending work becomes current and any changes to the original objects are made. This user can also make modifications to the pending work, but only a single level of pending work is maintained in the database. In this way geoManager provides a form of version management at a database level which is simpler than that proposed by Newell and Easterfield but easier to manage.

Pending work is marked as such in the database using the same status field which geoManager uses for indicating whether an object is checked out. This field can take the following values:

- ' ' – current
- 'C' – checked out for update
- 'P' – pending
- 'A' – current but affected by pending work

Only current objects (which are unaffected by pending work) can be checked out for update. The status field will then be changed from ' ' to 'C'. If an object is checked back in again directly the status field will be reset to ' '. If a set of objects is checked back in as pending then updates will be treated as follows. If an object is to be deleted then its status will be set to 'A'. If an object is to be modified then the original object record will have its status set to 'A' and another record will be added with details of the modified object with a status of 'P'. If a new object is to be added then a new record will be inserted with a status of 'P'. Separate tables maintain lists of all the objects which belong to each pending retrieval set. To cancel a pending retrieval set all the objects in that set with status 'P' are deleted, and all those with status 'A' have this changed to ' '. To commit a pending retrieval set, all objects with status 'A' are deleted and all objects with status 'P' have this changed to ' '.

The three types of retrieval mentioned earlier will retrieve objects with the following status values:

- Current objects only – ' ', 'C' or 'A'.
- Pending objects only – 'P'.
- Current and pending objects – ' ', 'C' or 'P'.

Only objects with a status of ' ' can be checked out for update.

Summary

In summary, geoManager provides a method of locking at object level to prevent concurrent update. Contention problems are minimised in the following ways:

- Locking at object level minimises the amount of data which needs to be locked for any transaction.
- Read only access is permitted to locked objects.
- Attribute updates can be handled as short transactions using geoManager Graphic Analysis.
- When contention does occur, the user who has checked out the objects which have caused the contention can be identified, so that the two users involved can try to resolve the situation if necessary.

In addition, geoManager provides the ability to store pending work in the database alongside current work, so that users can be aware of work which is in progress.

Data sharing and integration between applications

One of the main advantages of a relational database is that it is possible for multiple applications to access common data in a very flexible way. Since all geoManager data is stored in standard DB2 tables it can be easily accessed either by other applications or by flexible query tools such as QMF (Query Management Facility). This makes it possible to query attribute data stored in the GIS from any alphanumeric terminal within an organisation.

There are no real issues with regard to other applications reading the the geoManager data, but when it comes to updating the data then issues relating to long transactions arise, as was discussed in the section on concurrent update. Since the updates are being done to geographic data, which could be undergoing update by a long transaction via geoManager, it is necessary for the updating applications to recognise the geoManager status field and only update objects which are not checked out for update. Provided this rule is followed it is possible for other applications to directly update attribute data in the geoManager tables. Other applications should not, in general, directly insert records into, or delete records from, tables which are controlled by geoManager. This is because information about an object and its relationships is stored in multiple tables and to ensure data integrity the addition or deletion of objects should be done via geoManager. These sort of complex data integrity constraints are known as *semantic integrity constraints*, which cannot be enforced automatically within the relational database model itself, but must be enforced through applications (see Elmasri and Navathe, 1989). This is the sort of area where object-oriented databases, which are currently the subject of much research, may be able to give the application developer, and database administrator, more freedom in the future, by allowing such semantic integrity constraints to be stored within the database itself as 'rules' or 'methods'.

If non-GIS applications are going to be doing most of the updating on certain tables then it is possible to leave these tables outside the control of geoManager (but in the same database system) and just access the data as appropriate from geoManager or GPG. For example, one would probably keep a customer database outside geoManager but refer to this by storing customer reference numbers in appropriate objects within geoManager.

This is one area where a non-database approach which used a relational database to store all its alphanumeric data could provide a similar level of integration for alphanumeric data. The issues which have already been mentioned still need to be addressed, such as the management of long transactions. There is a danger if an

appropriate mechanism is not put in place to manage these that a non-GIS application could change an attribute value during the course of a long transaction, in a way which would conflict with what the GIS user was doing.

In the area of 'corporate GIS', geoManager Graphic Analysis gives significant benefits by allowing graphical data to be accessed by a very large number of users. Any user with a standard business graphics screen can view the graphics for a selected area, update related alphanumeric data, and perform simple forms of analysis. This type of facility would be much harder to implement if the graphics were not also stored in the database, especially as one starts to move to a distributed environment, as described in the next section.

Figure 2 shows an overview of some applications which could be integrated with GFIS in a typical environment. TSO, IMS, and CICS are all access methods which can access the DB2 database using the SQL query language. QMF (Query Management Facility) is a flexible query and reporting tool which enables users to analyse data using either the SQL or QBE (Query By Example) languages. AS (Application System) is a Decision Support Tool which, in addition to query and reporting features, provides functions in other areas including business planning, financial modelling, business graphics, project control and management and statistical analysis. AS can run directly against the DB2 database, or against a flat file such as the geoManager Open Format. The latter option is particularly useful for doing analysis on data within a specified geographic area using AS. Any maps, showing thematic information if appropriate, which are produced in GPG or geoManager Graphic Analysis, can be output in a standard graphical format which can be imbedded in AS reports if desired. These reports, including graphics, can be circulated to any number of people using standard office products. This is just a brief overview of some of the ways that other standard products can use the GIS data without having to write any special interfaces, because it is stored in a standard database management system.

Distributed database

The development of database software which can manage a database spread across multiple machines is something which major database vendors, such as IBM, are putting great efforts into. Various benefits come from being able to do this, such as being able to store each district's data locally for improved performance and availability, whilst still being able to access data from other districts in a transparent way (see Elmasri and Navathe for a more detailed discussion). Providing such distributed capability, whilst still maintaining the same function to manage data integrity and other issues, is an extremely complex task. For example, the rollback capability mentioned earlier must be able to work across multiple machines. If a complex transaction makes one update on one machine and is about to make a related update on a second machine when the transaction fails (for any reason, such as the failure of either database system or a network link), then the system must ensure that synchronisation is maintained between both machines.

The ability to provide a distributed database, with this sort of integrity, is perhaps the biggest single argument in favour of storing all aspects of geographic data in a standard database management system. The complexities of producing a true distributed database management system are such that it is difficult to see how any GIS developer could justify taking on this task independently. If a non-database approach is taken and some data is stored outside the database management system then it is not possible for the GIS to exploit any distributed function provided by the database management system.

Since geoManager stores all data in DB2 it can automatically take advantage of any distributed function in DB2 as it appears. There is already some basic distributed function in the latest version of DB2, and it is IBM's intention to greatly enhance this in future versions, first in terms of distribution across multiple DB2 systems on mainframe computers connected over a network, and ultimately including SQL databases on a whole range of hardware platforms from mainframes to PS/2s, within the framework of IBM's Systems Application Architecture (for further details see IBM, 1988).

Performance

As was stated earlier, the main reason for storing some data in a separate database is to optimise performance for graphical display or certain kinds of geographic analysis. It has also been pointed out already that these sort of operations are not done directly against data stored in DB2 by geoManager. Instead, data is extracted from geoManager into a format suitable for interactive display and analysis by GPG, geoManager Graphic Analysis or other GIS packages. Therefore the only performance question which needs to be examined in comparing the geoManager approach with a non-database approach is that of the time taken to retrieve a geographic area from the moment it is initially requested.

Clearly by using a non-database approach and storing the graphics in a specialised format suitable for immediate display, it is possible to achieve a very fast response to certain types of request. For example, if the data is stored as a set of regular map tiles then it should be possible to satisfy a request for a specific map from this set very quickly. The same is true for a request to view an area covering a small number of these maps. In an application where the data viewed is generally quite standard (for example, where it can be specified as one or more of a number of layers), and this is generally viewed at a reasonably constant scale, then this approach is likely to give good performance in terms of elapsed time from initial request to initial display.

In a corporate system rather more flexibility is required because different applications will need to display and analyse data at greatly differing scales. While some applications may need very detailed data at a large scale, other applications may require a sparse set of data, perhaps qualified by attributes, across a much wider area (for example, one might want to retrieve all crimes committed between 10 pm and midnight on a Friday or Saturday night during the last six months in county X). With a map-based (and possibly layered) system it is likely to be rather difficult to create this type of dataset. This is where an object-based system has an advantage, since this sort of request is easy to satisfy with an object-based approach. Of course the extra flexibility provided by an object-based approach does not come for free, since in order to display a map at any scale it is necessary to

retrieve each individual object in the requested area, rather than just accessing a single predefined file. Because of this, the initial retrieval of a geographic area at a large scale in an object-based system is likely to be slower than one can achieve with a map-based approach.

Spatial indexing in geoManager

The importance of performance issues has been recognised in the design of geoManager, and a major design point has been to optimise retrieval times from the database. In line with the general philosophy of geoManager, the techniques used have been designed so that they will exploit as much DB2 function as possible, and also take advantage of future enhancements. The key to efficient area retrieval from geoManager is a spatial indexing system based on a quadtree approach. For a general discussion of spatial indexing methods, see Samet (1988) or Vanzella (1988). The specific approach used by geoManager is similar to that described by Abel and Smith (1983), generally known as the Smallest Containing Quad Method or MX-CIF quadtree, with suitable modifications for a relational environment.

A very brief overview of the approach is given here. The area covered by the application is recursively subdivided into quad cells to a predefined level. Each quad cell can be assigned a code as illustrated in Figure 3. A unique key for each quad cell is given by concatenating this code with the level of the quad cell in the tree. The example illustrates this using three levels, but in practice one would use considerably more than this. The spatial index key assigned to each object is given by the key of the smallest quad cell which completely encloses it. The spatial index key is stored as an attribute of every object in the database.

This spatial index key has some important properties which allow us to use it to perform efficient area retrievals from the database. The single most important property is that, in general, objects which are geographically close together in the real world will be assigned spatial index keys which are similar. To exploit this fact, geoManager specifies to DB2 that it should use the spatial index key as a clustering index, which means that it will physically sort the data records on disk in spatial index key order, and try to maintain this physical clustering as far as possible even when records are inserted or deleted. One consequence of doing this is that *objects which are geographically close together in the real world are, in general, stored physically close together on disk.* This can considerably enhance performance when accessing a number of objects which are geographically close together, because of the way that disk access works. When a request is made to read a record from disk, a block of data is read into memory which contains a number of records, including the one which was requested. If a request is made to read another record subsequently, the system checks whether the requested record is already in memory because of a previous disk access. If it is then the record can be returned much more quickly because it can be read far more quickly from memory than from disk. This technique can be extended in various ways. One technique is caching, where the disk subsystem stores large pages of data in memory to increase the probability that a requested record will be in memory rather than on disk, and will therefore be able to be accessed much more quickly. DB2 also has the capability of further improving performance using a technique called prefetch, which involves asynchronously reading pages of data in from disk before they are required for processing. The appropriate pages to read in are determined by looking at the index which the query is following. The geoManager spatial index key allows all these methods of improving performance to be used automatically when doing area retrievals from the database.

As objects are inserted and deleted the physical clustering of objects will gradually be lost. Therefore it is necessary to reorganise the tables from time to time to restore the physical clustering. This is a standard function of DB2. The DB2 system allows the database administrator to obtain statistics about each table to enable him to decide when it is appropriate to reorganise that table.

Another property of the spatial index key which helps geoManager perform efficient extractions is that the key for any given quad cell is immediately followed in the ordered list of keys by all its descendents, that is all smaller quad cells entirely contained within it. This means that the quad cells which are candidates for containing objects in the requested retrieval area typically have key values which can be specified as a relatively small number of continuous ranges, rather than a large number of scattered values, which allows the retrieval to be specified using a relatively small number of SQL queries. Furthermore, these queries can be executed efficiently since each one includes a selection qualification which is a continuous range of values of a field which has a clustering index defined on it.

A more detailed discussion of the spatial indexing techniques used by geoManager is really beyond the scope of this article. Perhaps the most important thing to note is that the techniques used are designed to exploit the standard relational database indexing system as far as possible, which means that geoManager will automatically be able to take advantage of future hardware and software developments which enhance the performance of the relational database.

Conclusion

This discussion has looked at a wide range of database issues which apply to all corporate database systems, including corporate Geographic Information Systems. Some ways in which a standard relational database management system can help to tackle these issues have been highlighted, as have areas where additional function must be provided by the GIS. It has been proposed that many of the benefits which can be provided by standard relational database management systems can only be realised by storing all aspects of the geographic data in the database. This applies most of all to providing distributed database function.

An important distinction has been made between the storage and management of geographic data and the manipulation and analysis of geographic data. Many people have claimed that relational databases are not suitable for GIS, but they are normally talking about the manipulation and analysis of geographic data. The main assertion of this article is that commercial relational database management systems can offer significant benefits for the storage and management of geographic information, as outlined with reference to the approach taken by geoManager.

The role of geographic storage and management systems is likely to become increasingly important as GIS

develops. An increasing number of specialised GISs are likely to appear which are suitable for specific applications within an organisation, and it will become increasingly unlikely that a single GIS will be suitable for all the manipulation and analysis requirements of an organisation. However, it is vital that all the GIS applications which are used, as well as non-geographic applications, have a means of accessing data from a single consistent database. The trend towards such geographic database management systems is also likely to be encouraged as different organisations seek to solve the problems of using common databases.

REFERENCES

1. Abel, D.J. and Smith, J.L., A Data Structure and Algorithm Based on a Linear Key for a Rectangle Retrieval Problem, *Computer Vision, Graphics and Image Processing*, Number 24, 1983.
2. Elmasri, Ramez and Navathe, Shamkant B., *Fundamentals of Database Systems*, Benjamin/Cummings, Redwood City, California, 1989.
3. IBM UK Programming Announcement ZP88-0472, *Distributed Relational Data in Systems Application Architecture*, November 1988.
4. Newell, Richard G. and Easterfield, Mark, Version Management – the Problem of the Long Transaction, *Proceedings of the Mapping Awareness Conference, Oxford*, January 1990.
5. Samet, Hanan *Hierarchical representations of collections of small rectangles*, ACM Computing Surveys, Volume 20, Number 4, December 1988.
6. Vanzella, Luca, Classification of Data Structures for Thematic Data, *Technical Report TR 88-14*, Department of Computing Science, University of Alberta, June 1988.

The author would like to acknowledge helpful comments given at various stages in the preparation of this article by David Adler, Steven Brown, Paul Cocking, Peter Gee, Wilhelm Kerbl, Scott Kutz, Tim Lloyd and Kurt Mayer.

PETER BATTY is a Systems Engineer in the GIS Market Development Group at IBM, Warwick.

Editor – We are pleased to print this IBM response to the challenges laid down in our earlier database feature and in particular, to the article entitled 'Towards a blueprint for database vendors' which was described as being 'a valuable step towards a database specification'. We will be pleased to publish similar contributions from other suppliers and examples of live applications.

Reprinted from
MAPPING AWARENESS

Published by
MILES ARNOLD, HIGH WINDS, CASSINGTON, OXFORD OX8 1DL
TELEPHONE 0865 880236