Field Techniques Manual: GIS, GPS and Remote Sensing

- Section B: Data

Chapter 4: GIS Database Mechanics

4 GIS Database Mechanics

4.1 The underlying information in the GIS

A GIS is simply a database with graphical capabilities that displays its information within a given co-ordinate space. As with any database due attention has to be given to the way GIS data are formatted and stored. This configuration is referred to as the database structure or schema. Designing effective database structures can be very difficult but some of the more common concepts will be discussed here. A more in-depth look at database structure pertaining to GIS can be found in Burrough & McDonnell (1998) or for more on database structure itself the reader is referred to Roman (1999). The GIS can display co-ordinate information relative to other data or to digital images or maps. The information displayed, often referred to as a theme (event theme in ArcView), consists of data taken from the database (these data points are referred to as features) and each point has many details or attributes. Figure 4-1 shows a project consisting of the countries of Europe. The collection of countries is a theme, as would be a collection of cities, rivers or other geographic features. Each theme can have various details such as a country's size, population, currency etc. These descriptions about the theme are called attributes.

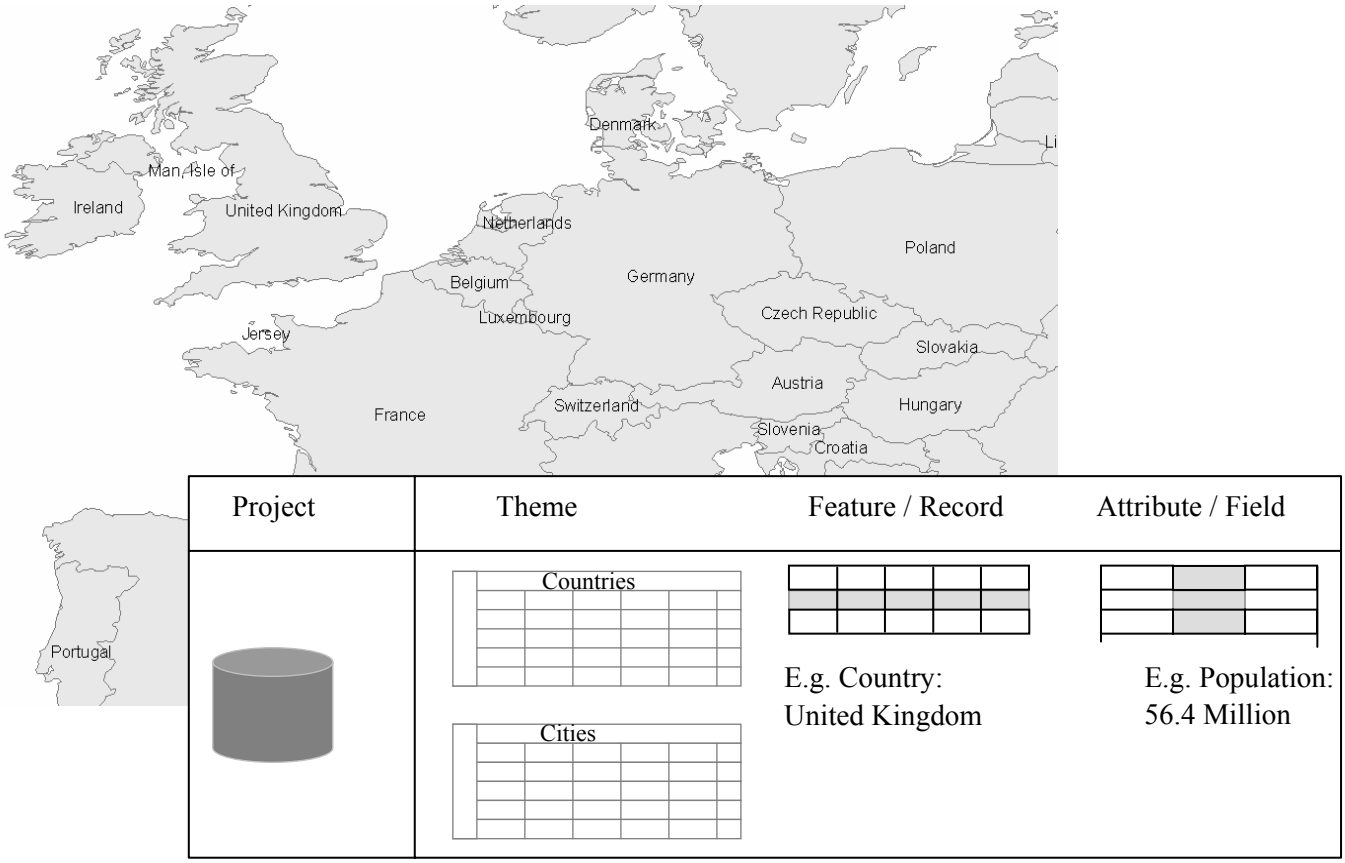

Figure 4-1 Relationship between themes and attributes. Data from ESRI ArcView example data.

These attributes will often come from an external database, which could be as simple as a Microsoft Excel spreadsheet containing GPS positional data, or it could be a large ORACLE database with hundreds of tables and gigabytes of data. A GIS can access these

data directly or the data can be exported into the GIS in a tabular format. What the database is and how the GIS accesses it is vitally important to any project. All themes require attributes for latitude and longitude (or equivalent) so the GIS can plot them as X Y co-ordinates. In Figure 4-2 GPS points from the Bogda Shan dataset are plotted above a Landsat ETM+ sub-scene. In the dialogue box various attributes associated with the localities theme.

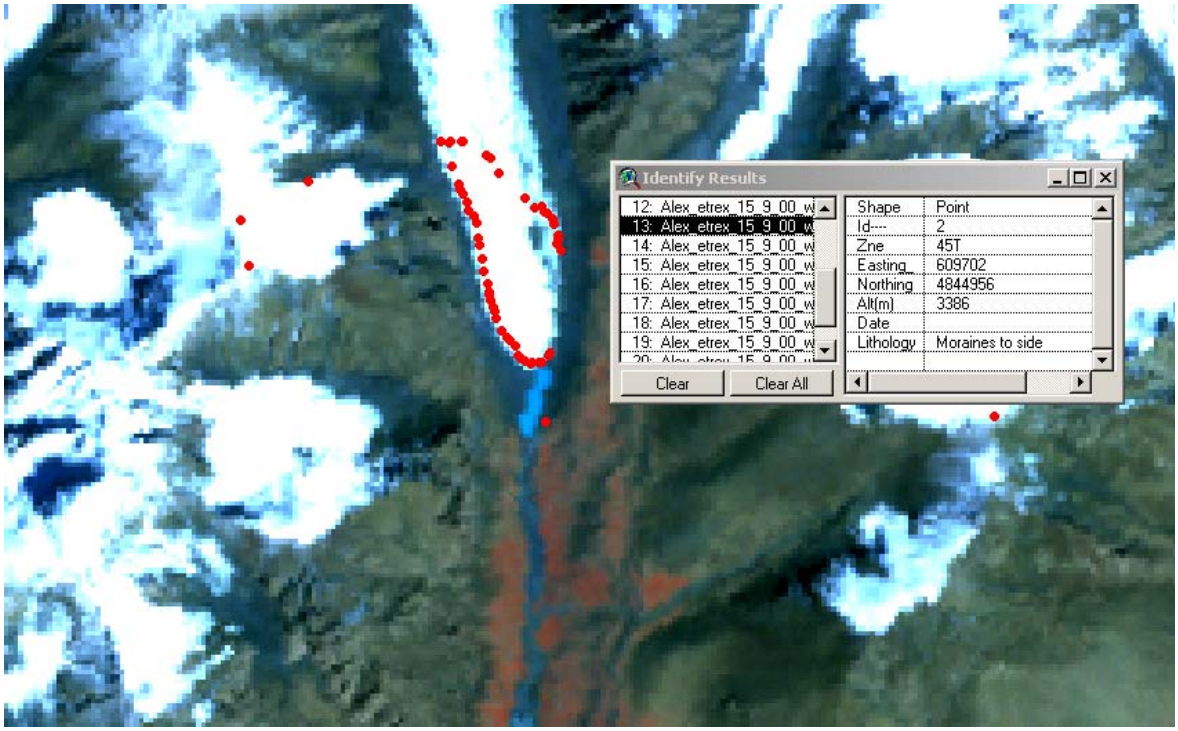

Figure 4-2 Figure showing Bogda Shan data and GPS co-ordinates with associated data.

The number and complexity of the attributes associated with the theme will influence how the team decides to store them.

4.2 Selecting the project database

The first decision to be addressed is the type of database to be used. There are two types of databases: file/server and server based solutions. *File/server databases* have a database file on a computer hard drive. This could be held locally on the expedition PC or on a shared network drive of a PC. Examples of this type of database include Microsoft Access *.mdb and *.mde files, Microsoft Fox Pro *.dbf or Borland Dbase *.dbf files. Microsoft Excel spreadsheets also take this format but, as discussed below, are not the same as databases. The other type of database is a *server database* where a computer is used as a dedicated database server that processes instructions and sends them back to a PC over a network. Examples of this type include ORACLE databases and Microsoft SQL Server. Server based solutions are more expensive, generally require more hardware and require a network connection. They are useful for high-end databases where large quantities of data need to be accessed by large numbers of people (>50 users). Server side solutions also benefit from automatic data backup and file security. File/server solutions are cheaper and generally better suited to small databases or where data access across network is unavailable. It is possible that the main database could be a server solution but that certain

tables are output as Dbase files onto a file/server connection for accessing by the GIS. This, however, runs the risk of losing data integrity within the GIS and underlying database as the two may rapidly become very different and require synchronising on a regular basis.

The second decision with databases is whether the system is to be relational or not. This question addresses the database engine, the system that handles all queries. The engines can either be a Database Management System (DBMS) or a Relational Database Management System (RDBMS). Relational databases automatically connect data in various tables and keep records in the correct order when sorting or querying. Relational databases take longer to set up and are often considered less flexible than non-relational databases but benefit from improved data integrity and substantial speed improvements.

In a non-relational database such as Microsoft Excel data entered has no connection with any other information. This means if a column is sorted it will destroy the integrity of your information. This is shown below in Figure 4-3. The left hand column is sorted but the right hand column stays the same. This means when plotted the data will no longer make sense. If this was in a relational database then both columns would automatically be sorted. The dangers of having your data accidentally scrambled should demonstrate the need for a relational system.

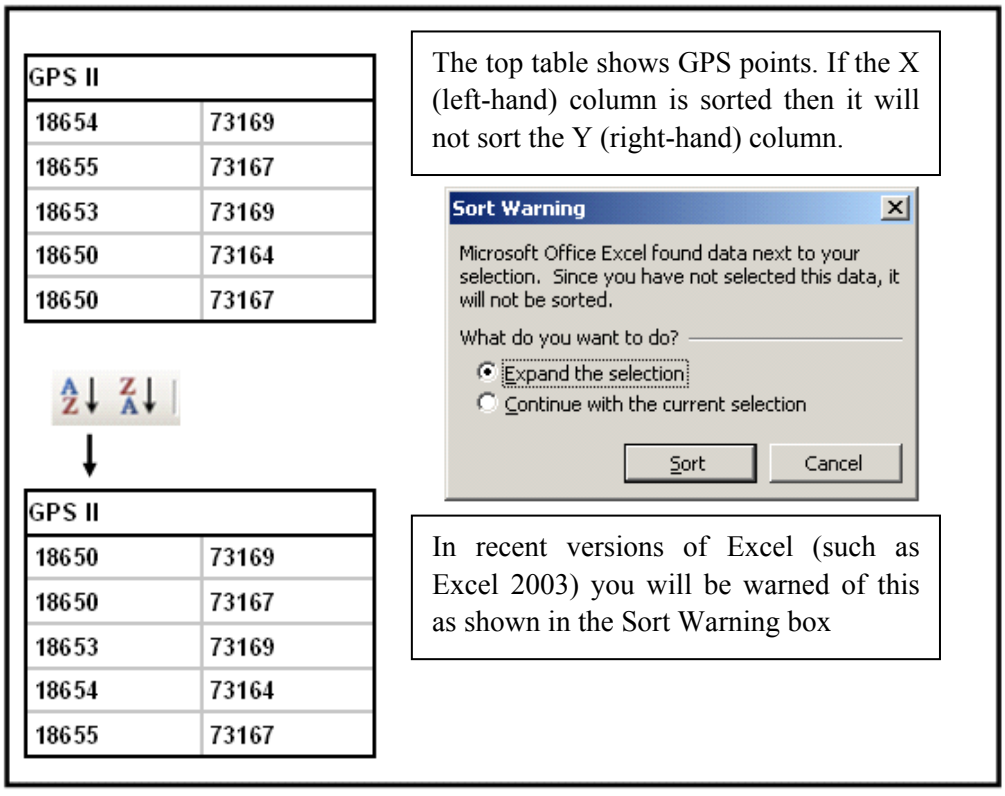

Figure 4-3 Disadvantages of non-relational systems.

What this means in practise is that many decisions have to be made in advance. The structure of a relational database is not very flexible but the way data is accessed is. A good example is that once data is entered into a non-relational database it can be very slow and time consuming to alter. If a relational database is set up correctly altering information on the fly is simple, rapid and accurate. For example, on the first few days in the field

assumptions may be made in the sampling and recording of data that later turn out to be incorrect. A rock type might be misidentified or a bias might be introduced into some other study. A relational database will allow one change to be made to the data that cascades through all relevant features instantly without searching through and potentially correcting the wrong data. For these reasons it is a very good idea to store your data relationally if possible.

4.3 Database configuration

When the type of database has been chosen it must be correctly configured. This process defines the database structure. The database structure or schema describes the number of tables, fields and the relationship of each one to another. The database structure has the biggest impact on the speed, flexibility and accuracy of the information. Databases are efficient systems for storing and retrieving data but each record has to be recorded in a set manner. Each individual field must be given a format that the database expects such as a number, string or date. When this format has been selected no record for that field can deviate from this selection. Careful consideration must therefore be given to the type of data required. Standard databases store data in one of six ways: integers, double numbers, strings, binaries, dates or memos. These data types are described below in Table 4-1.

Table 4-1 Data storage types for typical databases.

Data Types	**Restrictions**	**Uses**	**Speed / Flexibility**
Integers	2^{16} possible values. **Unsigned** approx ± 32,768. **Signed** + 32,768 to -32,767 inc 0). No decimals.	Used for integer attributes and to create internal database references.	Require small amounts of memory to store (2 bytes). A computer can sort & process integers quickly
Doubles	Allows up to 15 decimal places and up to 324 sig. figures.	A double number is the longest number a database can store	Requires 8 bytes of storage space and are larger and slow to sort.
Strings	A line of text containing up to 255 characters.	Descriptive fields	String fields can't be used for arithmetic processes.
Binaries	Binary (or Boolean) consists of a 1 or 0.	These fields are very quick to process & store.	Access uses Y or N boxes. ArcView uses True or False.
Dates	Commonly converted to a Julian Date format. Each day is given a number and the time is given as a fraction of this day.	Dates and times.	Very slow to query.
Memos (Blobs)	Memos are large files that can be stored externally and can be of any size.	Used for supporting notes and data.	Very slow to process in a database and should be used carefully

Some forms of data are easier to store and faster to process than other forms of data. Integer numbers are very useful to use, as they are both small and fast. In many cases this will not be acceptable because the information recorded will be of a more complex nature. Good database design allows the use of integers even when descriptions are more complex. An example that can demonstrate this is habitat mapping. Habitat mapping is an important fieldwork exercise being conducted by many expeditions. The example in Section 4.4 will show the difference between a good database structure for animal sightings and a poor structure.

4.4 Example of database structure designs

Habitat mapping is an important fieldwork exercise being conducted by increasing numbers of expeditions. This example will show the difference between a good database structure for animal sightings and a poor structure. While recording animal sightings in a game reserve using a GPS, commonly a description of an animal will be recorded with its corresponding GPS location as shown in Table 4-2.

Table 4-2 An example fieldslip in digital format.

species	habitat	waypoint	x co-ordinate	y co-ordinate	weather	altitude
impala	grassland	12	55234	233444	clear	500
buffalo	grassland	12	55234	233444	clear	500
cheetah	grassland	14	55520	232500	cloudy	480
impala	grasland	15	55420	232418	clear	450
buffalo	grassland	15	55420	232418	clear	450
giraffe	grassland	16	55435	232480	clear	411
buffalo	grassland	17	55412	235212	cloudy	422
cheetah	grassland	17	55412	235212	cloudy	422
impala	forest	18	55214	235444	cloudy	455
impala	forest	13	55280	233501	raining	200

Table 4-2 is poorly laid out in database terms: there have been several animal sightings at any one location but the information for that location is replicated. For example, the grid reference and altitude are all identical for any one GPS location so it is inefficient to write them multiple times, it is time consuming in the field and there is a significant potential for error. Putting one number wrong could seriously compromise a team's work. Also text fields are slow to sort; therefore we should find a method for replacing some text with integer numbers. We can do this by creating a table for all the variables.

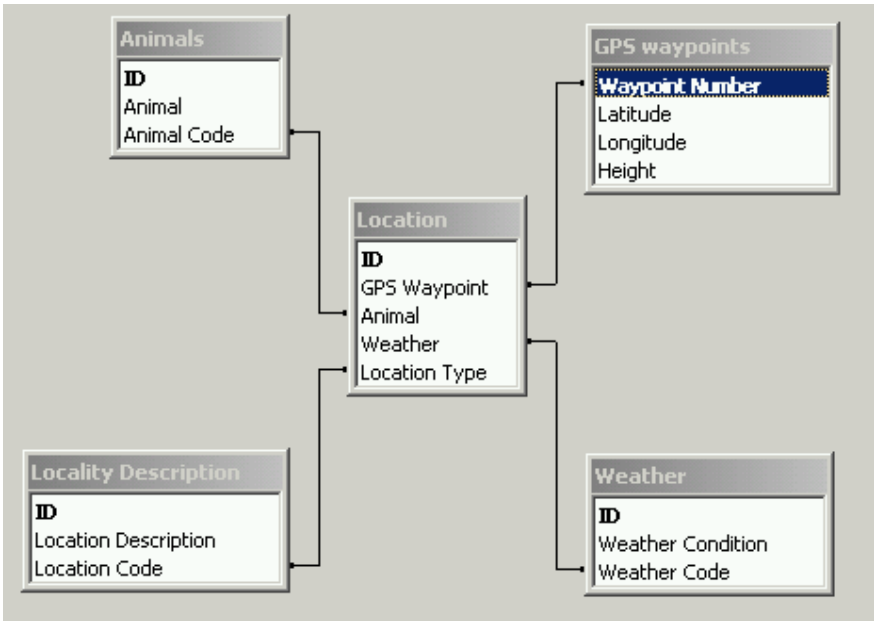

Figure 4-4 Relationship of data within the habitat mapping GIS.

In Figure 4-4 the GPS waypoints have been stored separately. This way data is not replicated. The first table contains 462 characters most of which are strings of text. By relating it into a database we can simplify it to less than 100 characters all of which are integer numbers as shown below in Table 4-3. This means that as the database grows in size its speed compared to the first method will grow exponentially.

Table 4-3 The database equivalent of the fieldslip. It is no longer necessary to record the latitude and longitude of each waypoint individually. By saying waypoint 12, the GIS can look up the co-ordinates for waypoint 12 from a separate table. Similarly for animal species, instead of recording the name each time, we can list all the animals in a separate table and reference them by a number.

id	waypoint	animal	weather	habitat
1	12	1	2	2
2	12	3	2	2
3	14	2	1	2
4	15	1	2	2
5	15	3	2	2
6	16	4	2	2
7	17	3	1	2
8	17	2	1	2
9	18	1	1	1
10	13	1	2	1

The species' numbers etc. can then be looked up in a database to create the same result as seen in the first table. The resultant query would look like Table 4-4, which is identical to Table 4-2.

Table 4-4 The result of a query on the re-structured database.

species	habitat	waypoint	x	y	weather	height
impala	Grassland	12	55234	233444	clear	500
buffalo	Grassland	12	55234	233444	clear	500
cheetah	Grassland	14	55520	232500	cloudy	480
impala	Grassland	15	55420	232418	clear	450
buffalo	Grassland	15	55420	232418	clear	450
giraffe	Grassland	16	55435	232480	clear	411
buffalo	Grassland	17	55412	235212	cloudy	422
cheetah	Grassland	17	55412	235212	cloudy	422
impala	Forest	18	55214	235444	cloudy	455
impala	Forest	13	55280	233501	raining	200

As discussed above a relational system reduces accidental errors. All the numbers are entered from a computer form where the animal type is selected from a drop-down list showing the relevant species. Only the number is entered into the table reducing the possibility of accidental typos and errors. In Table 4-2, the Impala at location 15 has its habitat spelt incorrectly. This is likely to result in incorrect analysis of data. If the typo were in the GPS location field, then the problems may go undetected and result in a spurious interpretation when animal sightings are plotted in the GIS.

The speed of a database when accessing data or performing queries is related to the amount of data in any field and the type of data used. In the previous example the size of the fields was reduced to a fraction by using integer numbers. This reduction in size is one way to speed up the database. The second speed increase is gained from the fact that the numbers were easier to process than the previous text string. All these help to make the database more efficient. There is also a major speed increase in entering data because fields with repeated data do not need to be re-typed. This is among the most important factors when establishing where data will be kept in the database. The process of removing redundant data is called normalisation. Normalisation works by separating tables into groups of data that are dependant on each other (a relationship called Functional Dependency) (Roman 1999). In the worked example above, the original field slip contained many fields that were not dependant on one another. The animal spotted is unlikely to be dependent on date, or exact grid location. For example impala could be spotted at many grid locations on many days. Normalisation requires that these tables be separated. Similarly, if we had details about an animal such as its colour etc. this should also be excluded into a separate table. There are six levels of normalisation - these place the database into the first normalized form up to the third normalised form, followed by the Boyce-Codd normalised form and finally the fifth and sixth normalised form. Each form represents a more efficient table design than the previous method. The increase in performance slows with each step in normalisation form and anything above third is generally not relevant.

The most common way to increase speed (usually covered by second normalisation form) is to give every record a unique identification. This is commonly referred to as an ID or Primary Key (this is usually required even before normalisation can begin). When a

database stores data it puts it in a sequential order according to the key. When an ordered list of sequential data is queried, the database examines the middle record first. If this is the correct record it stops, if not the database assess if the number it requires is greater than or less than the current value. The database then moves to the middle of the next section and repeats. In an unstructured data table the database starts from the beginning and reads through until the end, this method is very inefficient. The Bogda Shan Expedition discussed in Section 17.1, collected large amounts of field data. A GPS transect across a mountain range and an intermontane basin was constructed using GPS receivers. The expedition collected 12,855 waypoints and how that data was stored had a significant effect on the query speed. This is demonstrated in the Table 4-5. The reader is referred to Burrough and McDonnell (1998) for a more thorough explanation.

Table 4-5 Structured and unstructured queries.

	unstructured list	**structured list**
average number of searches to find one of *n* records in a database	(n+1)/2	$\log_2(n+1)$
average number of searches to find a record from the Bogda Shan Expedition database, containing 12,855 records	(12,855 + 1) / 2 = 6,428 searches	$\log_2(12{,}855+1)$ = 13.65 ≈ 14 searches

The advantage of a relational database to the expedition should now be obvious. It is worthy of note that for large projects a hybrid system between relational and GIS spatial data has been created. These spatial databases such as ARC SDE and ORACLE Spatial can be used to great effect but are generally too involved and expensive for an expedition. If the reader requires more information on the most effective method for storing and querying very large sets of complex spatial geographic data then they should consult a specific text book such as Rigaux *et al.* (2002). For most purposes a relational database combined with spatial queries within the database will suffice.

4.5 Using the database in the field

During fieldwork, the structure of the database will be critical to what is recorded. There is little point in configuring all fields as integers if this is not valid for the data being collected. On many occasions, decimal data will be essential for a project. In recent years the size of a database used on a desktop PC has become less of a concern, and hard drives are now commonly over 60 gigabytes in size. The size of the GIS on a high specification machine is not as relevant as in previous years. In the field, however, high specification PCs are not generally available and so smaller data loggers are important. These might include the memory of a GPS receiver or a handheld PDA or similar electronic device (see Chapter 13 on Field Equipment). These machines have comparatively low storage capabilities and make numerical annotation as important as ever. Complex processing in a GIS is not a significant concern on high-end computers but on low specification field computers, the processing has to be streamlined to work within the computer's capacity and often within the length of the battery time.

It is important that the setup of the database has been given sufficient thought. When a database has been configured it can be very difficult or even impossible to change the format of the data e.g. to change a string field to an integer field. In the habitat mapping example (Section 4.4), if a different species of animal were observed, such as seeing a lion, it would be easy to add, as this animal could become animal number 5 in the animal table. Changing the fundamental data structure is much more difficult than adding new records to a database. These decisions need to be made in advance of the expedition and adhered to in the field, changing the waypoint data from integer to decimal would be significantly more difficult.

Before the data can be imported into the database it needs to be collected in the field. Commonly data is collected in a field notebook and retyped into the database. This is a good method as it retains a hardcopy of the data in case the database becomes lost or corrupted. During the Bogda Shan Expedition data was written in notebooks and referenced to selected waypoints stored in the GPS. At base camp information from the notebooks was transferred to an Access database and linked to the GPS waypoints by joined fields. In Figure 4-5 the table on the right contains all the GPS data, the table on the left contains all the locality descriptions.

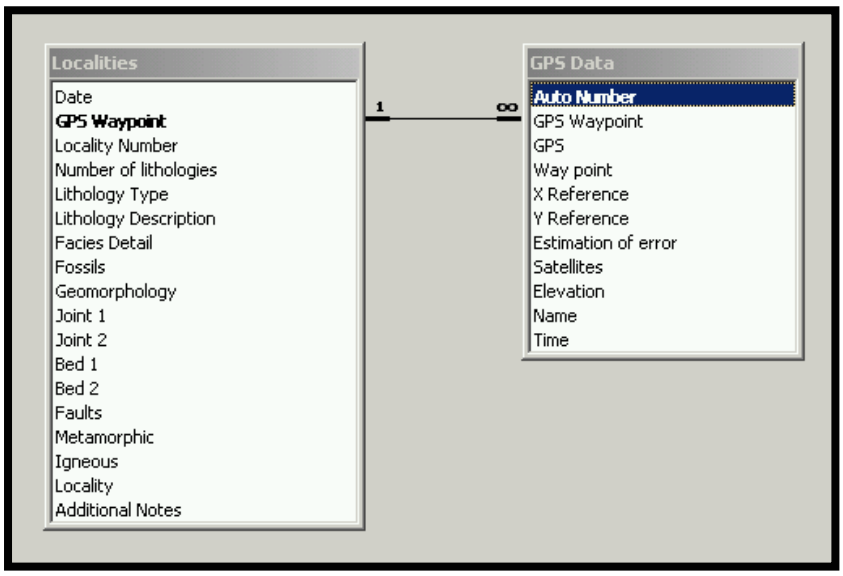

Figure 4-5 The relationship of GPS data to locality data in the Bogda Shan Access database. Note the relationship type one to many.

Figure 4-5 shows not just the relationship joining two numbering systems but the type of relationship. Joins can only exist between data of the same type, in this case two integer values. The data in the table on the right [GPS Data] has the symbol '∞' next to its join while the data in the [Localities] Table has the symbol '1' next to its join. This is referred to as a 'one to many' link. This means that there could be lots of GPS points but they can all relate to only one field locality. In the field four GPS receivers were commonly used. So at any one location four GPS waypoints could have been collected, though they all refer to the same location (*due to errors in the GPS receiver the latitude and longitude co-ordinates would be slightly different between them*). The GPS waypoints table could have up to four times as many records as the localities table. Also one locality description may

describe a field outcrop that may be many meters in length. There could be multiple GPS waypoints along the length of the exposure. All of these describe just one location, so we need to tell the database that there are lots of readings at one point and we do this with the one-to-many join. Other joins include the one-to-one join where there are an equal number of records in each table or a many-to-many link where the tables could both contain different quantities of data. Many-to-many links are more difficult and in most databases require a linking table between the joined tables.

4.6 Integrating existing data

In many cases the GIS being constructed for an expedition will build on some previous work. There might be an existing database of previous field results that need to be compared or a list of waypoints that need to be visited. There may even be an existing GIS that needs incorporating into the current version. The use of historical data can be essential to a good field work project, however, getting this existing data into the GIS can be very problematic. The way each database stores data is different and they are not all compatible. Very few standards exist but the format that the data is accessed in has been given some uniformity.

The ANSI and ISO standards have supported the use of the Structured Query Language (SQL) for data retrieval, which is important for data export and integration. Unfortunately, over time several slightly different versions of SQL have appeared. For example, Microsoft SQL Server, Visual FoxPro and ORACLE all have their own versions. Microsoft has made significant progress in the field of database integration and there are many techniques for getting disparate data sources to work together. More recent versions of Microsoft products (Access 2000 and SQL Server 2000 upwards) all integrate transparently. It is very easy now to have a small front end Access database referencing a larger GIS database behind the scenes without the need for separating the data and running export and import queries.

The second, and more useful, method of data integration is the Open Database Connectivity (ODBC) interface. ODBC is an Application Programming Interface (API) developed by Microsoft in 1992. It allows disparate data sources to talk to one another through a common syntax. Microsoft software can use an ODBC connection to retrieve data from a host of different sources. Once the data has been read by the ODBC API it can be exported to a Dbase *.dbf table which is perfect for adding to a new ArcView GIS project. ArcView can also reference a variety of other sources such as *.mdb files directly via a link. Alternatively if the data can be exported to a text file (ASCII) it can be re-imported to the database.

The problem with exporting and importing data is that referential integrity (the links between the data) can be lost. Also the data types (see Section 4.2) will not be retained. To restore data types a parsing filter can be used. Microsoft Excel has a very good parsing filter called Text to Columns found under the Data menu. This can reconvert data back to its original format if it has been lost. For example an integer data field will be changed to a string data field when exported as text. Selecting that column in Microsoft Excel and running the Text to Columns parsing filter over the data will return the data to its original form. When the data is in an Excel or Dbase format it can readily be imported into the GIS.

The biggest concern with historical data is the geographical co-ordinates. The GIS is concerned with the spatial relationship of data. With existing data it is not sufficient to import an X and Y co-ordinate without some knowledge of the datum and projection of the data. These concepts are discussed more thoroughly in the chapter on co-ordinate systems. It is important to be aware of them as they can considerably alter the quality of the data. Specifying an incorrect datum or importing data to a different projection can cause considerable problems.

4.7 Data import/export and cleaning

As discussed above, a GIS often does not exist in isolation. Data will have to be brought into the project and exported out of the project for reports, graphs and data sharing with other projects. Most databases have proprietary data formats. Problems can be avoided by exporting data to a common format, readable by many systems. Unfortunately, there are no GIS data standards. ASCII text files are useful but they are only raw data and they have no method for describing the information. The procedure of defining the data types is left to the automatic parsing filter or to the GIS administrator. Neither of these are ideal situations. A better solution is to use a data format that understands the types of information encoded but is not proprietary to a single system. Traditionally Dbase files have been very useful for this purpose. Most database engines will import a *.dbf file and they are a good method if nothing else is available. Dbase files are, however, far from perfect and a better solution is to use a totally format free import and export method. A common solution is to use HTML (Hyper-Text Markup Language), a text format that can be read by almost any computer, regardless of operating system and hardware. HTML script is structured into a series of blocks that are described by tags which tell the computer how to format the text.

HTML data format

<html>	*tells the computer the data format*
<head>	*tells the computer to format the following text as a heading type (usually bold)*
<body>	*tells the computer to format the rest of the text as the default text style.*

The HTML format is a text store in the same way as ASCII format. HTML can be used for data storage and many packages, such as Microsoft Office, support data exports to HTML format. HTML is good for text but often handles data such as spreadsheets and databases very badly and file size can inflate dramatically. For the complexities of GIS data we require a storage format that is none-proprietary, like HTML, but has the same advantages of the *.mdb, *.dbf etc. The data equivalent of the HTML text is XML (eXtensible Mark-up Language). XML replaces the default HTML tags (head, body, etc.) with user defined data types: it is the export method used by the most powerful data storage solutions. An example of an XML layout for a GIS database is shown below.

XML Data Format

```
</TABLEDEF>
        <COLDEF name = “Latitude” datatype = “Long” />
        <COLDEF name = “longitude” datatype = “Long” />
        <COLDEF name = “altitude” datatype = “double” />
```

XML will become increasingly common as it is integrated into the more common Microsoft Office programmes. A basic XML generator is included with Microsoft Office XP. This is a limited XML solution and a far superior one is bundled in Office 2003. XML will by 2006 become the default method of saving data in Microsoft applications, with proprietary file formats becoming redundant. For some time ESRI have published ArcXML, a protocol to control data exchange between the ArcIMS Spatial Server, the Application Server and the Server connectors. There is a GISci specific mark-up language in development called GML (Geographic Markup Language) that has been developed by the Open GIS Consortium. More details about the development of XML style solutions for GISci can be found at *gislounge.com/ll/xml.shtml*.

One perennial problem associated with GISci data is the order in which latitude and longitude are referenced. As already mentioned in Chapter 2 the convention used in speech of saying latitude and longitude (Y values followed by X values) is rarely used in written terms. More often we write longitudinal values before latitude values XY. In Excel spreadsheets XY is the common format for referencing cells (A1, B1 etc.). However, when delving more deeply into Excel with any form of programming the convention of R1C1 is used (YX). It is imperative you understand how the data is being output from the team's GPS and how it is used in the GIS. Failure to ensure the correct format can cause the data to be plotted in a skewed manner.

Regardless of the import/export formats used, there can still be a problem with the format or quality of the data. The process of preparing information to be placed into the database is called cleaning. Most existing data will have fields that are not required in the final database. Stripping out redundant data and re-ordering columns is important, as is reformatting or reprojecting data. Cleaning can take long periods of time and is essential to building a good GIS. Software for cleaning and parts of the cleaning practise are discussed in the chapter on GIS software.

4.8 Adding raw expedition data

A full relational database may be too complex for some expeditions. In this case the database may be a flat format series of tables. These tables are usually *.dbf format. ARC GIS has a good utility for managing these files called ArcCatalog. ArcCatalog allows the user to make connections to various data sources to pull them into the GIS. In Figure 4-6 the data available as .dbf is listed in the left hand pane and a preview of the data is shown in the right hand pane. There are three different symbols shown on the left hand pane. The symbols show tabular data, point shape files and polygon shape files. For shape files, it is possible to not only see the raw data as shown in Figure 4-6 but also a preview of the data in a spatial framework. This is done by changing the drop down at the bottom of the screen from Table to Geographical.

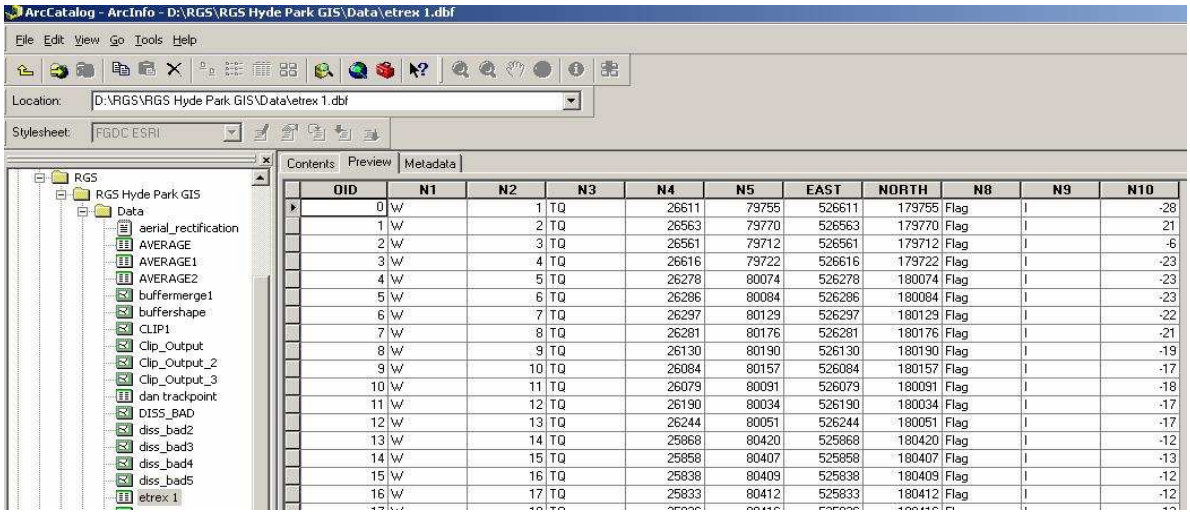

Figure 4-6 Example screen from ArcCatalog showing the available data for a project.

Right clicking on any of the tables listed in the left hand pane and selecting properties brings up a list of all fields associated with this table. The fields can then be customised in a manner similar to those discussed above in Section 4.2.

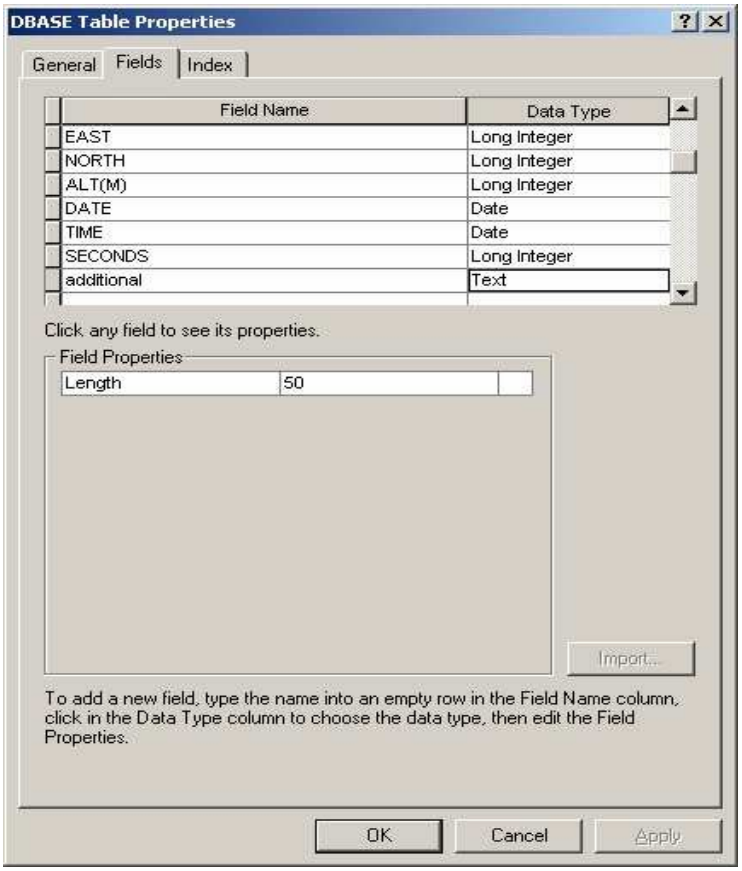

Figure 4-7 ArcCatalog data management window. Data can be managed from ArcCatalog. Additional fields can be added and their data types can be selected. Adding data fields like this can be useful to merge GPS data and data collected on field slips without the need for a more complex relational database. If there are only a few annotations to be made then this method is quicker. If there are a large number of GPS points and annotations then it is better to prepare a full database solution.

By using the method shown in Figure 4-7 details about the GPS waypoint can quickly be added to the data. These details might include animal sightings or waypoint descriptions. Managing your data with ArcCatalog is a simple way of bringing lots of data together.

A good companion to ArcCatalog is Microsoft Excel. It is in Excel that most of the actual data will be added. Excel is a good tool for this as it can open and export *.dbf files. A problem with Excel, as with all flat file systems, is that the data is nit relational and it is very easy to corrupt the information. Care must be taken to ensure that the fields or columns are not sorted incorrectly and that the data is held together and not mixed. It cannot be stressed strongly enough the benefits of keeping data in a relational database over individual *.dbf files.

4.9 Accessing project data

As discussed in Section 4.3 it is important to structure the data correctly. Previously this section has looked purely at storing data and using normalisation for improving data accuracy. Table 4-5 Structured and unstructured queriesshowed how structuring data can be important for the speed of large databases. This section will look at the need for good data structures for accessing the data in the GIS. It is important that the data entered into the database can be extracted and plotted into the GIS effectively. In a habitat mapping exercise it may be necessary to note the type and number of animals seen at a location. There is often a temptation to record this in one long line of information as shown below in Table 4-6:

Table 4-6 Basic record structure on a fieldslip.

id	waypoint	animal	habitat
1	12	2 x impala 1 x buffalo	Grass land

Though this is easy for a human to read it becomes very difficult for a computer to extract the animals from the animal column. In fact plotting this in the GIS would be very difficult indeed. A better example might be to put the animals into the column header. It becomes clear how this is a better structure because the GIS can then plot the data for waypoint 12 as two different attributes.

Table 4-7 A basic record structure favouring plotting data in a GIS.

id	waypoint	impala	buffalo	Habitat
1	12	2	1	Grass land

The disadvantage of this structure is that the GIS requires a column for every possible animal. This becomes problematic especially if the slip needs printing because the page could be very long with dozens of animals and therefore not very practical. Though the second example is better for the computer and the GIS, it is not very practical for a human. A hybrid of the two is needed for the database schema. It is very common to record data manually in the field using the original format but to transcribe it into the database in a different manner. The best method to compromise between the two is to split the data into

different tables; one containing data about the waypoint and one containing data about the animals, this is shown in Table 4-8.

Table 4-8 A relational data structure.

id	waypoint	habitat
1	12	grassland

waypoint	animal	number
12	impala	2
12	buffalo	1

A relational database tool such as Microsoft Access can be used to create this type of data. In the strictest terms this structure is not as far normalised as might be required because the animals are still listed as string values. As was shown in Table 4-2 string values are not ideal as they introduce error into any database. The best method for storing values is shown in Table 4-9. In some occasions it is perfectly acceptable to reformat Table 4-9 so that there is a tally against each animal id instead of a separate row for it.

Table 4-9 Many to many data schema.

id	waypoint	Habitat
1	12	Grassland
2	13	Grassland

waypoint	Animal_id
12	1
12	1
12	2

id	animal
1	impala
2	buffalo

The advantage of keeping data in a relational structure is that the data can be re-structured on demand to fit any of the tables shown in this section. Simple SQL queries can re-arrange the fields to form a structure suited to human reading or computer plotting as was shown in Table 4-8.

4.9.1 Understanding query operators

Now that we have our data as a table that can be easily read by the database, we can display it in the GIS. There are some important commands that need to be included in understanding selecting of data. The exact methods for implementing these commands (or operators) are shown below in Section4.9.2.

In the case of habitat study we might want to preferentially select a specific animal or exclude certain locations. These criteria are called the logical operators. The basic logical operators are AND, OR, NOT and XOR. They all have specific functions in choosing the correct data. The effect of the logical operators on the data presented in Table 4-4 is shown below.

If only the locations where impala and buffalo were seen we would use an AND query. This would leave only locations 12 and 15.

Table 4-10 Results of an AND query on the basic data.

species	habitat	waypoint	x	y	weather	height
impala	Grassland	12	55234	233444	clear	500
buffalo	Grassland	12	55234	233444	clear	500
impala	Grassland	15	55420	232418	clear	450
buffalo	Grassland	15	55420	232418	clear	450

Often this is a difficult concept for new users who expect the query to return locations 12,13,15,17 and 18. This is because the OR and AND operators are often confused because of their use in everyday language. Users often expect AND to return all impala and all buffalo sightings but it only returns the areas where both of them are found together. The OR query selects those where one or the other or both were observed.

Table 4-11 Results of an OR query on the basic data.

species	habitat	waypoint	x	y	weather	height
impala	Grassland	12	55234	233444	clear	500
buffalo	Grassland	12	55234	233444	clear	500
impala	Grassland	15	55420	232418	clear	450
buffalo	Grassland	15	55420	232418	clear	450
buffalo	Grassland	17	55412	235212	cloudy	422
impala	Forest	18	55214	235444	cloudy	455
impala	Forest	13	55280	233501	raining	200

The next logical operator is NOT and is used to exclude data. For example, sightings that were not conducted during clear conditions would result in the table shown below.

Table 4-12 Results of a NOT query on the basic data.

species	habitat	waypoint	x	y	weather	height
cheetah	grassland	14	55520	232500	cloudy	480
buffalo	Grassland	17	55412	235212	cloudy	422
cheetah	Grassland	17	55412	235212	cloudy	422
impala	Forest	18	55214	235444	cloudy	455
impala	Forest	13	55280	233501	raining	200

The final operator to be discussed here is the XOR query. XOR stands for Exclusive OR. When using an OR query the data returned included both the locations where both impala and buffalo were seen such as 12 and 15 as seen in Table 4.10 and the locations where one or the other is present such as 17, 18 and 13. XOR only returns 17, 18 and 13 but not 12 and 15.

Table 4-13 Results of an XOR query on the basic data.

species	habitat	waypoint	x	y	weather	height
buffalo	Grassland	17	55412	235212	cloudy	422
impala	Forest	18	55214	235444	cloudy	455
impala	Forest	13	55280	233501	raining	200

The basic logical operators can obviously be combined to return data in a more complex form by excluding some data and combining other data. Almost all operations in GIS spatial analysis involve the use of the simple Boolean operations, whose use in spatial analysis is shown in Chapters 3 and 7.

4.9.2 Implementing queries in a database

The rest of this chapter will concentrate on how the above described clauses can actually be used in SQL statements to select the required data. All the following examples use industry standard code that can be lifted directly into almost any GIS.

To plot the data onto the GIS it is important to understand the methods for selecting only the correct data. The Bogda Shan Expedition collected large quantities of geographical data. The biggest files generated were from downloaded GPS waypoints and trackpoints. The expedition collected 12,855 GPS points in a transect across the mountains. 11,233 of these had recorded height or elevation data. If the team was only interested in plotting the data that had accompanying height information, then a simple SQL query can be used. SQL queries usually contain at least two pieces of information, a command and a clause. A command tells the database what is required, this would most commonly be a command to select information without affecting the rest of the table, though it could be a command to delete or change the data.

The query shown below is selecting information about the GPS data according to a specific criterion; i.e. where the Elevation field contains data. This is referred to as the clause part of the query. The clause tells the database when to use its command. We are interested in the *select* command, when the Elevation field has a record of the elevation. The database assesses whether a record has data or not by describing it as Null (*empty*) or Not Null (*contains information*). Therefore our simple query would look like:

```
SELECT [GPS Data].Elevation
FROM [GPS Data]
WHERE ((([GPS Data].Elevation) Is Not Null));
```

The first line is the command section, in this case a simple select command, and we are interested in the *Elevation* field from the table *GPS Data*. The second line tells the database specifically the table to use and the third line contains the criterion. This is a very simple example; to be more useful to us we also want the database to return all the other fields in the table. The current query would simply return a list of elevation data. To return all fields where the GPS has recorded elevation data we need to specify them in the query.

```
SELECT [GPS Data].[Index], [GPS Data].[Sequential Numbering System], [GPS Data].GPS,
[GPS Data].[Way point], [GPS Data].[X Reference], [GPS Data].[Y Reference], [GPS
Data].[Estimation of error], [GPS Data].Satellites, [GPS Data].Elevation, [GPS Data].Name, [GPS
Data].Time
FROM [GPS Data]
WHERE ((([GPS Data].Elevation) Is Not Null));
```

Most often if all fields are required a wildcard character can be used to specify select all. It is the use of the wildcard character that makes the FROM statement important.

```
SELECT *,
FROM [GPS Data]
WHERE ((([GPS Data].Elevation) Is Not Null));
```

This now returns a replica of the original GPS Data table but with only 11,233 records. A query is transient; it does not make a hardcopy of the data and must recreate the list every time the list is required. This is useful if the table may change regularly but is slower than accessing an actual data table. To analyse the data after the Bogda Shan Expedition it would be more useful to have the fields containing GPS elevation data available as an actual table. To do this we can modify the SQL command slightly. We are still interested in GPS data where the receiver has recorded elevation information, but we want the database to put this data somewhere permanent. Consider the simple example again:

```
SELECT *, INTO [GPS Height Table]
FROM [GPS Data]
WHERE ((([GPS Data].Elevation) Is Not Null));
```

In the new example the command line has been altered to include the additional instruction INTO [GPS Height Table]. This instruction tells the database to take the selected fields and make a new permanent table called GPS Height Table. If the INTO table already exists this query will always destroy the old INTO table and make a new one of the same name. If we had a table set up in advance or if we were adding to a GIS that already existed we could add our data to that table without altering it by using an INSERT INTO command. An INSERT INTO command is often referred to as an append query.

```
INSERT INTO [GPS Data] (GPS Height Table)
SELECT *,
FROM [GPS Data]
WHERE ((([GPS Data].Elevation) Is Not Null));
```

The other type of query that can be used in analysing data is the UPDATE command. This changes data from one value to another. For example to change all co-ordinate from one system to another you might apply a formula to all the latitude and longitude values.

```
UPDATE [Waypoints] SET [Waypoints].[X ref] = ([Waypoints].[X ref]-10000)*1.41
```

These are the basic clauses and commands used in database structures. They allow data to be prepared and analysed by the GIS even if the information collected in the field was not in a structure the database is initially set up to use. The methods for doing this querying are detailed below in Table 4-14.

Table 4-14 Summary of commands and clauses for queries.

SQL Queries	Logical Operators		Commands	
	OR	Selects A or B (includes both)	SELECT	Selects data for a query but does not write the data anywhere.
	AND	Selects only where A and B are both equal to a criteria	INTO	Creates a new table for the SELECT query.
	NOT	Selects only where A and B are not both equal to a criteria.	INSERT INTO	Puts the results of the SELECT query into an existing table.
	XOR	Selects only where A or B is equal to a criteria but not where both of them are equal to the criteria.	UPDATE	Updates data in an existing table.