A Hydrographic Database built on Montage and S-PLUS

W. E. Farrell, J. Gaffney, J. Given, R. D. Jenkins
Science Applications International Corporation,
San Diego, California

N. Hall
National Oceanographic Data Center,
Southwest Liaison Office
San Diego, California

March 1994

Sequoia 2000 Technical Report 94/47
University of California
Berkeley, CA 94720

Contents

1. Summary

A powerful software environment for research can be created by combining database management systems and commercial software for analysis and display of data. This provides a richer programming environment and more sophisticated graphical possibilities than are offered by present database systems. It provides a data storage and retrieval system that adheres to a standard syntax, SQL, and which is more robust than file-based approaches.

Only recently has it become possible to combine these two kinds of software, but increasingly database and scientific application software has the open architecture that is conducive to the data sharing that makes such combinations possible. New database systems also accommodate the complex types of data of interest to scientists.

We have a way to link appropriate database software and scientific application software which is called the Generic Database Interface (GDI). In essence, it consists of a core set of functions, complemented by product-specific interface code which is unique to each database product and each application product. The GDI provides the mechanism for both submitting a SQL query to any supported database and also handling the data that are exchanged in satisfaction of the query. We illustrate the strength of this approach by applying it to tasks involving selection and display of ocean temperature data using the Montage database management system and the S-PLUS statistics application. The examples illustrate a few of the novel capabilities of this approach; they do not comprise a complete data processing application.

This work is part of the Sequoia 2000 project of the University of California and is an example of an end-to-end scenario involving vector data[1]. The oceanographic data and the Montage schema are described in Sections 2 and 3. The schema is essentially a relational mapping of a file structure well known in the community. Section 4 summarizes some of the advanced features of Montage which are showcased in the scenario examples and Section 5 summarizes pertinent features of S-PLUS and the Generic Database Interface. The Montage queries and excerpts of the S-PLUS scripts are given in Section 6 and Section 7 explains the software architecture.

There are three appendices. Appendix A has details about the scenario, presented around a dataflow model. Appendix B lists the database schema along with its entity-relationship model. The S-PLUS figures are collected in Appendix C.

[1] This scenario has a narrower scope than other Sequoia 2000 scenarios because we have not incorporated project activities in networks, file systems or hierarchical storage.

2. Ocean temperature data

The data, approximately 4000 temperature profiles and 50 temperature/salinity profiles, were provided to NOAA's National Oceanographic Data Center by Canada's Marine Environment Data Service (MEDS). The data span 4 months of time (April through July 1993) and are just a part of the data being assembled and distributed as part of the Global Temperature-Salinity Pilot Project (GTSPP). The GTSPP is an international cooperation to acquire all possible data from real-time and delayed sources, assess their quality, and rapidly distribute them. As a first step toward modernizing the practice of managing ocean data, the GTSPP is expected to grow into the major provider of reliable, continuously updated, and quality-controlled hydrographic data.

MEDS is coordinating the design and implementation of a GTSPP-standard format for data exchange. The format is an extensible, hierarchical way to structure ASCII files. It is designed to accommodate temperature, salinity, and other oceanographic variables. The format accommodates not only essential ancillary information, such as position, time, and instrument type, but also optional measurements, such as sea surface weather conditions (if they are available), and, most importantly, data quality. The processing history (such as screening, editing or quality assignment) is captured as well as the data center performing the work. As a result, while a typical record for one "station" (one bathythermograph drop, for example), may contain 20 depth/temperature pairs, the accompanying descriptive information may double the data volume.

The GTSPP format is more complex, more bulky, and more variable than other formats in use. It was chosen as the basis of our schema because it has the variety and completeness needed for modern oceanographic research. The procedures used to load the GTSPP data into the database, hydrodb, are described in Appendix A.

3. Database schema

It was assumed, as a practical requirement, that the database schema should preserve all the attribute names and the essential structure of the GTSPP format. New attributes could be added, but no attributes nor tables should be eliminated. This is to make it easy for users to switch from file-based approaches to the database approach to data management. It also simplifies creation of GTSPP-structured data sets from the database for distribution.

The schema, described in Appendix B, is relatively simple and has only eight tables. These are formally described in the Montage data definition language in the first part of the appendix. The second part shows the tables and attributes arranged in an entity-relationship picture of the schema. The picture graphically shows how the key *ix_master* is used to join the principal tables, *master* and *profiles*. *Ix-master* must be unique over the database, and a table *ui_seq* is used to hold the next available value. Table *months* is used to relate month names to calendar position:

the names are needed by *m_time*. The work so far has not utilized the subsidiary tables of the GTSPP format so the practicality of this part of the schema is untested.

Attributes were introduced to take advantage of special features of Montage that rely on the abstract data types pertaining to time, arrays and (Cartesian) location. For example, latitudes and longitudes have been combined into a single attribute, *Location*, of Montage data type *Pnt* (point). Montage types *arrayof()* are used to hold the depth, temperature and data quality vectors. The examples show how useful this storage method is. An attribute, *m_time*, of Montage type *abstime* was introduced to support queries involving time intervals.

4. Database management system

Montage is an object-relational database management system, derived from Postgres, the research system currently utilized in the Sequoia project. Hydrodb is implemented on Montage because we wanted SQL to be the query language. We also wanted professional support to be available, for it is our intention to develop this prototype into a production database. Most of the examples, with appropriate changes in syntax, will run under Postgres.

The query examples were devised for their relevance to the temperature data set and to illustrate aspects of Montage that distinguish it from most other database management systems. These special features are discussed in the subsequent paragraphs. The query examples execute quickly on the small database. A attributes have been indexed, but this is the only optimization performed. Optimization will become more important as the database grows.

a. Spatial queries

Spatial queries involve geometrical things such as points, polygons, lines, etc., which are familiar in geographic information systems but new to database management systems. Montage supports abstract data types and functions appropriate for plane geometry through its “spatial blade”. The underlying Cartesian geometry limits the usefulness of the spatial blade for spherical problems. But it is advantageous to use the spatial types, even if the geometry is inexact, because the spatial attributes can be indexed and the SQL is compact. For example, query 6.1.1(a) tests whether a location is within a latitude-longitude box. Query 6.1.1(b) tests whether a location is within a given distance (in the Cartesian approximation) of a fixed point.

b. Array data

Profile data are stored in the table *profiles* as single objects with the Montage type *arrayof*(). There are four arrays, the array of depths (*Depth_Press*), the array of temperatures (*Prof_Parm*), and two arrays with quality values. To have the array data available as distinct objects within the database is a more useful approach than either of the conventional alternatives[2]. Several examples illustrate how subscripts can be used in a SQL statement to select specific values from an array.

Arrays of integer or real numbers can have no more than 2,000 elements; for doubles the limit is 1000 elements. Conveniently, the GTSPP format accommodates large arrays by splitting them into 1,500-element partitions. The format has pointers that link the partitions, and a similar method could be used in the database. This has not been implemented for there are only 10 values in the average profile, and no profile in the present data has more than 100 elements.

c. New functions

The architecture of Montage allows new C functions to be added to the command language through a simple procedure called "registering"[3]. These are linked dynamically and can be invoked as part of any query. In this way it is possible to implement an algorithm that would be clumsy or impossible to express in SQL. For example, mathematics functions usefully enhance SQL. The Unix Trig(3M) library has been registered in hydrodb and is used in one of the examples.

Two interpolation functions were written for hydrodb. They return the temperature of the water between measurement points of a profile. Interp1() performs linear interpolation, interp2() performs parabolic interpolation.

Velocity(), quite a different type of function, combines two array attributes, the temperature and salinity, to form a new (non-database) quantity, the predicted sound speed profile. The function implements one of the standard equations-of-state for seawater. In a similar fashion, other equations for the sound speed, or equations for *in situ* density, could be provided.

A different way to embed equation-of-state functions in the database would have been to define auxiliary tables and calculate the velocity profile for all data in one batch calculation. What is wrong with this approach? Maybe nothing. Space is saved using the function approach, but this is not a great advantage for hydrodb which, at its largest, will be only a few gigabytes in size. There is a small disadvantage in speed to the function approach (but we have not timed it), but the oceanographic algorithms are simple enough the effect is probably small.

One reason to use functions is so that data administrators editing data and oceanographers utilizing data can work simultaneously. Using the function to calculate derived properties on demand gives assurance that the most current data

[2] One alternative is to save the measurement values as separate database records. The other is to save the data as files, and store pointers to the files in the database.

[3] SQL functions can also be registered. Other database management systems support SQL functions but it is rare to have supportable C functions.

are utilized[4]. Another reason for having DBMS functions handle the evaluation of derived data is to ensure consistent algorithms are employed by all users and, incidentally, to shrink slightly the amount of software they need to maintain. Furthermore, we can imagine the equation-of-state algorithms, themselves, becoming objects in the database. This would simplify the maintenance of the functions and facilitate incorporation of new algorithms in the future.

d. Virtual attributes

Virtual attributes are attributes of the schema defined by registered C or SQL functions. Values of virtual attributes are calculated dynamically at query time. We have added *Location* (type *Pnt*) and *m_time* (type *abstime*) as virtual attributes. They are defined by SQL functions acting on GTSPP variables *Latitude* and *Longitude* (for Location) and the four pertaining to date and time. These are introduced as virtual attributes because we want the associated GTSPP variables to have primacy in the database, yet we want them to be cast automatically into the more powerful Montage representations.

e. Time

Time is always difficult to handle because its human representation obeys complicated rules of arithmetic. An important use of hydrodb will be to calculate smoothed temperature-depth data through application of four-dimensional space-time filters. To do this efficiently requires an attribute which is continuous in time over the range of the data and on which mathematical operations can be performed. Time represented in calendar and clock notation, as it is defined by the GTSPP format, won't do. Montage supports a continuous time variable through the abstract data type *abstime*. In hydrodb the variable *m_time* is formed (as a virtual attribute) from the time elements of the GTSPP format. Hydrodb uses GMT.

5. S-PLUS and the Generic Database Interface

S-PLUS is a well-known statistics and graphics application. It is representative of a growing class of scientific software with the following attributes:

- A rich variety of scientific functions
- Easy methods for plotting data, usually under the X system
- A scripting language
- An interface-builder
- The ability to make dynamic links to foreign functions

[4] Transaction management, triggers, and stored procedures are other ways to handle concurrent access and data integrity.

While powerful tools for analysis and display of data, these applications are poor tools for storage and selection of data, particularly complex data sets or data sets with copious ancillary parameters.

Our Generic Database Interface (GDI) is software that bridges the gap between database software and software used for analysis and display of technical data. From the user's point of view, the GDI is a small number of functions which are called from the scientific application, S-PLUS in this case, or from C or Fortran programs. Some functions perform administrative operations (such as connecting to a database) in a standard manner. More important are the functions that submit queries to databases and the structures through which the data of the queries are exchanged. Objects of great complexity can be passed back and forth with ease.

By linking high-level application programs such as S-PLUS to a database such as Montage it is possible to build complete data processing programs in the scripting language of the application. The need to master SQL in addition to a scripting language will be an inconvenience to some and for them higher-level interfaces can easily be created by a number of mechanisms. We believe these interface builders augment but do not supplant the low-level approaches exemplified in this report.

6. Query examples

The following queries illustrate the features discussed previously. Familiarity with relational databases and SQL is assumed. These examples show some of the approaches that would be used to construct a complete data processing system with these techniques. All the examples are concerned with selecting and displaying data.

6.1 Queries through the Montage monitor

The following examples illustrate the selection of data according to location, profile depth and profile temperature at a specified depth.

6.1.1 Selection of data from a geographic region

The queries of this section illustrate selection of data based on location. The first two examples use the Contains() function of the spatial blade. The results are not exactly as intended because the Earth’s curvature has been neglected. The third example uses spherical trigonometry.

a. Select data by location with respect to a latitude-longitude limits

The following query prints the latitude and longitude[5] of profiles taken within a specified rectangular region. The "where" clause of the query tests whether the

[5] Hydrodb follows the GTSPP convention that western longitudes are positive. The sign is reversed in the queries so map plots make geographic sense.

database attribute *Location* (the latitude and longitude taken as a pair) is within the desired region. The function *Pnt* casts a latitude and a longitude into an object of type point. Two of these define the boundaries of a box. The box should not span the date line, since *Longitude* is discontinuous there. The same problem, however, occurs if the region of interest is defined in the usual way by separately testing the latitude and longitude against upper and lower bounds.

```
select
    Latitude as Lat, -Longitude as Lon, ix_master, Data_Avail
from
    master
where
    Contains(Box(Pnt(10,-175), Pnt(20,-165)), Location);
```

b. Select data with respect to distance from a given point

The following query prints the latitude and longitude of profiles taken within 6 degrees of 15N 170W. Again, the *Pnt* function casts a latitude and a longitude into a an object of Montage type *Pnt*, taken as the center of a circle. The underlying mathematics assumes a Cartesian geometry. On the earth this circle is warped into an ellipse because of the distortion of longitudinal distance with latitude.

```
select
    Latitude as Lat, -Longitude as Lon, ix_master, Data_Avail
from
    master
where
    Contains(Circ(Pnt(15, 170),6),Location);
```

c. Select data with respect to distance from a given point using spherical trigonometry.

The following query shows the correct way to select all measurement locations within 6 degrees of a reference point. As can be seen, the trigonometric functions are as easy to use as any other. The cosine rule could easily be turned into a registered function.

```
select
    Latitude as Lat, -Longitude as Lon,
    acos(cos((90-Latitude)/57.2958)*cos((90-15)/57.2958)
       + sin((90-Latitude)/57.2958)*sin((90-15)/57.2958)
      *cos((-Longitude+170)/57.2958))*57.2958 as dist_in_deg
from
    master
where
    (acos(cos((90-Latitude)/57.2958)*cos((90-15)/57.2958)
       + sin((90-Latitude)/57.2958)*sin((90-15)/57.2958)
      *cos((-Longitude+170)/57.2958))*57.2958 ) < 6.;
```

6.1.2 Select data according to location and depth extent.

This query prints the latitudes and longitudes of all profiles which extend deeper than 850 meters and which were taken within a specified latitude-longitude box. It illustrates how arrays are manipulated. It also illustrates the "join" between tables *master* and *profiles* through the key *ix_master*.

```
select
    t1.Latitude as Lat, -t1.Longitude as Lon, t1.ix_master,
    t2.Depth_Press[t2.No_Depths] as depth
from
    master t1, profiles t2
where
    t2.Depth_Press[t2.No_Depths] > 850 and
    t1.ix_master = t2.ix_master;
```

6.1.3 Select data according to location and temperature value.

This query uses an interpolation function to find locations where the temperature at 100 meters is warmer than 20°C.

```
select
    t1.Latitude as Lat, -t1.Longitude as Lon,
    interp1(100, t2.Prof_Parm, t2.Depth_Press) as Temp
from
    master t1, profiles t2
where
    t1.ix_master = t2.ix_master and
    Contains(Circ(Pnt(15,-170),6), t1.Location)  and
    interp1(100, t2.Prof_Parm, t2.Depth_Press) > 20 ;
```

6.2 Queries from S-PLUS

These examples illustrate selection and display of data from within the S-PLUS application. Certain initialization steps have been omitted.

The first example is discussed in its entirety. In subsequent examples the SQL queries are always given, but the rest of the S-PLUS script is only summarized. The figures produced by the examples are included in Appendix C.

6.2.1 Plot profile locations

This script maps the locations at which profiles are available. The first block builds a normal SQL query by pasting text into a string, q621. Variables are re-named for convenient citation within the argument lists of S-PLUS functions.

```
cat("Querying the database...\n")
q621 <- "select Latitude as lat, -Longitude as lon from master "
q621 <- paste(q621, " where Contains(Box(Pnt(-60,-180), Pnt(60,180)),
              Location)")
```

In the second block the query is submitted using the GDI function sdi.submit() and the result is returned in the S-PLUS structure m621.

```
      m621 <- sdi.submit(q621, -1)
```

The final block makes the plot. Region() is a function in S-PLUS we wrote to plot the map. Notice how the coordinates are referenced in the structure m621.

```
      title <- paste ("Q621: Profile Locations")
      region(m621$lat, m621$lon, -60, 70, 60, 270, title=title, nopoly=1)
```

6.2.2 Plot a histogram of the latitudes and longitudes of the data available for June.

This script draws histograms of the latitude distribution and longitude distribution of June 1993 measurement locations. In the first part, a string, q622, is formed with the desired SQL query.

```
      q622 <- "select Latitude as lat, -Longitude as lon from master "
      q622 <- paste(q622, "where Obs_Month::int=6 and Obs_Year::int=1993")
```

The query is submitted to Montage as before and the result returned in the structure m622. The following block plots the histogram of latitudes using the S-PLUS function hist().

```
      title1 <- "Q622: Latitude - Distribution for June"
      title2 <- "Latitude (degrees)"
      hist(m622$lat, main=title1, xlab=title2, ylab="Site Count")
```

6.2.3 Plot subsurface temperature profiles

This script obtains the locations of all available profile measurements and these are then plotted on an S-PLUS map. The user then draws a polygon on the map by choosing its vertices. The vertices are incorporated into a second SQL query which selects profile data for all locations falling within the polygon. Finally, temperature is plotted as a function of depth for the chosen locations.

The first query, string q623a, selects the location of every profile in the database. This is only feasible for the relatively small amount of data in the prototype.

```
      q623a <- "select Latitude as lat, -Longitude as lon from master"
```

The query is submitted and the set of latitude-longitude pairs returned in the structure m623. The locations are plotted on the map by the function region(), previously described. Region(), as used here, returns the coordinates of the picked vertices in the S-PLUS variable poly.

```
      poly  <- region(m623a$lat,  m623a$lon,  -60,  70,  60,  270,  title=t1)
      npoly <- length(poly$x)
```

The next block forms a SQL query that selects the data from all locations within the polygon. The start of the "where" clause of this query, shown below, illustrates how the first polygon vertex is specified to the Montage MakePoly() function.

```
q623b <- paste(q623b, " where a.ix_master  =  b.ix_master ")
q623b <- paste(q623b, " and Contains(MakePoly(")
q623b <- paste(q623b,"'(", round(poly$y[1],2), ",", round(poly$x[1],2),
                    sep="")
```

Additional vertices are specified by adding more text to the string q623b until the polygon is closed. This query is submitted and the result passed back in the S-PLUS structure m623b. Profiles are plotted, one to a panel, by an S-PLUS function written for this purpose, plot.dt().

```
t1 <- "Q623b: Selected Profile Data"
plot.dt(m623b,  t1)
```

6.2.4 Plot a histogram of the temperature at 100 meters for June.

This script uses one of the interpolation functions to calculate the sub-surface temperature beneath a specified region of ocean. A histogram is plotted of the temperature values.

The parabolic interpolation function, interp2(), is incorporated in the "select" clause. The "where" clause limits the time interval to the 6th month and the location to a specified latitude-longitude box. More delicate control over time windows would be accomplished with the database attribute *m_time*.

```
q624 <- "select interp2(100, b.Depth_Press, b.Prof_Parm) as temp "
q624 <- paste(q624, " from master a, profiles b ")
q624 <- paste(q624, " where a.ix_master = b.ix_master and
                a.Obs_Month::int=6 ")
q624 <- paste(q624, " and Contains(Box(Pnt(-10,140), Pnt(10,160)),
                a.Location)")
```

The query is submitted and results returned in m624. Then the histogram is drawn using a built-in S-PLUS function.

```
m624 <- sdi.submit(q624, -1)
hist(m624$temp, main=title1, xlab="Temperature (Celsius)",
     ylab="Site Count")
```

6.2.5 Plot the isotherms at a 100 meter depth

This script illustrates the use of temporary database tables, and entails two separate queries. Two queries are needed because one class of data, data recorded on buoys, needs to be treated differently from the other class, data recorded from ships. The buoys drift slightly, and their data must be smoothed so they do not bias the contours. Data smoothing is applied as an integral part of the SQL "select". First the locations and subsurface temperatures of buoy data are averaged and saved into a temporary table. Then the locations and subsurface temperatures of single-measurement stations are appended to the temporary table. The entire table is returned to S-PLUS and the temperatures then gridded and contoured using standard S-PLUS functions. The SQL part of the program is shown without the S-PLUS context to make it more readable.

An SQL query-string is formed that finds and averages buoy data in the region of interest. The buoys are known to have 5-digit *Cruise_ID* values beginning in 51 and 52. In the "select" clause, the built-in Montage function, avg() calculates the mean buoy location. Avg() also works when its argument is a set of temperatures returned by our registered function interp2(). The second line of the “where” clause ensures that the profile extends at least as deep as the target depth. The 'group by' clause ensures that all data from a given buoy are treated together.

```
create table temp_tbl as
select Cruise_ID, count(Cruise_ID),
    avg(a.Latitude) as lat, avg(-a.Longitude) as lon,
    avg(interp2(100, b.Depth_Press, b.Prof_Parm)) as temp
from master a, profiles b
where a.ix_master = b.ix_master
    and b.Depth_Press[b.No_Depths] >= 100
    and a.Obs_Month='06'
    and Contains(Box(Pnt(-10,140),Pnt(10,160)),a.Location)
    and (a.Cruise_ID like '51___' or a.Cruise_ID like '52___')
group by a.Cruise_ID
```

The query is submitted. This leaves a temporary table, *temp_tbl*, in the database holding the first part of the data. The variable m1 has information pertaining to the success of the query, but no data are returned.

```
m1 <- sdi.submit(q1, -1, 1)
```

A second query, with no averaging, is formed and submitted.

```
insert into temp_tbl
select Cruise_ID, '1',
    a.Latitude as lat, -a.Longitude as lon,
    interp2(100, b.Depth_Press, b.Prof_Parm) as temp
from master a, profiles b
where a.ix_master = b.ix_master
    and b.Depth_Press[b.No_Depths] >= 100
    and a.Obs_Month='06'
    and Contains(Box(Pnt(-10,140),Pnt(10,160)),a.Location)
    and a.Cruise_ID not like '51___'
    and a.Cruise_ID not like '52___'
```

The results of both queries have been accumulated in table *temp_tbl*. The third query returns the contents of *temp_tbl*, in the structure m625.

```
q625 <- "select temp, lat, lon from temp_tbl "
m625 <- sdi.submit(q625, -1, 1)
```

The S-PLUS functions interp() and contour() grid the data and draw the plot.

```
      fit <- interp(m625$lon, m625$lat, m625$temp)
      contour(fit, main="Q625: June Temperature at 100 meters",
             xlab="Longitude (degrees)",ylab="Latitude (degrees)")
```

7. Polyglot programming

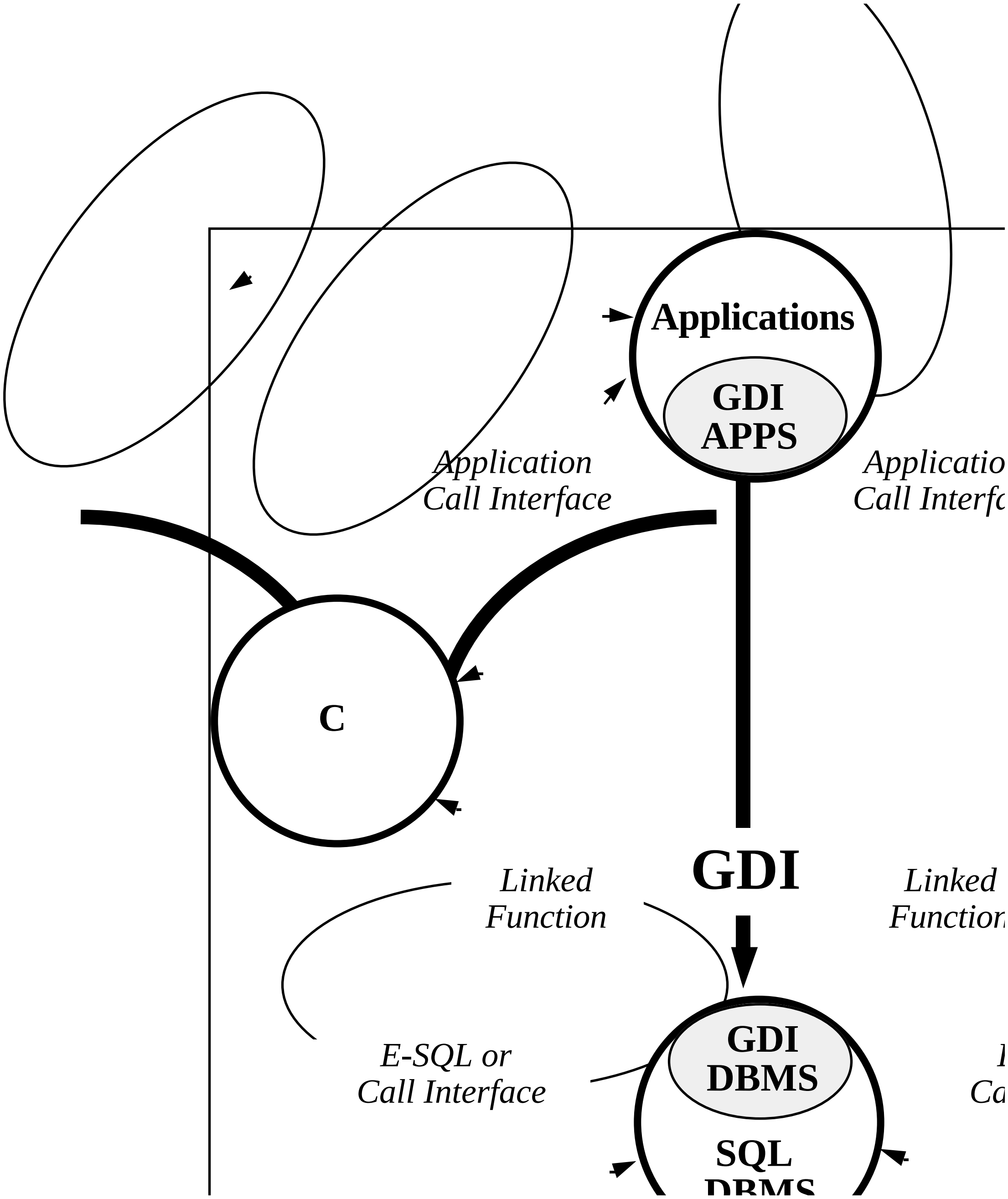
The software of this report illustrates a phenomenon of growing significance: in many respects the most productive technique for writing software for data analysis and display is to use a combination of languages. The traditional approach, adopt one language, usually C, and buy or develop function libraries for one's scientific domain, is becoming obsolete with the burgeoning of powerful software "environments". Members of the Sequoia community have themselves developed some of these environments, and the environments have been a significant part of the software architecture from the start.

There are a number of features usually shared by these environments: a rich collection of functions, a high-level language, an openness to C and Fortran, and, often, a toolkit for building graphical interfaces. Postgres, IDL, AVS, Tcl/Tk are familiar to the Sequoia project. Besides Montage and S-PLUS, essentially all database systems fit this model as well as Matlab, Mathematica, etc.

These systems have taken hold because, for many users, the environments have attained a better equilibrium than traditional languages between the extremes of flexibility and functionality. The power comes at a price, however. The proliferation of the languages, the incompatibility of the data interfaces, and the large amount of legacy software makes it hard, both for the computer scientists and the earth scientists, to judge the best software to adopt. Furthermore, the technology is rapidly changing, and one fears that the best choice now may be obsolete in the future.

Figure 1 shows some relationships between traditional languages and these "environments". In this picture SQL Database Management Systems (DBMSs) have been separated from Applications and in each of these those for which there is a generic database interface (GDI) are separated from the rest. Arcs link the four kinds of software, pointing from the caller (client) to the called (server) module. For example, vendors of database systems commonly provide two levels of support for C and Fortran programs, embedded SQL or a proprietary function library. This is represented by the arcs from the languages to the DBMS circle. Less frequently (there are examples in this paper) DBMSs permit SQL programs to execute a C or Fortran function. This capability is represented by arcs at the bottom of the page pointing in the opposite direction.

The top half of the page shows the analogous relationships between the "applications", as we have defined them here, and programs written in C and Fortran. As with database systems, not every application supports bi-directional interfaces to both programming languages. It is notable in the diagram that applications and databases are disconnected, other than through the generic database interface. We are not aware of any commercial software that bridges this gap in *either* direction.



The data loader (written in C) was by far the hardest software of this scenario. The registered functions (written also in C) each have about 200 lines of source code. The schema and all the database programs (written in SQL) total approximately 350 lines. There are 400 lines of S-PLUS scripts, 200 for the plotting program and 200 for all the scenario examples. Thus, with less than 1,500 lines of software a sophisticated scientific database with rudimentary graphical interface has been obtained. Appropriate use of several languages was the key to this economy.

8. Conclusions

Obviously, the science content of the query examples is not deep, but our purpose has been to explain database and software technology, not oceanography. The database approach has brought a substantial amount of hydrographic data into a state of easy accessibility in just a few months. The incorporation of S-PLUS has let us interrogate the database from a scripting language as well as display and analyze the returned sets.

The careful thought others put into defining the GTSPP format was a help in the beginning, but we don't yet understand all its features. A committee is continuing to refine the format, but future changes will be easy to incorporate because database management systems allow columns of tables to be added or deleted without the necessity of reloading all the data.

Montage is superior to other database systems with which we are familiar. It would be convenient to have support for spatial queries on a sphere. We were able to devise powerful SQL queries and S-PLUS scripts with a small fraction of the effort we would expect to spend on a C or Fortran program. An example was the matter of smoothing, in time, and contouring data that had been acquired both from moored and moving platforms. It was surprising to realize that many of the problems we experienced constructing queries reflected science issues, not software issues.

If all available hydrographic data were to be absorbed, this database could become three orders of magnitude larger. Furthermore, data collection continues, and if these future data are incorporated as well, each year would add 30 megabytes, three-times the amount of data presently installed. These larger data sets will raise issues that were not important with the test data: efficient indexing, expansion of the size of some tables, utility of virtual attributes.

8. Acknowledgments

Bonnie MacRitchie and Jean Anderson conceived the Generic Database Interface, developed the earliest implementations, and are continually helping extend its capabilities. Jean Anderson was our Montage DBA and wrote the C functions for interpolating data and calculating the *in situ* sound speed. We had several useful discussions with Warren White, Director of the JEDA Data Center at Scripps Institution of Oceanography. Melanie Hamilton (NODC) and Robert Keeley (MEDS) supplied the data and answered many questions, as did Doug Hamilton (NODC).

Appendix A. Scenario description

For didactic purposes, the scenario is presented as the orderly sequence of processes and flows diagrammed in the dataflow model, Figure A-1. Reading the diagram from left to right, the external inputs of the scenario are the hierarchical file holding data (measurements pertaining to 4,000 profiles) and a collection of E-mail and other documentation that describes the structure of the file and the allowed values of its fields. The file structure was mapped to a relational schema in the Montage data definition language. Some attributes were dropped from tables where they were redundant. Some new attributes were defined in order to take advantage of Montage abstract data types (*Location*, *m_time*). Two new tables were created (*months* and *ui_seq*), but these were associated with details of data management, not oceanography. We expect more attributes and tables will be added in the future, both to make querying easy and to support future versions of the GTSPP format.

A parser, process 1, translated the single, hierarchical, GTSPP file into multiple flat-files, one for each table in the schema. Coding the parser was an iterative process, for the file did not exactly match the advertised structure, and as we gained experience a number of quality checks were built-in.

Given the files holding the Montage schema and the data for the tables, it is easy to insert the schema into the database (process 2) and then copy the data from the files to the database (process 3). Table A-1 shows some time and size measurements for this part of the scenario. The elapsed times are acceptable, but faster performance would help. Assuming a linear increase in time with data size, if the entire hydrographic data collection is 1,000 times larger than this experimental data set, it will take a week to load the database.

The 3-fold expansion in data volume is notable, and should be investigated further. An inconsistency among tables is notable. Some increase in size is to be expected as efficiently-packed character data are expanded into standard types. The space taken by indices and system catalogs is small, but as the database expands more indices will be added. We think that the scale-up will be less than linear.

It is much slower to load the data by performing database inserts directly into Montage from the data parser. For example, our first method performed a database insert for each sequential record in the file. The time build the database was 6 hours (but this was on a 4-time slower SPARCstation 2). It was somewhat more efficient to read all related records of a single hierarchical cluster and then do the database inserts, but this only reduced the time by 30% to 4 hours.

Although it requires more free space on the file system, the two step procedure, in which step one is to create a file for each table, and step two to load the files one by one, is the only reasonable approach for large data sets.

Figure A.1

Data Flow model of the scenario

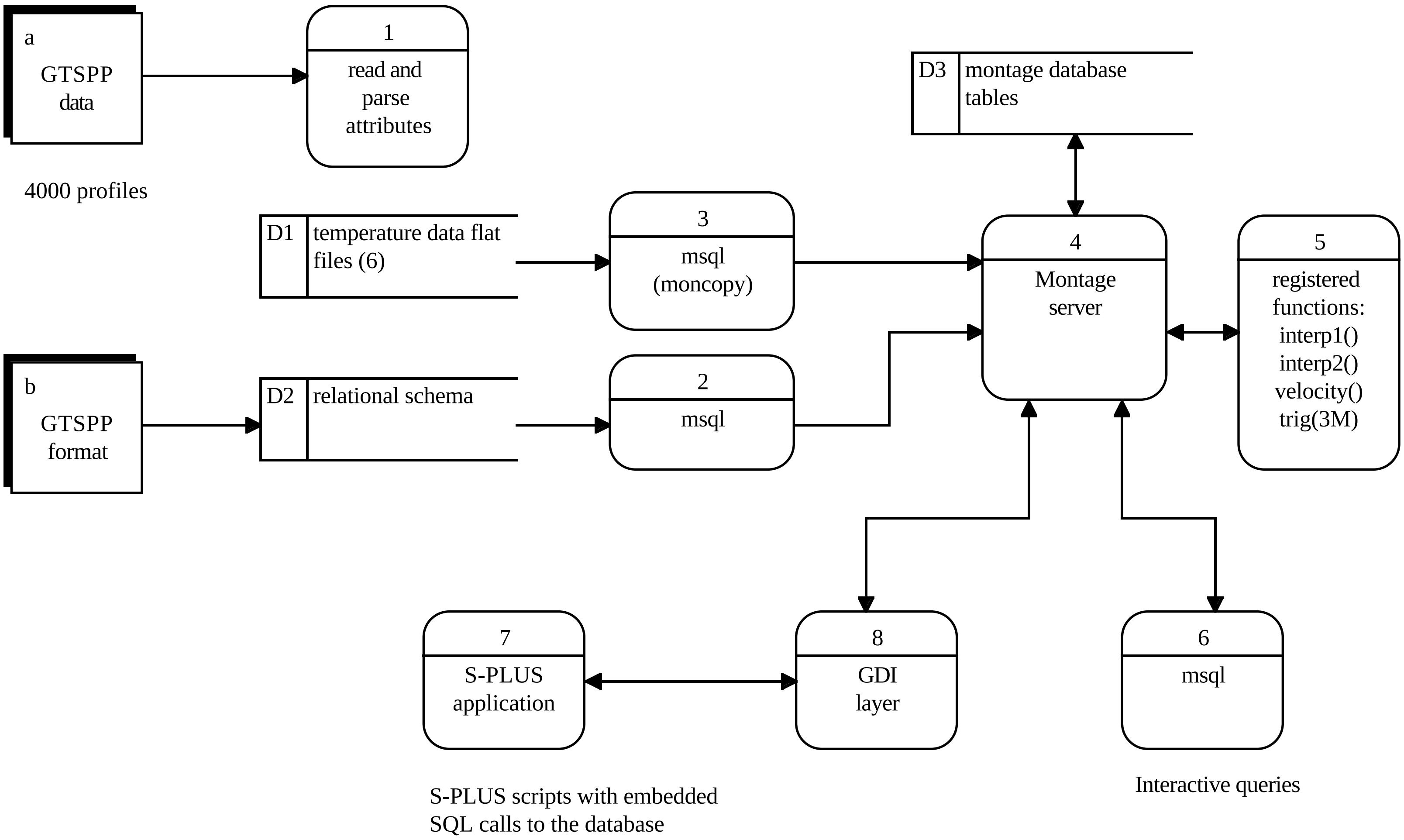


Figure A-1. Data Flows associated with hydrodb

Table A-1. Metrics associated with creating hydrodb.

Description	Time (seconds)	Size (megabytes)	Discussion
Size of original GTSPP data-file		2.44	
Time to parse into flat files	30		SPARCstation 10/41
Cumulative size of the flat files.		3.39	profiles 1.1 history_group 1.0 master 0.6 surface_codes_g. 0.6
Time to load using moncopy	570		writing records directly into the database using SQL "inserts" is nearly an order of magnitude slower.
Size of database (including a few hundred kilobytes for indices)		6.66	profiles 1.2 history_group 2.0 master 0.9 surface_codes_g. 2.0
Size with all system catalogs etc.		7.6	The system overhead will not change significantly as more data are added. Indices may become more important as they are used more intensively.

The Montage server, process 4, represents the database manager, and the database tables are represented by data store D3. User-defined functions we have added and which are dynamically linked, are represented by process 5, montage extensions. Functions interp1(), interp2(), and velocity() were written in the C language. SQL functions were written as well to define two virtual data types (m-time and Location) in the schema.

Processes 6 and 7, at the bottom of the figure, represent the software through which the user accesses the database. Section 6.1 has examples of queries submitted to the msql client program. Section 6.2 has examples of queries submitted through S-PLUS.

Appendix B. Database schema

This appendix gives the schema of hydrodb in the Montage data definition language. There are 8 tables, 6 inherited from the GTSPP format, and two we have added. There are approximately 50 attributes.

```
create table master of new type master_t (
        MKey                    integer,
        ix_master               integer,
-- Added "ix_master" unique primary key for master
        One_deg_sq              integer,
        Cruise_ID               char(5),
        Obs_Year                char(4),
        Obs_Month               char(4),
        Obs_Day                 char(4),
        Obs_Time                char(4),
        m_time                  abstime virtual,
-- note: m_time is virtual column (cf: function m_time  below)
        Data_Type               char(2),
        Iumsgno                 integer,
        Stream_Source           char(1),
        Uflag                   char(1),
        MEDS_Sta                integer,
        Location                Pnt virtual,
-- note: Location is of data type Pnt (requires Spatial Blade)
-- Location is a virtual column (c.f. function m_time above)
        Latitude                double precision,
        Longitude               double precision,
        Q_Pos                   smallint,
        Q_Date_Time             smallint,
        Q_Record                smallint,
        Up_Date                 integer,
        Bul_Time                real,
        Bul_Header              char(06),
        Source_ID               char(04),
        Stream_Ident            char(04),
        QC_Version              char(04),
        Data_Avail              char(01),
        No_Prof                 smallint,
        Nparms                  smallint,
        Nsurfc                  smallint,
        Num_Hists               smallint
);
create table profiles (
        ix_master               integer,
-- ix_master is foreign key for table profiles.
        Profile_Type            char(4),
        Profile_Seg             smallint,
        No_Depths               smallint,
        D_P_Code                char(01),
        Depres_Q                arrayof(int),
        Prof_Q_Parm             arrayof(int),
        Depth_Press             arrayof(real),
        Prof_Parm               arrayof(real)
-- note : use of array datatypes.
);
```

```
-- Added MKey to profile_info,surface_par_group,suface_codes_group
-- and history_group. This preserves the GTSPP relationship with master

create table profile_info  (
        MKey                    integer,
        No_Seg                  smallint,
        Prof_Type               char(04),
        Dup_flag                char(01),
        Digit_Code              smallint,
        Standard                smallint,
        Deep_Depth              char(05)
);
create table surface_par_group (
        MKey                    integer,
        Pcode                   char(04),
        Parm                    double precision,
        Q_Parm                  boolean
);
create table surface_codes_group (
        MKey                    integer,
        SRFC_Code               char(04),
        SRFC_Parm               char(10),
        SRFC_Q_Parm             char(01)
);
create table history_group (
        MKey                    integer,
        Ident_Code              char(02),
        PRC_Code                char(04),
        NODC_Version            char(04),
        PRC_Date                char(08),
        Act_Code                char(02),
        Act_Parm                char(04),
        Aux_ID                  real,
        Previous_Val            real
);
-- Create and load supporting tables.
create table ui_seq (
    next_ui_seq     integer
);
insert into ui_seq values (3000);
create table months (
    Obs_Month       char(4),
    month   char(3)
);
```

Figure B1

ER model of the schema

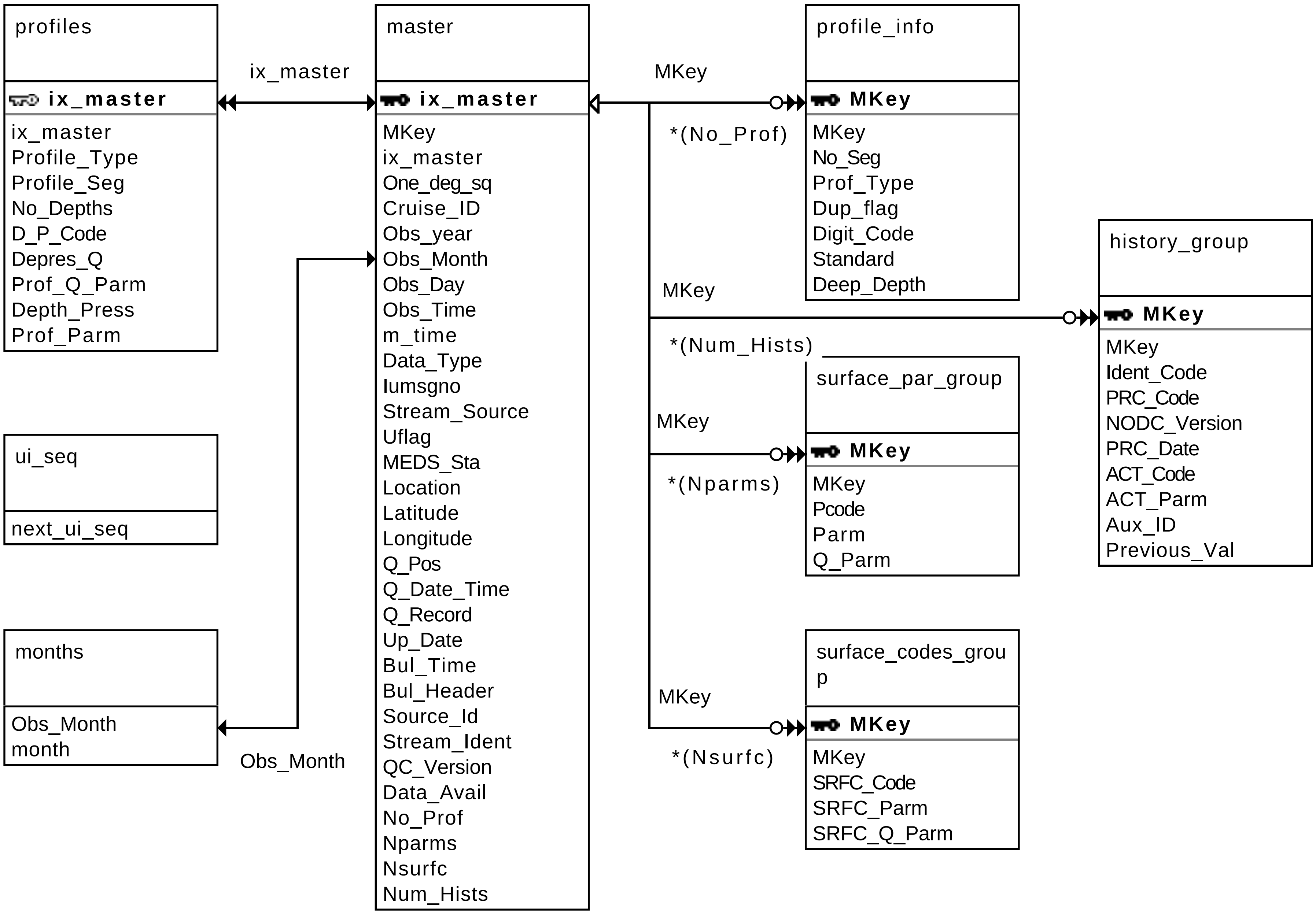


Figure B-1. Entity-relationship model of hydrodb schema

Appendix C. S-PLUS figures

Figure C1. q621.ps

Figure C2. q622.lat.ps

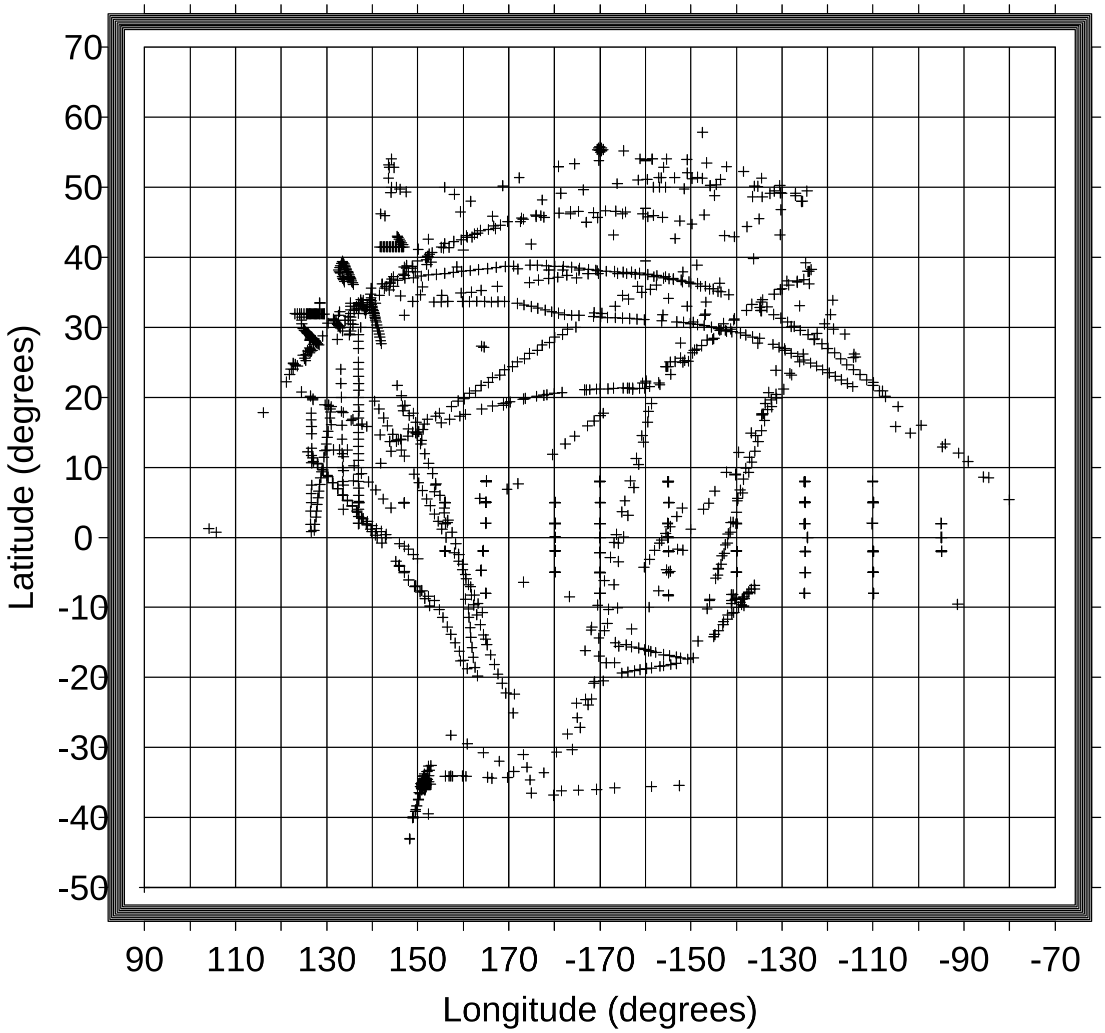
Figure D3 q623a.ps

Figure C4. q623b.ps

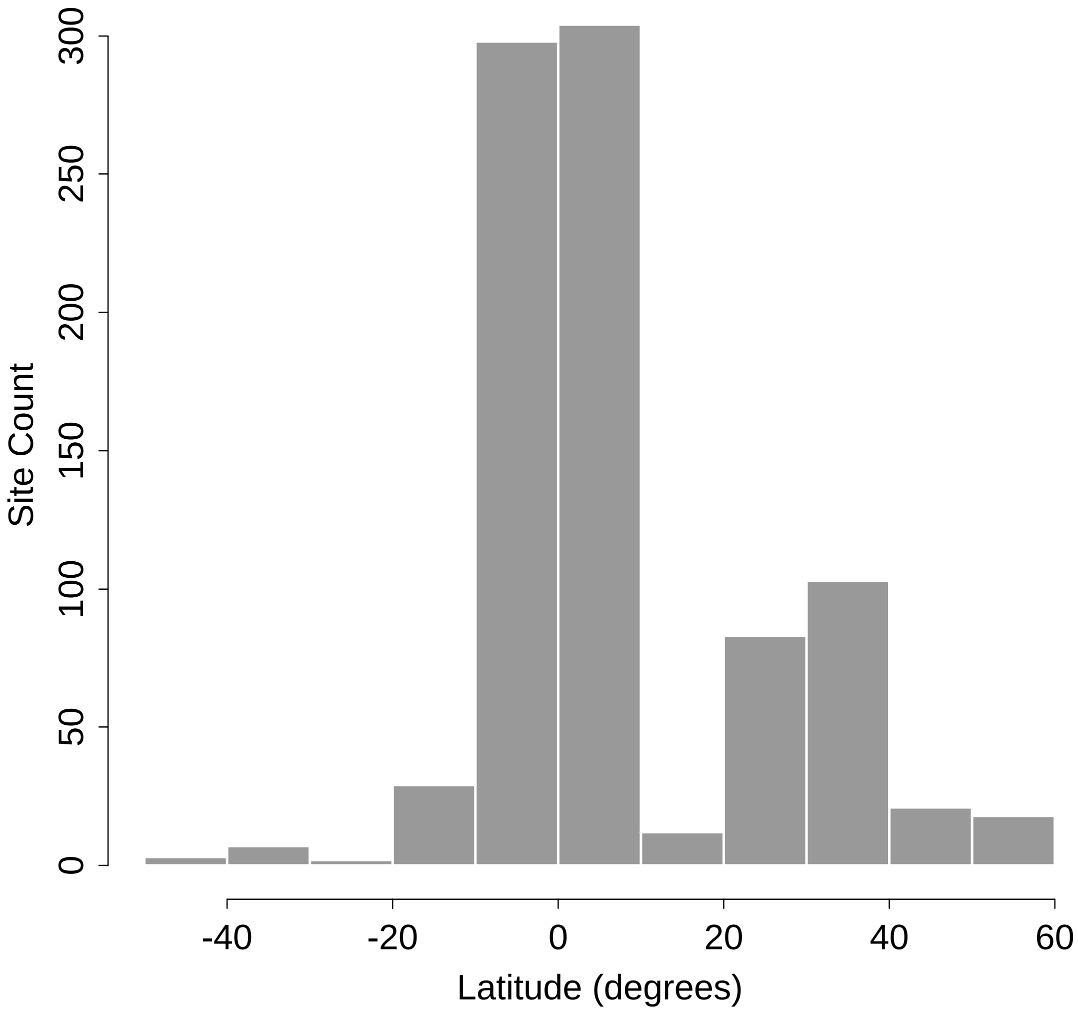
Figure C5. q624.ps

Figure C6. q625.ps

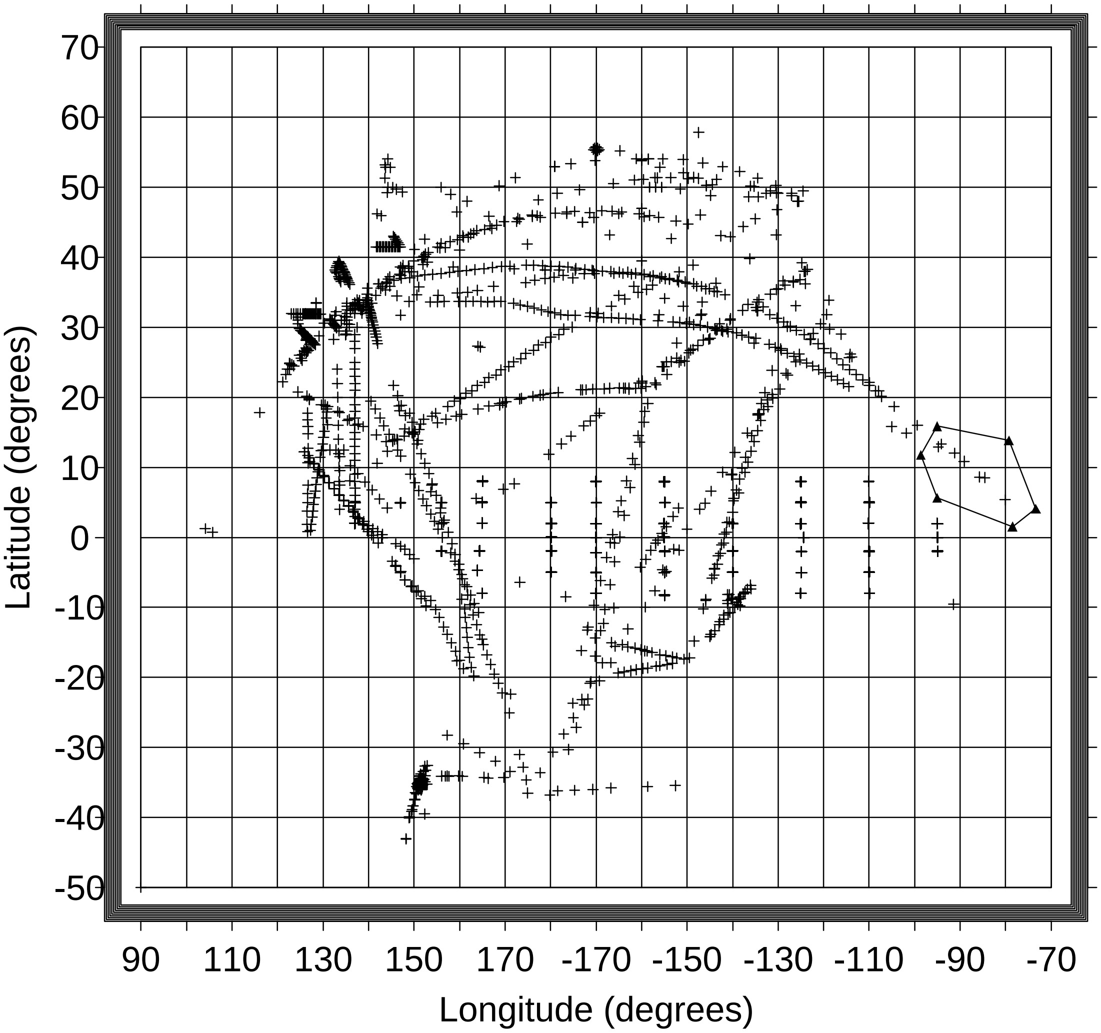
Q621: Profile Locations



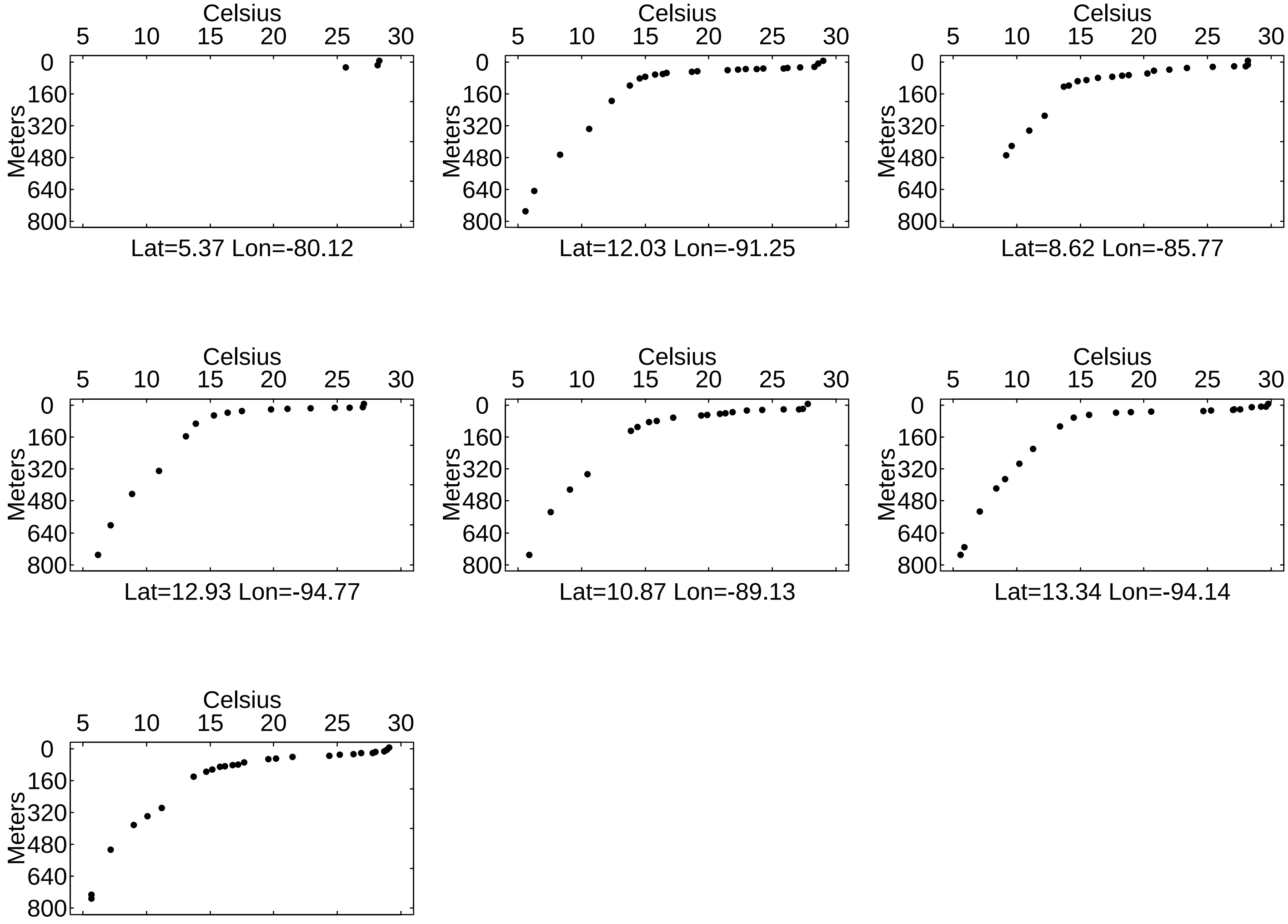
Q622: Latitude - Distribution for June



Q623a: Profile Locations



Q623b: Selected Profile Data



Lat=5.37 Lon=-80.12

Lat=12.03 Lon=-91.25

Lat=8.62 Lon=-85.77

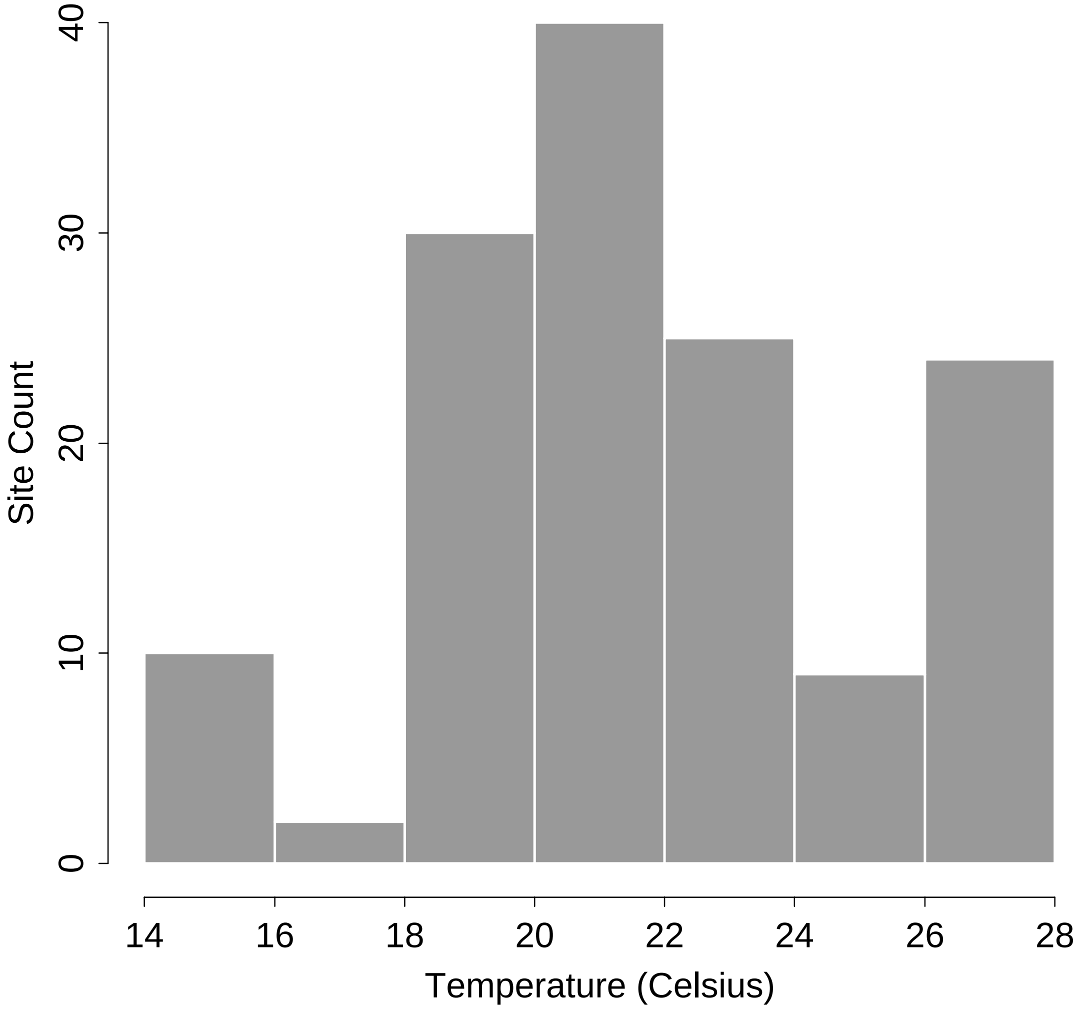
Lat=12.93 Lon=-94.77

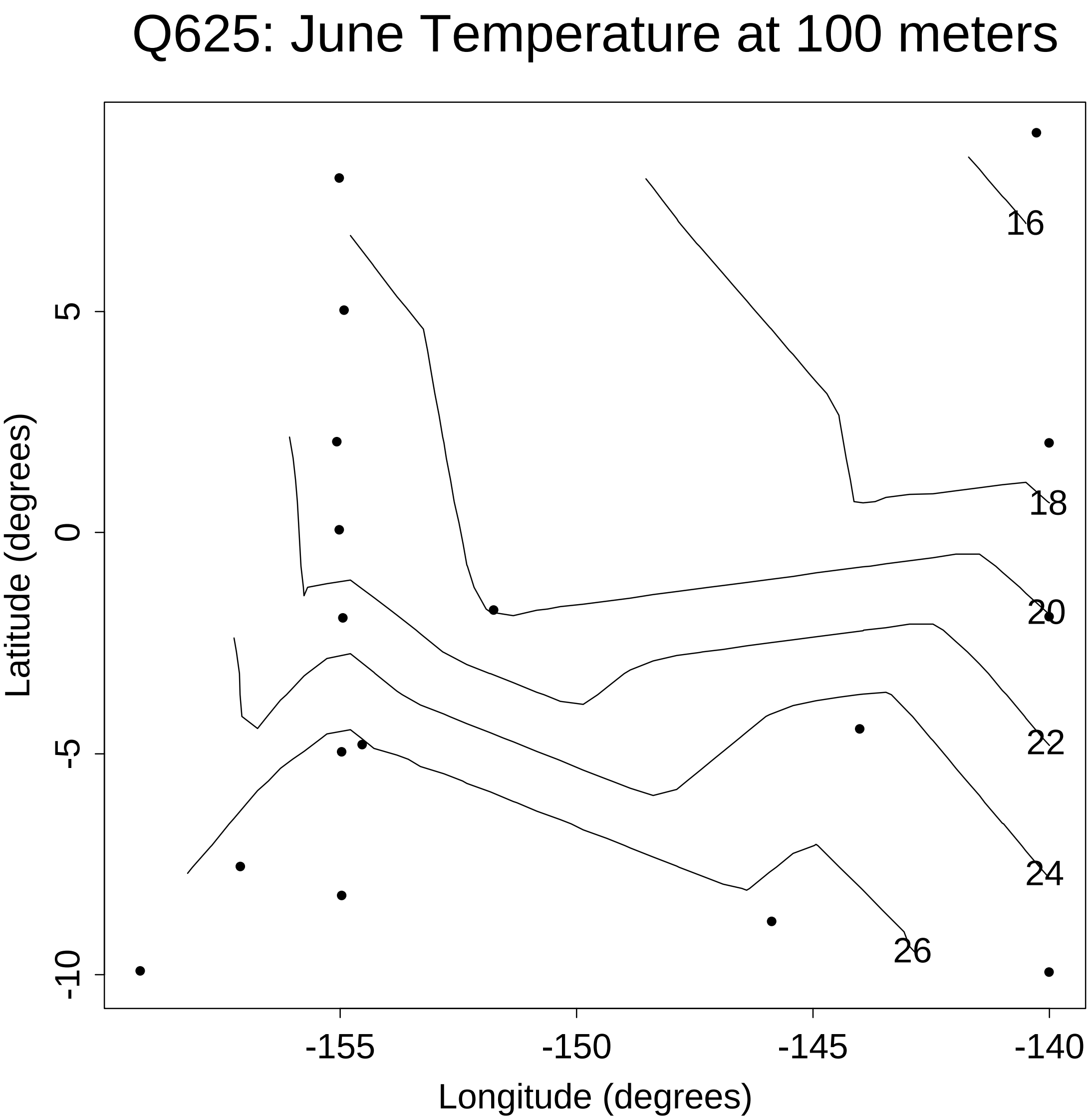
Lat=10.87 Lon=-89.13

Lat=13.34 Lon=-94.14

Lat=8.53 Lon=-84.61

Q624: June Temperature At 100 meters





Q625: June Temperature at 100 meters
Latitude (degrees)
Longitude (degrees)
5
0
-5
-10
-155
-150
-145
-140
16
18
20
22
24
26