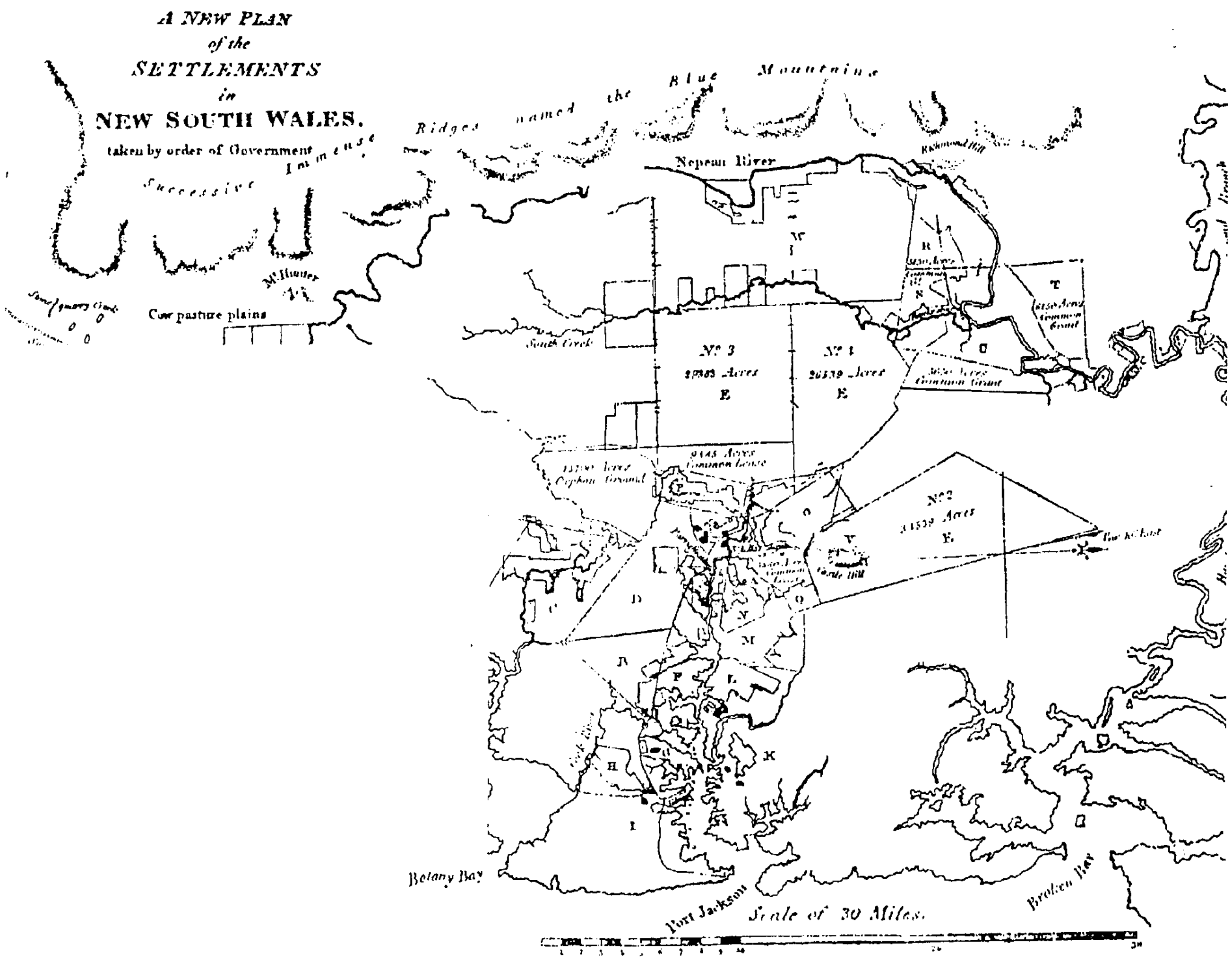
OBJECT–RELATIONAL DATABASE MANAGEMENT SYSTEMS (ORDBMS) FOR MANAGING MARINE SPATIAL DATA: ADCP DATA CASE STUDY

MOHAMAD ARIEF SYAFI'I



UNISURV S-58, 2000

Reports from

SCHOOL OF GEOMATIC ENGINEERING

THE UNIVERSITY OF NEW SOUTH WALES SYDNEY NSW 2052 AUSTRALIA

UNISURV REPORT S-58, 2000

OBJECT–RELATIONAL DATABASE MANAGEMENT SYSTEMS (ORDBMS) FOR MANAGING MARINE SPATIAL DATA: ADCP DATA CASE STUDY

MOHAMAD ARIEF SYAFI'I

Received: September 2000
Accepted: October 2000

SCHOOL OF GEOMATIC ENGINEERING
UNIVERSITY OF NEW SOUTH WALES
UNSW SYDNEY NSW 2052
AUSTRALIA

National Library of Australia

Card No. and ISBN 0 - 7334 - 1704 - 3

FOREWORD

The latter part of the 1990's has seen major database management systems vendors implementing spatial concepts and models into their software. These concepts were previously confined to the Geographic Information Systems area. However, object-oriented concepts and technology have made the inclusion of complex spatial objects into these systems possible.

In principle, these systems allow a rich set of queries to be undertaken, provided appropriate relationships can be defined and implemented in the database. We can expect to store features with coordinates in these systems and undertake queries that depend on coordinates and other spatial relationships, such as adjacency, intersection and containment. In GIS terms, these queries are probably still fairly simple. Nevertheless, the extensibility of these systems means that complex analysis functions can be implemented in the database. Most of the GIS vendors therefore have options to use the spatial capabilities of these new database systems.

The uptake of this type of technology will depend on the difficulty of implementing databases that can use the increased functionality, as well as depending on the improved query efficiency and expanded query capability offered. This project investigates these aspects of one particular marine application that uses spatial capabilities of the new database technology. It gives a useful description of the database design and implementation process for a multi-dimensional problem, including a comparison with using standard relational database design procedures to implement the same capabilities.

I believe this report is a useful introduction to the use of object-relational database technology for spatial applications. It describes the application, the new technology, the database design including data definitions, the Java and SQL implementation for loading data from the sensor into the database, as well as querying the database. Anyone interested in using this type of technology should find this report a useful description of the topic area, giving an overview of object-relational technology as well as the details of designing and implementing a spatial database in this context.

Ewan Masters

October 2000

ACKNOWLEDGMENTS

In the Name of Allah, the Most Gracious the Most Merciful. All Praises be to Allah, the Lord of the Universe, for His Help and Guidance to the Right Path throughout my life.

I wish to express my best gratitude to my supervisor, Dr. Ewan G. Masters, for his supervision during my study in the School of Geomatic Engineering, UNSW. His support, comment, invaluable suggestion and advice in the whole aspect of my study are gratefully acknowledged.

My gratitude is also expressed to A/Prof. Chris Rizos and A/Prof. A.W. Kearsley, for the encouragement and providing me some invaluable advice during my study. Many thanks are also expressed to all school office staff for their assistance especially in administrative matters. My special thank is expressed to Mr. Brian Donnelly and Mr. Christiaan Werner, the school system administrators for their assistance during my study.

My sincere gratitude also goes to the National Coordinating Agency for Surveys and Mapping (BAKOSURTANAL) of Indonesia for providing me an opportunity to study and to the Ministry of Research and Technology of Indonesia for supporting me with the scholarship.

I would like to thank to Australian Oceanographic Data Centre (AODC) staff especially Mr. Ben Searle (Head of AODC), Mr. Greg Reed (IT Manager of AODC) and Dr. Krystyna Jankowska (Data Manager of AODC) for their help, support, invaluable suggestion and discussion during my research at the AODC office. The AODC also provided me an opportunity to attend the Informix SQL and Formida Software training courses, which gave me the required basic knowledge on completing the research project.

I wish to express my sincere appreciation to the CSIRO, Division of Physical Oceanography, especially Mr. Terry Byrne for providing me the ADCP data set. My sincere appreciation is also expressed to Mr. Peter Carew of Navigate GIS Consultants. for the discussion and invaluable suggestion on completing my project.

Finally, I wish to express my deep gratitude to my beloved wife, Hanin and my lovely kids, Hanif and 'Ulya, for their support, encouragement, and patience when accompanying me during the study. I am also indebted to my parents who have always encouraged me to study.

Abstract

In the last two decades, the relational data model and Relational Database Management Systems (RDBMS) have been dominant in GIS and other spatial data management applications. However, this technology is not without disadvantages, especially when dealing with complex objects. The process of normalization in the relational model generally leads to the creation of relations that do not correspond to entities in the 'real world'. The fragmentation of 'real world' entity into many relations is inefficient, leading to many join operations being required during query processing. In the database context, the join operation is one of the most costly operations to perform.

The Object-Relational Database Management Systems (ORDBMS) technology has recently emerged in response to the increasing complexity of database application such as GIS. It extends the SQL of Relational DBMS to support object-oriented features (extensibility, encapsulation, inheritance). These capabilities enable users to store a complete spatial object such as point, line (arc) or polygon into a single column of a table. Furthermore, it enables users to store the complex attributes' data associated with a spatial object in the same table. These new approaches may lead to an effective and efficient spatial data management that can be applied in the marine area.

For a case study, a set of water current velocity profiles measured by Acoustic Doppler Current Profiler (ADCP) is used. This data can be considered as a complex object in the marine area and provides an example on how ORDBMS could be used to store and manage spatial objects with complex attributes. A database application has been created using Java application programs to provide some examples of database transactions and queries.

Table of Content

List of Acronyms

ADCP	Acoustic Doppler Current Profiler
ADMS	ADCP Data Management Systems
ADT	Abstract Data Types
ANSI	American National Standard Institutes
BLOB	Binary Large Objects
CAD	Computer Aided Design
CLOB	Character Large Objects
CSIRO	Commonwealth Scientific and Industrial Research Organisation
DBMS	Database Management Systems
DDL	Data Definition Language
DML	Data Manipulation Language
EER	Enhanced-Entity Relationship
ER	Entity Relationship
GIS	Geographic Information Systems
GPS	Global Positioning System
ISO	International Organisation for Standardisation
IUS	Informix Universal Server
JDBC	Java DataBase Connectivity
OLTP	On-Line Transaction Processing
OODBMS	Object-Oriented Database Management Systems
OODM	Object-Oriented Data Model
ORDBMS	Object-Relational Database Management Systems
ORDM	Object-Relational Data Model
RDBMS	Relational Database Management Systems
SQL	Structured Query Language
UDF	User-Defined Functions
UDR	User-Defined Routines
UDT	User-defined Data Types

Chapter I

Introduction

1.1. Background.

Geographic Information Systems (GIS) have been widely used in many applications in the last few years either in land or marine applications. The problems that can be solved by GIS have increased (according to) the improvement of GIS technology. In offshore applications there is much marine spatial information that can be analysed and visualised by GIS technology such as bathymetry, basic oceanographic data sets (salinity, temperature and current), marine geology and geophysics, sea level, waves, biodiversity, and so on. All of this information is important for supporting decision making in related marine activities. Thus, the marine spatial data management underlying the GIS also plays an important role on the decision-making processes.

There are numbers of database management systems (DBMSs) that can be used to manage marine spatial data. Based on its underlying data model, the database management systems can be categorised into *relational, object-oriented,* and the most recent database technology *object-relational database management systems.* All of these database management systems provide their own advantages and disadvantages depending on the applications.

For years, *Relational Database Management Systems (RDBMS)* have been used as a common database management system for storing and managing data in GIS. However, this traditional DBMS is not without disadvantages, especially when dealing with complex objects (namely, objects that require more than one structure to represent it) such as spatial data and its attributes. The process of normalization in relational model generally leads to the creation of relations that do not correspond to entities in the 'real world'. The fragmentation of 'real world' entities into many relations is inefficient, leading to many join operations during query processing. In a database context, a join operation is one of the most costly operations to perform. However, RDBMS can still be used to manage spatial data for particular purposes.

In response to the increasing complexity of database applications such as GIS, two new data models have emerged: *Object-Oriented Data Model (OODM)* and *Object-Relational Data Model (ORDM)*. The Object-Oriented Database Management System (OODBMS) is actually ideal for storing, managing and retrieving complex objects. The object-oriented data model, which the OODBMS is based on, enables users to model objects that correspond to entities in the 'real world'. However, for most GIS users and vendors, it is hard to migrate from RDBMS to pure OODBMS, because they should leave their previous investments on RDBMS and they have to establish new investments on OODBMS. They also have to convert all their existing data stored on RDBMS to OODBMS, in order to be able to use the existing data in the new database system.

The Object-Relational Database Management System (ORDBMS), which is based on object-relational data model, can be an alternative to the above situation. The ORDBMS provides the necessary data modeling, scalability, and robustness to store, manage, and retrieve highly interrelated, complex, and varied information. By ORDBMS, the GIS user can gain object-oriented benefits while storing their data in relational structures. The users may also keep the existing data in relational structure without converting them into the new structure. The ORDBMS enables users to store the data in both object-relational structure or in relational structures if they want. The ORDBMS also offers the possibility of accessing complex data via WWW because they allow fast access to huge spatial (complex) data.

1.2. Objectives.

The objective of this study is to develop a database application to store, manage and retrieve marine spatial information, in particular water current velocity profile measured by Acoustic Doppler Current Profiler (ADCP), in an Object-Relational Database Management System (ORDBMS). The application is developed using Java Programming language. The Informix Dynamic Server with Universal Data Option version 9.14, that support object-relational data model, has been used to store the ADCP data. For a case study, the CSIRO Division of Physical Oceanography in Hobart has provided a set of ADCP data within an area of 20° x 20° (30°-40° S and 150°-170° E). ADCP data was chosen as the main data in this study since it represents the complexity of marine spatial information.

1.3. Report Structures.

This report consists of six chapters and five appendices.

- **Chapter I Introduction.**

  Chapter one describes the background of this project, the objectives to be achieved and the structure of this report.

- **Chapter II Object-Relational Database Management System (ORDBMS).**

  Chapter two describes the most recent object-relational database management system technology. It begins by introducing the classification of the available database management systems and the history of database management systems generation. The main features of ORDBMS are then explained in detail. The overview of SQL3, as the new standard query language that support the ORDBMS, is described afterward. An application of ORDBMS for spatial data management is described at the end of this chapter.

- **Chapter III Acoustic Doppler Current Profiler (ADCP).**

  This project used a set of water current velocity measured by Acoustic Doppler Current Profiler (ADCP). Chapter three described ADCP data in detail. It starts by describing the Doppler effect concept and how the ADCP used this Doppler effect technique to measure water current velocity. The ADCP data is actually a three-dimensional current velocity vector and forms a profile of current velocities over a water column. This is described in sections 3.3 and 3.4. Having knowledge of ADCP data structures and viewing ADCP data as complex object is helpful to understand the reasons for storing and managing ADCP data into a database management system, which is described at the end of the chapter.

- **Chapter IV ADCP Data Management System (ADMS) Development.**

  Chapter four describes the development processes of ADCP Data Management System (ADMS). It is a database application in java programming language to store, manage and retrieve ADCP data as a complex object in Informix Dynamic Server with Universal Data Option version 9.14. The processes include planning, requirement and collection analysis, database design, application design, implementation, data conversion and loading, and testing. Some appendices are attached as a supplement to the materials covered in chapter four.

- **Chapter V Discussion.**

  Chapter five discusses how a complex object such as ADCP data can be stored, managed and retrieved in an object-relational database management system. The discussion focuses on the main features of ORDBMS, such as extensibility, support complex object and inheritance, using ADCP as a case study. Some comparison to its predecessor technology of relational DBMS, as well as some benefits of storing a complex object in ORDBMS are also discussed. Some examples from ADCP database are also provided to have a clear discussion on the topics.

- **Chapter VI Conclusion.**

  Chapter six summarises what has been done in the project and concludes with some approaches that can be applied in marine spatial data management.

Chapter II

Object-Relational Database Management Systems (ORDBMS)

2.1. Introduction.

For many years, *Relational Database Management Systems (RDBMS)* have been used as a common database management system for storing and managing data in many applications. However, traditional RDBMS is designed to handle large volume of transactions, against simple data such as character and numeric data and to build simple relationships between data. This simplicity makes RDBMS not ideal for storing, managing and retrieving complex data such as images, video, sound, or spatial data (data that can be registered to a location on earth surface). The term complex data can be thought as any data that requires more than one structure to represent it.

The *Object-Relational Database Management System (ORDBMS)* has emerged to overcome the difficulties with the Relational DBMS when dealing with complex data. The ORDBMS extend the capabilities of Relational DBMS to have object-oriented features, that are required to model complex objects, while storing the data as tables with rows and columns (relational structures). As with traditional relational DBMS, queries to the data stored in ORDBMS can also be performed using a *Structured Query Language* or *SQL*. A new SQL standard (commonly referred to as SQL3 or SQL99) has been developed as an extension to the 1992 ISO SQL standard (commonly referred to as SQL2 or SQL92) to support these object-oriented features. Thus, the ORDBMS are relational in nature because they support SQL but they are also object-oriented in nature because they support complex data.

2.2. DBMS Classifications.

Stonebraker (1996) has proposed a four quadrants view of DBMSs classifications as illustrated in figure 2.1. The horizontal axis shows data complexities that an application may require, which are divided into simple data and complex data. The vertical axis differentiates whether an application requires query capabilities, which are divided into two choices, "query" and "no query". Depending on its characteristic, an application fits into at least one of the four quadrants. However, many applications may fit more than one quadrant.

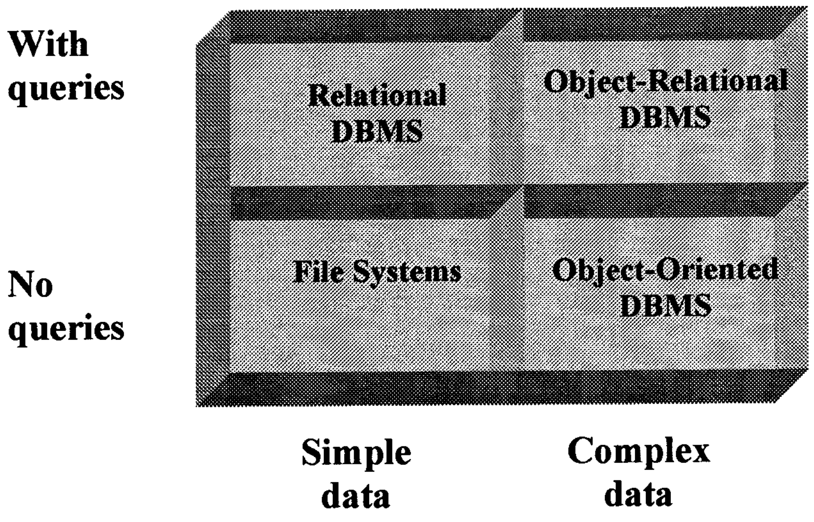


Figure 2.1. Classification of DBMSs.

In the lower left quadrant are those applications that process simple data and do not need querying of the data (using SQL). Most of the text editors, such as Microsoft Word, Word Perfect, Frame Maker or vi (a Unix text editor), fall in this quadrant. Such kinds of text editor deal with simple data such as characters and numeric and require no queries.

The upper left quadrant is those applications that process simple data and also have requirements for complex querying. Many traditional business applications fall into this quadrant and a Relational DBMS may be the most appropriate database management system. There are hundreds of relational DBMS available in the market for both mainframe and microcomputer environments. Oracle, Informix, MS-Access, FoxPro are some examples of relational DBMSs available in the market. Relational DBMS have a standard language for querying the data, which is called *Structured Query Language* or *SQL*.

The lower right quadrant occupies applications that process complex data and has no significant requirement for querying the data. The term complex data are often referred to as "objects" or "rich data types" because they represent complex internal structures and usually require special functions, or methods, to create and manipulate the data. For this type of application, for example Computer-Aided Design (CAD) packages, an Object-Oriented DBMS may be an appropriate choice of database management system.

Finally, in the top right quadrant are those applications that process complex data and have complex querying requirements. A query of an image by content is an example of this condition. As an illustration, a client would like to find images within a database, which contain a specific object. The location of the object might be a good key to search all images that match to the queries. Because it has complex data and query requirements, this type of application belongs on the upper right quadrant of the matrix and Object-Relational DBMS is an appropriate database management system to handle them.

2.3. Database Management System Generations.

Database management systems have been developed since 1960 to overcome a problem of managing vast amount of data in many applications. The database management systems development can be divided into three generations. The first generation of database management system was developed in the period of 1960 to 1970s as a file-based system to handle large amount of data. The second generation of database management system was initiated by the development of relational DBMS in period of 1970s. The third generation database management systems have emerged in response to the weakness of relational DBMS on handling complex data. The object-oriented DBMS and object-relational DBMS represent the third generation database management systems.

2.3.1. First-Generation Database Management Systems.

The development of database management systems was started in 1960s when Apollo moon-landing project was initiated. The database was developed as **a file-based system** to handle and manage the vast amount of information that the project would require. The prime contractor of the project, North American Aviation (NAA - now Rockwell

International), developed a software known as GUAM (Generalised Update Access Method). GUAM was based on the concept that smaller components come together as parts of larger components, and so on, until the final product is assembled. This Structure, which conforms to an upside-down tree, is also known as a **hierarchical structure**. In mid 1960s, IBM joined NAA to develop GUAM into what is known as *IMS (Information Management Systems).*

Another significant development of database system in mid 1960s was initiated by the development of *IDS (Integrated Data Store)* from General Electric. IDS used a different structure than IMS, which is known as **network structure**. This structure was developed partly to address the need to represent more complex data relationships than could be modelled with hierarchical structures, and partly to impose a database standard. To help establish such standard COnference on DAta SYstems Languages (CODASYL) formed a List Processing Task Force in 1965, subsequently renamed the Data Base Task Group (DBTG) in 1967. The term of reference for the DBTG was to define standard specifications for an environment that would allow database creation and manipulation. The DBTG proposal identified three components:

- The *network schema* - the logical organisation of the entire database as seen by the Database Administrator (DBA), which includes a definition of the database name, the type of each record, and the components of each record type.
- The *subschema* - the part of the database as seen by the user or application program.
- A *data management language* to define the data characteristics and the data structure, and to manipulate the data.

For standardisation, the DBTG specified three distinct languages:

- A *schema Data Definition Language (DDL),* which enable the DBA to define the schema.
- A *subschema DDL,* which allows the application programs to define the parts of the database they require.
- A *Data Manipulation Language (DML),* to manipulate data.

These two mainstream database approaches, **hierarchical structure (IMS)** and **network structure (CODASYL),** represented the **first-generation of DBMSs**. However, these two models have some fundamental disadvantages:

- Complex programs have to be written to answer even simple queries based on navigational record-oriented access.
- There is minimal data independence.
- There is no widely accepted theoretical foundation.

2.3.2. Second-Generation Database Management Systems.

The second-generation database management system was initiated by the development of *relational database model, which* was introduced by E.F. Codd of the IBM Research Laboratory in 1970. The relational model approach was addressed to overcome the disadvantages of the former approaches of hierarchical and network models. In the relational model, data is perceived as tables with rows and columns. Many experimental relational DBMSs were implemented thereafter, with the first commercial products appearing in late 1970s and early 1980s. Of particular note is the System R project at IBM's San Jose Research Laboratory in California. This project was designed to prove the practicality of the relational model by providing an implementation of its data structures and operations, and led to two major developments:

- The development of a *structured query language* called *SQL*, which has became the standard language for relational DBMSs.
- The production of various commercial relational DBMS products during the 1980s; for example, DB2 and SQL/DS from IBM, ORACLE from Oracle Corporation.

Nowadays, there are many relational DBMSs available for both mainframe and microcomputer environments. Some other examples for multi-user relational DBMSs are CA-OpenIngres from Computer Associates, Informix from Informix Software inc. Examples of microcomputer-based relational DBMSs are Access and FoxPro from Microsoft, Paradox and Visual dBase from Borland.

Relational DBMSs has been used for many years as a dominant DBMS especially in the area of traditional business applications, such as order processing, inventory control, banking and airline reservations. A Relational DBMS has its strength in its simplicity, its suitability for Online Transaction Processing (OLTP), and its support for data independence. However, relational DBMSs have proven inadequate for advanced database applications that require object-oriented features such as user-defined data type, encapsulation, inheritance, polymorphism, dynamic binding of methods, complex objects including non-first normal form objects, and object identity. These features are extensively used by many advanced-database applications include: Computer-Aided Design (CAD), Computer-Aided Manufacturing (CAM), Computer-Aided Software Engineering (CASE), Office Information System (OIS) and Multimedia Systems, Digital Publishing or Geographic Information Systems (GIS).

2.3.3. Third-Generation Database Management Systems.

In response to the increasing complexity of database applications that could not be adequately handled by relational DBMS, two new data models have emerged: the Object-Oriented Data Model (OODM) and Object-Relational Data Model (ORDM). As its name, these data models have the main characteristics of supporting object-oriented features such as user-defined data types and user-defined function, complex data, and inheritance. These two data models represent the **third generation of database management** systems.

In this project, the later data model of object-relational data model and object-relational DBMS are used to manage the water current velocity measured by Acoustic Doppler Current Profiler (ADCP). This data can be considered as complex data in the marine area. Thus, a DBMS with object-oriented features would appear to be appropriate to be used. The next section explains some features of ORDBMS.

2.4. ORDBMS Features.

Object-Relational Database Management Systems (ORDBMS) combine the relational and object-oriented DBMS capabilities. ORDBMS has object-oriented features such as extensibility (support user defined data types and Routines), support complex data, and inheritance, while storing the data in relational structures. The term relational structure means that the data is perceived as tables with rows and columns. The object-oriented capabilities of an ORDBMS can be found in SQL mode. A new SQL standard, referred to as SQL3 or SQL99 has been developed to support this object-oriented features of ORDBMS. Although the object-relational structure extends the capabilities of the relational structure, the users can still implement their data model as a traditional relational database if they desire.

The ORDBMS stores objects by automatically decomposing them into table entries. What the user sees looks like an object, but underneath, it uses the features of relational model (storing data in rows and columns). This allows for objects on the front end, where they are most visible to the users, while keeping the features of a relational database on the back end.

As with any other object-oriented programming languages, ORDBMS has the common features or capabilities of such kind of OO programming languages. Some of these common capabilities, which are found in SQL mode, will be discussed briefly in the following sections.

2.4.1. Extensibility.

In traditional relational database, users are limited to use only the built-in data types that the database server defines and supports. Consequently, users can store and access only those types of data that the built-in data type support. The built-in data type is atomic data type; that is, it cannot be broken into smaller pieces. Some examples of built-in data types are character (either fixed or variable length), numeric, date, decimal, float, integer, money, real, etc.

The extensibility feature of ORDBMS extend the capabilities of the database server that enables users to define *new data types (user-defined data types),* as well as the access methods and functions to support them, and *user-defined routines*. This allows a user to store and manage complex data such as spatial data, time series, image, audio, video, large text documents and so forth. A data type is a descriptor that is assigned to a variable or column and that indicates the type of data that the variable or column can hold. User-defined data types are treated identically to built-in data types. Values of the user-defined data types may be stored, examined, using queries or function calls, passed as arguments to database functions, and indexed in the same way as the built-in data types.

2.4.1.1. User-defined data types (UDT).

There are several different ways to define a new data type. Users can create a new data type by redefining some of the behaviour of the existing data type or they can create a new data type by defining the behaviour of that new data type. In the first case, some additional behaviour can be defined to the existing data type such as additional operations that are valid to the new data type, new operator class for indexing or additional casting function to convert between data types. Since it is created from existing data types, its internal structure (which provides the format of the data type) has already known by the database server and the database server will automatically define implicit cast between the new data type and its source types. In the second case, the users are allowed to define new data types and define the behaviour of these new data types to the database server. As the first case, the behaviours those are defined for the new data types include their internal structure, operation, operator class and casting function.

There are several categories of user-defined data types: *simple distinct types, abstract types with complex structure,* and *named-row types*. The named row data types are also referred to as complex data types and will be explained in the next section. Except the abstract data type, these user-defined data types are created from the existing data types (either built-in or UDT).

A *distinct data type* is created by renaming an existing data type to differentiate it from the base type. For example we may create a distinct type of `latitude` from an existing data type of `float` with a length of 12. The following SQL statement illustrate the example:

```
CREATE DISTINCT TYPE latitude AS float(12);
```

The DBMS treats any column defined as a `latitude` type as a different data type than a column defined as a generic `float` type, even though under the covers they are both float types with the same physical representation.

An *abstract data type (ADT)* is a custom-coded type that represents arbitrarily complex internal structures (Informix call ADT as *'opaque'* data type). Since it is custom-coded, the DBMS does not know the structure of this ADT. Thus, the user who created this ADT should determine the internal structure (usually in C language) and provides a set of external attributes and functions through which to access the data. This is implementing a form of *encapsulation*, namely the application does not have to understand the internal structure of the type in order to create and manipulate the data. For example, we might want to create a circle as an ADT of `circle` data type. This data type includes `(x,y)` coordinate, to represent the centre of the circle and a radius value. The following shows the internal data structure of the circle data type written in C language:

```
typedef struct
{
  double   x;
  double y;
} point_t;

typedef struct
{
  point_t  center;
  double radius;
} circle_t;
```

The internal representation for circle requires three double values: two double values for x and y that form the centre of `point_t` data type, and one for the `radius`. Because each `double` value is 8 bytes, the total internal length for the `circle_t` structure is 24 bytes. The following SQL statement register this new data type to the DBMS:

```
CREATE OPAQUE TYPE circle_t (INTERNALLENGTH = 24);
```

The user should also register the support functions in order to be able to access and manipulate the data as well as casting to or from other data types.

In object-relational system, a data type is defined as a stored representation of particular kind of information together with the appropriate operators and functions for the information. The flexibility of object-relational systems makes user-defined data types a powerful means to model complex database applications. But by themselves, new data types are not very useful unless users can perform operations on instances of the type. This issue will be discussed in the next section.

2.4.1.2. User-Defined Functions (UDF).

In a traditional RDBMS, there are only built-in operators provided by the database server such as arithmetic and comparison operators. In ORDBMS, the users are allowed to use the built-in operators as well as creating type-specific operations that are valid to the data type they have created (user-defined). This user-defined function can be written in either SQL or a general-purpose third-generation programming language such as C/C++ or Java, and then register them to the system.

User-defined functions define the methods by which applications can create, manipulate, and access the data stored in the defined data types. User-defined functions also support the notion of *overloading* or *polymorphism*, which refers to the concept of using the same name of functions but different inputs or arguments. This function will be executed based on the data type of the arguments.

2.4.2. Support Complex Data.

The term complex data means any data that requires more than one structure to represent it. The data structure of a complex object may be known by the database server or the users may define and register it to the database. Such an object is referred to as *structured complex object*. Sometimes, the database system does not know the data structure of complex data and only the application program knows the data structure. This situation is referred to as an *unstructured complex object*. In database context, unstructured complex objects are sometimes known as *Binary Large Objects (BLOB)*.

2.4.2.1. Structured Complex Data.

A *structured complex* data type is built from a combination of one or more existing data types, either built-in or other complex data types, with an SQL type constructor. An important characteristic of a structured complex data type is that we can easily access each of its component data types using an SQL statement. There are two kinds of structured complex data type: *collection types* and *row types*, as describe in figure 2.2.

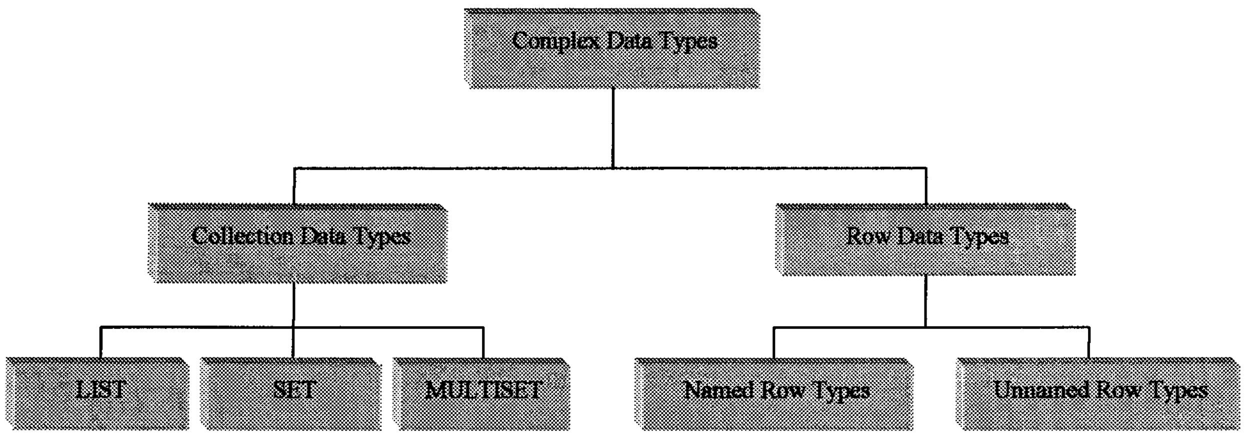


Figure 2.2. Complex Data Types.

2.4.2.1.1. Collection data types.

A collection data type is a group of elements (array) of the same data type, which can be atomic or complex data type. This data type allows multiple values to be stored in a single column of a table. The requirements for elements with ordered position and uniqueness among the elements determines whether the collection is a *LIST, SET or MULTISET.* A *LIST* is a group of ordered elements, each of which need not be unique. A *SET* is a group of elements, each of which is unique and the order of the elements is ignored. A *MULTISET* is a group of elements, each of which need not be unique and the order of the elements is ignored.

A group of sequence points that form a polygon is an example of LIST data type. Within the LIST, each point is unique and the order of the points is a necessity to form a polygon. A scattered set of sampling locations may be considered as SET data type. In this situation, the order of the sampling locations is not important.

2.4.2.1.2. Row data type.

Row data type is a group of related data fields, of any data type, that form a template for a record. The assignment of a name to the row type determines whether the row type is a *named row type* or an *unnamed row type*. A *named row type* is a group of data fields that are defined under a single name. This name is unique within the database. A *field* refers to a component of a row type and should not be confused with a column, which is associated with tables only. The fields of a named row type are analogous to the fields of a C-language structure or members of a class in object-oriented programming. For example, a geographic position is a complex type, since it can be subdivided into horizontal and vertical (height) positions. It is acceptable for the components of a complex type to be complex types themselves and form a hierarchy; for example, the horizontal position of a geographic position can be further subdivided into two atomic data: latitude and longitude as shown in figure 2.3.

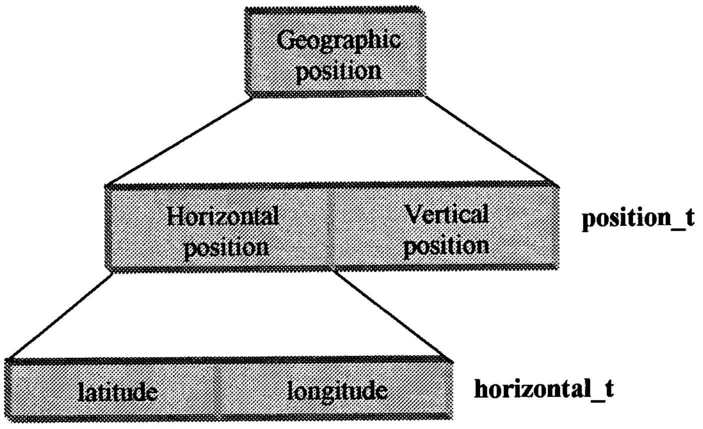


Figure 2.3. An example of complex type

An unnamed row type is a group of related data fields that is not assigned any name. Unlike named row types that are identified by their name, unnamed row types are identified by their structure. To construct a named row type, indicate the name of the row type along with the name and data type of the constituent members. An unnamed row type can be created by the same way but without assigning the name of the type. Figure 2.4 shows how to create a named row type:

```
Create row type horizontal_t (
       Latitude  float,
       longitude float);
```

```
Create row type position_t (
        Horizontal horizontal_t,
        Height     float);
```

Figure 2.4. Example of creating a named row typed.

A named row type can also be assigned to a table to create a typed table. Only a typed table may have inheritance properties.

2.4.2.2. Unstructured Complex Data.

Unstructured complex data is also referred to as Binary Large Object (BLOB). This includes maps, image, audio, video, large text documents and so forth. The database server does not know the structure of this data type. Thus, using an application program is the only way to access and manipulate this data type. The BLOB data types are managed by the database in separate space optimised specifically for large objects. The column of a BLOB type only stores the address of the BLOB data in the storage memory. The BLOB data type provides several performance enhancements such as random access, enabling the database to perform physical I/O on the requested portion of the data. Working with BLOB data type in an application program is very similar to that of a file systems, allowing existing file-based applications to more easily use BLOB data types.

2.4.2.3. Using Complex Data Types.

In ORDBMS, a complex data type can be used in the same way as built-in or abstract data types. For example, we can use complex data type as:

- column types.

- routine argument types and return types.
- field types in other complex types.

2.4.3. Inheritance.

Inheritance is the process that allows a type or a table to acquire the properties of another type or table. The type or table that inherits the properties is called *subtype* or *subtable*. The type or table whose properties are inherited is called *supertype* or *supertable*. The subtype or subtable inherits all properties that are defined on the supertype or supertable such as structure, behaviours (routines, aggregates, operators, constraint definitions, referential integrity, access methods, storage options, triggers, etc.), and indexes.

2.4.3.1. Type Inheritance.

Type inheritance is only applied to named row data types. This type inheritance enables us to group several named row types into a *type hierarchy* in which each subtype inherits the representation (data fields) and the behaviours (routines, aggregates, and operators) of the super type under which it is defined. Suppose we construct a data type with two field members called `person_t`:

```
Create row type person_t (
          Name       varchar(30),
          Address    address_t);
```

Figure 2.5. Creating a named row type.

We can now construct two additional data types, employee_t and student_t which are sub types of supertype person_t:

```
Create row type employee_t (
        Salary     int,
        Startdate  date)
Under person_t;

Create row type student_t (
        course     varchar(30),
        Gpa        float)
Under person_t;
```

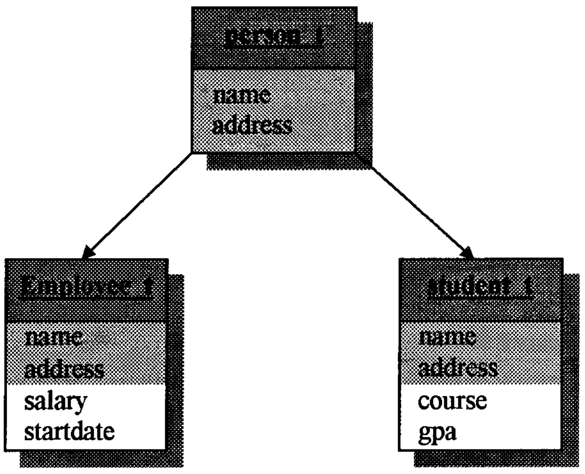


Figure 2.6 Inheritance hierarchy

The subtypes employee_t and student_t inherit all of the attributes and properties from its supertype person_t. The type employee_t inherits name and address from person_t and then specifies to additional attributes salary and startdate. Similarly, student_t inherits the same attributes from person_t and adds two additional attributes of its own (course and gpa). If the supertype person_t has another property such as routines, aggregates, constraint definitions etc, they will be inherited by the subtypes too. Figure 2.6 shows the schema of the above inheritance hierarchy.

2.4.3.2. Table Inheritance.

As mentioned above, the data inheritance only applies to data types. If a table is constructed that is not of a named type, then this table will be of an anonymous type and cannot utilise inheritance. Therefore, we need to construct types and assign them to tables rather than merely creating tables. In the former case, we can leverage inheritance while in the latter case, we cannot. The following statement constructs the typed tables of person, emp, and student:

```
Create table person of type person_t;

Create table emp of type employee_t
Under person;

Create table student of type student_t
Under person;
```

Similarly to type inheritance, the subtable of emp and student inherit all attributes possessed by person table and thus create a table hierarchy, as shown in figure 2.6. The scope of SQL statement in table hierarchy is the table and all tables underneath it or specifically the subtables. A query on a table may retrieve data from its subtables but never from tables, which are higher up in the hierarchy (supertables)

2.4.3.3. Functions Inheritance.

A function is a task that the database server performs on one or more values. There are built-in functions such as cos(), abs() etc., and user-defined functions. Unlike the built-in functions which can be used anywhere by the system, the user-defined function can only be used by the type that the function is attached to and its underneath type.

The inheritance behaviour of a user-defined function is determined by its arguments. As in type inheritance, the functions attached to any node (type) of inheritance hierarchy will be inherited to its underneath types. This means that if the ORDBMS is asked to evaluate a function and there is no function with the correct name and arguments, then the system searches the type hierarchy for a supertype on which the function is defined with the proper arguments. If one is found then the system uses this function.

It is permissible to have many functions with the same name but have different input arguments. For example a function called `sum(...)` applied to integer data types would perform conventional addition, while `sum(...)` applied to spatial data types would perform vector addition. This is called as *function overloading* or *polymorphism.* This polymorphism functionality is very convenient for users because potentially there can be one function for each data type. By this functionality, users may make one function for different data types as its arguments.

2.5. Overview of SQL3.

Structured Query Language or *SQL* is a standard language that is used to direct all operations on the relational database. It composed of statements, each of which begins with one or two keywords that specify a function such as SELECT, CREATE, UPDATE, DELETE etc.

2.5.1. Brief history of SQL.

SQL was invented at IBM in 1970s along with the introduction of relational database structure by E.F. Codd. Later, other vendors began to provide similar products for non-IBM computers. Each SQL implementation was slightly different from the IBM version and from other vendors. In 1986, a committee called X3H2, sponsored by American National Standards Institutes (ANSI), issued the SQL1 standard which defines a core set of SQL features and the syntax statements such as SELECT. Some database vendors had implemented some extension to this SQL standard to take an advantage of local hardware and software features. Another SQL standardisation was published in 1987 by International Organisation for Standardisation (ISO). In 1989, ISO published an addendum that defined an 'Integrity Enhancement Feature'.

The first major revision to the ISO standard was occurred in 1992, which is referred to as SQL2 or SQL92. Although some features have been defined in the standard for the first time, much of this has already been implemented, in part or in similar form, in one or more of the many SQL implementations. Features that are added to the standard by the vendors are called extensions.

The requirements to support object-oriented features in relational database have created an effort to add this functionality to SQL92 standard. In 1999, ANSI (X3H2) and ISO (ISO/IEC JTC1/SC21/WG3) SQL standardisation committees issued a new standard of SQL to support object-oriented data management, by adding features to SQL 92 standard. This new SQL standard is referred to as SQL3 or SQL99.

2.5.2. SQL3 features.

The SQL3 standard is fully compliant to SQL92 standard with some additional features to support object-oriented requirement of database application. The parts of SQL3 that provides the primary basis for supporting object-oriented data modelling are:

- *User-defined data types (UDT)*, either abstract data types (ADT), distinct types or named row types.
- *User-defined functions* and *procedures* (all together referred to as user-defined routines, UDR).
- Type constructors for *unnamed row types* and *reference types.*
- Type constructors for *collection data types* (LIST, SET and MULTISET).
- Support for large objects either *Binary Large Objects (BLOB)* or *Character Large Objects (CLOB).*

In the SQL92 standard, the users are restricted to only use the built-in data types that the database server provided such as CHARACTER, CHARACTER VARYING, BIT, BIT VARYING, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION, DATE, TIME, TIMESTAMP, and INTERVAL.

The new SQL3 data types give a relational database more flexibility in what can be used as a type for a table column. For example, a column may now be used to store the new type BLOB, which can store very large amount of data as raw bytes. A column may also be of type CLOB, which is capable of storing very large amount of data in character format. In addition to this BLOB and CLOB data types, the users are able to randomly access as well as update only on the requested portion of the data. The new collection data types make it possible to use multi-values or an array as a column value. Even the new user-defined data types, named row types and distinct types, can now be stored as column values.

2.6. Spatial Data Management using ORDBMS.

2.6.1. Spatial Data Overview.

The term spatial data means any data that can be registered to a location on the earth surface. In other word, it has geographic location. There are several main characteristics that spatial objects should have:

- *Location:* The objects exist at some known point on the surface of the earth.
- *Form:* The objects have a geometric representation. There are three basic spatial objects: point (0D), line (1D) and polygon (2D).

- *Attributes:* properties that describe the spatial objects.
- *Spatial relationship*: relationship between objects.

2.6.2. The use of ORDBMS for Managing Spatial Data.

In the last two decades, the relational data model and Relational Database Management Systems (RDBMS) have been commonly used in GIS and any other spatial data management applications. However, the RDBMS has some weakness for managing spatial data as a complex object. This was because the relational data model was designed to support very limited data types ie. numeric, character and date, that make it not ideal for storing, managing and retrieving complex objects. The first Normal Form (1NF) rule of normalisation process in relational model requires any complex object to be broken down into atomic values. This may lead to the creation of relations, which is inefficient since it requires many join operations during query processing. In addition, another applications such as CAD and image processing systems have developed their own data formats, requiring anyone who wanted to integrate GIS, CAD and raster image data to use multiple data translators.

The object-relational database management system (ORDBMS) provides facilities to create user-defined data types (UDT) and user-defined functions (UDF) that enable users to store and manage complex spatial data, as an object, along with data from other sources such as CAD and raster image in the same database. Creating UDT and UDF for the spatial object introduce some new approaches of spatial data management that provides two key benefits:

- The spatial data types can be treated as any other built-in data types such as numeric, character and date, enabling us to create UDF that work with spatial data as an argument parameters in performing spatial-based analysis.
- The spatial database application can be developed as 'thin' as possible by reducing application server tasks, since the database server may perform most of the required tasks rather than implementing them in the application server.

2.6.3. Storing spatial data as complete objects.

There are three basic spatial objects to represent a geometric feature of an entity: *point, line* and *polygon*. Any geographic features can be derived from these three basic objects. The ORDBMS enables us to store the spatial objects in a single column, without decomposing it into atomic values, as should be done in RDBMS. Furthermore, we may perform a spatial analysis based on this spatial object data type. For example, we may want to know the hospitals located at a radius of 10 km from a particular location (35S, 151E). The following SQL statement provides an example of the situation:

```
SELECT hosp_name, hosp_address
FROM hospital
WHERE  Within(hosp_position, '(-35,151)'::point, 10000);
```

The above SQL statement will return the hospital names (hosp_name) and its address (hosp_address) from hospital table that are located within radius of 10 km from the specified location (35S, 151E). The within key word in the WHERE condition statement is a user-defined function (UDF) to perform spatial analysis of proximity against a spatial object of point data type. The hosp_position is also a spatial object of `point` data type represent the geographical position (lat, long) of the hospitals. As can be seen, the ORDBMS provides an efficient way of performing spatial analysis in the database server with a single SQL statement, instead of performing the analysis in the application.

2.7. Summary.

The object-oriented features of ORDBMS provide a better way of modelling complex objects than RDBMS. It allows storing any complex object as a complete object into a single column of relation by utilizing user-defined data types (UDT) and user-defined function (UDF) capabilities. Moreover, ORDBMS provides a new approach on spatial data management that enables users to store the spatial data and the associated attributes (either simple or complex) into the same relation. This can be a benefit since users can create a simple SQL statement to manage complex spatial data as well as performing simple steps of spatial analysis. To examine the capabilities of ORDBMS in spatial data management, a set of water current velocity profiles measured by Acoustic Doppler Current Profiler (ADCP) is used in this project. The ADCP data may represent the complexity of spatial data in marine area and ORDBMS would be an appropriate database management system to manage them.

Chapter III

Acoustic Doppler Current Profiler (ADCP)

The Acoustic Doppler Current Profiler (ADCP) measures the current velocity and its direction by transmitting high frequency acoustic (sound) waves, and then determining the *Doppler frequency shift* of the return signal scattered from assemblages of "drifters" in the water column. The sound wave is transmitted from a four-beam transducer in different directions (either vessel hull-mounted or fixed positioned). Three beams are required to determine the three-dimensional current velocity (two horizontal and one vertical velocity components). The fourth beam is required to examine the quality of the measured velocity.

This chapter will discuss about how the ADCP works and estimates the three-dimensional water current velocities vector, the ADCP data as complex object and some reasons why we need to store and manage ADCP data into a database management systems.

3.1. Overview of Doppler Effect.

The *Doppler Effect* is a change in the observed sound pitch resulted from relative motions between the sound source and receiver. If a sound source A moves relatively to a receiver B, the sound transmitted from the sound source A will have different frequency when it is observed at position of receiver B. A good example of this situation is the sound made by a train as it passes by. The whistle has a higher pitch as the train (object A) approaches the observer (object B), and a lower pitch as it goes away from the observer. This change of pitch is directly proportional to how fast the train is moving. Therefore if we measure the pitch and how much it changes, then we can calculate the speed of the train.

A sound wave is produced by the change of pressure in air, water or solids. When there is a change of pressure at any point, this will be propagated to all directions in the media. In case of water the propagation velocity depends on its frequency (f) and its wavelength (λ). Mathematically, this relationship can be expressed by the following equations:

$$C = f\,\lambda \tag{3.1}$$

To provide more illustration of Doppler effect, imagine we are next to some water and watching waves pass by. While standing still, we see eight waves pass in front of us in a given interval. Now, if we start walking toward the waves, more than eight waves will pass by in the same interval. Thus, the wave frequency appears to be higher. If we walk in other direction, fewer than eight waves pass by in the same interval, and frequency appears to be lower. This is the phenomenon of *Doppler effect*, and the wave frequency difference appears, when we are standing still and when we are walking, is called by *Doppler shift*.

The same situation happens in sound propagation in water. The receiver will hear the sound with a higher frequency when moving towards the sound source and a lower frequency when moving away from the sound source. In this situation, the Doppler shift can be expressed by the following equations:

$$F_D = F_S\,(V / C) \qquad (3.2)$$

Where:

F_D = the Doppler frequency shift.

F_S = the frequency of the sound when everything is still.

V = the relative velocity between the sound source and the sound receiver (m/sec).

C = the speed of sound in water (m/sec).

As can be seen from equation (3.2), the Doppler shift increases if the sound source/receiver moving towards one another (relative velocity increases), and decreases if the sound source/receiver moving away one another (relative velocity decreases). The Doppler shift also increases if the frequency of the sound increases and decreases if the frequency of the sound decreases.

3.2. How ADCP Measures the Water Current Velocity.

ADCP use the Doppler effect to measure the water current velocities. By transmitting the sound wave at a fixed frequency and listening to echoes returning from the sound scatterers in the water, the three-dimensional water current velocities can be estimated (Figure 3.1). These sound scatterers can be found anywhere in the ocean as small particles or plankton. They float in the water and on average they move at the same velocity as the water movement.

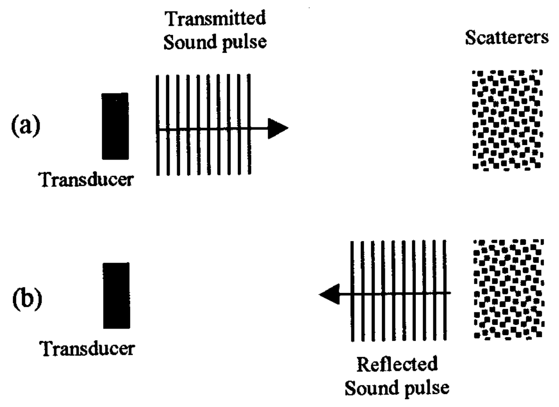


Figure 3.1. The ADCP transmit the sound wave (a). The scatterers reflect the sound wave back to ADCP (b).

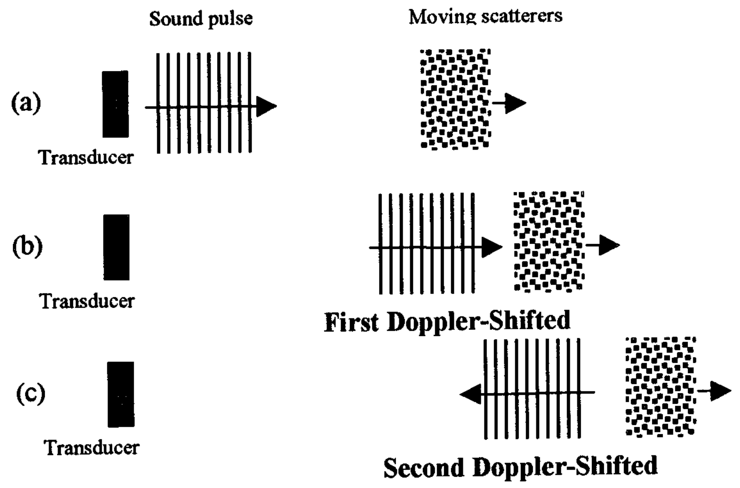


Figure 3.2. The backscattered sound heard by the ADCP is Doppler-shifted twice.

When sound scatterers move toward the ADCP, the transmitted sound heard by the scatterers is Doppler-shifted to a higher frequency, or vice versa. The amount of this shift is proportional to the relative velocity between the ADCP and scatterers (Figure 3.2). Part of this Doppler-shifted sound reflects backward or is "backscattered" to the ADCP, and part of them reflects to other directions. The backscattered sound appears to the ADCP as if the scatterers were the sound source. In this situation, the ADCP hear the backscattered sound Doppler shifted a second time. Therefore, since the ADCP both transmit and receives sound, the Doppler shift is doubled, changing equation (3.2) to:

$$F_D = 2\ F_S\ (\ V\ /\ C\ ) \qquad (3.3)$$

The Doppler shift only works when there is a change in distance between sound source and receiver. This is known as *a radial motion*. If the sound source and receiver are moving at fixed relative position to one another, there will no Doppler shift. Mathematically, this means that the Doppler shift results from the velocity component in the direction of the line between the sound source and receiver. In ADCP case, this adds a new term, cos(α), to equation 3.3:

$$F_D = 2\ F_S\ (\ V\ /\ C\ )\ \cos(\alpha) \qquad (3.4)$$

Where α is the angle between the relative velocity vector and the line between the ADCP and scatterers (Figure 3.4).

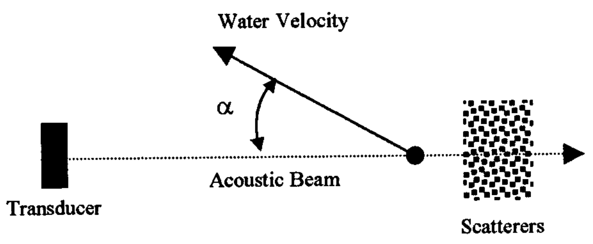


Figure.3.4.Doppler shift occurs if there is a radial motion.

3.3. Three-Dimensional Current Velocity Vectors.

As mentioned in section 3.2, each beam of the ADCP measures a single velocity component of the current. Thus, to obtain three-dimensional velocity components (two horizontal and one vertical velocity components), we require a minimum of three measurements from three beams simultaneously. These beams are configured to point in different directions to enable measuring different velocity components. For example, if the beam point east and another north, the ADCP will measure east and north velocity components. If the beams point in other directions, trigonometric relations can convert the measured current velocities into east and north components.

To use such kinds of multiple beams and apply trigonometric relations, one must make an assumption that the water currents are horizontally homogenous over layers of constant depth. The trigonometric assumption will not work if the current velocities are not the same at the measured place. Fortunately, this horizontal homogeneity assumption is reasonable in the ocean, rivers and lakes.

Generally, most of the ADCP have four beams rather than the minimum requirement of three beams. The fourth beam behaves as a redundant system to evaluate whether the assumption of horizontal homogeneity is reasonable. It is a built-in means to estimate data quality. This data quality is represented by the error velocity, which is computed from the existence of the fourth beam. Figure 3.5 illustrate how to compute three velocity components and error velocity using the four acoustic beams of an ADCP.

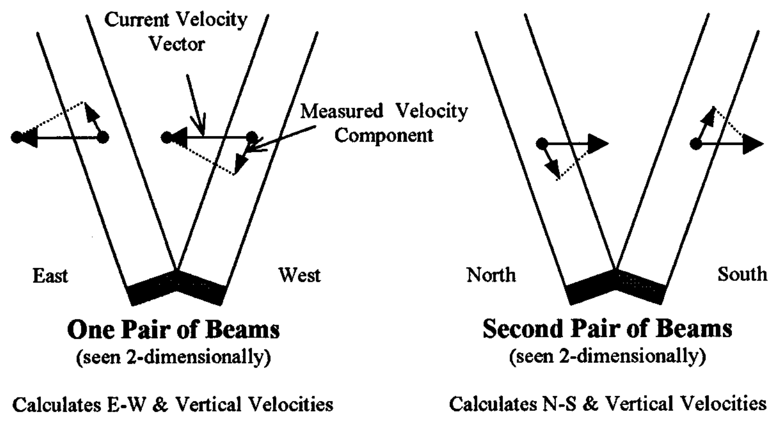


Figure 3.5. Two pairs of beams construct a three dimensional current velocity.

As shown in figure 3.5., one pair of beams produces one horizontal velocity (E-W) component and the vertical velocity component. The second pair of beams produces a second horizontal velocity (N-S) component, which is perpendicular to the first one, as well as the vertical velocity component. Thus there are two horizontal velocity components and two vertical velocity components. As three velocity components are required to form three-dimensional current velocity (east, north and vertical velocity components), the remaining vertical velocity component is useful to produce the error velocity.

The error velocity is the difference between the two vertical velocity components. This represents the homogeneity of the current over layers of constant depth, which means the quality of the measured current velocity. Since the current velocity measurement is strongly depend on the scatterer over water layer, the homogeneity of scatterer density at a constant depth plays a key role to the quality of the measured current velocity. Figure

3.6 shows two different conditions at a constant level of depth. In the first situation (a), the velocity is the same in all four beams which means the water layer is relatively homogenous. In the second situation (b), the velocity in one beam is different to the other three. This means that the current velocity is homogenous in all four beams. The error velocity in the second situation will, on average, be larger than the error velocity in the first situation. The in-homogeneity in the second situation may be caused by either the ADCP beam being bad or the actual currents are different. It does not a matter which one the cause is. The key point is that the error velocity can detect errors due to in-homogeneities of the water layers, as well as errors caused by malfunctioning equipment.

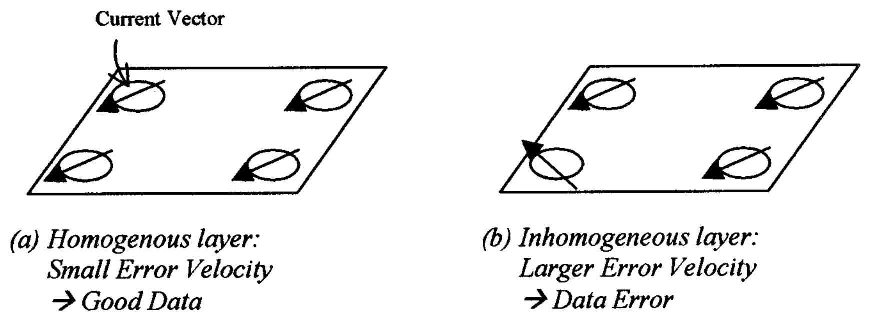


Figure 3.6. Homogeneity of scatterers' density in the water layer determine the quality of the measured current velocity.

3.4. Current Velocity Profile.

Unlike a conventional current meter, which measures water current at a point in a water column, the ADCP measures current velocity profiles along a water column. ADCP breaks up the velocity profile into several uniform segments called *depth cells* or often called *bins*. Each bin is comparable to a single current meter. Therefore, an ADCP profile is like a string of current meter uniformly spaced on a mooring current meter system (see figure 3.7). Based on this analogy, we can make several definitions:

- *Depth cell size* = distance between two consecutive current meters.
- *Number of depth cells* = number of current meter.

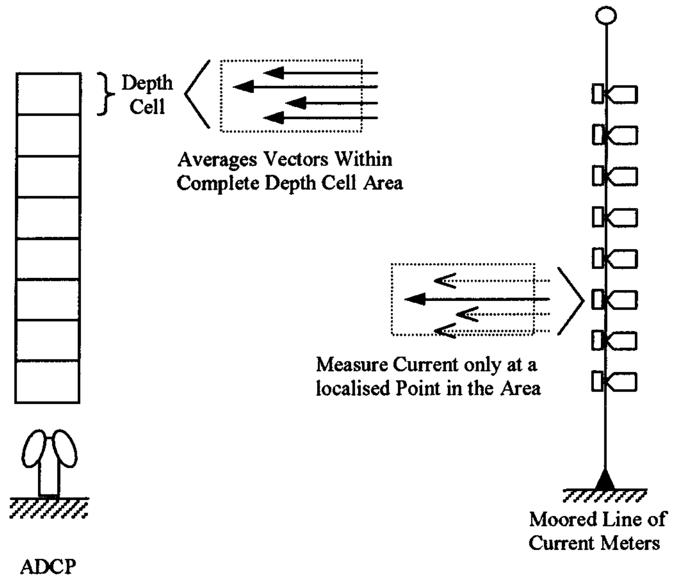

Depth Cell
Averages Vectors Within Complete Depth Cell Area
Measure Current only at a localised Point in the Area
Moored Line of Current Meters
ADCP

Figure 3.7 An analogy of ADCP profile to a mooring current meters system

However, this analogy is partly right, since there are two important differences between ADCP profile and a mooring current meters system. The first difference is that the depth cells size in an ADCP profile are always uniformly spaced while in a mooring current meter can be spaced at irregular interval. This regular spacing is intended to ease data processing and interpretation. Like any other data collections, it is much easier to process data collected at regular sample rate rather than irregular sample rate. The same benefit applies to measurements in current velocity profile.

The second difference is that an ADCP measures average velocity over the depth range of each depth cell, while the current meter measures current only at the current meter position. This averaging reduces some random errors that may occur during the measurements such as the effects of spatial aliasing. Aliasing in a time series is the process where signals at frequencies higher than the time series can resolve, are mistaken for low frequency signals. The effect of the aliasing is equivalent to increasing the noise level of the lower frequency signals resolved by the time series. By averaging over the range of the depth cell, the ADCP effectively rejects this kind of error.

Like any other measurements, the velocity measurement uncertainty from a single-measurement (called *a single-ping*) of ADCP is too large to meet most measurement requirements. Therefore, averaging data from several pings of measurements will reduce the measurement uncertainty to acceptable level. The ADCP data that come from averaging several pings of measurements is called an *ensemble*.

There are two kinds of error contributing to velocity uncertainty: *random error* and *bias*. Random error is the variation of a single-ping of measurement to its actual current value, while bias is the difference between the mean current value and its actual value. The averaging will reduce errors caused by random errors but not bias. Typically, the order of a single-ping random error is range from a few cm/sec to as much as 50 cm/sec. The size of this error depends on factors such as the ADCP frequency, depth cell size, the number of pings averaged together and beam geometry. Bias is typically on the order of 0.5 to 1.0 cm/ sec. This bias depends on a variety of factors including temperature, mean current speed, signal/noise ratio, beam geometry errors, etc. Unfortunately, it is not yet possible to measure ADCP bias and to calibrate or remove it in post-processing.

An ADCP system can calculate the ensemble averages inside of the ADCP, in the Data Acquisition System, or in both. It is possible, for example, to average ensembles of several pings in the ADCP and to send the results to Data Acquisition System, which then computes averages of these ensembles.

3.5. ADCP Data Structures.

The *ADCP Data Acquisition System* records the measured data from ADCP as ensembles. One ***ensemble*** is result from averaging several pings of measurements. Then, the *ADCP Data Processing System* may post-process the raw data by re-averaging ADCP ensembles over the desired averaging interval and stores the processed data to a binary file. One file of processed ADCP data containing a sequence of ensembles within the same cruise line is called a *transect*. A transect and the contained ensembles all together holds several kinds of data: configuration parameters, measurement information and bins data of current profiles. For a case study, this project uses a set of ADCP data that is provided by CSIRO. This data set is a processed ADCP data stored in CSIRO ASCII format. Figure 3.8 shows an example of ADCP data in CSIRO ASCII format.

In figure 3.8, the first three rows are a header record which contains a configuration parameters for that transect (the first and second rows are left blank). The data came after the header record are ensembles data, which contain measurement information and current profile in several bins.

```
   0         1         2         3         4         5         6         7         8
   01234567890123456789012345678901234567890123456789012345678901234567890123456789012
 1
 2   60   8   8   4                          1   3   6  0.50  0.60
 3
 4  07-APR-1994 02:40:00100  16  0.395  0.002   D 151.553 -33.982100  166  0  0 1200
 5  -0.50 -0.10 5.9  99 -0.50 -0.11 5.9  99 -0.49 -0.10 7.7  98 -0.49 -0.12 6.0  98
 6  -0.49 -0.13 6.0  97 -0.46 -0.14 6.3  97 -0.38 -0.15 6.3  97 -0.31 -0.12 6.4  96
 7  -0.32 -0.12 6.3  97 -0.33 -0.08 5.6  96 -0.35 -0.09 6.1  96 -0.41 -0.11 6.4  97
 8  -0.43 -0.06 5.5  97 -0.43 -0.05 5.5  97 -0.43 -0.06 5.4  97 -0.43 -0.06 5.5  67
 9  07-APR-1994 03:00:00100  20  2.205 -0.982   D 151.575 -33.990100  212  0  0 1200
10  -2.31  0.99 6.6  96 -2.31  0.97 5.9  96 -2.32  0.97 5.9  94 -2.32  0.95 6.2  94
11  -2.31  0.96 5.8  94 -2.26  0.99 5.9  93 -2.25  1.00 6.2  94 -2.26  1.05 6.7  94
12  -2.29  1.03 6.0  93 -2.26  1.03 6.2  93 -2.22  1.07 6.2  93 -2.25  1.06 6.2  94
13  -2.25  1.05 5.6  93 -2.29  1.04 5.4  93 -2.29  1.06 5.9  93 -2.26  1.10 6.0  92
14  -2.21  1.12 6.1  88 -2.15  1.12 5.3  73 -2.13  1.12 4.9  73 -2.12  1.16 5.0  59
```

Figure 3.8. CSIRO ASCII format of ADCP data

The following are some interpretation of the CSIRO ASCII format of ADCP data:

Row no.	**Description**
1-3	Header record (configuration parameter).
4	Measurement information for the first ensemble profile.
5 – 8	The first ensemble data contains current profile in several bins. Each bin holds 4 parameter values.
9	Measurement information for the second ensemble profile.
10-14	The second ensemble data contains current profile in several bins. Each bin holds 4 parameter values.

3.5.1. Configuration Parameters.

The Configuration parameters are set up before the measurement is started. This configuration parameter is very useful to know the situation during measurements and provide information for further data processing and analysis. For most users, the first four values in the second row are important to calculate the depth of bins and the maximum depth of sampling. These first four are number of bins to sample, bin length (in vertical meters), pulse length (in vertical meters) and delay after transmit (in vertical meters; also known as depth to first bin, DTFB). In the example the number of bins to sample is 60, the bin length is 8 m, the pulse length is 8 m, and the DTFB is 4 m.

3.5.2. Measurement Information.

The measurement information are very useful to know the situation during measurements and provide information for further data processing and analysis. The measurement information comes at the beginning of ensemble data. It contains:

- Date and time of ensemble's measurement.
- percentage of averaging period covered by acceptable ensembles (AvgCover).
- bottom-most accepted bin in ensemble's profile.
- Ship velocity components (East-West and North-South (u and v) components) in m/s.
- Navigation type code:
    - B = Bottom track velocity
    - P = GPS position-derived velocity
    - D = GPS direct velocity
    - Unc = No Navigation velocity (uncorrected)
    - rel = "navigation" velocity is actually that of some reference layer, so corrected velocities will be relative to that layer.
- Ensemble Position (longitude and latitude).

- percentage of interfix period for which there was bottom depth information (bottom coverage).
- mean bottom depth of ensembles for which a bottom depth was available.
- mean of bottom track error velocity (BTError).
- mean of ensemble percent good bottom track pings (BT correction only).
- integration period in seconds

In the example, the first ensemble profile began at 02:40 on 07-APR-1994 (UTC), AvgCover is 100%, the deepest good bin is bin 16, the GPS ship's velocity is (u, v) = (0.395,0.002) m/s, only Direct GPS velocities were used, the mean position during this ensemble profile was 151.553 E –33.982 S, bottom information was available for 100% of the profile, mean bottom depth was 166 m, and averaging period was 20 minutes (1200 seconds).

3.5.3. Bins Data.

The measurement of current velocity profile is based on a uniform segment called bins. The number of bins contained in an ensemble may vary depending on the bottom depth, but the maximum number of bins to be measured is set in the configuration parameters variable (bin to sample). Each bin in an ensemble contains four parameters, as follows:

- East-West components of the current velocity (east velocity) relative to the ship in m/s (+vel = east),
- North-South components of the current velocity (north velocity) relative to the ship in m/s (+vel = north),

- quality control value (avqc) and,
- attendance percentage or the percentage of the profile period for which there was good data in the bin (pctok).

3.5.3.1. Current Velocity Components.

The water current velocity in each bin is measured relatively to the ship velocity. The absolute velocity components can be obtained using the following equations:

$$\text{Ucorrected}(j) = \text{unav} + u(j) \tag{3.5}$$

$$\text{Vcorrected}(j) = \text{vnav} + v(j) \tag{3.6}$$

Where :

Ucorrected (j), Vcorrected (j)	=	corrected velocity components (east and north velocity) of bin j.
unav, vnav	=	ship velocity components (east and north ship velocity).
u,v	=	measured velocity components (east and north velocity).

As can be seen, the current velocity is a vector, that is, it has a magnitude and direction. We can find the magnitude (Vmag) and direction (Vdir) using the following equations:

$$\text{Vmag} = (u^2 + v^2)^{1/2} \tag{3.7}$$

$$\text{Vdir} = \text{atan}(u/v) \tag{3.8}$$

3.5.3.2. Quality Control Statistic.

The quality control value (avqc) contained in each bin is calculated from the major data quality statistic produced by the ADCP logging system, using the following equation:

```
avqc = %Good / ( Verr + 0.05 )
```
(3.9)

where:

`%Good`	= Percent Good pings after logging system screening.
`Verr`	= RMS Error Velocity in m/s.

The `avqc` has a possible range value of 0-20, and an expected range of 0-10, with values of 0 to 4 indicating very poor data, and values above 8 being very good data.

The other quality statistic, called `pctok`, comes from the profile integrating process. It is also referred to as the "attendance percentage". For each bin, it is the percentage of the pro-file period for which the data was acceptable. That is, if the data in a given bin (depth level) was usable in only half the ensembles during a given integration period, then for that bin `pctok` is 50.

3.5.3.3. Bin Depth.

Bin depth is the depth to the center of a bin, measured in vertical meters. The bins do not store the bin depth data. The bin depth is approximately calculated from several configuration parameters, using the following equation:

```
depth(j) = draught + (plen + blen)/2 + delay + blen*(j-1) + blen/10
```
(3.10)

Where:

`Draught`	=	4 m
`blen`	=	bin length
`plen`	=	pulse length
`delay`	=	delay after transmit (also known as DTFB - Depth To First Bin).

The depth bins are generated by the instrument using the assumption of a sound speed of 1475 m/s. The above approximation can therefore be refined by correcting for the approximate real sound speed, that is, by multiplying the above-derived depth by (*estimated real sound speed*) / 1475. This sound speed estimate would be made by estimating the mean temperature, salinity and depth for the main study area.

3.6. ADCP Data as a Complex Object.

As mentioned in Chapter II, the term complex object means any data that requires more than one structure to represent it. Unlike an atomic object, which is an object that cannot be subdivided into smaller components, a complex object may be composed of several atomic objects or other complex objects. In the 'real world', an object can be viewed as a single object, but combines with other objects in a set of complex A-PART-OF relationships. For example, a geographic coordinate is a complex object as it composed of three other objects latitude, longitude, and height. However, in the real world it can be viewed as a single object of point. The latitude, longitude and height are parts of point object respectively (A-PART-OF relationship).

An ensemble of ADCP data can be considered as a complex object since it composed of several other objects. An ensemble contains all measured values that come together with ADCP measurement. For simplicity, an ensemble can be divided into two objects: *spatial* and *aspatial objects*. The *spatial* object of an ensemble of ADCP data is actually a point object that represent the ensemble position in a particular coordinate system. A series of ensembles within a cruise line can also be considered as a line object of transect. Meanwhile, the *aspatial* object is actually all attributes associated with the ensemble position, representing all measured values contained in the bins as well as related measurement information and configuration parameters. Figure 3.9 and 3.10 show the member of spatial and aspatial objects of an ensemble.

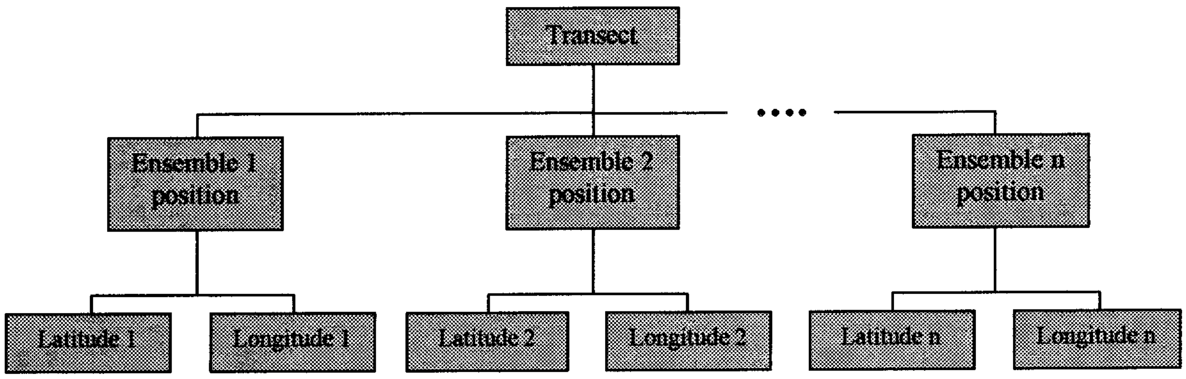


Figure 3.9 Spatial object components

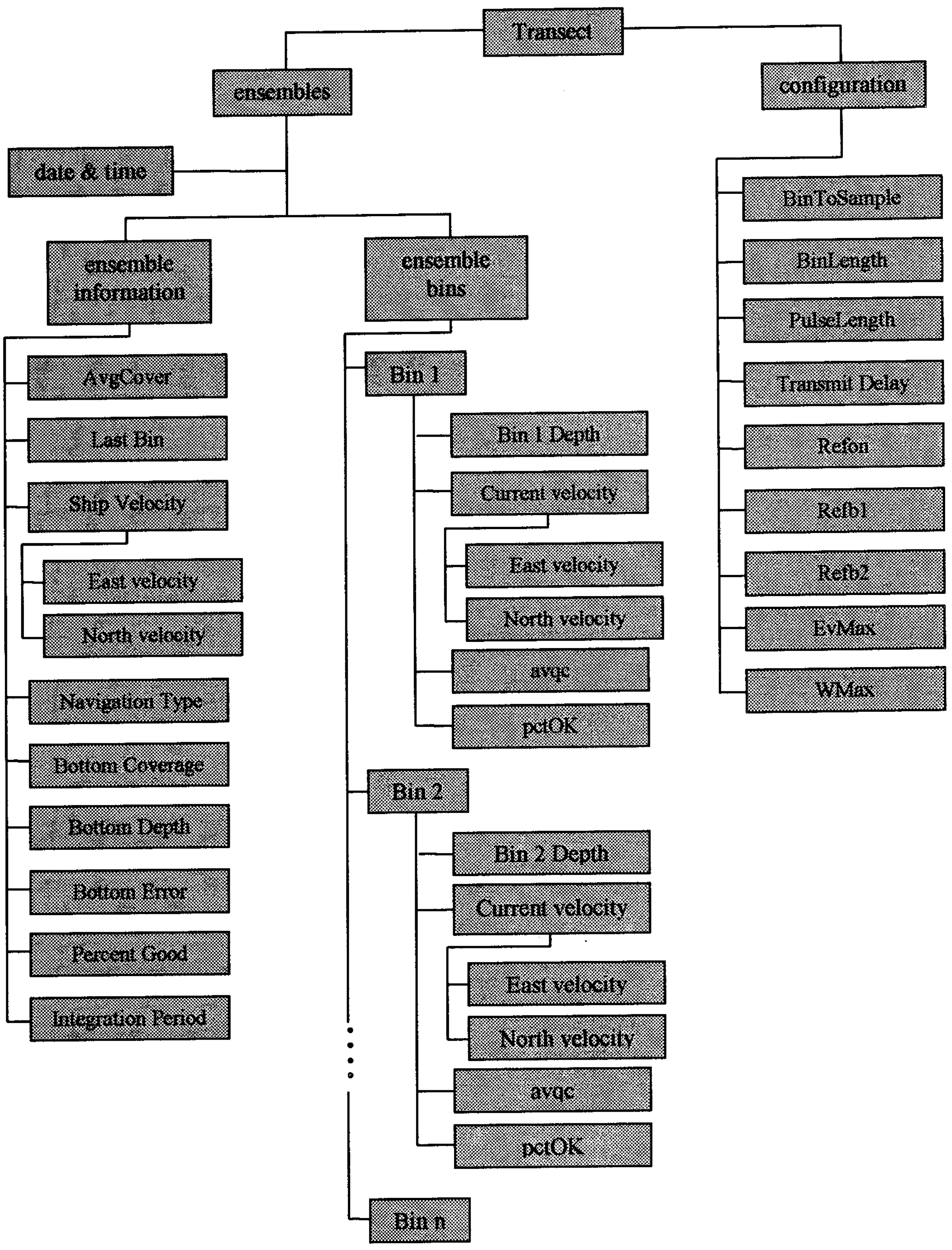


Figure 3.10 Aspatial object components of ensembles in a transect

3.7. The Reasons to Store and Manage ADCP Data in a Database Management System (DBMS).

So far, many oceanographic data centres store ADCP data on file based systems. This would not be any problem if the data centre is the only user of ADCP data. Problems arise if users from other organizations want to access the data. There are various user categories that use ADCP data for their own purposes. Different users may require different types of ADCP data. The data centre, however, should provide any required information for a variety of users. For example, a user may require current velocity data at 5 m depth interval within a particular area boundary, some other at 20 m depth interval at different area boundary. In a file based systems, we have to find each file that contain the data within the specified area boundary, extract the required data and interpolate them according to the specified depth interval. All of these steps are carried out by an operator and requires a long time to finish. In a database management system (DBMS), such a query is carried out behind the scenes. What we need to do is send the SQL query to the DBMS via the network according to the specified criterion, and get the result in just a few seconds.

The following are some reasons to store and manage ADCP data using a database management system:

- *Spatially distributed.*

  ADCP data are usually measured along the cruise line of the ship (transect). The measured data are then averaged within a specified time interval or specified distance to form data ensembles. Thus the positions of ADCP ensembles are spatially distributed according to the cruise line. In a file based data storage, any data query should be based on the cruise line, since ADCP data that came from a cruise line of measurement may be stored in a file or more. This kind of data management makes it difficult to query ADCP data based on a specified area boundary that may intersect one or more cruise lines as well as the whole or part of the cruise lines. Database management systems may overcome this situation by allowing users to query ADCP data based on spatial queries.

- *Vertically distributed.*

  An ensemble of ADCP data is also vertically distributed at a particular depth interval. Each set of data at certain depth is called *a bin*. An ensemble may have bins up to maximum of 128 bins. The number of bins may vary between each ensemble depending on the bottom depth and bin interval. In query processing, some users may only require the data at particular depth or depth range. Again, the same problem will arise if the ADCP data is stored in a file-based system. A database management system may overcome this situation by providing convenient access for variety of users via SQL query.

- *Complex data.*

  Since a lot of data are associated with ADCP measurement, some users may only require some part of the data (for example velocity magnitude and velocity direction only), some other may require the whole data of ensembles for further processing (including all ensemble information and configuration parameters). A database management system provides more convenient ways to facilitate this kind of query rather than a file based system. Moreover, in dealing with complex data, the recent object-relational database management system offers a better way to store and manage ADCP data.

Chapter IV

ADCP Data Management Systems (ADMS) Development

4.1. Introduction.

As mentioned in the previous chapter, Acoustic Doppler Current Profiler (ADCP) data is an important data set in marine area. This valuable data set is often required in any decision-making processes in marine related activities. ADCP data is considered to be complex data since it requires more than one structure to represent it. The complexity of ADCP data requires an appropriate data management system and applications to enable users to access, analyze and visualize the data in a convenient ways. The ADCP Data Management Systems (ADMS) is developed to provide a better service for various users in marine communities.

This chapter explains the development stages of ADCP data management system. In general, the stages of ADMS development can be describe as follows:

- *Database planning.* This involves planning how the stages of ADMS development can be realised most efficiently and effectively.
- *System definition.* This involves specifying the scope and boundaries of database application, its users, and application areas.
- *Requirements collection and analysis.* This involves the collection and analysis of the requirements of users and application areas.

- *Database design.* This includes the conceptual, logical, and physical design of the ADCP database.
- *Application design.* This involves designing the user interface and the application programs that used and process the ADCP database.
- *Implementation.* This involves creating the external, conceptual, and internal database definitions and the application programs.
- *Data conversion and loading.* This involves converting the ADCP data of CSIRO format and loading the data as well as all related information into the ADCP database.
- *Testing.* This involves testing the ADMS for errors and validated against the requirements specified by the users.

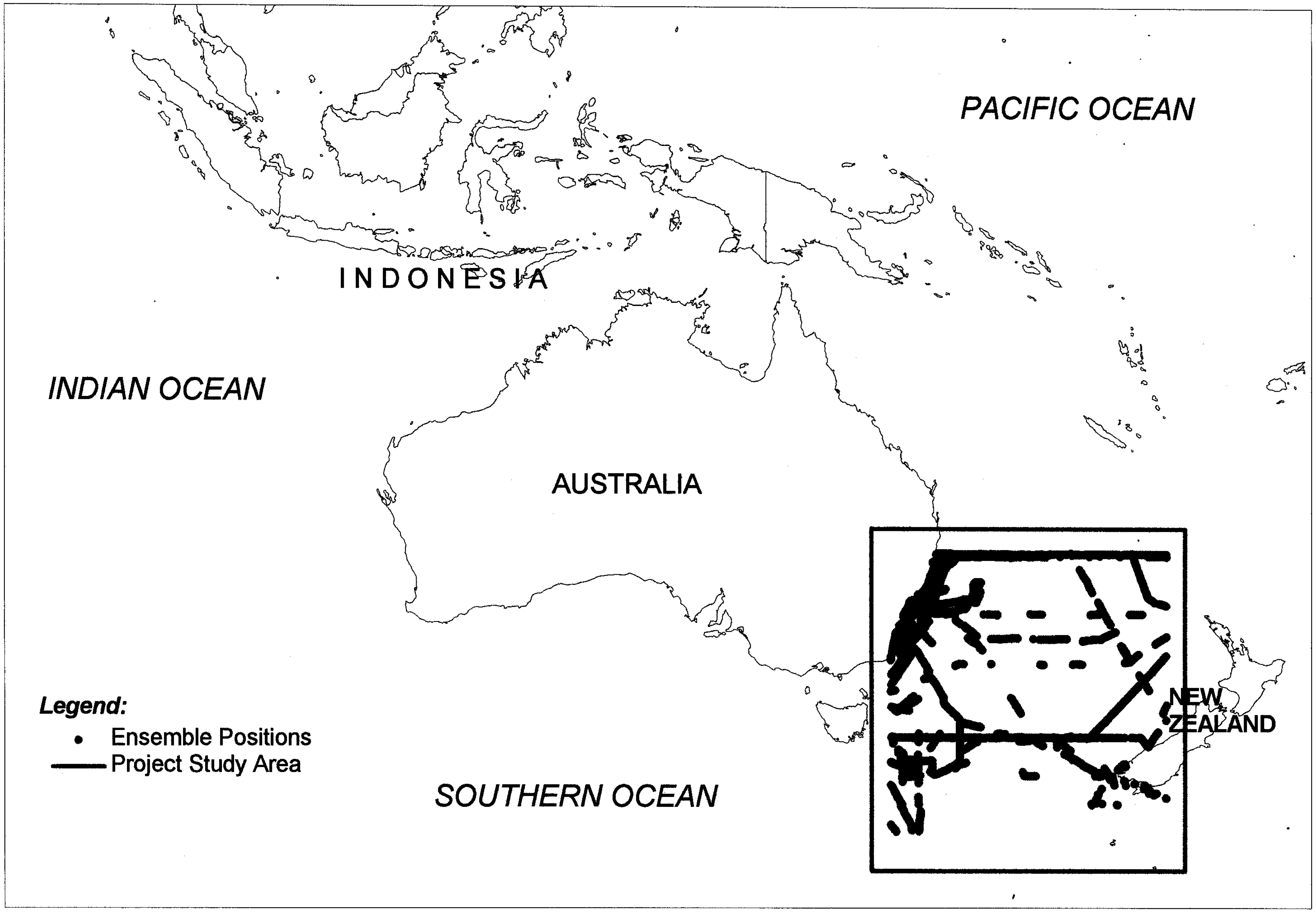
4.2. Planning.

The planning stage is an important stage in the ADMS development. In this stage, all work to be done and the required resources are defined. Good planning will make the whole development processes work as efficiently and effectively as possible. This planning stage also includes the identification of user's requirements, how data will be collected, what data format will be used, as well as evaluating available information systems technologies to determine the best possible resources to be used.

4.2.1. Data Collection.

For a case study, this project used the ADCP data set available at CSIRO Division of Marine Research, in Hobart. This data sets consists of 37 transect files of ADCP data within 20° x 20° area boundary (30° - 50° S and 150° - 170° E). The ADCP data used in

PROJECT STUDY AREA



this project is a data set that had been processed to remove any obvious errors. This data set is stored in ASCII format. Figure 4.1 shows the map of the project area.

4.2.2. Resources.

The following resources are used in completion of the project.

- Informix Dynamic Server with Universal Data Option ver 9.14, including Geodetic Data Blade Extension.
- Java^{TM} 2 SDK version. 1.2.2.

4.3. Requirements Collection and Analysis.

The requirement collection and analysis were carried out to produce the requirement specifications for the user's view. The users were categorized into two parts: the data center where ADCP data is held and public users including education, government institutions, military, private companies as well as foreign users as part of data exchange programs.

4.3.1. Data Requirements:

- Each bin of an ensemble has velocity parameters (east velocity and north velocity components), as well as quality control parameter and percentage of the profile period for which there was good data in the bin. Each bin within an ensemble is identified by bin depth. An ensemble may compose of one up to 128 bins at particular bin depth interval.

- An ensemble stores a series of bins and one distinct piece of measurement information for that ensemble. The measurement information includes percentage of averaging period covered by acceptable ensembles, bottom-most accepted bin in ensemble profile, ship's East-West velocity for the profile (m/s), ship's North-South velocity for the profile (m/s), navigation type code, percentage of interfix period for which there was bottom depth information, mean bottom depth of ensembles for which a bottom depth was available, mean of bottom track error velocity, mean of ensemble percent good bottom track pings, and integration period.
- A Transect or a cruise line has several ensembles separated at particular distances. Each transects hold a measurement configuration for all ensembles in that transect. The measurement configuration includes number of bins to sample, length of bins (in vertical meters), length of pulse (in vertical meters), delay after transmit (in vertical meters), real-time reference layer averaging setup, threshold for ping by ping data rejection on basis of error velocities (m/s), vertical velocity real-time threshold (m/s) and bandwidth data screen threshold.

4.3.2. Transaction Requirements.

The main transactions required by the data center as well as public user include:

- Inserting ADCP data into the database, when new data comes. The data may come from the data center or other organizations.
- Inserting all new related information if any, such as cruise information, vessel, instrumentation, survey area, survey period, personnel's in charge (both in measurement and processing).

- Display part or all available transects lines as well as ensemble positions.
- Produce a list of water current velocities at particular depth within selected area.
- Load one or more complete ensembles of ADCP data for further analysis and processing using external application programs.
- Load data in the original format for further processing.
- List institutions that carried out the survey for particular ADCP data.
- List the cruise information as well as vessel information used at ADCP measurements.
- List ensemble positions that meet particular conditions.
- List some or all ensemble elements such as ensemble information, ensemble bins

4.4. Database Design.

Database design is performed to assist in the understanding of meaning (semantics) of the data and to facilitate communication about information requirements. The design represents:

- Each user's perspective of the data,
- The nature of the data it self, independent of its physical representations,
- The use of data across application areas.

In this project, the database design is divided into three stages, as follows:

- *Conceptual database design:* to build the conceptual representation of the database, which includes identification of important entity and relationship types.

- *Logical database design :* to translate the conceptual representation to the logical structure of the database, which includes designing the relations.
- *Physical database design :* to represent how the logical structure is to be physically implemented (as tables) on the target database management system (DBMS).

The above database design stages of the development of ADMS are explained in the following sections.

4.4.1. Conceptual Database Design.

The conceptual database design includes the process of identifying all available entity types, attributes as well as its attribute domains, and relationships between entities. Entity types are objects that have independence existence. Attributes are properties that describe the entity types whereas a relationship is an association between entity types. In some cases, a relationship type may have attributes. This stage also includes the process of identifying all possible candidate keys. Among those candidate keys, one is chosen as an appropriate primary key for each entity types. The primary key uniquely identifies the record in the relation. Tables 4.1 and 4.2 show all available entity types for ADCP database and relationship type respectively. The complete entity list can be found in appendix A.

The documentation of components that are required in the data modelling process is important prior to building the ADCP database. Data modelling is useful to visualize the entities as well as the relationship between them. This enables users (either the data centre or public users) to review and evaluate the data model to meet their requirements.

In this project, the data modelling is performed using Enhanced Entity-Relationship (EER) modelling. This modelling is chosen since it is the most appropriate data modelling to model a complex data such as ADCP data in an Object-Relational Database Management System (ORDBMS).

Table 4.1. List of entity types used in ADCP database.

Entity name	Description	Occurrence
Institutes	• Institution that performed ADCP measurement. • Vessels owner	• An institution may perform one or more projects of ADCP measurement survey. • An institution may own one or more vessels
Projects	ADCP measurement survey	One project may consists of one or more cruise lines within particular area boundary
Cruises	trips within a project	One project may have one or more cruises
Vessels	The vessel used for ADCP measurement	One vessel may be used to performed one or more ADCP measurement survey
Countries	The country where institution is located	An institution may be located in one or more countries
Instrumentations	Instrumentation used in ADCP measurement either ADCP or Navigational instruments	One ADCP measurement uses one type both ADCP instrument and navigation instrument
Transects	Cruise line where ADCP measurement is performed	• A transect has a series of discreet ensembles. • A transect has one configuration parameter for all ensembles within it.
Ensembles	ADCP measured value	An ensemble contains one date and time of measurement, one measurement information (EnsembleInfo) and a series of bins (EnsembleBins)
EnsembleInfo	Measurement information contained in each ensemble.	One ensemble contains one measurement information.
EnsembleBins	A series of bins within ensembles	Each ensemble has one up to 128 bins
EnsembleConfig	Measurement configuration	Ensemble within a transect has the same configuration parameters

Table 4.2. List of Relationship Types of ADCP Database.

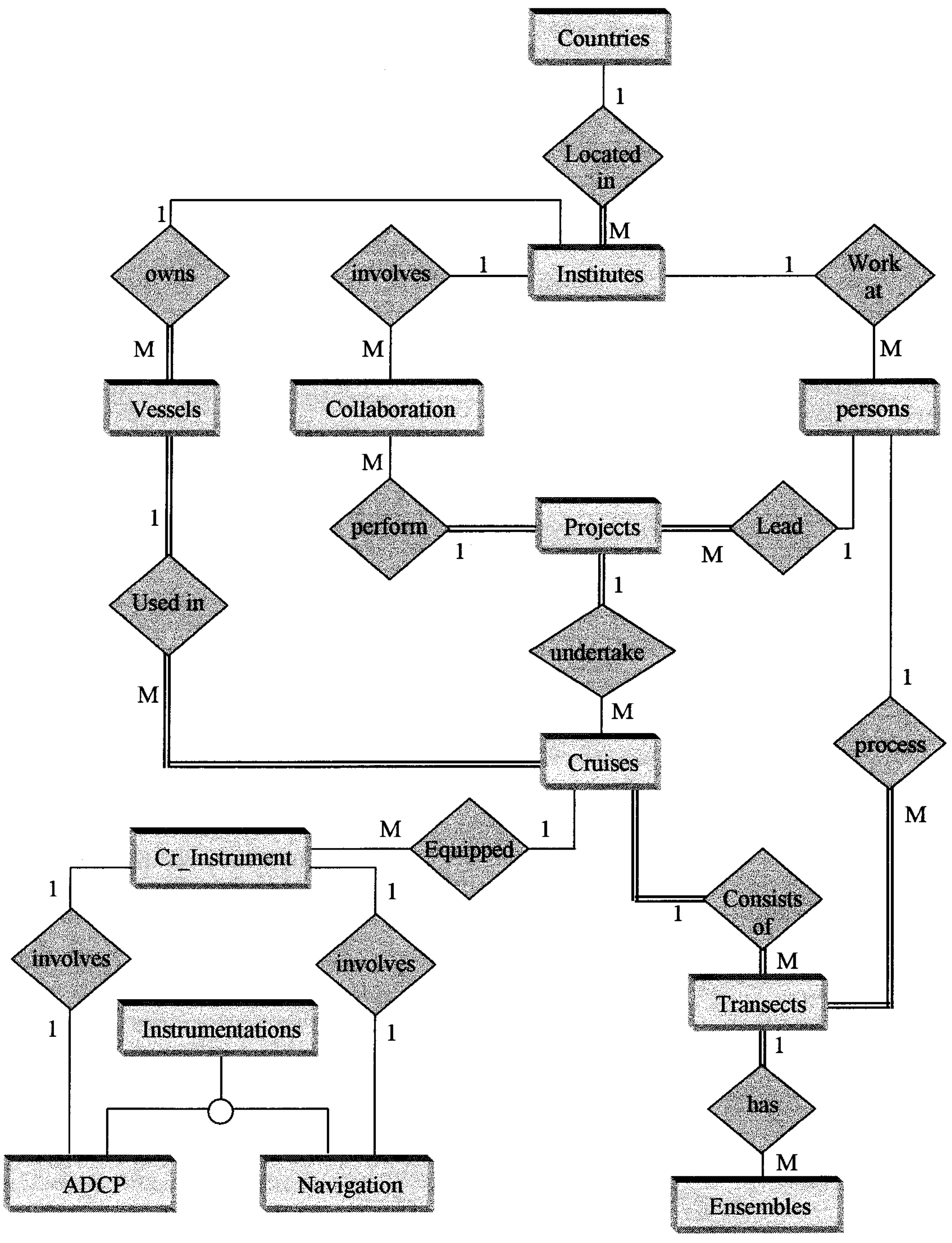
Entity Type	**Relationship Type**	**Entity Type**	**Cardinality Ratio**	**Participation**
Institutes	• own • Located in • Perform	• Vessels • Countries • Projects	1 : M M : 1 M : N	P : T T : P P : T
Projects	Undertake	Cruises	1 : M	T : P
Cruises	• Equipped • Consist of	• Instrumentations • transects	M : N 1 : M	T : P T : T
Vessels	Used in	Cruises	1 : M	T : T
Transects	Has	Ensembles	1 : M	T : P
Persons	• Work at • Lead • Process	• Institutes • Projects • Transects	M : 1 1 : M 1 : M	P : T P : T P : T

Notes : P = partial participation, T = Total participation.

Unlike modelling in traditional relational data model where all attributes should be decomposed into atomic values, the modelling in object-relational data model enables us to maintained a complex data as an object without decomposing it into atomic values. Of course, this violates some of the normal form rules of normalisation process that a traditional relational data model has to meet. However, this provides some benefits such as reducing join operation between relations, enabling a single data access to spatial data and its complex attributes, which has an implication of increasing data access performance. A more detail discussion about this would be found in chapter 5.

4.4.2. Logical Database Design.

The logical database design is carried out to refine the conceptual data model in the previous stage by removing data structures that are difficult to implement in the DBMS.



Notes:

▭	Entity Type	———	Partial Participation
◇	Relationship Type	═══	Total Participation

Figure 4.2. Entity-Relationship (ER) model for ADCP database.

In the relational data model, the process of normalisation is often used to validate this logical model. However, the object-relational data model does not require fulfilling all normalisation rules. To some extent, it is better to keep a complex object as it is, without decomposing it into atomic values as required in the first normal form of normalisation rules. Chapter 5 will discuss some reasons for doing this.

Meanwhile, on the completion of the database design stage, we should have a model that is correct, comprehensive and unambiguous. This objective is achieved by undertaking some activities, if any, such as removing many to many relationships, removing complex relationships, validating the data model using normalisation as well as against transactions. These activities provide a better ER diagram as shown in Figure 4.2.

In this logical database design stage, the mapping between ER diagram into relations is derived. This mapping is DBMS independent and it is used to create the entire ADCP database using *Database Definition Language (DDL)* regardless any target DBMS. The following are ER to relations mapping for the entire ADCP database:

Countries(co_id, co_name)
Primary Key co_id

Institutes(inst_id, inst_name, address, co_id)
Primary Key inst_id
Foreign Key co_id **references** Countries

Persons(pers_id, pers_name, inst_id)
Primary Key pers_id
Foreign Key inst_id **references** Institutes

Projects(p_id, p_name, p_area, survey_period, pers_id)
Primary Key p_id
Foreign Key pers_id **references** Persons

Collaboration(coll_id, inst_id, p_id)
Primary Key coll_id
Foreign Key inst_id **references** Institutes
Foreign Key p_id **references** Projects

Vessels(vs_id, vs_name, inst_id, vs_photo)
Primary Key vs_id
Foreign Key inst_id **references** Institutes

Cruises(cr_id, cr_name, s_port, e_port, p_id, serial_no)
Primary Key cr_id
Foreign Key p_id **references** Projects

Cr_Instrument(cri_id, cr_id, navigation, adcp)
Primary Key cri_id
Foreign Key cr_id **references** Cruises
Foreign Key navigation **references** Instrumentation (serial_no)
Foreign Key adcp **references** Instrumentation (serial_no)

Instrumentation(serial_no, instrument_name, manufacturer, model, firmware_no)
Primary Key serial_no

ADCP(transmit_freq, td_config, beam_angle, beam_width)
Under Instrumentation

Navigation(differential, p_code, ant_offset)
Under Instrumentation

Transects(tr_id, config, cr_id, pers_id, tr_notes)
Primary Key tr_id
Foreign Key cr_id **references** Cruises
Foreign Key pers_id **references** Persons

Ensembles(ens_id, ens_position, ens_data, tr_id)
Primary Key ens_id
Foreign Key tr_id **references** Transects

The mapping involves some user-defined data types (UDT) as well as user-defined functions (UDF) to handle the complexities of ADCP data. These capabilities enable users to store a complex data within a single column within a relation (table). The UDT created in ADCP database as well as its field data members are shown in table 4.3, while the UDF are shown in appendix C.

Table 4.3. User Defined Data Types for ADCP database.

UDT	Field Names	Data Type	Description
Velocity_t	EastVelocity NorthVelocity	Smallfloat Smallfloat	Holds velocity data components
Bin_t	BinDepth BinVelocity Avqc PctOK	Smallfloat Velocity_t Smallfloat Integer	Holds measured value in particular depth
EnsInfo_t	AvgCovered ShipVelocity NavType Bcover BottomDepth BTError PctGood IntegrationPeriod	Integer Velocity_t Varchar(3) Integer Integer Integer Integer Integer	Measurement information of an ensemble
EnsConfig_t	Draught BinsToSample BinLength PulseLength TransmitDelay Refon Refb1 Refb2 EvMax WMax	Smallfloat Integer Integer Integer Integer Integer Integer Integer Smallfloat Smallfloat	Measurement configuration for each transect
EnsBins_t	Bins	LIST(Bin_t not null)	A series of bins in an ensemble at particular depth interval
Ensemble_t	EnsDateTime EnsInfo EnsBins	DateTime year to second EnsInfo_t EnsBins_t	Hold all measured ADCP data as well as measurement information in a particular position.

Address_t	Street
	City
	State
	Phone
	Fax
	Zip
		Varchar(40)
		Varchar(20)
		Varchar(3)
		Varchar(15)
		Varchar(15)
		Varchar(4)
			Hold address data of institutes

Name_t	Firstname Lastname	Varchar(15) Varchar(15)	Hold person name
Offset_t	X Y Z	Smallfloat Smallfloat Smallfloat	Hold antenna offset data of navigation instrument relative to the ADCP transducer.

4.4.3. Physical Database Design.

The physical database design determines how the data model, represented in the logical database design, will be stored in a specific target DBMS. Since it is dependent on the target DBMS, the way the data is stored will be different for different DBMSs. In this project, the ADCP data is stored in *Informix Dynamic Server with Universal Data Option version 9.14*, which is referred to as *Informix Universal Server (IUS)*. This DBMS supports Object-Relational Database Management System (ORDBMS) technology that enables user to gain all functionality of Relational DBMS technology as well as object oriented features such as extensibility, support complex data, and inheritance.

This project also utilized the *Informix Geodetic Data Blade* extension module to support the spatial part of ADCP data. This extension module enables us to store spatial objects such as point, line or polygon in a single column of a relation. There are two spatial objects in ADCP database that use *Informix Geodetic Data Blade* Data Type: **ensemble position** and **boundary definition of project area**.

Besides the translation of logical data model for target DBMS, the physical design also includes the design of the constraint for each relation, choosing primary and or secondary index methods, as well as designing security mechanisms. In this project the *R-tree* index method is used as a secondary indexes for ensemble position (`ens_position`) column in the ensembles relation. The ensemble position used the `GeoPoint` data type of *Informix Geodetic Data Blade.*

The implementation of the physical database design is performed using SQL Data Definition Language (DDL) that is precompiled in a Java programming language. Appendix B shows the complete list data definition language used to create the user defined data type (UDT) as well as tables for ADCP database in Informix Universal Server (IUS) version 9.14.

4.5. Application Design.

The application design stage is actually a parallel activity to the database design stage. It involves designing the application programs (user interface) to access the data in the database, and designing the data access methods (transaction design). A database application program or user interface is a tool for the users to interact with the database and performing transactions as specified in the requirement collection and analysis. In ADCP database, there are two categories of transaction: *retrieval transaction* and *update transactions* depend on the type of users.

The *retrieval transaction* is opened to all users. However, some users may only have privilege to access public data sets, while some other specific users may access restricted data set. The data centre that maintains the data set drives this user access policy. The *update transaction* privilege is restricted to the data centre that maintains the ADCP database only. It involves inserting new records, deleting old records, or modifying records in the database.

The ADCP Data Management System (ADMS) application programs to support the ADCP database is developed using *Java™ 2 SDK ver 1.2.2*. Java programming language is chosen to be used in the ADMS development since it has an advantage of platform independence. This capability is important on developing a cross-platform application that can be run by various users. The following are some of the main tasks of the ADMS:

- Reading the ADCP data from the original format in ASCII file format and wrapping it into ensemble object prior inserting the data into the database.
- Inserting new ADCP data as well as all related information (cruise, project, institution, instrumentation, and personnel in charge).
- Sending queries to the database as well as receiving query results.
- Displaying query result either as tables, list, forms or graphical displays.
- Loading selected data into original format or output standard format for data exchange.

Several Java classes are required to implement the above tasks. These Java classes utilize the *Java DataBase Connectivity Application Programming Interface (JDBC API)* in order to be able to connect to the ADCP database. The JDBC used in this project is the Informix JDBC API version 2.0, which is available from Informix and can be downloaded freely via the Internet. Informix JDBC API is a Java API for accessing virtually any of tabular data stored in Informix. There are some other JDBC specific to particular DBMS as well as standard JDBC-ODBC that can be applied to any DBMS. The JDBC consists of a set of classes and interfaces written in Java programming language that provides a standard API. Figure 4.3 shows the connectivity between ADMS, JDBC and ADCP database.

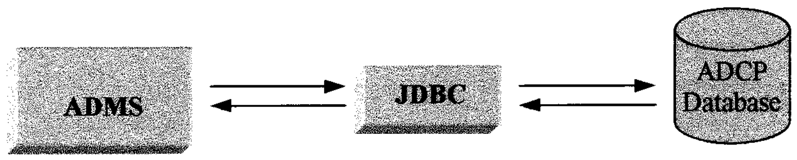


Figure 4.3. ADMS, JDBC and ADCP database connectivity.

4.5.1. Type Mapping.

The ADCP database utilizes custom mapping of SQL user-defined types (UDTs). A custom mapping maps a UDT to a class in the Java programming language where each attribute of the UDT is mapped to the corresponding field in the class. However, a class in a custom mapping must implement the SQLData interface. The SQLData interface is a special interface in Java programming language that is used only for the custom mapping of SQL user-defined types. The mapping from Java classes to some of Informix Geodetic DataBlade Data Type is also defined to facilitate spatial based queries and analysis. Table 4.4 lists the Java classes and the corresponding Informix Geodetic DataBlade Data Types, while table 4.5 lists the Java classes and the corresponding UDT defined for the ADCP database. A more detail description on these Java classes can be found in Appendix D.

Table 4.4. Mapping from Java class to Informix Geodetic DataBlade Data Type.

Java class	Informix Geodetic DataBlade Data Type
GeoCoords	GeoCoords
GeoAltRange	GeoAltRange
GeoTimeRange	GeoTimeRange

Table 4.5. Mapping from Java class Type to SQL3 UDTs in ADCP database

Java class	Informix UDT
Velocity	Velocity_t
Bin	Bin_t
EnsembleInfo	EnsInfo_t
EnsembleConfig	EnsConfig_t
EnsembleBins	EnsBins_t
Ensemble	Ensemble_t
Address	Address_t
Name	Name_t
Offset	Offset_t

The custom mapping enables us to store a Java class as an object directly into the database, as well as retrieving a UDT object directly from the database. Instead of working with the whole object on an ensemble, someone may want to work with some part of the ensemble data. This is enabled by accessing some of the class member of `Ensemble` Java class (in the application) or the attribute member of the `Ensemble_t` UDT (in the database). Each field member of a Java class can be accessed using the methods associated with the Java class. The same situation is also applied to the UDT in the ADCP database where each attribute member can be accessed using the available functions as part of SQL Queries. To facilitate this purpose, seventy functions have been created. Accessing an attribute member of a UDT can be perceived as accessing a single column in relational DBMS. This is similar to accessing a single column in a relational database management system. The difference is that, it does not require any join operation when we use functions to access attribute members of a UDT in object-relational DBMS, while in relational DBMS join operation is a necessity. Appendix C & D list all methods associated with the Java class as well as UDT's functions defined for the ADCP database.

4.6. Implementation.

The implementation stage is a physical realization of the database and application designs as described in the previous sections. This implementation is achieved using the Data Definition Language (DDL) to create database schema and empty database files. The Data Manipulation Language (DML) is also used to facilitate database transactions, either update transaction or retrieval transaction. In this project, either DDL or DML is precompiled in Java programming language and implemented via *Informix Java Database Connectivity Application Programming Interface (JDBC API) version 2.0*. Several Java classes have been created to facilitate ADCP database creation as shown in table 4.6.

Table 4.6. List of Java Classes to create ADCP database.

Java Classes	Description
CreateDB	Create empty ADCP database
CreateTables	Create all tables in ADCP database
CreateFunction	Create User-defined Functions (UDF) in ADCP database

The classes must be run in sequence, otherwise it may generate errors. One should noticed that after creating the empty ADCP database, the Informix Geodetic DataBlade must be registered to the database using the DataBlade Manager, prior to creating the tables. This enables the database to utilise the functionality of Geodetic DataBlade on manipulating the spatial data of ADCP. A shell script can be created to simplify the procedure and it must contain the following command lines:

```
java CreateDB
blademgr < adcp.txt
java CreateTables
java CreateFunction
```

The above script will create the ADCP database, register the Informix Geodetic DataBlade, create the required tables as well as all user-defined functions. The `adcp.txt` as shown in the second line of the script is a redirect input to the Blade Manager (`blademgr`) and it contains the following command lines:

```
set confirm off
list adcpdb
register geodetic.2.11.UC3 adcpdb
```

The following is an example of DDL to create the `ensembles` table:

```
create table ensembles (
     ens_id          serial8 not null primary key
                     constraint pk_ensembles,
     ens_position    GeoPoint,
     ens_data        ensemble_t,
     tr_id           integer references transects
                     constraint ens_fk_tr);
```

Figure 4.4. Data definition language (DDL) for creating the `ensembles` *table.*

As can be seen in figure 4.4., the ensembles table stores the spatial data and its attributes (aspatial data) in a single table. The ensemble position, as the spatial data, is stored in column `ens_position` of `GeoPoint` data type while the ensemble data of ADCP, as the attributes of ensemble position, is stored in a column called `ens_data` of `ensemble_t` data type. This can be a benefit since the spatial data and it's complex attributes can be retrieved in a single data access. The complete DDL for all tables, as well as user define data types, in the ADCP database can be found in Appendix C.

As part of the application, several Java classes as well as it's associated methods were created to facilitate the database transactions. A Java class is simply a type code for an instance of a class defined in the Java programming language that is stored as a database object. These Java classes are custom mapped to SQL3 user-defined types (UDTs) via the Informix JDBC so that they can be stored directly into the database as any other built in data types. In addition, the custom mapped UDTs can be used in the query as part of the SQL statement. The following query is an example of retrieving ensemble positions and it's complete ensembles data using SQL statement:

```
Select ens_position, ens_data from ensembles
Where ensYear(ens_data) between 1990 and 1992;
```

Figure 4.5. SQL query example

Several functions were also created in Informix Universal Server associated with each user-defined data type. These functions are used to access each element of UDTs in ADCP database. Furthermore, these functions can also be used as part of the SQL statement in Java application programs. They have the same functionality to the corresponding methods of Java classes. For example, the function `ensYear()` in figure 4.5, returns the ensemble year of measurement. Appendix C lists the complete available functions in ADCP database as well as corresponding methods of the ADCP Java classes.

In the application side, the ADCP Java classes are also used to store the query results. These results can then be used in the application such as graphical displaying, table listing, form listing or further analysis. For example, the query results of the SQL statement in figure 4.5. can be stored in `GeoCoords` and `Ensemble` classes respectively, as shown in the following fragments of a Java program:

```
pstmt = IfxCon.prepareStatement(query);
rs = pstmt.executeQuery();

while(rs.next())
{
   GeoCoords EnsPositon = (GeoCoords) rs.getObject(1);
                              // get the object returned by the first column, cast it to GeoCoords class and store it in the
                              // EnsPosition variable of GeoCoords class.
   Ensemble Ens = (Ensemble) rs.getObject(2);
                              // get the object returned by the second column, cast it to Ensemble class and store it in the
                              // Ens variable of Ensemble class.
   Ens.printEnsemble(); // print the ensemble data to standard output
}
```

Figure 4.6 Java class for materializing query results.

In the above fragment, the variable `EnsPosition` stores the first column of the query result (`ens_position` of `GeoPoint` data type) while the variable `Ens` stores the second column of the query result (`ens_data` of `ensemble_t` data type).

4.7. Data Conversion and Loading.

The original ADCP data comes in a particular format that can not be inserted into the database directly. This requires data conversion before loading the data into the database. This project used ADCP data provided by CSIRO, which is available in an ASCII file format. Several Java classes were created to facilitate this data conversion and loading. Table 4.5 lists all Java class classes used for data conversion and loading in this project.

Table 4.6 Java classes that are used for data conversion and loading.

Java Class	Description
InsertCountries.class	Inserting countries data into `Countries` table.
InsertInstitutes.class	Inserting institution information that performs ADCP survey or owns the survey vessel into `Institutes` table.
InsertPersons.class	Inserting personnel data that lead a project or process the ADCP data into `persons` table.
InsertProjects.class	Inserting projects information into `projects` table.
InsertCollab.class	Inserting institutions that perform collaboration in one or more projects into `collaboration` table.
InsertVessels.class	Inserting survey vessel information that is used in ADCP survey into `vessels` table.
InsertADCP.class	Inserting ADCP instrument specification that is used in an ADCP survey into ADCP table.
InsertNavigation.class	Inserting navigation instrument specification that is used in an ADCP survey into `Navigation` table.
InsertCruises.class	Inserting cruise information into `cruises` table.
InsertCI.class	Inserting information of ADCP and navigation instrument that are used in a cruise into `CruiseInstrument` table.
RCsiro.class	Reading ADCP data from files (in ASCII format) as well as inserting the data into `Transects` and `Ensembles` table directly.

4.8. Query Examples.

Some testings has been performed to find errors against the application and validate the database transactions. There were some errors found in this testing stage. All significant errors had been fixed either by refining the application program or improving the database transaction strategy.

As part of the testing stage, some queries had been performed through the database application that is developed using Java application program, to evaluate that both the database and the application are working well. The following are several examples of the SQL statements and the application fragment to materialise the query result.

- *Query 1.*

```
SELECT ens_id, Coords(ens_position), ens_data
FROM ensembles
WHERE ensmonth(ens_data) = 9
AND ensyear(ens_data) = 1993;
```

The above SQL statement returns the ensemble id (`ens_id`), the ensemble position (`ens_position`) which is composed of latitude and longitude, and the ensemble of ADCP data (`ens_data`) for any ensemble that are measured during September 1993. The `ensmonth()` and `ensyear()` are user-defined function that return the month and year of ensemble measurement date respectively. The `ensmonth()` and `ensyear()` take `ensemble_t` user-defined data type as input parameters (see Appendix D for detail).

The following is a fragment in the database application program to materialise the above query result:

```
long EnsId = rs.getLong(1);
GeoCoords enspos = (GeoCoords) rs.getObject(2);
Ensemble ens = (Ensemble) rs.getObject(3);
enspos.printGeoCoords();
ens.printEnsemble();
```

As can be seen, the returned values are stored in the corresponding variables/classes in the database application. Once they are stored in the Java variables/classes, they can be used by the users for any purposes such as displaying, plotting, printing, saving or analysis. In the above example the latitude and longitude of the ensemble position as well as the ensemble of ADCP data are printed out to a standard output (monitor screen) using `printGeoCoords()` and `printEnsemble()`methods respectively.

- *Query 2*

```
SELECT ens_id, Coords(ens_position), ens_data
FROM ensembles
WHERE intersect(ens_position, setdist(
'(((-30,150),(-35,155),(-40,165),(-50,170)),
any,any)'::GeoString, 15000));
```

The above SQL statement returns the ensemble id (`ens_id`), the ensemble position (`ens_position`) which is composed of latitude and longitude, and the ensemble of ADCP data (`ens_data`) for any ensemble that are located within 15 km of the specified line `((-30,150),(-35,155),(-40,165),(-50,170))`. The `setDist()` is a Geodetic DataBlade function to create buffer or proximity at the specified distance (in this case 15 km). The `intersect()` in `WHERE` condition is a Geodetic DataBlade operator function to overlay (intersect) two spatial objects (in this case ensemble position and the specified line).

To materialise the query result the same fragment, as described in query 1, is used since the returned values are also the same.

```
long EnsId = rs.getLong(1);
GeoCoords enspos = (GeoCoords) rs.getObject(2);
Ensemble ens = (Ensemble) rs.getObject(3);
enspos.printGeoCoords();
ens.printEnsemble();
```

- *Query 3.*

```
SELECT Coords(ens_position), velocityMag(ens_data, 30),
       velocityDir(ens_data,30)
FROM ensembles
WHERE intersect(ens_position, setdist(
'(((-30,150),(-35,155),(-40,165),(-50,170)),
any,any)'::GeoString,15000));
```

The above SQL statement returns the ensemble position (`ens_position`) which is composed of latitude and longitude, the velocity magnitude and the velocity direction at 30 m depth for any ensemble that are located within 15 km of the specified line (`(-30,150),(-35,155),(-40,165),(-50,170)`). The velocityMag() and velocityDir() are user-defined functions to calculate the velocity magnitude and velocity direction at the specified depth of an ensemble of ADCP data. These two functions take the `ensemble_t` user-defined data type and depth value as input parameters (see Appendix D for details).

The following is the corresponding Java fragment to materialise the query result in the database application:

```
GeoCoords enspos = (GeoCoords) rs.getObject(1);
float VMag = rs.getFloat(2);
float VDir = rs.getFloat(3);
enspos.printGeoCoords();
System.out.println("VMag: " + VMag + "    VDir: " + VDir);
```

- *Query 4*

```
SELECT Coords(ens_position), ens_data, config
FROM ensembles e, transects t
WHERE inside(ens_position,
'((-40, 155),(-35,160),any,any)'::GeoBox)
AND e.tr_id = t.tr_id ;
```

The above SQL statement returns the ensemble position (`ens_position`) which is composed of latitude and longitude, the ensemble of ADCP data (`ens_data`) and the configuration parameter (`config`) for any ensemble positions that are located inside the specified boundary box `((-30,150),(-35,155),(-40,165),(-50,170))`. The `inside()` is a Geodetic DataBlade operator function to perform spatial analysis whether one object is located inside another object.

The following is the corresponding Java fragment to materialise the query result in the database application:

```
GeoCoords enspos = (GeoCoords) rs.getObject(1);
Ensemble ens = (Ensemble) rs.getObject(2);
EnsembleConfig config = (EnsembleConfig) rs.getObject(3);
enspos.printGeoCoords();
config.printEnsembleConfig();
ens.printEnsemble();
```

4.9. Summary.

In this project the ADCP database had been developed to evaluate some approaches of ORDBMS on managing marine spatial data. Some user-defined data types (UDT) were created to facilitate object-based transactions to enable users to query part or whole ensemble of ADCP data. Appendix B lists the UDT created for ADCP database as well as its attribute members. Several user-defined functions (UDF) have also been created to enable a user to retrieve the UDT's attribute members as well as performing data manipulation and analysis in the database server. Appendix C lists all user-defined functions that can be used for these purposes.

A database application written in Java programming language had also been created to optimise the benefits of ORDBMS approaches on managing ADCP data. This database application includes several Java classes and it's associated methods to enable users sending queries in SQL3 statements as well as materialising the query results. Once the query results are stored in the Java variables or classes, the users can use them for any purposes according to their requirements. The next chapter discusses various approaches of using ORDBMS for managing ADCP data as a spatial data that can be applied to other marine spatial database management.

Chapter V
Discussion

Object-Relational Database Management Systems (ORDBMS) have emerged to overcome the difficulties of Relational DBMS in response to the increasing complexity of database applications. The ORDBMS extends the capabilities of Relational DBMS to have object-oriented features while storing the data in relational structures. The object-oriented features of ORDBMS include extensibility, supporting complex data (data that is composed from other existing data types either atomic or complex data types), and supporting inheritance. This chapter discusses some features of ORDBMS compared to its predecessor technology of Relational DBMS. The discussion uses examples from ADCP database.

5.1. Extensibility.

The extensibility feature of ORDBMS provides a freedom to the database designer to create their own data types and functions to meet their requirements. Creating new data types can be achieved by extending the existing data types or defining new data types as well as functions to support them. The ADCP data can be divided into two kinds of data, the *spatial* and *aspatial (attributes)* data. This project used user-defined data types (UDT) and functions (UDF) provided by Informix Geodetic DataBlade module to handle the spatial part of the ADCP data. Several other UDT and UDF were created by extending the existing data types to support the complexity of ADCP data (discussed in section 5.2).

5.1.1. User-defined Data Types (UDTs).

The Informix Geodetic DataBlade module enables us to store and manipulate – in an Informix Universal Server database – objects in space referenced by latitude and longitude, and provide additional attributes representing an altitude range and a time range. With this datablade module, we can create several spatio-temporal objects such as points, line segments, strings, rings, polygons, boxes, circles and ellipses. The Informix Geodetic DataBlade module provides data types that represent these spatial objects on an ellipsoidal representation of the earth.

In this project, the ADCP data only dealt with one spatial object of points. This spatial object represents an ensemble position of ADCP measurement. The `GeoPoint` data type of Informix Geodetic DataBlade is used to hold the ensemble position of ADCP data and store it into a single column of `ens_position` in `ensembles` table. Several functions have been provided to access the attribute members of this spatial data as well as to manipulate them. It also enables spatial query using spatial operators to be performed - in SQL statement - such as *beyond, inside, intersect, outside* and *within*, as shown in the following example:

```
SELECT  ens_position, ens_data
FROM ensembles
WHERE inside(ens_position,
'((-35,160),(-40,165),ANY,ANY)'::GeoBox);
```

Figure 5.1. Retrieving spatial data and its attributes based on spatial query.

The SQL statement in figure 5.1. returns ensemble positions (`ens_position`) and its attributes of ADCP data (`ens_data`) from `ensembles` table where the ensemble positions are located inside the specified boundary box (SouthWest corner: 35° S and 160° E; NorthEast corner: 40° S and 165° E). The `ens_position` column is a `GeoPoint` data type that holds the latitude and longitude values of the ensemble position. The following is a representation of `GeoPoint` data type:

```
GeoPoint((lat, long), altrange, timerange)
```

Where :

`lat` and `long`	are the latitude and longitude of the `GeoPoint.`
`Altrange`	is the altitude range.
`Timerange`	is the time range.

The `ens_data` column is an `ensemble_t` data type that holds an ensemble of water current velocity profile (explained in section 5.2). The `GeoBox` is an Informix Geodetic DataBlade data type that represents an area bounded by four orthogonal edges parallel to the coordinate axes (defined by SouthWest and NorthEast corners).

5.1.2. User-defined Functions (UDFs).

A function is merely a task that the database server perform on one or more values. There are *built-in functions* such as cos(), max(), avg(), etc., and *user-defined functions (UDFs)*. A UDF is a function created by the user to support a specific task on either built-in or user-defined data types. Unlike the built-in functions which can be used anywhere by the system, the user-defined function can only be used by the type that the function is attached to and its subtypes.

This project utilizes two kind of UDFs; the UDFs that are provided by the Informix Geodetic DataBlade to manipulate the spatial data as well as performing spatial analysis, and the UDFs that are created in this project to manipulate the ADCP data. For example, we might want to retrieve all ensemble positions and its associated ADCP data within a distance of 15 km along a proposed underwater pipe line. The following SQL query will provide the desired data:

```
SELECT ens_position, ens_data
FROM ensembles
WHERE       intersect(ens_position,
            SetDist((((-30,150),(-40,160),(-45,165),
            (-50,170)),ANY,ANY)::GeoString, 15000:: GeoDistance));
```

Figure 5.2. Example of spatial analysis using SetDist function of Informix Geodetic DataBlade.

The following are external representation of SetDist function as well as GeoString and GeoDistance as the arguments of the function in figure 5.2:

- `SetDist` function:

  ```
  SetDist(GeoString, GeoDistance)
  ```

- `GeoString` data type:

  ```
  (((lat1, long1),(lat2,long2),. . .,(latn,longn)), altrange, timerange)
  ```
  or
  ```
  GeoString(((lat1,long1),(lat2,long2),...,(latn,longn)),altrange,timerange)
  ```

Where:

`GeoString`	a connected, non-branching sequence of line segments (may intersect itself).
`GeoDistance`	domain-enforcing type for distance.
`lat` and `long`	are the latitude and longitude of the nodes.
`altrange`	is the altitude range.
`timerange`	is the time range.

The flexibility to create user-define function is a great feature of ORDBMS. It enables a lot of the application tasks to be carried out in the database server rather than in the client application. This reduces the amount of data sent to the client application as well as transaction load. It also reduces the access time of a transaction which means increasing the database and application performances.

5.1.3. Comparison to Relational Structures.

As a comparison to relational structures, the creation of user-defined data types and user-defined function is not supported. In relational DBMS, the users are restricted to only use the built-in data types and functions that the database server defined such as numeric, character and date. The built-in data type is atomic; that is, it cannot be broken into smaller pieces. As a consequence, any complex objects should be decomposed into atomic values before storing them into columns of a relation. For example, a geographic position that is composed of latitude and longitude should be decomposed into atomic values of built-in data type. Thus, the ensemble position of ADCP data should be stored into two columns of `numeric` data type in a relation. Decomposing a complex object into atomic values is one of the *normalization* processes, in order to meet the requirement of *First Normal Form (1NF)*. 1NF is the situation where a relation contains only atomic (or single) values at the intersection of each row and column. The purpose of normalisation is to produce a set of relations with desirable properties and ensures no update anomalies will occur. However, the process of normalization in the relational model generally leads to the creation of relations that do not correspond to entities in the 'real world'.

The extensibility feature of ORDBMS provides a better real-world representation of an object rather than the traditional relational DBMS. As can be seen in the previous example (figure 5.1), we can view a point as an object (composed of latitude and longitude) rather that viewing it as two numeric values representing latitude and longitude. Moreover, with ORDBMS we can use the UDTs as any other built-in data types in the database as well as SQL statements.

5.2. Complex Data.

As mentioned in the Chapter III that ADCP data is considered as complex data since it composed from several other data (measurement information, bins data etc.) and it requires more than one structure to represent it. The object-relational database management system (ORDBMS) technology is an appropriate database management system to store and manage this kind of complex data. Using this technology, it is possible to store an ensemble data in a single column of a relation rather than decomposing the data into atomic values of several relations.

5.2.1. Storing Complex Data into a Single Column of a Relation.

Storing complex data into a single column of a relation can be perceived as storing nested relations into a single column. For example, a complex data of an ensemble position of type `GeoPoint` is stored in column `ens_position` of `ensembles` relation. The `GeoPoint` holds another complex data of `GeoCoords`, which is composed of `GeoLatitude` and `GeoLongitude;` `GeoAltRange`, which is composed of `LowAltitude` and `HighAltitude`; and `GeoTimeRange`, which is composed of `LowTime` and `HighTime`. In case there are only one value of altitude and time, the values of `LowLatitude` and `HighAltitude` as well as `LowTime` and `HighTime` will be the same. In this situation, the `GeoCoords`, `GeoAltRange` and `GeoTimeRange` can be perceived as another relation, that holds atomic data respectively, within a single column of `ens_position`.

Another example is the `ens_data` column of `ensembles` relation that holds a complex data of an ensemble of ADCP data. An ensemble data is stored as `ensemble_t` data type (UDT) and holds another complex data of ensemble date and time of measurement of `DateTime` data type; measurement information of `EnsInfo_t` data type (UDT); and a series of Bins data of `EnsBins_t` data type (UDT). The suffix `_t` is used to name a user-defined data type (UDT) that are created in this project. The `EnsInfo_t` holds several atomic values of measurement information (`AvgCovered, NavType, Bcover, BottomDepth, BTError, PctGood, and IntegrationPeriod`) and one complex data `ShipVelocity` of `velocity_t` data type, which holds EastVelocity, and `NorthVelocity` component values.

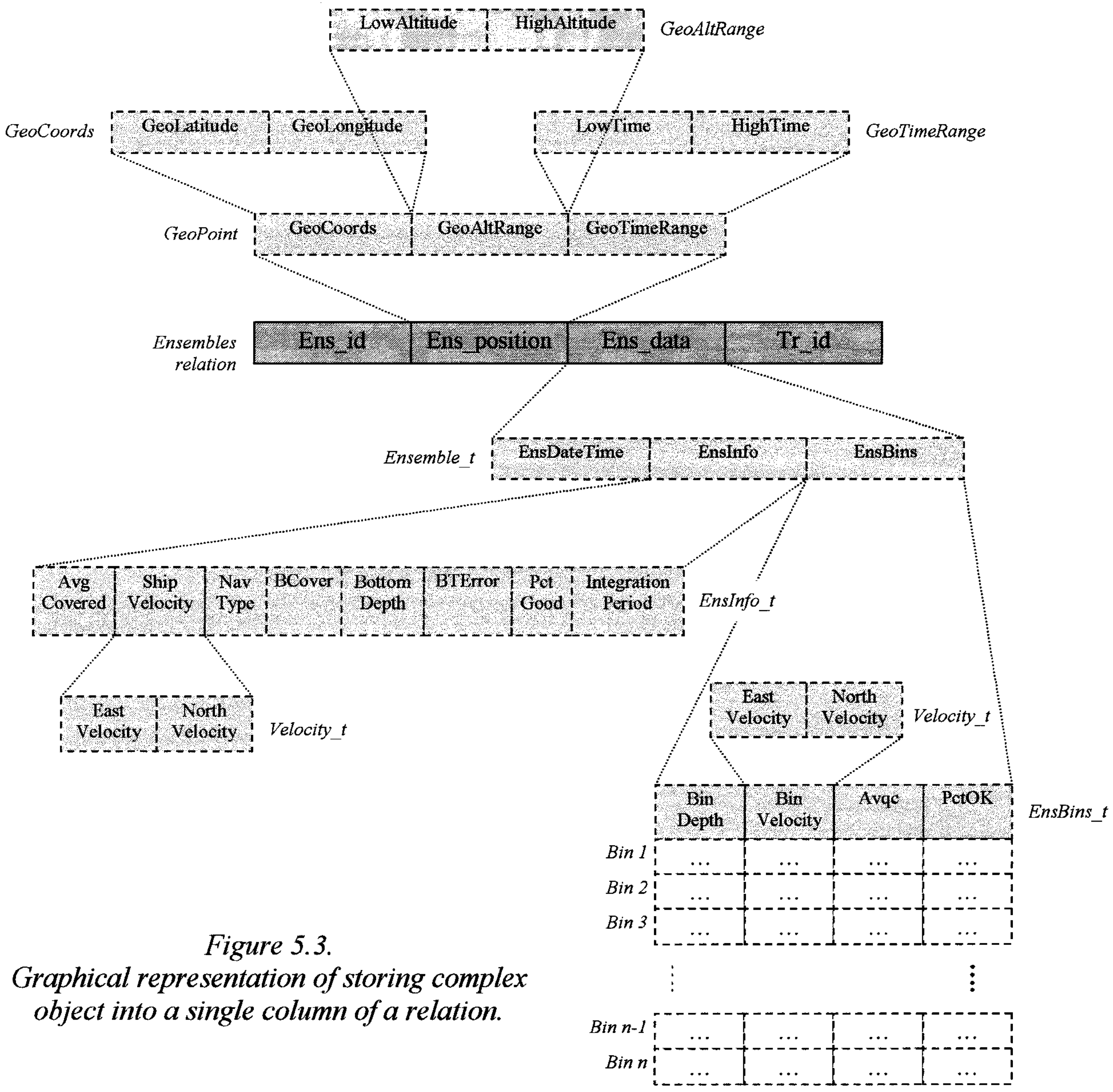


Figure 5.3.
Graphical representation of storing complex object into a single column of a relation.

Meanwhile, the `EnsBins_t` holds a series of complex data of `Bin_t` data type. The `Bin_t` data type composed of three atomic values of Bin data (`BinDepth, Avqc,` and PctOK) and one complex data of water current value, which holds EastVelocity and NorthVelocity components. One important thing here is that, the EnsBins_t is a resizable data type. It can hold one up to 20 bins. The limited maximum number of bins this data type can hold is due to the limitation of Informix database on handling row and collection data types where the EnsBins_t data type is built from. The Informix DBMS restricted the number of data that row or collection data types can hold up to 32 Kbytes. There is a solution to overcome this problem. One should make an opaque data type which can hold unlimited size of ADCP data. Unfortunately, this was not done in this project due to the limited time of the project completion. Figure 5.3 shows a graphical representation of storing complex data into a single column of a relation.

5.2.2. Accessing Attribute Members of a Complex Data.

There are two methods of accessing each attribute member of a UDT: by using *cascading dot* and *accessor functions*. We can use a cascading dot if we know the data structure of a UDT. However, for most users who do not know the UDT's data structure, the accessor functions provide a more convenient way of accessing an attribute member of a UDT. The following two SQL statements provide examples of accessing ShipVelocity components contained in an ensemble:

- Using *cascading dot.*

```
SELECT   ens_data.ensinfo.shipvelocity.eastvelocity,
         ens_data.ensinfo.shipvelocity.northvelocity
FROM ensembles
WHERE tr_id = 9001;
```

- Using *accessor functions.*

```
SELECT   eastShipVelocity(ens_data), northShipVelocity(ens_data)
FROM ensembles
WHERE tr_id = 9001;
```

Figure 5.4 Examples of accessing attribute member of a UDT using cascading dot and accessor function methods in SQL statements.

The SQL statements in figure 5.4 return the same value of `ShipVelocity` components contained in an ensemble that is stored in `ens_data` column of `ensembles` relation. As can be seen, the second statement that uses accessor functions looks simpler than the first one. This can be understood since the second statement used the function that has the same name with it's returned value. Moreover, using functions may retrieve an aggregate value of some data members. For example, we may retrieve the `ShipVelocity` magnitude and direction that are calculated from `ShipVelocity` components, as shown in the following example:

```
SELECT  ShipVelocityMag(ens_data), ShipVelocityDir(ens_data)
FROM ensembles
WHERE tr_id = 9001;
```

Figure 5.5. An example of aggregate functions within an SQL statement.

In this project, 70 accessor and aggregate functions have been created to facilitate convenient database transactions of ADMS. The complete list of the functions can be seen in Appendix C.

5.2.3. Comparison to Relational Structure.

As a comparison to relational structure, any complex objects should be broken into smaller pieces of atomic values, as required by the First Normal Form (1NF) rule of normalisation process. As mention in section 5.1.3, the process of normalisation in the relational model generally leads to the creation of several relations with relationships between relations, which may be complicated.

The ability of ORDBMS to support complex object leads to the creation of simple tables in the database. Simple table means that the number of columns in a table as well as the number of tables in the database can be reduced to a minimum level. As an impact, the SQL statements to access complex data can be written as simple as possible. For example, the ensemble of ADCP data is stored in a single column of `ens_data` in `ensembles` table (see Figure 5.3). One instance of ensembles occupies only one row in the table. If an ensemble holds 20 bins of current velocity profile, then the ensemble will be composed of 110 atomic values (10 atomic values of measurement information and 100 atomic values of bin data (20 bins x 5 atomic data each)). However, all of these data are stored in a single column, one row per instance.

In relational structures, two separate tables are required to store the ensemble of ADCP data (see Figure 5.6) and each column of the tables can only store an atomic value of built in data type. Thus, decomposing the ensemble of ADCP data into atomic values is a necessity. These two tables will store the measurement information and the bins data of current velocity profile. If an ensemble has 20 bins, then a ten-column table is required to store measurement information (`Info` table) and another five-columns table is required to store the current velocity profile (`Bins` table). This situation excludes one column of foreign key in each table to represent the relationship between tables. Moreover, the Bins table requires 20 rows to hold one instance of 20 bins ensemble.

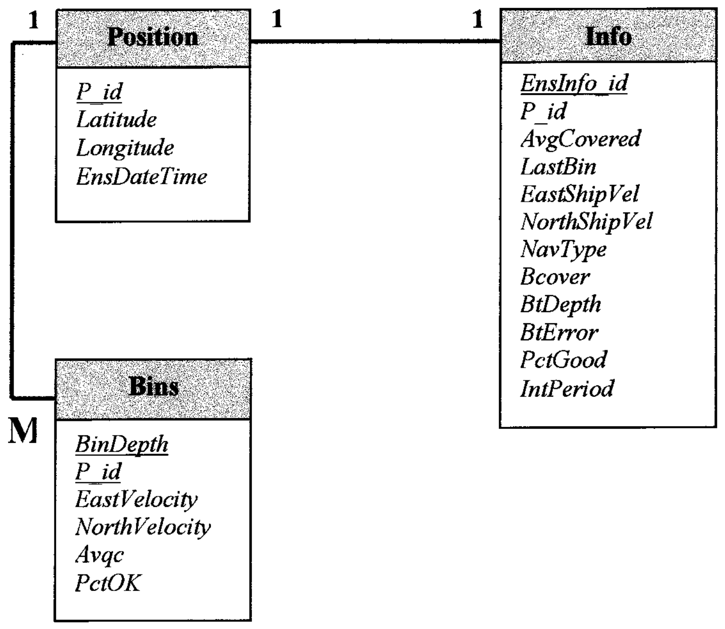


Figure 5.6. Relations that are required to store ensembles of ADCP data in relational DBMS.

As can be seen in this example, the ORDBMS provides an efficient way of storing and managing complex data than the RDBMS. This can be considered as a new approach of managing spatial data such as ADCP data that can be applied in marine spatial data management.

5.2.4. Benefits of Storing Complex Object into a Single Column of a Relation.

Some new approaches for spatial data management have been introduced by the emerging ORDBMS technology. One of these approaches is storing a complex object into a single column of a relation. This approach could not be implemented in the former RDBMS where a column of a relation could only store an atomic value of built-in data type. There are some benefits of storing a complex object into a single column of relation as explained in the following sections.

5.2.4.1. Storing the spatial data and its associated complex attributes in the same relation.

The Object-Relational Database Management System (ORDBMS) enables users to store the spatial data and its complex attributes in a single relation. This allows a single data access to retrieve the spatial data and its attributes. For example, the `ensembles` relation in ADCP database stores the spatial data of ensemble position in a column `ens_position` of `GeoPoint` data type, and it's complex attributes of ensemble data in a column `ens_data` of `ensemble_t` data type. The query to these two attributes can be performed in one SQL statement as shown in the following example (rewritten from Figure 5.1):

```
SELECT ens_position, ens_data
FROM ensembles
WHERE inside(ens_position,
      '((-35,160),(-40,165),ANY,ANY)'::GeoBox);
```

Figure 5.7. Retrieving spatial data and its attributes based on spatial query.

The SQL statement in figure 5.7, returns ensemble positions (`ens_position`) and its attributes of ADCP data (`ens_data`) from ensembles relation where the ensemble positions located inside the specified boundary box. As can be seen, this statement retrieves the spatial data and it's associated attributes in a single data access.

As a comparison to the relational structure, the spatial data and its attributes are usually stored in separate relations. This requires join operations between relations to retrieve the attributes based on the spatial data queries.

5.2.4.2. Reducing joins operation between relations.

Storing a complex data into a single column may have a benefit of reducing the number of relations that are required to store the data. This has an implication of reducing join operation between relations. In the relational structure, where a complex data should be decomposed into atomic values, several relations are required to store them and the join operation is a necessity to access the data across relations. In a database context, a join operation is considered as one of the most expensive operation to perform. For example, an ensemble of ADCP data is stored into a single column of a relation in ORDBMS, but it requires at least three relations to store the same data in RDBMS, as can be seen in Figure5.6, join operations are required to access data from these relations.

5.3. Inheritance.

Inheritance is the process that allows a type or a table to acquire the properties of another type or relation. The type or relation that inherits the properties is called the *subtype* or *subtable*. The type or relation whose properties are inherited is called the *supertype* or *subtable*. The subtype or subtable will inherit all properties that are defined on the supertype or supertable such as structure, behaviors (routines, aggregates, operators, constraint definitions, referential integrity, access methods, storage options, triggers, etc.), and indexes. In this project, inheritance is applied to adcp and navigation tables that inherit the properties of instrumentation table. It should be a typed-table in order to utilise inheritance, thus the adcp, navigation and instrumentation tables are typed-tables of adcp_t, navigation_t, instrument_t respectively. Figure 5.8 shows the inheritance hierarchies of the specified tables.

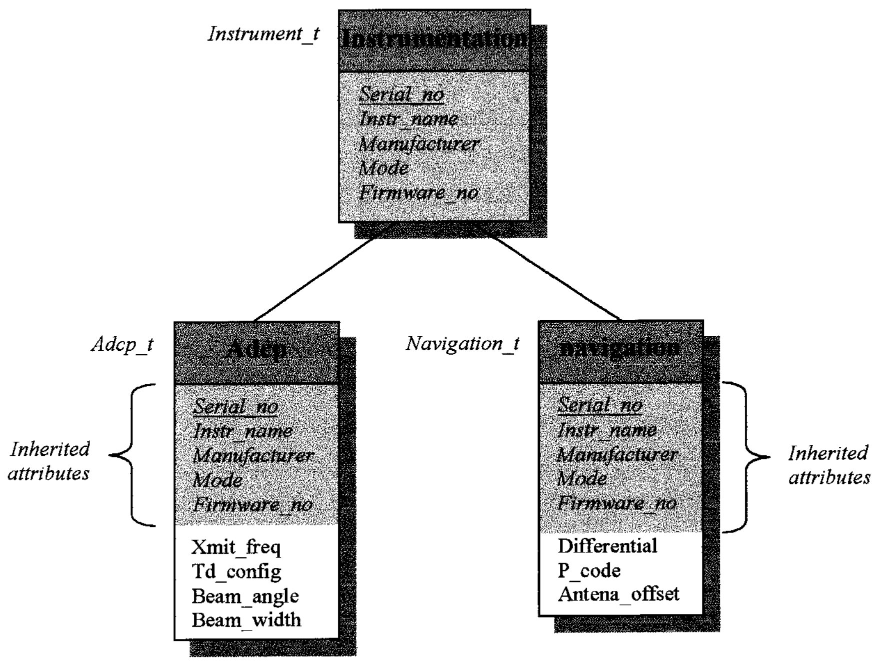


Figure 5.8 Inheritance hierarchies.

5.3.1. Comparison to Relational Structures.

Inheritance is not supported in relational DBMS. Three separate relations should be created to holds all attributes of instrumentation information both for ADCP and

Navigation instruments. Moreover, joins between tables should be performed to query all data regarding ADCP and Navigation instruments.

5.4. Some New Approaches on Marine Spatial Data Management.

The object-oriented features of ORDBMS provide flexibility in data modelling especially when dealing with complex data such as marine spatial data. Unlike relational data modelling, where any complex object should be broken down into atomic values before storing them into columns of relations, the object-relational data modelling enables us to model complex objects that correspond to entities in the 'real world'. Moreover, the ORDBMS allows us to store an entity as a complete object into a single column of a relation. As can be seen in the example (figure 5.6), the ensemble position of ADCP data as the spatial data is stored in the same table with its complex attributes of water current velocity profiles, each of which in a single column. This is a new approach to spatial data management that leads to the creation of simple tables as well as simple database application.

The extensibility feature of ORDBMS, that allows us to create user-defined data types (UDT) and user-defined functions (UDF), enables us to perform most of the application tasks in the database server rather than in the application server. This situation may have an implication of creating a 'thin' database application and reducing the transaction load in the network that may lead to improve the application's performance. The following points summarise some new approaches that can be applied to marine spatial data management as well as any other spatial data management in general:

- The spatial data and its complex attributes can be stored as a complete object in a single column respectively, in the same relation. If there are more than one object of attributes, they can be stored in separate columns.
- Accessing each component member of the spatial data as well as its complex attribute can be performed using user-defined functions.
- Most of the spatial analysis tasks can be implemented in the database server instead of in the application server by creating user-defined functions that work with spatial data.

Chapter VI

Conclusion

The recently developed Object-Relational Database Management System (ORDBMS) technology has increased the ability of a database application, such as spatial data management and GIS, to handle complex data in a better way than its predecessor technology of Relational DBMS. The object-oriented features of ORDBMS enable users to model objects that correspond to entities in the 'real world'. In relation to marine spatial data management, the ORDBMS provides several benefits that allow us to store, manage and retrieve spatial data - as well as its associated complex attributes - effectively and efficiently.

The water current velocity profiles measured by Acoustic Doppler Current Profiler (ADCP) is one of the complex data types in the marine field that can be considered as spatial data since it has geographic position. The ORDBMS is an appropriate data management system to store and manage marine spatial such as ADCP data. The object-oriented features of ORDBMS provide some new approaches on marine spatial data management that may provide an effective and efficient method of data management.

The following are some new approaches that can be applied to marine spatial data management:

- In ORDBMS, an entity in the 'real world' can be stored as a complete object in a single column of a relation. There is no requirement to decompose an entity of complex object into atomic values as required by the First Normal Form (1NF) rule of normalisation process in the relational model. This provides an appropriate data modelling of an entity that corresponds to the 'real world' phenomena. In ADCP case, the ensemble position as well as its complex attributes of water current velocity profile are stored in a single column respectively.

- The spatial data (geographic position) and its attributes can be stored in the same table, each of which is stored in a single column. In ADCP data case, the ensemble position - as the spatial data - is stored in the same table with its attributes of water current velocity profile.
- Most of the spatial analysis tasks can be performed in the database server by creating user-defined functions that work with the spatial data, instead of performing the analysis in the client application. However, the same functions can also be created in the application program to provide users a freedom of performing analysis tasks either in the database server or in their application programs. In this project, 70 user-defined functions have been created to facilitate some analysis tasks in the database server as well as the application program. The Java programming language has been used to develop the application programs.

The new approaches of ORDBMS on marine spatial data management provides several benefits as summarized in the following:

- The object-oriented features of ORDBMS enables users to reduced the number of tables and columns that are required for managing marine spatial data. In ADCP database, one table of four columns is required to store the ensemble position and the water current velocity profile (the other two columns are for ensemble id and the foreign key of transect id respectively).

- Storing an entity as a complete object into a single column of a relation enables users to create a simple SQL query to retrieve and manipulate any complex data stored in the database.
- Storing the geographic position as the spatial data and its attributes in the same tables enables users to retrieve the spatial data in a single disk access, reducing many join operation between tables as found in the relational DBMS, which means reducing access time.
- The extensibility features of ORDBMS may lead to the creation of 'thin' database applications since most of the application tasks are performed in the database server via user-defined functions rather than in the application programs. This situation may also reduced the transaction load since the data to be returned to the users are the analysis result instead of the whole data to be analysed.

References

1. Bancilhon, F. (1996), *Object, Relational, Object-Relational & Relational-Object,* New York, SIGS Publication (www.sigs.com).
2. Basu, A., and Nalamotu, C. (1997), *Marine Geographic Information System for the Exclusive Economic Zone*, Marine Geodesy, 20: 255-265.
3. Bobbitt, Andra M., 1997, *GIS Analysis of Remotely Sensed and Field Observation Oceanographic Data,* Marine Geodesy, 20:153-161.
4. Chamberlin, Donald D., 1996, *Anatomy of An Object-Relational Database*, DB2 Online Magazine, Winter 1996 (www.db2mag.com).
5. Cokelet, E.D., 1996, *ADCP-Referenced Geostrophic Circulation in the Bering Sea Basin*, Journal of Physical Oceanography, July 1996.
6. Connolly, T., 1999, *Database Systems – A Practical Approach to Design, Implementation, and Management,* Addison Wesley Longman.
7. Cook, Rick, 1997, *Is a Hybrid Database in Your Future?,* Sun World Online, February 1997.
8. Date, C.J., 1998, *Back to the Relational Future*, DBSummit, September 1998.
9. Davis, Judith R., 1996, *Informix Universal Server - Extending the Relational Database Management System to Manage Complex Data*, Database Associate Int., November 1996.
10. Dick, Timothy, 1997, *Object/Relational Database Vendors Map Their Strategies*, PC Week, January 1997.
11. Dinger, Kari Richards, , *Lowered ADCP (LADCP) Application Note*, RD Instrument Research Report (www.adcp.com/resrep/ladcp/ladcp.htm).
12. Elmasri, R., and Navathe, S.B., 1994, *Fundamentals of Database Systems*, Addison Wesley Longman.
13. Embley, David W., 1998, *Object Database Development, Concepts and Principles*, Addison Wesley Longman, Inc.
14. Falconer, Robin K.H., 1990, *Experience with Geographic Information Systems (GIS) in the Marine World,* The Hydrographic Journal, 58:19-22.

15. Fowler, Cindy, 1998, *Geographic Information Systems, Mapping, and Spatial Data for the Coastal and Ocean Resource Management Community*, Surveying and Land Information Systems, 58:135-140.
16. Frazer, Donald, 1998, *Object/Relational Grows Up*, DB Summit, September 1998.
17. Fussel, Mark L., 1997, *Foundations of Object-Relational Mapping,* Chi Mu Publications (www.chimu.com)
18. Gold, Christopher M., 1995, *Spatial Data Structure Integrating GIS and Simulation in a Marine Environment,* Marine Geodesy, 18:213-228.
19. GreenWood, Len, 1995, *The PostRelational Database, The case for NF2*, Database and Network Journal, 25:3-5.
20. Grimes, Seth, 1998, *Modeling Object/Relational Databases,* DBMS Magazine, April 1998.
21. Grimes, Seth, 1998, *Object Relational Reality Check,* Database Programming and Design, July 1998.
22. Group, Gartner, 1995, *Universal Server: RDBMS Technology for the Next Decade*, Informix Corp., June 1995.
23. Group, Gartner, 1998, *Application Partitioning with Informix-New Era (TM),* Informix Corp.
24. Hall, Robert K., 1995, *Geographical Information Systems (GIS) to Manage Oceanographic Data for Site Designation and Site Monitoring,* Marine Geodesy, 18:161-171.
25. Humphreys, R.G., 1989, *Marine Information Systems,* The Hydrographic Journal, 54:19-21.
26. Informix, 1997, *An Introduction to Informix Universal Server Extended features Training Manual,* Informix.
27. Informix, 1998, *Developing DataBlade Modules for Informix Dynamic Server with Universal Data Option,* Informix.
28. Informix, *Extending & Enhancing ERP Applications by Using Complex Data Types*, Informix.

29. Informix, 1997, *Extending Informix Universal Server, Data Types version 9.1*, Informix.
30. Informix, 1997, *Informix-Enterprise Command Center, A Complete Enterprise Data Management Solution*, Informix.
31. Informix, 1997, *Informix Geodetic DataBlade Module, User's Guide version 2.1*, Informix.
32. Informix, 1996, *Informix and Illustra Merge to Create Informix Universal Server*, Informix Corp.
33. Informix, 1998, *Informix Object-Relational Database Leads the Industry with 1,000 Customers*, Informix Corp.
34. Keeler, Michael, 1997, *Treasure Maps for Decision Support*, Database Programming and Design, November 1997.
35. Kumar, S., and Nor, A., 1997, *Oracle8 Object Relational database: An Overview*, Oracle Corp., June 1997.
36. Li, Rongxing, 1993, *Development of an Integrated Marine Geographic Information System*, Marine Geodesy, 16:293-307.
37. Mason, J.Y.M., 1993, *Management of Hydrographic and Oceanographic Databases using GIS*, The Hydrographic Journal, 68:21-26.
38. McClure, Steve, 1997, *Object Database vs. Object-Relational Database*, IDC Bulletin, April 1997.
39. Miller, Glenn, 1999, *Informix Basics*, Prentice Hall.
40. OSMOS, *Object-Relational Technology and OSMOS*, Business White Paper, OSMOS Corp (www.osmos.com).
41. OSMOS, *OSMOS Object-Relational Database Management System*, Technical White Paper, OSMOS Corp (www.osmos.com).
42. RD Instruments, 1995, *User's Manual for the RD Instruments BBLIST.EXE Program (For Use with Broadband ADCP)*, RD Instruments Corp.
43. RD Instruments, 1994, *User's Manual for the RD Instruments Transect Program (For Use with Broadband ADCP)*, RD Instruments Corp.
44. RD Instruments, 1995, *User's Manual for the RD Instruments Play Back Program (For Use with Broadband ADCP)*, RD Instruments Corp.

45. RD Instruments, 1988, *RD-VM Acoustic Doppler Current Profilers, Operation and Maintenance Manual*, RD Instruments Corp.
46. Renhnhackkamp, Martin, 1997, *Extending Relational DBMSs*, DBMS Magazine, December 1997.
47. Rosenblatt, Bill, 1996, *Informix Leaps to Objects*, SunWorld, May 1996.
48. Rudin, Ken, 1997, *Are Object/Relational Databases Scalable*, PCWeek, June 1997.
49. Rudin, Ken, 1997, *Quantifying Scalability*, PC Week, November 1997.
50. Sanchez, Angela, 1997, *Informix Dynamic Server with Universal Data Option: Best Practices*, Prentice Hall.
51. Sontek, *ADP Principles of Operation*, Sontek Corp.
52. Stephens, R., 1997, *Current Measurements and Forecasting for Deep Water Site Investigation*, The Hydrographic Journal, 84:21-25.
53. Stonebraker, M., 1997, *Limitations of Spatial Simulators for Relational DBMSs*, Informix Corp.
54. Stonebraker, M., 1998, *Object-Relational DBMS, The Next Wave*, Informix Corp.
55. Stonebraker, M., 1996, *Object-Relational DBMSs, The Next Great Wave*, Morgan Kaufmann.
56. Stonebraker, M., 1997, *Performance Penalties for Simulating Object-Relational DBMSs*, Informix Corp.
57. Taylor, Art, 1999, *Informix Power Reference*, Prentice Hall.
58. Weber, J.L., 1998, *Using Java™2 Platform, Special Edition*, Que Publishing.
59. White, Seth., 1999, *JDBC™ API Tutorial and Reference, Second Edition – Universal Data Access for the Java™2 Platform*, Addison Wesley Longman.
60. Wilson, John T., 1996, *Bathymetric Surveys of Morse and Geist Reservoirs in Central Indiana Made with Acoustic Doppler Current Profiler and Global Positioning Systems Technology*, U.S. Geological Survey.

Appendix A

List of Entity Types, Relationship Types and Attributes of Entity Types Used in ADCP Database

A.1. List of entity types.

Entity name	Description	Occurrence
Institutes	• Institution that performed ADCP measurement. • Vessels owner	• An institution may perform one or more projects of ADCP measurement survey. • An institution may own one or more vessels
Projects	ADCP measurement survey	One project may consists of one or more cruise lines within particular area boundary
Cruises	trips within a project	One project may have one or more cruises
Vessels	The vessel used for ADCP measurement	One vessel may be used to performed one or more ADCP measurement survey
Countries	The country where institution is located	An institution may be located in one or more countries
Instrumentations	Instrumentation used in ADCP measurement either ADCP or Navigational instruments	One ADCP measurement uses one type both ADCP instrument and navigation instrument
Transects	Cruise line where ADCP measurement is performed	• A transect has a series of discreet ensembles. • A transect has one configuration parameter for all ensembles within it.
Ensembles	ADCP measured value	An ensemble contains one date and time of measurement, one measurement information (EnsembleInfo) and a series of bins (EnsembleBins)
EnsembleInfo	Measurement information contained in each ensemble.	One ensemble contains one measurement information.
EnsembleBins	A series of bins within ensembles	Each ensemble has one up to 128 bins
EnsembleConfig	Measurement configuration	Ensemble within a transect has the same configuration parameters

A.2. List of Attributes for each Entity Types.

Table Name	Attribute Names	Data Type	Constraints

Countries	co_id
	co_name
		Varchar(10)
		Varchar(30)
			Not null primary key

Institutes	inst_id
	inst_name
	Address
	co_id
		Varchar(10)
		Varchar(50)
		Address_t
		Varchar(10)
			not null primary key
			
			foreign key references countries

Persons	pers_id pers_name inst_id	serial Name_t Varchar(10)	Not null primary key Foreign key references institutes
Projects	p_id p_name p_area survey_period co_id	serial Varchar(50) GeoPolygon GeoTimeRange Varchar(10)	Not null primary key foreign key references countries
Collaboration	Coll_id Inst_id P_id	Serial Varchar(10) Integer	Not null primary key Foreign key references institutes Foreign key references projects

Vessel	vs_id
	vs_name
	inst_id
	vs_photo
		Varchar(10)
		Varchar(40)
		Varchar(10)
		BLOB
			not null primary key
			
			foreign key references institutes

Cruises	cr_id
	cr_name
	vs_id
	s_port
	e_port
	p_id
	serial_no
		Varchar(10)
		Varchar(50)
		Varchar(10)
		Varchar(50)
		Varchar(50)
		Integer
		Varchar(10)
			Not null primary key constraint pk_cruises
			
			Foreign key references vessels
			
			Foreign key reference projects
			Foreign key reference instrumentation

Transects	tr_id tr_config cr_id pers_id tr_notes	Serial Ensconfig_t Varchar(10) Varchar(10) CLOB	not null primary key foreign key references cruises foreign key references Persons. foreign key references processing
Ensembles	ens_id ens_position ens_data tr_id	Serial8 GeoPoint Ensemble_t integer	not null primary key R-tree index not null foreign key references transects

Instrumentation	Serial_no
	Instrument_name
	Manufacturer
	Model
	Firmware_number
		Varchar(10)
		Varchar(30)
		Varchar(30)
		Varchar(10)
		Varchar(10)
			Not null primary key

ADCP	transmit_freq Transducer_config Beam_angle Beam_width	Smallint Varchar(10) Smallint Smallint	
Navigation	Differential P_code Antena_offset	Boolean Boolean Offset_t	

A.3. List of Relationship Types.

Entity Type	Relationship Type	Entity Type	Cardinality Ratio	Participation
Institutes	• own • Located in • Perform	• Vessels • Countries • Projects	1 : M 1 : M M : N	P : T T : P P : T
Projects	Undertake	Cruises	1 : M	T : P
Cruises	• Equipped • Consist of	• Instrumentations • transects	M : N 1 : M	T : P T : T
Vessels	Used in	Cruises	1 : M	T : T
Transects	Has	ensembles	1 : M	T : P
Persons	• Work at • Lead • Process	• Institutes • Projects • Transects	M : 1 1 : M 1 : M	P : T P : T P : T

Notes : P = partial participation, T = Total participation.

Appendix B

Data Definition Languages Of User-defined Data Types (UDT) and Relations For ADCP Database in Informix Dynamic Server With Universal Data Option version 9.14

B.1. Data Definition Languages of User-defined Data Types (UDT)

UDT Name	Data Definition Language

Velocity_t	`create row type velocity_t (`
	`    EastVelocity    smallfloat,`
	`    NorthVelocity   smallfloat);`

bin_t	`create row type bin_t (`
	`    BinDepth        smallfloat,`
	`    BinVelocity     velocity_t,`
	`    Avqc            smallfloat,`
	`    PctOK           Integer);`

Ensinfo_t	`create row type ensinfo_t (`
	`    AvgCovered      Integer,`
	`    ShipVelocity    velocity_t,`
	`    NavType         Varchar(3),`
	`    BCover          Integer,`
	`    BottomDepth     Integer,`
	`    BTError         Integer,`
	`    PctGood         Integer,`
	`    IntegrationPeriod  Integer);`

ensconfig_t	`create row type ensconfig_t (`
	`    Draught         smallfloat,`
	`    BinsToSample    Integer,`
	`    BinLength       Integer,`
	`    PulseLength     Integer,`
	`    TransmitDelay   Integer,`
	`    Refon           Integer,`
	`    Refb1           Integer,`
	`    Refb2           Integer,`
	`    EvMax           smallfloat,`
	`    WMax            smallfloat);`

ensbins_t	`create row type ensbins_t (`
	`    Bins            LIST(bin_t not null));`

UDT Name	Data Definition Language
ensembles_t	create row type ensemble_t (EnsDateTime Datetime year to second, EnsInfo ensinfo_t, EnsBins ensbins_t);
address_t	create row type address_t (street Varchar(40), city Varchar(20), state Varchar(3), phone Varchar(15), fax Varchar(15), zip Varchar(4));

B.2. Data Definition Languages of Relations

Table Name	Data Definition Language
Countries	create table countries (co_id Varchar(10) not null primary key constraint pk_countries, co_name Varchar(30));
Institutes	create table institutes (inst_id Varchar(10) not null primary key constraint pk_inst, inst_name Varchar(50), address address_t, co_id Varchar(10) references countries constraint inst_fk_co);
Persons	create table persons (pers_id Serial not null primary key constraint pk_persons, pname name_t inst_id Varchar(10) references institutes constraint pers_fk_inst);
Projects	create table projects (p_id Serial not null primary key constraint pk_projects, p_name Varchar(50), p_area GeoPolygon, survey_period GeoTimeRange, co_id Varchar(10) references countries constraint p_fk_co);
Collaboration	Create table collaboration (Coll_id Serial notu null primary key, Inst_id Varchar(10) references Institutes, P_id Integer references Projects);

Table Name	**Data Definition Language**
Vessels	create table vessels (vs_id Varchar(10) not null primary key constraint pk_vessels, vs_name Varchar(40), inst_id Varchar(10) references institutes constraint vs_fk_inst, vs_photo BLOB);";
Cruises	create table cruises (cr_id Varchar(10) not null primary key constraint pk_cruises, cr_name Varchar(50), vs_id Varchar(10) references vessels constraint cr_fk_vs, s_port Varchar(50), e_port Varchar(50), pers_id Varchar(10) references person constraint cr_fk_proc);";
Instrumentation	Create table instrumentation of type instrument_t Primary key (serial_no);
ADCP	Create table adcp of type adcp_t Under instrumentation;
Navigation	Create table navigation of type navigation_t Under instrumentation;
Transects	create table transects (tr_id serial not null primary key constraint pk_transects, config ensconfig_t, cr_id Varchar(10) references cruises constraint tr_fk_cr, pers_id integer references persons constraint tr_fk_pers, tr_notes CLOB);
Ensembles	create table ensembles (ens_id serial8 not null primary key constraint pk_ensembles, ens_position GeoPoint, ens_data ensemble_t, tr_id integer references transects constraint ens_fk_tr);";

Appendix C

List of Accessor Functions in ADCP Database

The following are the accessor functions associated with ADCP database. These functions can be used to access each element, parts of or the whole ensemble object. The functions can be run as part of SQL Queries either from database specific application (such as DBAccess) or any external programming languages. In this project, the functions are run from java applications programs.

Return Value	Function Name
`Int`	**avgCovered**(*ei* ensinfo_t) Get the percentage of averaging period covered by acceptable ensemble from set of measurement information data.
`Int`	**avgCovered**(*ens* ensemble_t) Get the percentage of averaging period covered by acceptable ensemble from contained in each ensemble.
`Smallfloat`	**avqc**(*b* bin_t) Get the quality control value of this bin.
`Smallfloat`	**avqc**(*ens* ensemble_t, *d* smallfloat) Get the quality control value of this ensemble at d depth bin.
`Bin_t`	**binAtDepth**(*eb* ensbins_t, *d* smallfloat) Get the bin object at the specified depth.
`Bin_t`	**binAtDepth**(*ens* ensemble_t, *d* smallfloat) Get the bin object at the specified depth contained in this ensemble.
`Smallfloat`	**binDepth**(*b* bin_t) Get the bin depth of this bin.
`Int`	**binInterval**(*eb* ensbins_t) Get the bin interval in this ensemble bins object, in vertical metre.
`Int`	**binInterval**(*ens* ensemble_t) Get the bin interval in tthis ensemble, in vertical metre.
`Int`	**binLength**(*ec* ensconfig_t) Get the length of bin in vertical metres.
`Int`	**binSize**(*eb* ensbins_t) Get the number of bins contained in the ensemble associated with this ensemble bins object.
`Int`	**binSize**(*ens* ensemble_t) Get the number of bins contained in this ensemble.
`Int`	**binsToSample**(*ec* ensconfig_t) Get the number of bins to sample of each ensemble.
`Int`	**bottomCover**(*ei* ensinfo_t) Get the percentage of interfix period for which there was bottom depth information.

Return Value	Function Name
Int	**bottomCover**(*ens* ensemble_t) Get the percentage of interfix period for which there was bottom depth information.
Int	**bottomDepth**(*ei* ensinfo_t) Get the mean bottom depth of the ensemble associated with this measurement information, for which a bottom depth was available.
Int	**bottomDepth**(*ens* ensemble_t) Get the mean bottom depth of this ensemble, for which a bottom depth was available.
Int	**btError**(*ei* ensinfo_t) Get the bottom track error velocity of the ensemble associated with this measurement information (used for bottom track correction).
Int	**btError**(*ens* ensemble_t) Get the bottom track error velocity of this ensemble (used for bottom track correction).
Smallfloat	**draught**(*ec* ensconfig_t) Get the transducer depth correction in meter.
Smallfloat	**eastShipVelocity**(*ei* ensinfo_t) Get the east velocity component for this ensemble from measurement information.
Smallfloat	**eastShipVelocity**(*ens* ensemble_t) Get east velocity component contained in this ensemble.
Smallfloat	**eastVelocity**(*b* bin_t) Get the east current velocity component contained in this bin object, relative to the ship.
Smallfloat	**eastVelocity**(*ens* ensemble_t, *d* smallfloat) Get the east current velocity component contained in this ensemble object at d bin depth, relative to the ship.
EnsBins_t	**ensBins**(*ens* ensemble_t) Get a series of bins data contained in each ensemble.
DateTime Year to Second	**ensDateTime**(*ens* ensemble_t) get date and time of measurement of each ensemble.
Int	**ensDay**(*ens* ensemble_t) get date of measurement of each ensemble.
EnsInfo_t	**ensInfo**(*ens* ensemble_t) Get the measurement information for each ensemble.
Int	**ensMonth**(*ens* ensemble_t) Get month of measurement of each ensemble.
Int	**ensYear**(*ens* ensemble_t) Get year of measurement of each ensembles.
Smallfloat	**evMax**(*ec* ensconfig_t) Get the threshold for ping by ping data rejection on basis of error velocities, in m/s.
Bin_t	**firstBin**(*eb* ensbins_t) Get the first bin of this ensemble bins object.
Bin_t	**firstBin**(*ens* ensemble_t) Get the first bin of of the ensemble bins object in this ensemble.

Return Value	Function Name
Bin_t	**interpolateBin**(*b1* bin_t, *b2* bin_t, *d* smallfloat) Interpolate a bin object at the specified depth.
Bin_t	**lastBin**(*eb* ensbins_t) Get the last bin of this ensemble bins object.
Bin_t	**lastBin**(*ens* ensemble_t) Get the first bin of the ensemble bins object in this ensemble.
Float	**latitude**(*position* GeoPoint) Get the latitude of ensemble position.
Float	**longitude**(*position* GeoPoint) Get the longitude of ensemble position.
Varchar	**navType**(*ei* ensinfo_t) Get the navigation type used in this ensemble from measurement information.
Varchar	**navType**(*ens* ensemble_t) Get the navigation type used in this ensemble measurement.
Smallfloat	**northShipVelocity**(*ei* ensinfo_t) Get the north velocity component for this ensemble from measurement information.
Smallfloat	**northShipVelocity**(*ens* ensemble_t) Get north velocity component contained in this ensemble.
Smallfloat	**northVelocity**(*b* bin_t) Get the north current velocity component contained in this bin object, relative to the ship.
Smallfloat	**northVelocity**(*ens* ensemble_t, *d* smallfloat) Get the north current velocity component contained in this ensemble object at d bin depth, relative to the ship.
Int	**pctOK**(*b* bin_t) Get the "attendance percentage", or the percentage of the profile period for which there was good data in this bin.
Smallfloat	**pctOK**(*ens* ensemble_t, *d* smallfloat) Get the "attendance percentage", or the percentage of the profile period for which there was good data in this ensemble at d bin depth.
Int	**pulseLength**(*ec* ensconfig_t) Get the length of pulse used in the ADCP measurement, in vertical metres.
Velocity_t	**shipVelocity**(*ei* ensinfo_t) Get the ship velocity object for this ensemble from measurement information.
Velocity_t	**shipVelocity**(*ens* ensemble_t) Get the ship velocity object contained in this ensemble.
Smallfloat	**shipVelocityDir**(*ei* ensinfo_t) Get the ship velocity direction from measurement information.
Smallfloat	**shipVelocityDir**(*ens* ensemble_t) Get the ship velocity direction for this ensemble.
Smallfloat	**shipVelocityDir**(*vel* velocity_t) Get the ship velocity direction from velocity object.
Smallfloat	**shipVelocityMag**(*ei* ensinfo_t) Get the ship velocity magnitude for this ensemble from measurement information.
Smallfloat	**shipVelocityMag**(*ens* ensemble_t) Get the ship velocity magnitude for this ensemble.

Return Value	Function Name
`Smallfloat`	**shipVelocityMag**(*vel* velocity_t) Get the ship velocity magnitude from velocity object.
`Int`	**transmitDelay**(*ec* ensconfig_t) Get the delay after transmit in vertical metres.
`Smallfloat`	**velocityDir**(*b* bin_t) Get the current velocity direction of the velocity object contained in this bin object, relative to the ship.
`Smallfloat`	**velocityDir**(*ens* ensemble_t, *d* smallfloat) Get the current velocity direction of the velocity object contained in this ensemble object at d bin depth, relative to the ship.
`Smallfloat`	**velocityMag**(*b* bin_t) Get the current velocity magnitude of the velocity object contained in this bin object, relative to the ship.
`Smallfloat`	**velocityMag**(*ens* ensemble_t, *d* smallfloat) Get the current velocity magnitude of the velocity object contained in this ensemble object at d bin depth, relative to the ship.
`Smallfloat`	**wMax**(*ec* ensconfig_t) Get vertical velocity real time threshold, in m/s.

Appendix D

List of Java Class and Methods for ADCP Database

D.1. Ensemble Class

Constructors:

Ensemble()   Construct an empty ensemble object.
Ensemble(EnsembleInfo *ensinfo*, EnsembleBins *ensbins*)   Construct an ensemble from measurement information and a series of bins measurement data.

Methods:

Return Value	Function Name
`EnsembleInfo`	**ensInfo**() Get the measurement information for each ensemble.
`EnsembleBins`	**ensBins**() Get a series of bins data contained in each ensemble.
`TimeStamp`	**ensDateTime**() get date and time of measurement of each ensemble.
`int`	**ensYear**() Get year of measurement of each ensembles.
`int`	**ensMonth**() Get month of measurement of each ensemble.
`int`	**ensDay**() get date of measurement of each ensemble.
`int`	**binsSize**() get number of bins contained in this ensemble.

D.2. EnsembleConfig Class

Constructors:

EnsembleConfig() Construct an empty configuration parameter object.
EnsembleConfig(String *header*) Construct a configuration parameter object from header data of CSIRO ADCP data file.

Methods:

Return Value	Function Name
`float`	**draught()** Get the transducer depth correction in meter.
`int`	**binsToSample()** Get the number of bins to sample of each ensemble.
`int`	**binLength()** Get the length of bin in vertical metres.
`int`	**pulseLength()** Get the length of pulse used in the ADCP measurement, in vertical metres.
`int`	**transmitDelay()** Get the delay after transmit in vertical metres.
`float`	**evMax()** Get the threshold for ping by ping data rejection on basis of error velocities, in m/s.
`float`	**wMax()** Get vertical velocity real time threshold, in m/s.

D.3. EnsembleInfo Class

Constructors:

EnsembleInfo() Construct an empty measurement information object.
EnsembleInfo(String *info*) Construct a measurement information object from CSIRO ADCP data file.

Methods:

Return Value	Function Name
int	**avgCovered()** Get the percentage of averaging period covered by acceptable ensemble from set of measurement information data.
Velocity	**shipVelocity()** Get the ship velocity object contained in this ensemble.
float	**eastShipVelocity()** Get east velocity component contained in this ensemble.
float	**northShipVelocity()** Get north velocity component contained in this ensemble.
float	**shipVelocityMag()** Get the ship velocity magnitude for this ensemble.
float	**shipVelocityDir()** Get the ship velocity direction for this ensemble.
String	**navType()** Get the navigation type used in this ensemble from measurement information.
int	**bottomCover()**
int	**bottomDepth()** Get the mean bottom depth of this ensemble, for which a bottom depth was available.
int	**btError()** Get the bottom track error velocity of this ensemble (used for bottom track correction).

D.4. EnsembleBins Class

Constructors:

EnsembleBins() Construct an empty ensemble bins object.
EnsembleBins(Vector *binsdata*) Construct a series of bins ensemble from bins data of CSIRO ADCP data file.
EnsembleBins(EnsembleConfig *ensconfig*, Vector *binsdata*) Construct a series of bins ensemble from bins data of CSIRO ADCP data file with the specified configuration parameter.

Methods:

Return Value	Function Name
int	**binSize()** Get the number of bins contained in this ensemble.
int	**binInterval()** Get the bin interval in this ensemble, in vertical metre.
Bin_t	**firstBin()** Get the first bin of the ensemble bins object in this ensemble.
Bin_t	**lastBin()** Get the first bin of the ensemble bins object in this ensemble.
Bin_t	**binAtDepth(**float *depth*) Get the bin at the specified depth in this ensemble.

D.5. Bin Class

Constructors:

Bin() Construct an empty bin object.
Bin(float *eastvelocity*, float *northvelocity*, float *avqc*, int *pctOK*) Construct a bin from measured current velocity components, quality control value and attendance percentage as float data input, with unspecified bin depth.
Bin(float *depth*, float *eastvelocity*, float *northvelocity*, float *avqc*, int *pctOK*) Construct a bin from measured current velocity components, quality control value and attendance percentage as float data input at specified bin depth.
Bin(String *eastvelocity*, String *northvelocity*, String *avqc*, String *pctOK*) Construct a bin from measured current velocity components, quality control value and attendance percentage as String data input, with unspecified bin depth.
Bin(String *depth*, String *eastvelocity*, String *northvelocity*, String *avqc*, String *pctOK*) Construct a bin from measured current velocity components, quality control value and attendance percentage as String data input at specified bin depth.

Methods:

`Return Value`	**Function Name**
`float`	**eastVelocity**() Get the east current velocity component contained in this bin object, relative to the ship.
`float`	**northVelocity**() Get the north current velocity component contained in this bin object, relative to the ship.
`float`	**velocityMagnitude**() Get the current velocity magnitude of the velocity object contained in this bin object, relative to the ship.
`float`	**velocityDirection**() Get the current velocity direction of the velocity object contained in this bin object, relative to the ship.
`float`	**binDepth**() Get the bin depth of this bin.
`float`	**avqc**() Get the quality control value of this bin.
`int`	**pctOK**() Get the "attendance percentage", or the percentage of the profile period for which there was good data in this bin.

D.6. Velocity Class

Constructors:

Velocity(float *eastvelocity*, float *northvelocity*) Construct a velocity object from each velocity components as float data inputs.
Velocity(String *eastvelocity*, String *northvelocity*) Construct a velocity object from each velocity components as String data inputs.

Methods:

Return Value	Function Name
float	**eastVelocity**() Get the east velocity component contained in this velocity object.
float	**northVelocity**() Get the north velocity component contained in this velocity object.
float	**velocityMagnitude**() Get the velocity magnitude of the velocity object.
float	**velocityDirection**() Get the velocity direction of the velocity object.

D.7. GeoAltRange

Constructors:

GeoAltRange() Construct an empty GeoAltRange object.
GeoAltRange(double *h*) Construct a GeoAltRange from height value.
GeoAltRange(double *lo*, double *hi*) Construct a GeoAltRange object from lowest and highest height values.

Methods:

Return Value	Function Name
double	**AltLow**() Get the lowest altitude value from GeoAltRange.
double	**AltHigh**() Get the highest altitude value from GeoAltRange.

D.8. GeoCoords

Constructors:

GeoCoords() Cinstruct an empty GeoCoords object
GeoCoords(double *lat*, double *lon*) Construct a GeoCoords object from latitude and longitude as double values.
GeoCoords(String *lat*, String *lon*) Construct a GeoCoords object from latitude and longitude as String values.

Methods:

Return Value	Function Name
`void`	**setGeoLatitude**(double *lat*) Set the GeoLatitude of GeoCoords to lat.
`void`	**setGeoLongitude**(double *lon*) Set the GeoLongitude of GeoCoords to lon.
`double`	**getGeoLatitude**() Get the GeoLatitude from GeoCoords object.
`double`	**getGeoLongitude**() Get the GeoLongitude from GeoCoords object.

D.9. GeoTimeRange

Constructors:

GeoTimeRange() Construct an empty GeoTimeRange object.
GeoTimeRange(long *t*) Construct GeoTimeRange object using a millisecond value.
GeoTimeRange(long *tlo*, long *thi*) Construct GeoTimeRange object from lowest and highest time value in millisecond
GeoTimeRange(String *t*) Construct GeoTimeRange object using a String value.
GeoTimeRange(String *tlo*, String *thi*) Construct GeoTimeRange object from lowest and highest time value in String

Methods:

Return Value	Function Name
`String`	**timeLow**() Get the Lowest Time as String from GeoTimeRange object.
`String`	**timeHigh**() Get the Highest Time as String from GeoTimeRange object.
`long`	**timeLowValue**() Get the Lowest Time value as millisecond from GeoTimeRange object.
`long`	**timeHighValue**() Get the Highest Time Value as millisecond from GeoTimeRange object.

S34. R. C. Mullin, "Data Update in a Land Information Network", Unisurv S34, 168 pp, 1988.

S35. B. Merminod, "The Use of Kalman Filters in GPS Navigation", Unisurv S35, 203 pp, 1989.

S36. A. R. Marshall, "Network Design and Optimisation in Close Range Photogrammetry", Unisurv S36, 249 pp, 1989.

S37. W. Jaroondhampinij, "A model of Computerised Parcel-Based Land Information System for the Department of Lands, Thailand," Unisurv S37, 281 pp, 1989.

S38. C. Rizos (Ed.), D. B. Grant, A. Stolz, B. Merminod, C. C. Mazur "Contributions to GPS Studies", Unisurv S38, 204 pp, 1990.

S39. C. Bosloper, "Multipath and GPS Short Periodic Components of the Time Variation of the Differential Dispersive Delay", Unisurv S39, 214 pp, 1990.

S40. J. M. Nolan, "Development of a Navigational System Utilizing the Global Positioning System in a Real Time, Differential Mode", Unisurv S40, 163 pp, 1990.

S41. R. T. Macleod, "The Resolution of Mean Sea Level Anomalies along the NSW Coastline Using the Global Positioning System", 278 pp, 1990.

S42. D. A. Kinlyside, "Densification Surveys in New South Wales - Coping with Distortions", 209 pp, 1992.

S43. A. H. W. Kearsley (Ed.), Z. Ahmad, B. R. Harvey and A. Kasenda, "Contributions to Geoid Evaluations and GPS Heighting", 209 pp, 1993.

S44. P. Tregoning, "GPS Measurements in the Australian and Indonesian Regions (1989-1993)", 134 + xiii pp, 1996.

S45. W.-X. Fu, "A Study of GPS and Other Navigation Systems for High Precision Navigation and Attitude Determinations", 332 pp, 1996.

S46. P. Morgan et al, "A Zero Order GPS Network for the Australia Region", 187 + xii pp, 1996.

S47. Y. Huang, "A Digital Photogrammetry System for Industrial Monitoring", 145 + xiv pp, 1997.

S48. K. Mobbs, "Tectonic Interpretation of the Papua New Guinea Region from Repeat Satellite Measurements", 256 + xc pp, 1997.

S49. S. Han, "Carrier Phase-Based Long-Range GPS Kinematic Positioning", 185 + xi pp, 1997.

S50. M. D. Subari, "Low-cost GPS Systems for Intermediate Surveying and Mapping Accuracy Applications", 179 + xiii pp, 1997.

S51. L.-S. Lin, "Real-Time Estimation of Ionospheric Delay Using GPS Measurements", 199 + xix pp, 1997.

S52. M. B. Pearse, "A Modern Geodetic Reference System for New Zealand", 324 + xviii pp, 1997.

S53. D. B. Lemon, "The Nature and Management of Positional Relationships within a Local Government Geographic Information System", 273 + xvi pp, 1997.

S54. C. Ticehurst, "Development of Models for Monitoring the Urban Environment Using Radar Remote Sensing", 282 + xix pp, 1998.

S55. S. S. Boey, "A Model for Establishing the Legal Traceability of GPS Measurements for Cadastral Surveying in Australia", 186 + xi pp, 1999.

S56. P. Morgan and M. Pearse, "A First-Order Network for New Zealand", 134 + x pp, 1999.

S57. P. N. Tiangco, "A Multi-Parameter Radar Approach to Stand Structure and Forest Biomass Estimation", 319 + xxii pp, 2000.

S58. M. A. Syafi'i, "Object–Relational Database Management Systems (ORDBMS) for Managing Marine Spatial Data: ADCP Data Case Study", 123 + ix pp, 2000.

MONOGRAPHS

Australian prices include postage by surface mail.
Overseas prices include delivery by UNSW's air-lifted mail service (~2-4 weeks to Europe and North America).
Rates for air mail rates through Australia Post on application.
Australian prices include GST.

(Prices effective September 2000)

		Price Australia (incl. GST)	**Price** Overseas
M1.	R. S. Mather, "The Theory and Geodetic Use of some Common Projections", (2nd edition), 125 pp, 1978.	$ 16.50	$ 15.00
M2.	R. S. Mather, "The Analysis of the Earth's Gravity Field", 172 pp, 1971.	$ 8.80	$ 8.00
M3.	G. G. Bennett, "Tables for Prediction of Daylight Stars", 24 pp, 1974.	$ 5.50	$ 5.00
M4.	G. G. Bennett, J. G. Freislich & M. Maughan, "Star Prediction Tables for the Fixing of Position", 200 pp, 1974.	$ 8.80	$ 8.00
M8.	A. H. W. Kearsley, "Geodetic Surveying", 96 pp, 1988.	$ 13.20	$ 12.00
M11.	W. F. Caspary, "Concepts of Network and Deformation Analysis", 183 pp, 2000.	$ 27.50	$ 25.00
M12.	F. K. Brunner, "Atmospheric Effects on Geodetic Space Measurements", 110 pp, 1988.	$ 17.60	$ 16.00
M13.	B. R. Harvey, "Practical Least Squares and Statistics for Surveyors", (2nd edition, reprinted with corrections), 319 pp, 1998.	$ 33.00	$ 30.00
M14.	E. G. Masters and J. R. Pollard (Eds.), "Land Information Management", 269 pp, 1991. (Proceedings LIM Conference, July 1991).	$ 22.00	$ 20.00
M15/1	E. G. Masters and J. R. Pollard (Eds.), "Land Information Management - Geographic Information Systems - Advance Remote Sensing Vol. 1", 295 pp, 1993 (Proceedings of LIM & GIS Conference, July 1993).	$ 33.00	$ 30.00
M15/2	E. G. Masters and J. R. Pollard (Eds.), "Land Information Management - Geographic Information Systems - Advance Remote Sensing Vol. 2", 376 pp, 1993 (Proceedings of Advanced Remote Sensing Conference, July 1993).	$ 33.00	$ 30.00
M16.	A. Stolz, "An Introduction to Geodesy", 112 pp, 1994.	$ 22.00	$ 20.00
M17.	C. Rizos, "Principles and Practice of GPS Surveying", 565 pp, 1997.	$ 46.20	$ 42.00

Credit Card Orders / Payment

from / to

School of Geomatic Engineering

The University of New South Wales

Name and postal address of ordering person/organisation

____

____

____

____

____

Date : ______________

Day time phone number : __________________

Credit Card debit for

	$
	$
	$
	$
Total	$

Please debit my credit card: ☐ Mastercard ☐ Visa ☐ Bankcard

Card Name: __________________________ Signature: __________________________

Card No:

Expire Date : __________

Note: UNSW requires that you attach a photocopy of your credit card to this order