공학석사 학위논문

# 선박 내 정보의 통합관리를 위한 다단 데이터베이스 설계 및 구현

A Design and Implementation of Multistage Database for Integrated Information Management on Shipboard

지도교수 박 휴 찬

2011년 2월


한국해양대학교 대학원

컴 퓨 터 공 학 과

서 정 민

KOREA MARITIME UNIVERSITY
1945
한국해양대학교

공학석사 학위논문

# 선박 내 정보의 통합관리를 위한 다단 데이터베이스 설계 및 구현

A Design and Implementation of Multistage Database for Integrated Information Management on Shipboard

지도교수 박 휴 찬

2011년 2월


한국해양대학교 대학원

컴 퓨 터 공 학 과

서 정 민

本 論文을 徐正民의 工學碩士 學位論文으로 認准함.

위원장 이 장 세 인

위 원 이 서 정 인

위 원 박 휴 찬 인

2010년 12월 24일

한 국 해 양 대 학 교 대 학 원

# 목 차

# 그림 목차

# 표 목차

# A Design and Implementation of Multistage Database for Integrated Information Management on Shipboard

Jeong-Min Seo

*Department of Computer Engineering*
*Graduate School of*
*Korea Maritime University*

Abstract

Ships are equipped with several kinds of navigation equipments which generate huge amount of data. Although these data have been mainly used for the safe navigation of ships, their usability can be enhanced if they are managed on a database. Because these data have different formats according to the kinds of equipments, however, an integrated database is required. This thesis presents a design and implementation of such database.

International Electro-technical Commission(IEC) is preparing the standard, IEC 61162-450, which specifies data exchange between shipboard equipments. This thesis adopts the standard as the basis of integrated information management. Data in the form of sentences specified in the standard, NMEA 0183 ver. 4.0, are collected from the shipboard networks, and then transformed and saved in a database. The database, in turn, can provide several information services to applications.

However, the database may become full soon, because huge amount of data are saved continuously. To cope with this problem, this thesis proposes a multistage database methodology which can always keep the database under

full by means of sampling mechanism. The data of the database are sampled, and then saved in another database, called second stage database. To prevent overflow of the database, some old data which already sampled can be deleted from the database. By this way, shipboard data are stably collected and managed, and then applied to the safe and efficient navigation of ships.

To verify the functionality of the methodology, a prototype system has been implemented and tested. It is expected to become a component of e-Navigation implementation in the future.

# 제 1 장 서 론

예로부터 선박은 대륙과 대륙 간의 이동을 위해서 중요한 교통수단이 되어 왔으며, 이러한 선박의 운항은 날씨 또는 방향과 같은 여러 가지의 요소에 의해 영향을 받는다. 따라서 이들 요소에 대해 예측을 하는 것은 중요하며 이를 위해서는 선박에 장착된 여러 장비들로부터 얻을 수 있는 각종 데이터가 필요하다[1]. 예를 들면, 바람의 방향 및 세기는 풍향·풍속계를 통하여 측정될 수 있으며 위도와 경도 데이터는 GPS를 통하여 측정될 수 있다. 이와 같이 각 장비에서 발생된 데이터는 실시간으로 이용할 수 있으며 이 데이터를 보존하면 모니터링 및 데이터 검색, 예측 시스템에 있어 유용하다[2,3].

이러한 목적에 따라 과거에는 데이터 보관을 위해서 각각의 데이터를 일지에 기록하였다. 그러나 이 방법은 사람에 의해 직접 기록되기 때문에 일정 간격으로 정확하게 데이터를 기록하기 쉽지 않으며 기록된 데이터의 신뢰도가 낮아질 수밖에 없다. 또한 전자 데이터를 통해 선박 제어, 자율 운항, 상황 발생 시 정보 제공 등의 기능을 할 수 있는 디지털 선박을 위한 장비가 개발되어 더 많은 데이터를 측정할 수 있게 되었다[4-6]. 데이터의 양이 급격히 증가함에 따라 사람이 기록하는 방법으로는 데이터 관리가 거의 불가능해졌다.

따라서 현재는 점차 전자적 데이터를 관리할 수 있는 방법들이 제안되고 있으며 이들 방법은 데이터베이스를 기반으로 하고 있다[7,8]. 하지만 지금까지 제안된 방법들은 사용자가 필요로 하는 소수의 특정 데이터들의 관리를 위한 데이터베이스를 설계하는 것이 일반적이며 또한 각 장비에서 발생되는 데이터마다 형태가 각각 다르기 때문에 통합적으로 관리하는 것이 쉽지 않다. 그러나 선박의 안전한 운항을 위해서는 장비에서 발생하는 데이터를 통합 관리할 필요가 있다.

국제해사기구(IMO : International Maritime Organization)의 81차 회의에서 제안된 e-Navigation에서는 사용자의 의사 결정, 작업 부하 관리, 안전 및 보안이 확보된 운항 등을 위하여 선박에서 발생하는 데이터의 통합 관리를 결정하였다[9]. 이에 따라 데이터 통합 관리 및 상호 교환할 수 있는 방법이 MiTS(Maritime information Technology Standard)에서 연구되어 왔다

[7,10-11]. 그러나 MiTS에서 최근까지 사용하던 표준인 IEC 61162-4 시리즈는 기능이 과다하고 처리과정이 너무 복잡하여 구현 및 적용 사례가 적다[12,13]. 따라서 IEC(International Electro-Technical Commission)에서는 현재 IEC 61162-4 시리즈를 대체하기 위하여 이더넷 기반의 IEC 61162-450을 제정하고 있으며 이 표준은 NMEA (National Marine Electronics Association) 0183 버전 4.0을 기반으로 한다[14,15]. 선박의 장비들로부터 발생된 데이터에 대하여 NMEA 0183 형식으로 변환된 데이터는 통합 관리를 위한 데이터베이스의 설계에 훨씬 효율적이다[16].

한편, 변환된 데이터가 존재하고 이에 따라 통합 관리를 위한 데이터베이스를 설계하더라도, 데이터가 계속 저장되어 일정량을 초과하게 되면 데이터베이스 서버는 자신의 역할을 제대로 수행할 수 없다. 이러한 문제를 해결하는 방법으로 데이터베이스 서버의 용량을 증가시키거나 데이터 삭제를 하는 방법이 있으며 두 가지 방법 모두 장단점이 있다.

우선 데이터베이스 서버의 용량을 증가시키는 방법은 보존된 데이터의 분실 없이 용량 초과를 해결할 수 있지만 용량 증가를 위한 비용과 용량 증가 횟수의 문제가 있다. 또한 서버 관리자가 수동적으로 관리를 해 주어야 하지만 선박에 승선하지 않으므로 관리가 힘들다. 한편, 데이터 삭제 방법은 일정 시간이 지난 데이터의 일정량을 지워 나가는 방법으로 데이터베이스 서버 용량 증가 방법에 비해 적은 비용과 전문가 없이 자동적으로 용량을 확보할 수 있는 장점이 있다. 그러나 유용한 데이터가 삭제될 수도 있으므로 이를 보완할 방법 또한 필요하다.

따라서 본 논문에서는 MiTS 내에서 NMEA 0183 센텐스 형식의 데이터를 통합 관리하는 다단 데이터베이스를 설계한다. 또한 다단 데이터베이스의 효율적인 관리를 위한 각종 모듈, 즉 데이터 관리를 위한 저장, 삭제, 샘플링 모듈과 이를 검색하기 위한 모듈을 구현하여 다단 데이터베이스의 성능을 시험한다.

본 논문의 구성은 우선 2장에서 통합 관리 데이터베이스의 설계의 바탕이 되는 MiTS 및 NMEA 0183 센텐스에 대하여 설명한다. 3장에서는 이를 기반으로 데이터베이스 및 각종 모듈에 대한 설계를 다룬다. 4장에서는 이를 구현 및 시험하며 5장에서 결론 및 향후 과제로 끝을 맺는다.

# 제 2 장　관련 연구

최근의 선박에서는 다양한 장비로부터 전자적인 데이터를 발생하며, 이를 활용한 많은 연구가 진행되고 있다. 본 논문에서 논하고자 하는 전자적 데이터의 통합 관리도 그 중 하나이다.

이를 위하여 본 장에서는 통합 관리 데이터베이스를 구축하기 위한 관련 연구로 MiTS와 NMEA 0183 센텐스에 대하여 살펴본다.

## 2.1 MiTS(Maritime information Technology Standard)

e-Navigation에서 데이터의 통합 관리 및 상호 교환의 역할을 하고 있는 MiTS가 채택하던 표준은 원래 IEC 61162-4 시리즈(400, 401, 402, 410, 420)이었다[12]. 그러나 IEC 61162-4 시리즈는 기능이 과다하고 처리과정이 너무 복잡하여 구현 및 적용 사례가 거의 없다. 따라서 MiTS에서는 새 표준으로 IEC에서 현재 제정 중인 IEC 61162-450을 채택하였다.[14]. 새롭게 채택된 이더넷 기반 표준인 IEC 61162-450에서는 NMEA 0183 버전 4.0을 기반으로 하고 있다[15].

한편, MiTS 시스템은 다수의 게이트웨이(gateway), 미들웨어(middleware), 응용시스템(application), 데이터베이스 서버(database server)로 나누어진다. 게이트웨이는 각 장비에서 발생하는 NMEA 2000 또는 NMEA 0183 형식의 데이터를 전송하며, 미들웨어는 이를 MiTS 형식으로 변환하여 전송한다[15,17]. 또한 응용 시스템 및 데이터베이스 서버는 이를 수신하여 각 용도에 맞게 처리한다[18].

그림 2.1은 MiTS 시스템의 구성을 나타낸다.

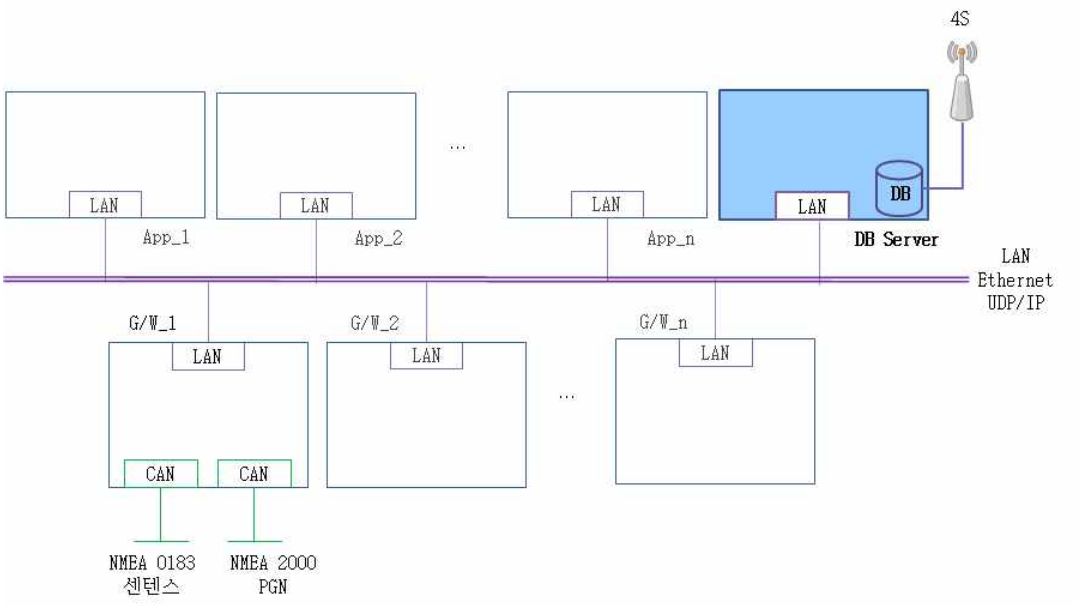


그림 2.1 MiTS 시스템의 구성

Fig. 2.1 MiTS system architecture

본 논문에서는 이와 같은 MiTS 시스템의 구성 중에서 진하게 표현된 데이터베이스 서버에 관한 내용에 대하여 논한다. MiTS 내에서 데이터베이스 서버는 데이터의 통합 관리와 상호 교환을 하는 역할을 하고 있다. 따라서 미들웨어를 통하여 전송된 데이터는 통합 관리를 위한 데이터베이스에 저장되며 이는 타 시스템에서도 다양한 용도로 활용 가능하다. 대표적인 예로, 선박과 선박 또는 선박과 육상 간의 통신을 담당하고 있는 4S(Ship to Ship, Ship to Shore) 시스템에서 선박의 주요 정보를 특정 기간에 보고하는 데 사용되는 noon report 가 있다[19]. 또한 응용 시스템 내에서도 필요에 따라 데이터베이스 서버에 접근하여 데이터를 활용 가능하다.

## 2.2 NMEA 0183 센텐스

NMEA 0183은 선박에 장착되는 각종 장비를 포함하여 해양에서 쓰일 수 있는 장비들의 전자적인 인터페이스를 정의하고 있다. 따라서 각종 장비로부터 발생되는 데이터에 대하여 정의를 하고 있으며, 이 표준에서는 데이터가 센텐스로 표현된다.

### 2.2.1 센텐스의 구조

센텐스는 몇 개의 필드(Field)로 구성되어 있으며 필드는 최소 1개 이상이어야 한다. 가장 앞의 필드는 주소 필드를 의미하며 Talker_id 및 센텐스 포맷터로 구성되어 있다. 나머지 필드는 고정된 길이의 값(fixed value), 수치 데이터(numeric value), 텍스트(text), 혹은 널(null) 값도 가능하다. 그러나 필드별 제약 조건 중에 널 값을 포함할 수 없다고 명시된 경우에는 해당 필드에 널 값을 쓸 수 없다.

센텐스는 최대 길이가 82 바이트로 이루어져 있으나 센텐스의 시작을 표시하는 “$” 혹은 “!”, 센텐스의 끝을 표시하는 <CR><LF>를 제외하면 최대 79 바이트를 사용 가능하다. 한 번에 하나의 센텐스만 전송할 수 있는 단일 센텐스의 경우에는 최대 79 바이트의 데이터만 전송 가능하다.

NMEA 0183에서 언급되는 센텐스의 형태는 표 2.1과 같은 형태로 이루어져 있다.

표 2.1 NMEA 0183 센텐스의 구조

Table 2.1 NMEA 0183 sentence structure

| 필드 | 설명 |
|---|---|
| “$” 또는 "!" | 센텐스 시작 |
| <주소 필드> | Talker_id 및 센텐스 포맷터 |
| [","<데이터 필드>]<br>…<br>[","<데이터 필드>] | 0개 이상의 데이터 필드 |
| "*"<체크섬 필드> | 체크섬 필드 |
| <CR><LF> | 센텐스의 끝 |

### 2.2.2 센텐스의 종류

NMEA 0183의 최근 버전인 버전 4.0에서의 센텐스 포맷터는 이전 버전에서 사용되었던 센텐스 포맷터를 포함하여 152개를 제시하고 있다. 각각의 센텐스 포맷터에 대한 자세한 내용은 NMEA 0183의 8, 9, 10장 및 부록 H를 통하여 언급하고 있다. 일반 목적의 센텐스(Approved General Sentences), AIS(Automatic

Identification System)에 관한 센텐스, AtoN(Aids To Navigation)에 관한 센텐스, 태그 블록 센텐스에 대하여 각 장에서 다루고 있으며 본 논문에서는 일반 목적으로 사용되는 센텐스에 대해서만 다룬다.

일반 목적으로 사용되는 센텐스는 102개의 센텐스 포맷터가 제시되어 있다. 이 중에는 수심 데이터를 나타내는 DPT, 바람의 풍향과 속도를 나타내는 MWD, 위경도 및 대지침로, 대지속도 등에 대한 정보를 갖는 RMC, 여러 변환기의 정보를 한 센텐스에 가지는 XDR 등이 있다. 한편, 같은 데이터 타입이라 하더라도 여러 센텐스 포맷터에 따로 포함될 수도 있고 하나의 센텐스 포맷터에 단위를 달리하여 포함될 수도 있다. 실제 대지침로와 대지속도의 경우에는 RMC 및 VTG에 모두 포함되어 있으며, MWD의 경우 풍향과 풍속의 데이터를 단위별로 가지고 있기도 하다. 그림 2.2는 본 논문에서 사용하는 센텐스 포맷터 중에서 DPT, MWD, XDR의 형식을 보여주고 있다.

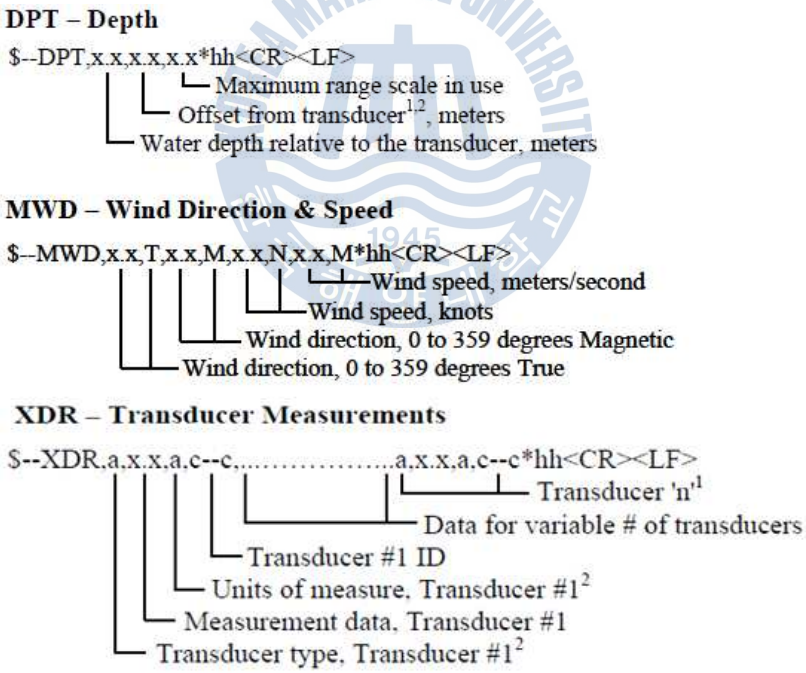


그림 2.2 NMEA 0183 센텐스 포맷터의 구조(DPT, MWD, XDR)

Fig. 2.2 Structure of sentence formatter(DPT, MWD, XDR)

그림 2.2에서 x.x는 소수를, T, M, N, A와 같은 문자는 각 단위 혹은 상태

표시를, c--c는 가변적 길이의 문자열을 의미한다. XDR의 경우 다른 센텐스 포맷터와는 달리 4개 필드 단위로 여러 변환기의 정보를 가질 수 있으며 변환기의 개수도 여러 개가 가변적으로 저장될 수 있다.

표 2.2는 이들 센텐스 포맷터의 예를 나타낸 것이고 표 2.3은 XDR 변환기의 타입과 단위에 대한 정보를 나열한 것이다.

표 2.2 각 센텐스 포맷터의 예(DPT, MWD, XDR)

Table 2.2 Example of sentence formatter(DPT, MWD, XDR)

| 포맷터 | 예시 |
|---|---|
| DPT | $SDDPT,12.345,11.111,23.45*16<CR><LF> |
| MWD | $YXMWD.11.123,T,33.333,M,22.222,N,44.444,M*35<CR><LF> |
| XDR | $YXXDR,C,12.333,C,Temperature,F,13.666,H,Frequency*41<CR><LF> |

표 2.3 XDR 타입 및 단위 정보

Table 2.3 XDR type & unit information

| 변환기 | 타입 필드 | 단위 필드 |
|---|---|---|
| 온도 | C | C |
| 각 변위 | A | D |
| 선 변위 | D | M |
| 주파수 | F | H |
| 힘 | N | N |
| 압력 | P | B, P |
| 유출률 | R | l |
| 유속 | T | R |
| 습도 | H | P |
| 부피 | V | M |
| 일반 | G | none(null) |
| 전류 | I | A |
| 전압 | U | V |
| 스위치 또는 밸브 | S | none(null) |
| 염분 | L | S |

표 2.3에 나열된 것은 기본적으로 NMEA 0183에서 정의된 것이며, 사용자의 필요에 따라 추가적으로 정의하여 사용할 수는 있으나 이미 정의된 것과 중복되지 않아야 한다.

# 제 3 장　다단 데이터베이스 서버 설계

MiTS의 다단 데이터베이스 서버에는 데이터의 통합 관리를 위한 다단 데이터베이스가 구축될 뿐만 아니라 데이터베이스에 저장된 데이터를 관리하는 각종 모듈이 같이 동작한다.

그림 3.1은 다단 데이터베이스 서버의 구성을 나타낸 것이다.

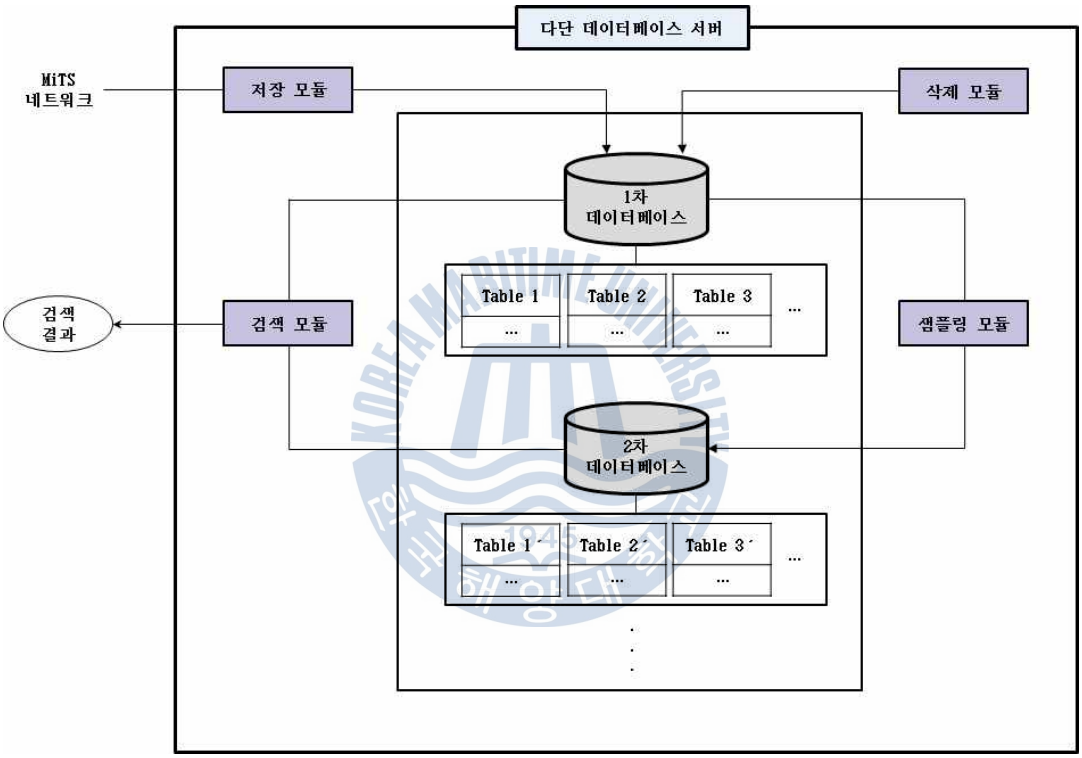


그림 3.1 다단 데이터베이스 서버의 구성

Fig 3.1 Architecture of multistage database server

다단 데이터베이스 서버에 존재하는 다단 데이터베이스는 선박의 여러 장비로부터 발생한 데이터를 저장하기 위한 1차 데이터베이스 및 데이터베이스 용량 초과로 인한 문제를 방지하고자 백업 목적으로 사용되는 2차 이상의 데이터베이스가 존재한다. 본 논문에서는 데이터를 샘플링하고 이를 저장하기 위한 백업 데이터베이스를 2차 데이터베이스까지 설계하였다.

또한 다단 데이터베이스에 저장된 데이터를 통합 관리하기 위하여 설계 및 구현된 모듈로는 저장, 샘플링, 삭제, 검색 모듈이 있다. 본 논문에서는 저장 모듈 및 삭제 모듈은 1차 데이터베이스에 데이터를 저장하거나 삭제한다. 삭제 모듈은 최종 데이터베이스가 n차 데이터베이스라면 n-1차 데이터베이스까지 데이터를 삭제한다. 본 논문에서는 최종 데이터베이스가 2차 데이터베이스이므로 1차 데이터베이스에서만 데이터를 삭제한다. 데이터 샘플링 모듈은 1차 데이터베이스의 데이터를 샘플링한 후 2차 데이터베이스에 저장하며, 삭제 모듈과 같이 최종 데이터베이스는 샘플링 대상에서 제외된다. 따라서 최종 데이터베이스인 2차 데이터베이스를 제외하고 1차 데이터베이스에서만 샘플링 한다. 검색 모듈은 구축된 모든 데이터베이스의 데이터를 검색하고 결과를 출력하는 역할을 한다.

## 3.1 다단 데이터베이스의 설계

본 장에서는 MiTS 형식으로 전달되는 데이터를 통합 관리하는 데 있어 데이터베이스 서버의 용량을 효율적으로 활용할 수 있도록 다단 데이터베이스를 설계하되, 그림 3.1에서 보여주듯이 2차 데이터베이스까지 설계한다.

### 3.1.1 1차 데이터베이스

그림 3.2는 MiTS의 형식에 맞게 저장할 수 있도록 설계된 1차 데이터베이스의 E-R 다이어그램을 나타낸 것이다.

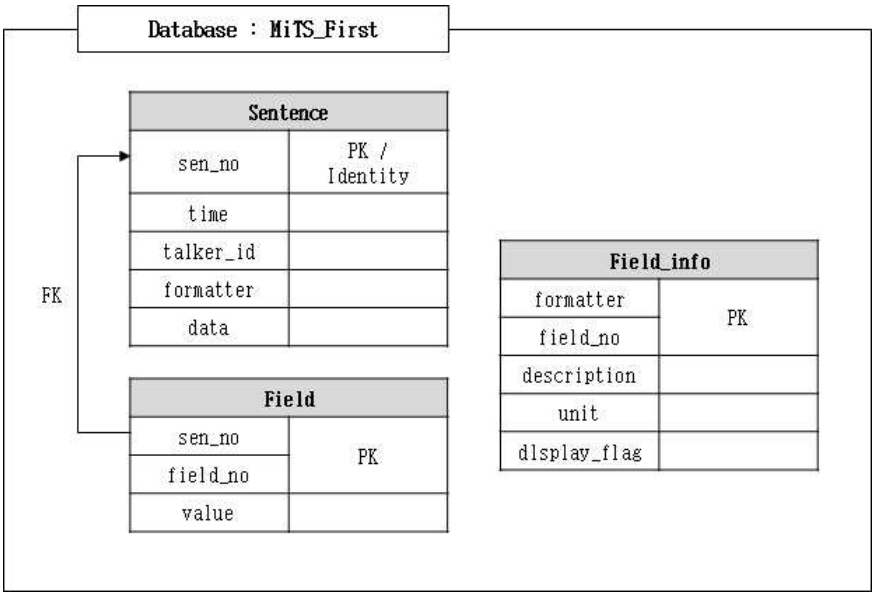


그림 3.2 1차 데이터베이스의 E-R 다이어그램

Fig 3.2 E-R diagram of first database

*Sentence* 테이블은 각 장비에서 발생하는 데이터를 순차적으로 저장하기 위한 테이블이다. 이 테이블의 *data* 칼럼에는 센텐스 형태로 전송된 데이터가 저장되는데 이 칼럼에 저장되는 데이터의 형태는 표 2.1의 데이터 필드를 저장하게 된다. 이 데이터 필드는 풍향, 수압, 기온 등 세부적으로 데이터 검색을 하기에는 불편하므로 이 *data* 칼럼에 저장된 데이터를 콤마 단위로 분리한다.

*Field* 테이블은 콤마 단위로 분리된 데이터들을 저장함으로써 좀 더 수월한 데이터 검색을 할 수 있도록 한다. 그러나 센텐스 포맷터 이름과 데이터만을 저장할 경우 어떤 필드의 값인지 알 수 없으므로 필드 번호까지 저장한다.

*Field_info* 테이블은 사용자가 각 센텐스 포맷터의 필드에 대한 상세 정보를 얻기 위한 테이블로서, 사용자가 관리할 센텐스 포맷터의 정보가 저장되어 있다. 또한 이 테이블에는 이후에 검색 모듈에서 데이터를 검색할 때, 각 필드에 대해서 필드별 검색 가능 여부를 구별하는 *display_flag* 칼럼이 존재한다. 예를 들어, 그림 2.2에서 MWD의 T, M과 같은 필드는 단위 필드이므로 검색을 하여서는 안 된다. 그러나 이 단위 필드가 검색 모듈에 표시되면 사용자의 부주의로 인하여 선택될 수도 있으므로 미리 검색 가능한 필드를 구별한다.

한편, 이 테이블은 선박의 장비가 새로운 센텐스 포맷터를 요구하지 않는 한 변경되지 않으며, 데이터베이스 관리자는 장비를 추가할 경우 필히 이 테이블에 관련 센텐스 포맷터의 정보를 입력해야 한다. 그렇지 않을 경우 데이터를 검색할 수 없다.

다음 표 3.1부터 표 3.3까지는 각 테이블에 해당하는 칼럼 이름과 그 의미를 나타낸다.

표 3.1 *Sentence* 테이블의 칼럼 및 의미

Table 3.1 Column and description of *Sentence* table

| 칼럼명 | 설명 |
|---|---|
| *sen_no* | 저장된 센텐스의 일련번호. 자동증가 |
| *time* | 센텐스가 저장된 시각 |
| *talker_id* | 센텐스가 발생된 장비의 ID |
| *formatter* | NMEA 0183 센텐스 포맷터 |
| *data* | 장비에서 발생된 데이터 |

표 3.2 *Field* 테이블의 칼럼 및 의미

Table 3.2 Column and description of *Field* table

| 칼럼명 | 설명 |
|---|---|
| *sen_no* | 저장된 센텐스의 일련번호 |
| *field_no* | 센텐스 포맷터에서의 필드 번호 |
| *value* | 필드의 실제 값 |

표 3.3 *Field_info* 테이블의 칼럼 및 의미

Table 3.3 Column and description of *Field_info* table

| 칼럼명 | 설명 |
|---|---|
| *formatter* | NMEA 0183 센텐스 포맷터 |
| *field_no* | 센텐스 포맷터에서의 필드 번호 |
| *description* | 필드에 대한 상세한 설명 |
| *unit* | 각 필드의 단위 |
| *display_flag* | 검색 모듈에서의 검색 가능 여부 확인 |

### 3.1.2 2차 데이터베이스

1차 데이터베이스가 장비에서 전송된 데이터의 저장을 위한 데이터베이스인 데 반해, 2차 데이터베이스(혹은 그 이상)는 앞의 데이터베이스에 대한 용량 초과를 사전에 방지하기 위하여 샘플링 한 데이터를 저장하는 역할로 사용되는 데이터베이스이다. 따라서 이 데이터베이스는 1차 데이터베이스의 구조와 유사해야 한다.

물론 데이터베이스 설계 과정에서 필요에 따라 2차 데이터베이스의 구조는 추가 및 변경 가능하지만 데이터가 변경되어서는 안 되므로 샘플링 데이터를 저장하는 역할에 영향을 주지 않는 선에서 추가해야 한다. 본 논문에서는 샘플링 관련 정보를 가지고 있는 테이블을 추가하여 사용하였으며 그림 3.3은 2차 데이터베이스의 E-R 다이어그램을 나타낸 것이다.

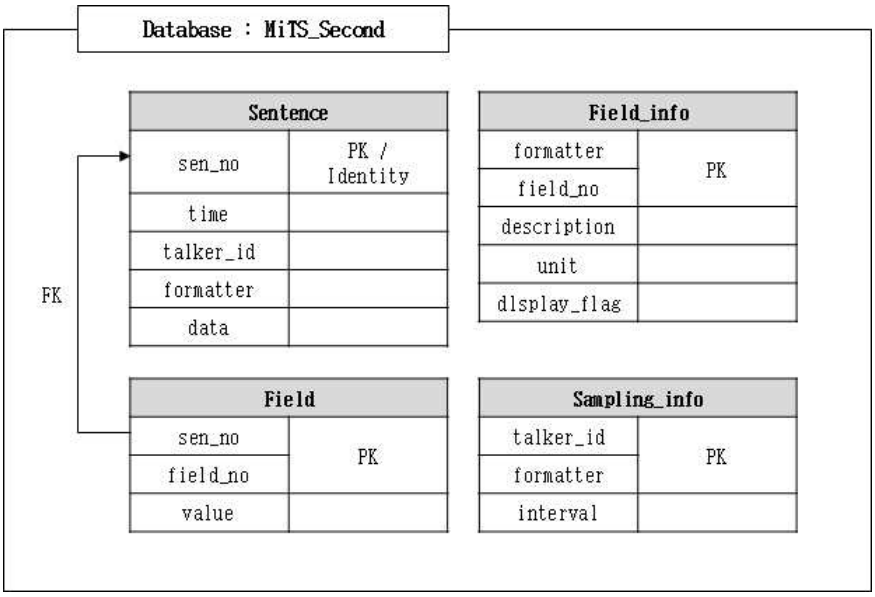


그림 3.3 2차 데이터베이스의 E-R 다이어그램

Fig 3.3 E-R diagram of second database

그림 3.2와 비교하면 *Sentence*, *Field*, *Field_info* 테이블의 구조는 일치한다. 그러나 이 데이터베이스는 각각의 센텐스 포맷터에 대해서 샘플링을 하기 위한 정보를 가지고 있는 *Sampling_info* 테이블을 추가적으로 가지고 있다. 이 *Sampling_info* 테이블의 각 칼럼의 이름과 의미는 표 3.4에 나타나 있다.

표 3.4 *Sampling_info* 테이블의 칼럼 및 의미

Table 3.4 Column and description of *Sampling_info* table

| 칼럼명 | 설명 |
|---|---|
| *talker_id* | 센텐스가 발생된 장비의 ID |
| *formatter* | 센텐스 포맷터에서의 필드 번호 |
| *interval* | 각 센텐스 포맷터의 샘플링 간격 |

*Sampling_info* 테이블에서 센텐스 포맷터만으로 간격을 설정하는 것이 아닌, Talker_id까지 고려하는 이유는 같은 데이터 타입을 측정하더라도 각 장비별로

발생하는 시간 간격이 다를 수 있기 때문이다. 따라서 각 장비마다 사용하는 센텐스 포맷터마다 간격을 설정할 수 있다. 또한 *Sampling_info* 테이블에는 *interval* 필드가 존재하는데, 이 필드는 시간 또는 이벤트 발생과 같은 샘플링 방법에 따라 달라질 수 있으며 본 논문에서는 일정한 시간 간격에 따라 샘플링 하기 위하여 *interval* 필드를 사용하였다.

한편, *Sampling_info* 테이블의 경우도 *Field_info* 테이블과 마찬가지로 관리자에 의해서 샘플링 관련 데이터가 사전에 입력되어 있어야 한다.

## 3.2 데이터베이스 모듈의 설계

### 3.2.1 데이터 저장 모듈

저장 모듈은 MiTS 네트워크를 통하여 변환된 데이터를 수신하고 저장하는 역할을 한다. 모듈이 동작하게 되면 우선 데이터베이스에 접속하고 소켓을 초기화 시킨 후 데이터 수신을 준비한다.

데이터 수신 준비가 완료되면 데이터가 전송되는지 계속 확인한다. 데이터가 전송되면 이를 수신하며 이 때 수신된 데이터는 *Sentence* 및 *Field* 테이블의 각 칼럼에 맞게 저장되어야 한다. 따라서 모듈에서도 *Sentence* 및 *Field* 테이블의 구조와 일치하는 구조체들을 선언하여 각각에 맞게 저장한다.

한편, 1차 데이터베이스에서 *Sentence* 테이블과 *Field* 테이블을 제외한 나머지 테이블인 *Field_info* 테이블은 1차 데이터베이스를 설계한 3.1.1절에서도 언급했듯이 데이터 저장 모듈에 의해 데이터가 저장되지 않고 서버 관리자에 의하여 사전에 직접 입력되어 있어야 한다.

그림 3.4는 데이터 저장 모듈에 대한 흐름도이며 Insert_Sentence() 및 Insert_Field() 함수는 *Sentence* 및 *Field* 테이블에 각 구조체의 데이터를 저장하기 위한 함수에 해당한다.

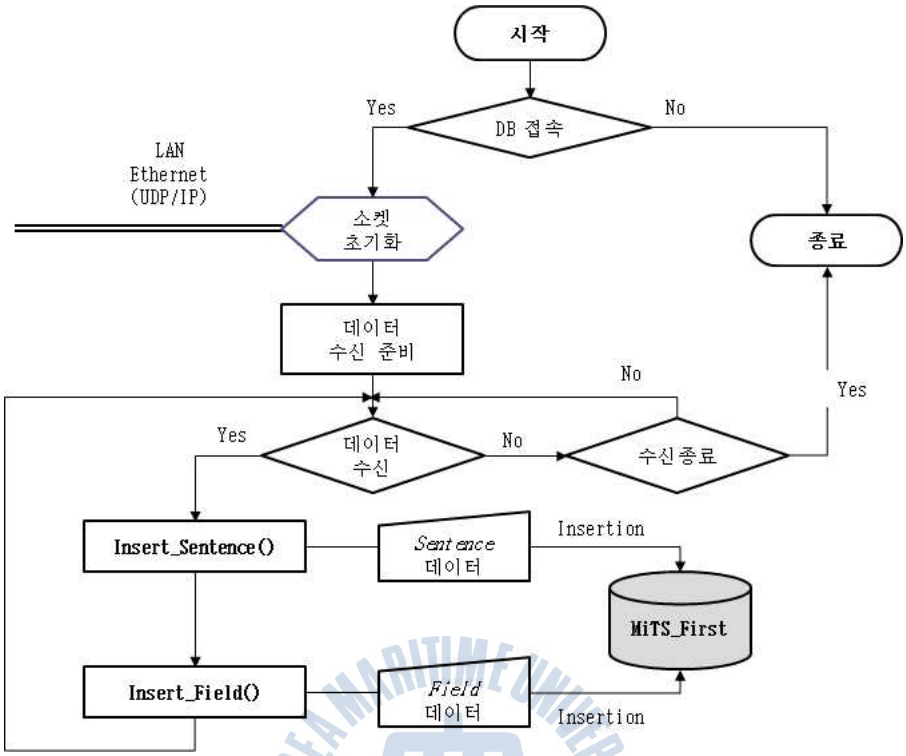


그림 3.4 데이터 저장 모듈의 흐름도

Fig 3.4 Flow chart of data storage module

MiTS 네트워크를 통하여 수신된 데이터는 표 2.2의 센텐스 포맷터의 예와 같은 형식을 가지고 있다. 각각 수신된 센텐스 포맷터가 각 테이블에 저장된 예는 그림 3.5와 같으며 화살표는 그림 3.4에서 Insert_Field() 함수의 기능과 같다.

한편, 그림 3.5에서 *Sentence* 테이블에 저장된 수심 데이터는 다시 *data* 필드를 콤마 단위로 분리하여 *Field* 테이블에 저장하였으며 Field_info 테이블의 경우는 NMEA 0183 표준의 내용에 따라 사전에 저장한 것이다.

XDR의 경우에는 여러 변환기의 정보가 하나의 센텐스에 전송되지만 하나의 변환기당 4개의 필드가 반복되는 형식을 가지고 있다. 여러 변환기의 정보가 한 센텐스에 가지고 있는 경우는 각 변환기 별로 따로 저장하는데 이는 *data* 필드를 분리하는 이유와 같이 검색이 쉽지 않기 때문이다. 이 때 혼란을 줄이기 위하여 센텐스 포맷터는 XDR 뒤에 변환기 타입을 붙여 저장한다.

**Sentence**

| sen_no | time | talker_id | formatter | data |
|---|---|---|---|---|
| 1 | 2010-12-07 14:01:02.003 | SD | DPT | 12.123,34.555,10.012 |
| 2 | 2010-12-07 14:01:03.009 | YX | XDR_C | C,12.34,C,Temperature |
| 3 | 2010-12-07 14:01:03.009 | YX | XDR_F | F,59.111,H,Frequency |

**Field**

| sen_no | field_no | value |
|---|---|---|
| 1 | 1 | 12.123 |
| 1 | 2 | 34.555 |
| 1 | 3 | 10.012 |
| 2 | 1 | C |
| 2 | 2 | 12.34 |
| 2 | 3 | C |
| 2 | 4 | Temperature |
| 3 | 1 | F |
| 3 | 2 | 59.111 |
| 3 | 3 | H |
| 3 | 4 | Frequency |

**Field_info**

| formatter | field_no | description | unit | display_flag |
|---|---|---|---|---|
| DPT | 1 | Water Depth relative to the transducer | meters | True |
| DPT | 2 | Offset from transducer | meters | False |
| DPT | 3 | Maximum range scale in use | meters | False |
| XDR_C | 1 | Temperature_Type | None | False |
| XDR_C | 2 | Temperature_Value | None | True |
| XDR_C | 3 | Temperature_Unit | None | False |
| XDR_C | 4 | Temperature_ID | None | False |
| XDR_F | 1 | Frequency_Type | None | False |
| XDR_F | 2 | Frequency_Value | None | True |
| XDR_F | 3 | Frequency_Unit | None | False |
| XDR_F | 4 | Frequency_ID | None | False |

그림 3.5 데이터가 저장된 예

Fig 3.5 Example of data storage

### 3.2.2 데이터 샘플링 모듈

데이터베이스에서 데이터 통합 관리를 하는 경우, 다수의 데이터 발생 빈도로 인하여 소수의 데이터만을 관리하는 경우에 비해 용량이 초과되기까지 걸리

는 시간이 짧다. 따라서 데이터베이스의 용량이 초과되기 전에 용량을 확보하여야 한다.

서론에서도 언급했듯이 이를 위한 해결책으로 데이터베이스 서버의 용량을 수동적으로 증가시키는 방법과 모듈을 통해 자동적으로 용량을 확보하는 방법이 있다. 이 때 비용, 해결 방법 측면을 모두 고려하면 모듈 내에서 자동적으로 용량을 확보하는 편이 더 효율적이며 모듈을 통한 용량 확보를 하더라도 데이터를 백업해야 데이터 통합 관리가 가능하다. 이 때 데이터 백업의 경우에 몇 가지 고려 사항이 있다.

우선 전체 백업을 할 것인지, 부분 백업을 할 것인지에 대한 것이다. 전체 백업은 데이터를 통합 관리하기 좋으나 많은 비용이 들며, 부분 백업은 일부적인 통합 관리만 가능하지만 비용이 적게 든다. 따라서 둘을 비교하면 부분 백업이 더 효율적이다. 그러나 부분 백업을 하더라도 어느 부분의 데이터를 백업할 것인지를 결정해야 한다. 이 때 특정 기간에 대해서만 백업한다면 그 외의 기간에 대해서는 검색이 불가하다. 그러나 일정 간격으로 데이터를 샘플링 하는 경우 신뢰도가 떨어질 수 있으나 전반적인 기간에 대한 검색이 가능하다. 사용자는 과거 데이터를 검색할 때 분, 초 단위까지 검색하는 경우가 드물다는 점을 감안하면 일정 간격으로 샘플링 하는 편이 좋다.

그리고 두 번째 문제는 샘플링을 할 시기의 문제이다. 언제 데이터를 삭제할 것인지의 문제인데, 데이터를 삭제할 때 샘플링 한다면 한 번에 많은 양을 샘플링 해야 한다. 많은 양의 데이터를 한 번에 샘플링 하는 것은 시간 측면에서 상당히 비효율적이므로 *Sampling_info* 테이블에 저장된 간격마다 각 센텐스 포맷터별로 계속 샘플링 하여 저장하는 것이 훨씬 효율적이다.

마지막으로, 샘플링 방법에 관한 것이다. 3.1.2절에서 언급한 것과 같이 시간 간격으로 샘플링 하더라도 언제 어떤 데이터를 샘플링 할 것인지의 문제가 존재한다.

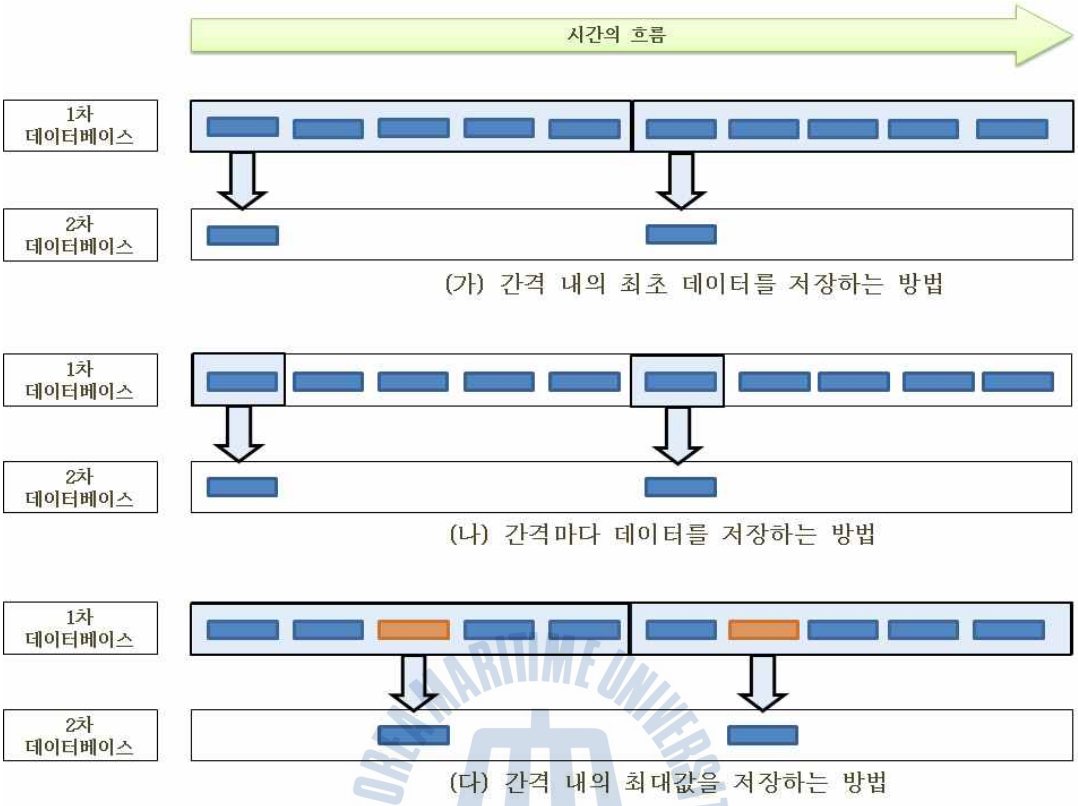


그림 3.6 데이터 샘플링 기법의 종류

Fig 3.6 Kinds of data sampling technique

그림 3.6은 시간 간격으로 데이터 샘플링을 하기 위한 몇 가지 방법의 예를 나타낸 것이다. (가)의 방법은 일정 간격이 지난 후에 그 간격 내에서 가장 최초의 데이터를 샘플링 하는 방법이다. (나)의 방법은 일정 간격마다 샘플링 하는 방법으로 가장 일반적인 방법이며 (다)의 방법은 (가)의 방법과 유사하지만 최초의 데이터가 아닌, 일정 간격 내에서의 최대값을 저장하는 방법이다. 그러나 (나)의 방법은 일정 간격으로 샘플링 시 데이터가 존재하지 않으면 샘플링이 되지 않을 수 있으며 (다)의 방법은 수치 데이터에만 가능하다는 한계가 있다.

따라서 본 논문에서는 (가)의 방법을 사용하여 일정 간격으로 계속 샘플링하되, 가장 최초의 데이터를 샘플링 하도록 설계하였다. 이 때 샘플링 간격은 데이터의 발생 빈도에 따라 사용자가 설정 변경 가능하도록 하였다. 예를 들어

자주 변할 수 있는 온도의 경우는 간격을 짧게 주어야 하며 이에 비해 위경도 값은 변화가 적으므로 온도보다는 간격을 넓게 줄 수 있도록 한다.

그림 3.7은 시간의 흐름에 따라 저장된 각각의 데이터를 샘플링 하는 과정을 나타낸 것으로서 DPT와 MWD를 각각 10분과 30분마다 샘플링을 한다고 가정한 것이다. 1차 데이터베이스에 저장된 각 데이터들은 각각의 간격에 따라 샘플링 되어 2차 데이터베이스에 저장되는데 그림 3.7에서는 12시부터 각 간격에 따라 샘플링하게 된다. 또한 샘플링을 할 때 간격 내 가장 최초의 데이터 하나만 저장한다. 결과는 그림 3.7의 2차 데이터베이스와 같이 나타난다.

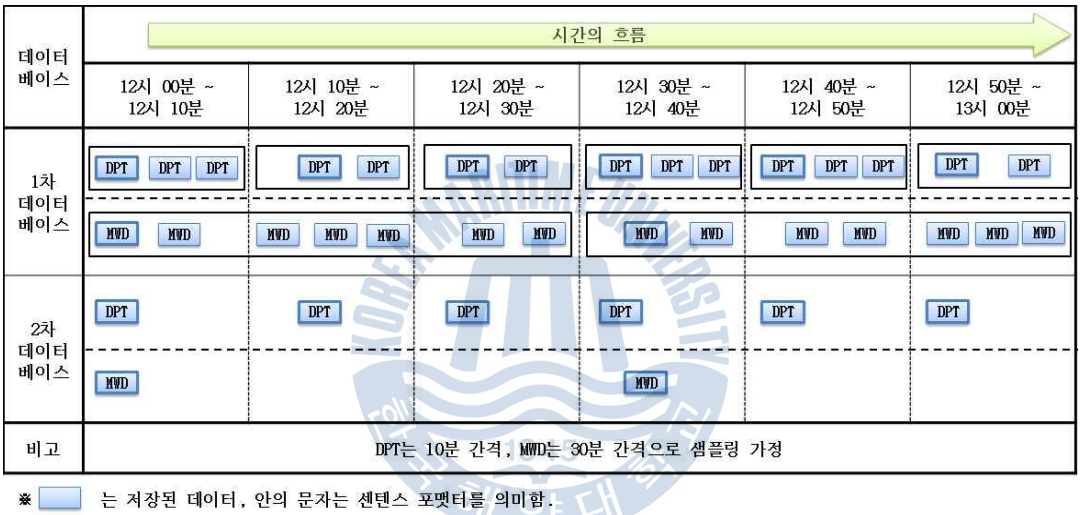


그림 3.7 데이터 샘플링 과정의 예

Fig 3.7 Example of Data sampling processing

이에 따라 일정 간격 내 가장 최초 시간의 데이터를 샘플링 하는 데이터 샘플링 모듈의 흐름도는 그림 3.8과 같이 표현하였다.

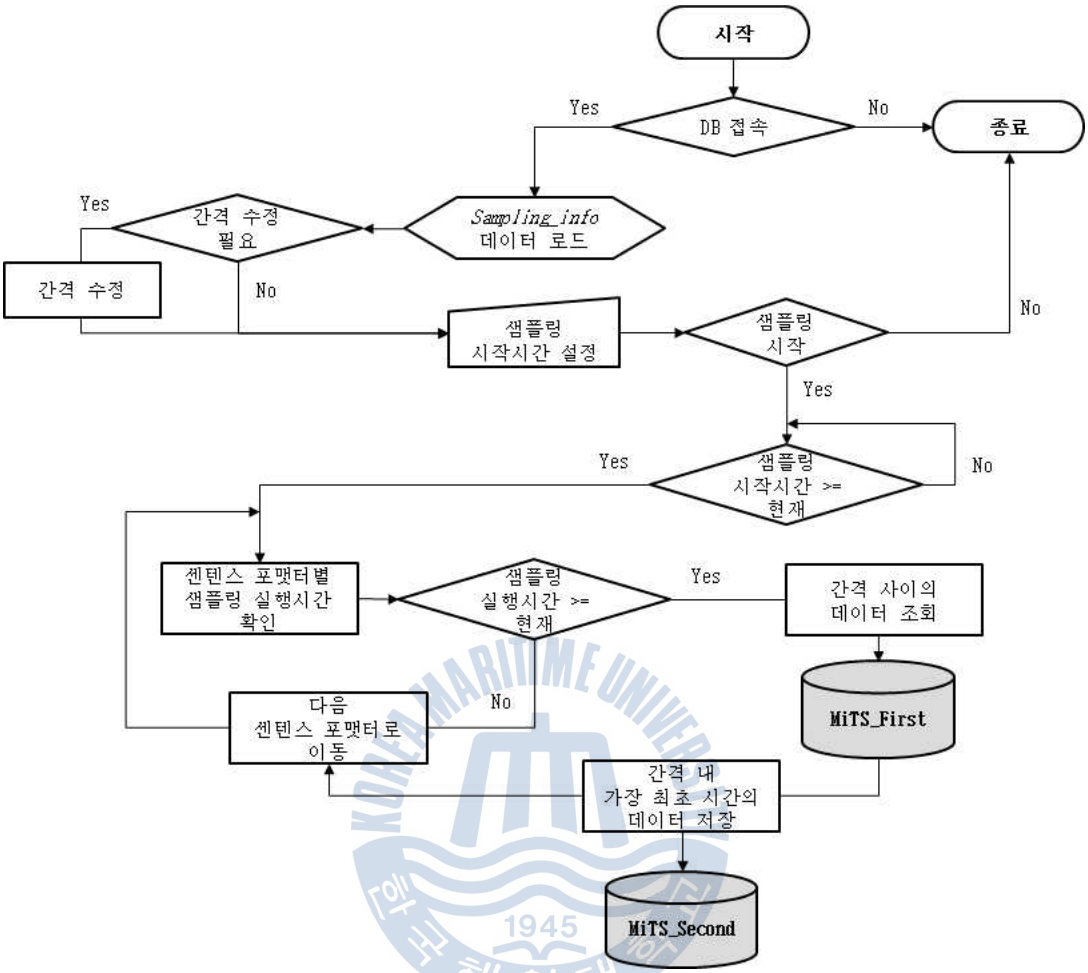


그림 3.8 데이터 샘플링 모듈의 흐름도

Fig 3.8 Flow chart of data sampling module

샘플링 모듈은 시작과 함께 데이터베이스에 접속한 후 *Sampling_info*에서 샘플링을 위한 데이터를 읽어 온다. 3.1.2절의 2차 데이터베이스 설계에서 이미 *Sampling_info*는 사용자가 사용할 센텐스 포맷터에 대해서 사전에 입력해 두었다고 가정하였으므로 데이터가 존재하지 않는 경우는 제외한다. 데이터 로드가 끝나면 간격 변경 및 샘플링 시작 시간을 설정할 수 있다. 간격의 경우는 사용자가 변경하고자 하는 센텐스 포맷터에 대해서 샘플링 시작 이전에만 변경이 가능하므로 모든 설정 후에 샘플링을 시작한다. 간격 변경을 원치 않는 경우에는 변경하지 않아도 무관하다.

샘플링이 시작되면 1초 단위로 계속 센텐스 포맷터별로 샘플링 시간을 확인하여 샘플링 시간이 된 센텐스 포맷터에 대해서는 1차 데이터베이스에서 센텐스 포맷터별 간격 사이의 데이터 중 가장 최초 시간의 데이터를 가져온다. 이를 2차 데이터베이스에 저장한 후 다음 센텐스 포맷터를 확인한다. 샘플링이 가능한 상태라면 샘플링을 하고 만약 샘플링이 아직 가능하지 않다면 그 다음 센텐스 포맷터를 확인한다. 마지막 센텐스 포맷터를 확인한 후에는 다시 처음의 센텐스 포맷터를 확인한다.

한편, 그림 3.8의 흐름도에 표현된 것과 같이 현재 시간 이후에 대해서만 샘플링이 가능하도록 설계하였으므로 데이터 샘플링 모듈이 동작하지 않은 시간에 저장된 데이터에 대해서는 샘플링을 수행하지 않는다.

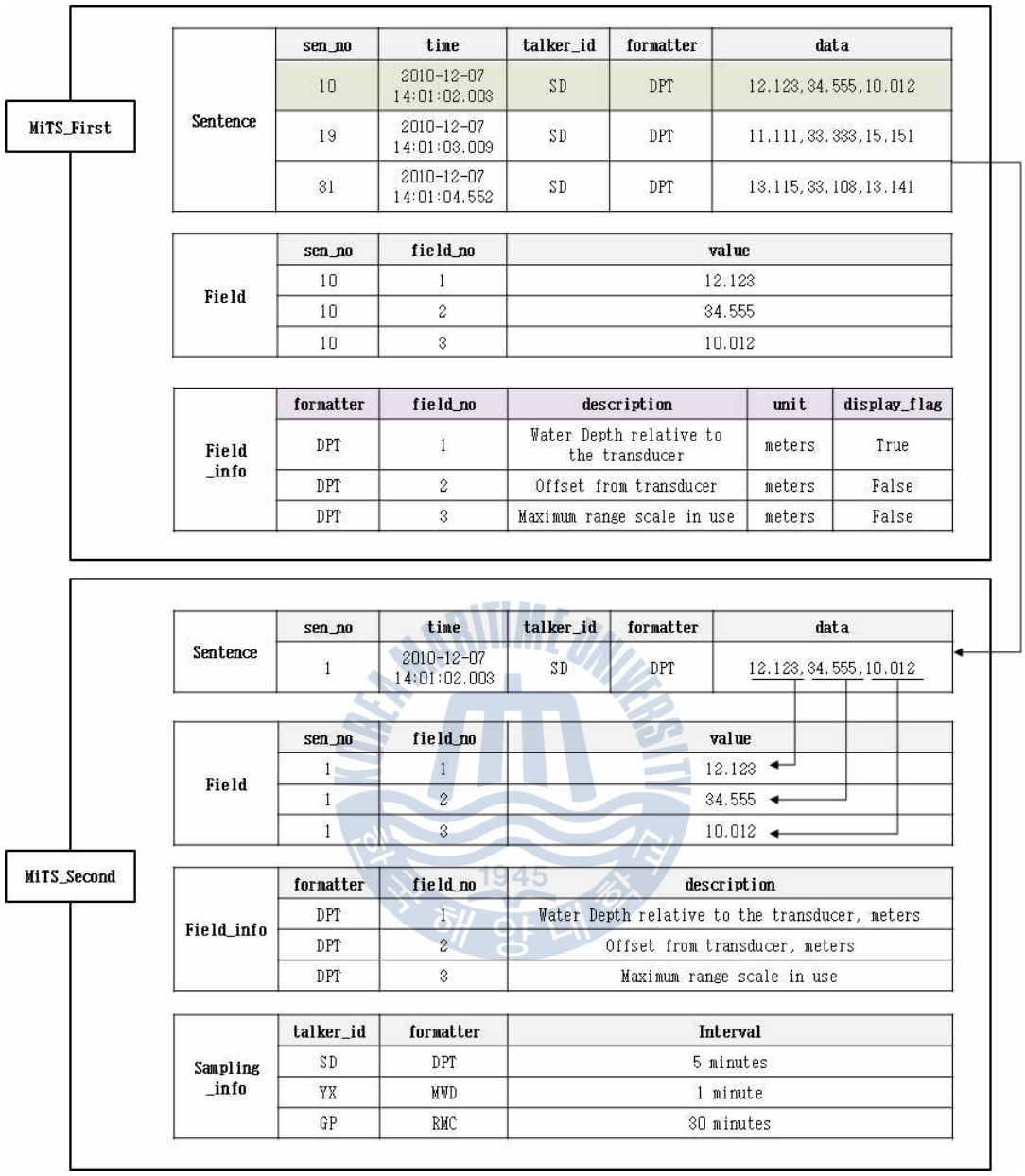


그림 3.9 데이터가 샘플링 된 데이터베이스의 예

Fig 3.9 Example of sampled database

```
/* OdbcConnection Con = new OdbcConnection();
   OdbcCommand Com = new OdbcCommand();
   OdbcDataReader R = new OdbcDataReader(); */

private void Data_Sampling(String _Talker_ID, String _Formatter, String
_From, String _To)
{
    if (Con.State == ConnectionState.Closed)
        MiTS_First_Connection();

    Com = new OdbcCommand("select * from sentence where talker_id = '" +
    _Talker_ID + "' and formatter = '" + _Formatter + "' and time in
    (select min(time) from sentence where talker_id = '" + _Talker_ID +
    "' and formatter = '" + _Formatter + "' and " + " (time between '" +
    _From + "' and '" + _To + "'));", Con);

    R = Com.ExecuteReader();
    if (R.HasRows) {
        R.Read();

        Insert_Sentence(R.GetDateTime(1).ToString("yyyy-MM-dd HH:mm:ss.fff"),
        R.GetString(2), R.GetString(3), R.GetString(4));
        Int32 Count = Count_Data();
        Insert_Field(Count, R.GetString(4));
    }

    R.Close();
}
```

그림 3.10 데이터 샘플링 함수의 구현

Fig 3.10 Implementation of data sampling function

그림 3.9는 데이터 샘플링 된 데이터베이스의 예를 보여주며 그림 3.10은 데이터 샘플링을 실행하는 함수를 구현한 것이다.

### 3.2.3 데이터 삭제 모듈

3.2.2절에서도 언급했듯이 1차 데이터베이스에 데이터가 계속 저장될 경우 데이터베이스는 용량 초과 상태에 이를 수 있기 때문에 일정량의 데이터가 저장되면 데이터를 삭제하여 용량을 확보하여야 한다.

그러나 데이터 샘플링을 거쳐 데이터를 삭제한다고 하더라도 1차 데이터베이스의 데이터를 100% 삭제한다면 샘플링이 되지 않은 시간의 데이터까지 완전히 삭제되어 데이터 검색이 불가능해진다. 따라서 샘플링이 되고 난 이후의 데이터에 대해서만 삭제한다.

우선 1차 데이터베이스에서 데이터가 일정량이 되는지를 확인할 필요가 있는데, 다음 그림 3.11과 그림 3.12의 SQL문들을 통하여 확인할 수 있다. 이들 SQL문은 MS-SQL 내에서 사용 가능하며, 그림 3.11은 데이터베이스의 전체 용량을 확인하는 SQL문이며 그림 3.12는 현재 용량을 확인하는 SQL문이다. 현재 용량을 전체 용량으로 나누어 100을 곱하면 현재 데이터베이스가 얼마의 데이터를 저장하고 있는지를 백분율로 나타낼 수 있으며 용량 단위는 MB(Mega Byte)로 일치시켰다.

```
select convert(nvarchar(15), convert (bigint, maxsize) * 8 / 1024)
from [MiTS].[dbo].sysfiles
where name = 'MiTS_First'
```

그림 3.11 데이터베이스의 전체 용량을 확인하기 위한 SQL문

Fig 3.11 SQL query to check database's total capacity

```
select str(convert(dec(15),sum(size)) * 8192 / power(2,20),10,2)
from [MiTS_First].[dbo].sysfiles;
```

그림 3.12 데이터베이스의 현재 용량을 확인하기 위한 SQL문

Fig 3.12 SQL query to check database's current capacity

현재 데이터베이스에 용량이 얼마나 사용 중인지 확인 후 일정량 이상이 되지 않는다면 용량이 초과될 때까지 확인하여 일정량이 된 이후에 삭제한다.

한편, 지속적인 데이터베이스 용량을 확인하기 위하여 삭제 모듈은 기본적으

로 무한 루프를 실행하여야 한다. 그러나 무한 루프는 CPU 사용량을 급격하게 증가시키므로 동시에 다른 모듈을 동작시키기 어렵다. 데이터베이스 용량이 극히 작거나 데이터 저장 속도가 매우 빨라서 데이터베이스의 용량 변화가 쉽게 발생하는 경우에는 용량 확인을 자주 해야 한다. 그러나 그렇지 않은 경우라면 Sleep 함수를 사용하여 문제를 해결 가능하다. 용량이 급격하게 증가하지 않는다면 지속적으로 확인을 할 필요는 없으므로 용량 확인 횟수를 줄이는 것이 효율적이다.

한편, 데이터를 삭제할 때에는 얼마나 삭제할 것인가를 계산하여 삭제하도록 한다. 이 때, 데이터를 얼마나 삭제하느냐에 대한 기준은 데이터의 양 또는 저장 시간이 될 수 있으며 두 가지 방법에 대하여 모두 장단점이 존재한다.

데이터 저장량을 기준으로 일부를 지우는 방법은 시간을 기준으로 하는 방법보다 용량 확보 측면에서는 능률적이지만 같은 시간에 발생한 데이터의 경우 일부만 지워질 수 있다. 특히 XDR의 경우 그림 3.5와 같이 변환기 별로 분리하여 저장하므로 용량 기준으로 삭제할 경우 일부 변환기의 데이터만 삭제 될 수 있다. 또한 데이터 저장량 기준의 삭제 방법은 시간 기준의 삭제 방법에 비하여 샘플링이 되지 않은 데이터를 삭제할 가능성이 크다. 데이터베이스에 저장된 전체 시간의 일부를 지울 경우에는 동시에 들어온 데이터가 같이 삭제되지만 데이터가 지속적으로 들어오지 않을 때에는 삭제되는 데이터가 적어 용량이 많이 확보되지 않을 수 있다. 그러나 샘플링이 되지 않은 데이터를 삭제할 확률은 용량에 따른 삭제에 비하여 작다. 따라서 본 논문에서는 저장량이 90%가 될 경우에 데이터의 저장량보다는 데이터가 저장된 시간을 계산하여 50%의 데이터를 삭제하도록 설계하였다.

데이터가 삭제되는 시간의 범위를 구하는 방법은 다음과 같다. 예를 들어, 현재 시간으로부터 샘플링 최대 간격 전까지, 즉 모든 센텐스 포맷터 중에서 샘플링 최대 간격이 30분이라면 현재 시간으로부터 30분 전까지는 샘플링이 되지 않은 데이터가 존재하므로 삭제 범위에서 제외시키고 나머지 데이터에서 50%를 삭제한다. 그러나 샘플링 최대 간격보다 현재 1차 데이터베이스에 저장된 데이터의 총 기간이 작은 경우에는 샘플링과 상관없이 전체 데이터 기간의 50%를 삭제한다.

그림 3.13을 통하여 데이터 삭제 모듈의 전체적인 흐름도를 확인할 수 있다.

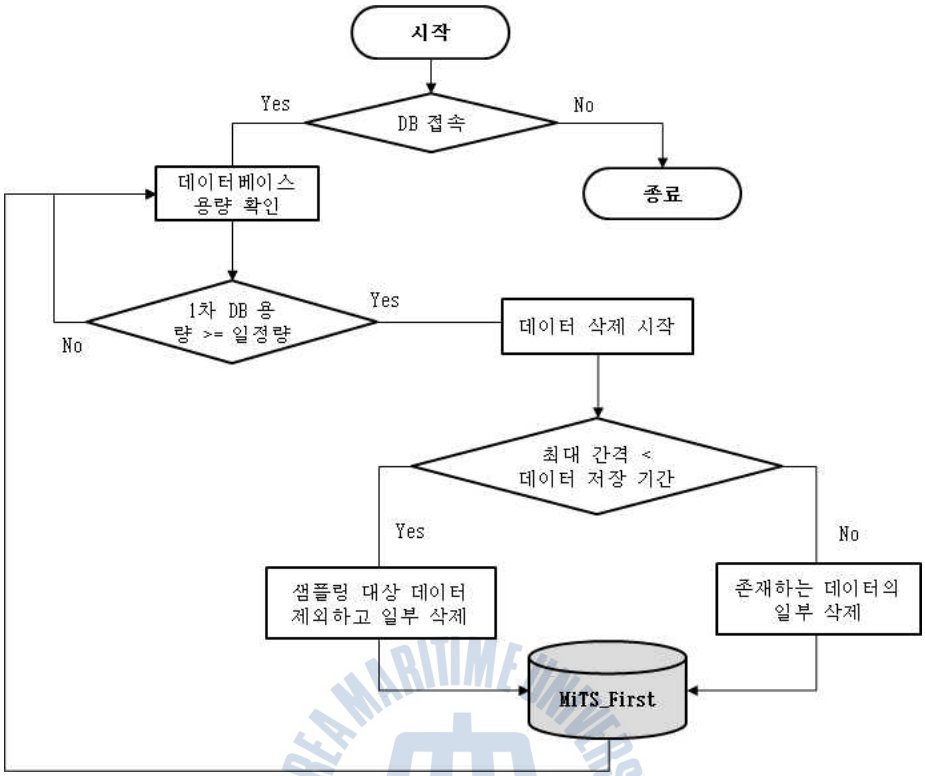


그림 3.13 데이터 삭제 모듈의 흐름도

Fig 3.13 Flow chart of data deletion module

한편, 데이터 삭제를 하더라도, 데이터베이스의 용량은 줄어들지 않고 오히려 늘어난다. 데이터 삭제 시 트랜잭션 로그가 발생하기 때문이며, 적은 용량의 트랜잭션 로그가 발생한다면 문제가 없으나 데이터를 몇 천 개 이상 삭제하는 경우 트랜잭션 로그의 용량은 급격하게 늘어난다. 따라서 트랜잭션 로그를 삭제해 줄 필요가 있다.

그림 3.14에 구현된 데이터 삭제 함수를 통하여 데이터 삭제 과정 및 트랜잭션 로그의 삭제 과정을 확인할 수 있다. 중간의 DELETE문은 데이터를 삭제하는 과정이며 FK(foreign key) 설정에 의하여 *Sentence* 테이블보다는 *Field* 테이블을 우선 삭제한다. 밑의 BACKUP문 및 DBCC(Database Console Command)문은 트랜잭션 로그를 비우고 데이터베이스의 용량 확보를 위하여 사용된다.

```
static void Delete_Data(DateTime StartTime, DateTime EndTime) {
    using (OdbcConnection Con = Init_MiTS_FirstConnection()) {
        if (Con.State == System.Data.ConnectionState.Closed)
            Con.Open();

        OdbcCommand Com = new OdbcCommand();
        Com.Connection = Con;

        Com.CommandText = "DELETE FROM field WHERE sen_no IN (SELECT sen_no
        FROM sentence WHERE time BETWEEN '" + StartTime.ToString("yyyy-MM-dd
        HH:mm:ss") + "' AND '" + EndTime.ToString("yyyy-MM-dd HH:mm:ss") +
        "');";
        Com.CommandTimeout = 300;
        Com.ExecuteNonQuery();

        Com.CommandText = "DELETE FROM sentence WHERE time BETWEEN '" +
        StartTime.ToString("yyyy-MM-dd HH:mm:ss") + "' AND '" +
        EndTime.ToString("yyyy-MM-dd HH:mm:ss") + "';";
        Com.CommandTimeout = 300;
        Com.ExecuteNonQuery();

        Com.CommandText = "BACKUP LOG MiTS_First WITH no_log;";
        Com.ExecuteNonQuery();

        Com.CommandText = "DBCC SHRINKDATABASE (N'MiTS_First');";
        Com.ExecuteNonQuery();

        Com.CommandText = "DBCC SHRINKFILE(N'MiTS_First', TRUNCATEONLY);";
        Com.ExecuteNonQuery();
    }
}
```

그림 3.14 데이터 및 트랜잭션 로그 삭제를 위한 함수의 구현

Fig 3.14 Implementation of function to delete data and transaction log

### 3.2.4 데이터 검색 모듈

그림 3.15는 데이터 검색 모듈의 흐름도를 나타낸 것으로서 사용자가 검색하고자 하는 데이터의 기간, 데이터 타입, 센텐스 포맷터, 간격 등의 특정 조건을 부여하면 해당하는 결과를 출력한다. 데이터 검색 시 우선적으로 1차 데이터베이스를 검색하며 데이터가 부족하거나 없는 경우 2차 데이터베이스에서 추가적으로 검색한다.

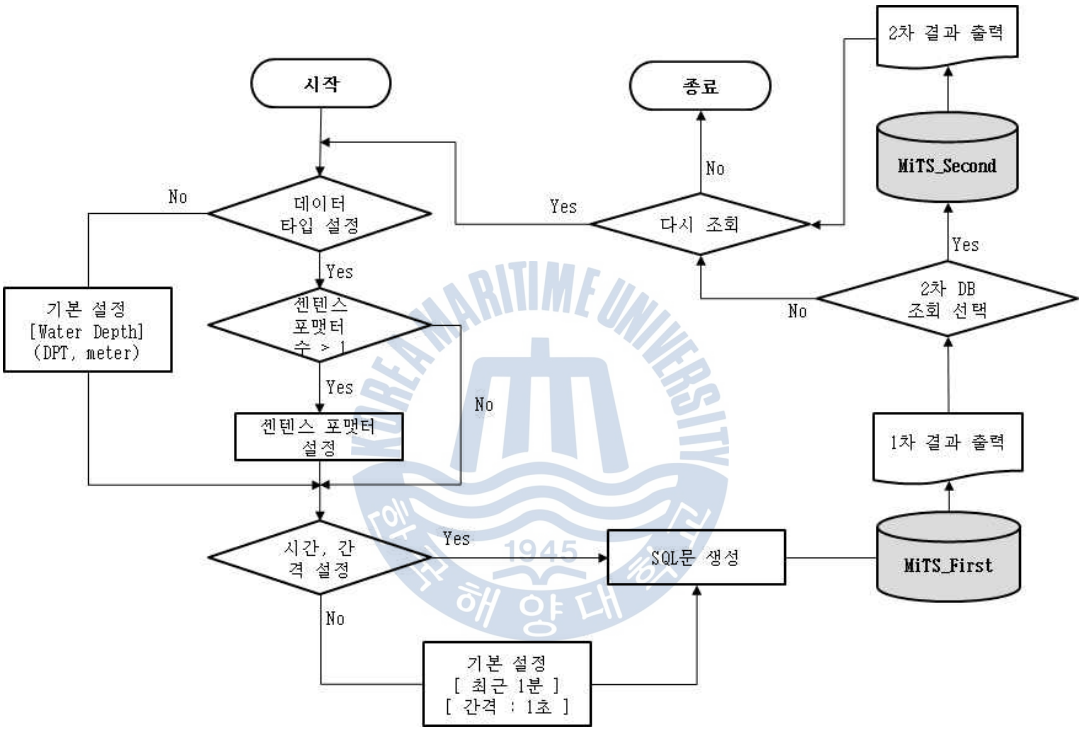


그림 3.15 데이터 검색 모듈의 흐름도

Fig 3.15 Flow chart of data retrieval module

데이터 검색 모듈의 사용자 인터페이스를 통하여 사용자는 자신이 검색하기 원하는 데이터의 조건을 설정 가능하며, 이 때 데이터 타입, 단위, 센텐스 포맷터와 같은 데이터 자체의 관련 정보 및 검색 기간, 간격 등의 시간 정보를 입력 가능하다.

데이터를 검색할 때 같은 간격 내에 둘 이상의 데이터가 존재한다면 샘플링 방법과 마찬가지로 간격 내에서 최초의 데이터만을 출력하도록 한다. 그림

3.16은 이와 같은 방법으로 특정 시간의 수온 데이터를 분 단위별로 검색하기 위한 SQL문의 예를 나타낸 것이다.

```
select time, value as [Water Temperature]
from sentence, field
where sentence.sen_no = field.sen_no
and formatter = 'MTW' and field_no = 1 and time in
 (select min(time)
  from sentence, field
  where (formatter = 'MTW' and field_no = 1
         and sentence.sen_no = field.sen_no) and convert(char(16),time,120) in
    (select distinct substring(convert(char(16),time,120),1,16)
     from sentence, field
     where (time between '2010-12-20 12:00:00.000'
            and '2010-12-20 17:00:00.000')
     and formatter = 'MTW' and field_no = 1
     and sentence.sen_no = field.sen_no)
  group by convert(char(16),time,120))
order by time;
```

그림 3.16 특정 조건을 검색하기 위한 SQL문의 예

Fig 3.16 Example of SQL query to retrieval for specific condition

그림 3.16에서, 우선 특정 시간 이내의 분 단위별 수온 데이터를 검색하기 위하여 특정 시간 내에 수온 데이터가 존재하는 시간을 분 단위별 리스트로 추출한다. 만약 1분 내에 수온 데이터가 없다면 나타나지 않으며, 중복된다면 하나만 표시된다. 이 리스트는 각 분 단위별로 최초의 수온 데이터를 찾는데 이용될 수 있으며, 각 시간별로 최초에 발생한 수온 데이터는 시간 순으로 정렬된 뒤 결과 출력에 쓰인다.

데이터를 검색할 때에는 우선적으로 1차 데이터베이스에서만 검색한다. 그러나 1차 데이터베이스를 검색한 후에 데이터가 존재하지 않거나, 혹은 검색 기간 중 일부 기간에 해당하는 데이터가 1차 데이터베이스 내에서 삭제되었을 경우 사용자는 2차 데이터베이스 검색을 선택할 수 있다. 2차 데이터베이스에서

는 사용자가 설정한 검색 기간 중 1차 데이터베이스에서 데이터가 출력된 기간을 뺀 나머지 기간에 대해서 검색하며, 이를 그림으로 나타내면 그림 3.17과 같다.

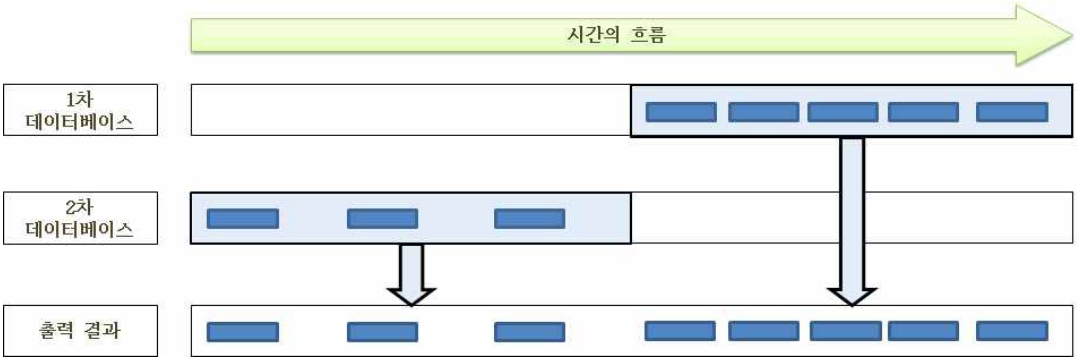


그림 3.17 2차 데이터베이스 검색을 통해 얻는 결과

Fig 3.17 Result from second database

# 제 4 장 구현 및 시험

본 논문에서 설계된 다단 데이터베이스의 시험을 위하여 구현된 각 모듈은 Microsoft Windows 7에서 개발하였다. Microsoft Visual Studio 2008의 C++ 및 C#을 이용하였으며, 검색 모듈은 추가적으로 SoftwareFX의 Chart FX 7 .NET 컴포넌트를 사용하였다. DBMS(Database Management System)는 Microsoft SQL Server 2005 SP2를 이용하여 데이터베이스를 구축하였다.

시험 환경은 Microsoft Windows 7(VGN-Z45LD Notebook) 및 Windows Server 2008(PowerEdge 2950 Server)에서 수행하였다.

## 4.1 데이터베이스의 구현

본 절에서는 3.1절에서 설계한 내용을 바탕으로 데이터베이스를 구축하였으며 이를 시험하기 위하여 아래의 센텐스 포맷터를 가지고 시험하였다.

- **DPT** : Depth
- **MTW** : Water Temperature
- **MWD** : Wind Direction & Speed
- **RMC** : Recommended Minimum Specific GNSS Data
- **TXT** : Text Transmission
- **VHW** : Water Speed and Heading
- **VTG** : Course Over Ground & Ground Speed
- **XDR** : Transducer Measurements

그림 4.1과 그림 4.2를 통하여 이 센텐스 포맷터들의 정보를 미리 입력한 *Field_info* 테이블과 *Sampling_info* 테이블의 일부를 확인할 수 있다.

| formatter | field_no | description | unit | display_flag |
|---|---|---|---|---|
| DPT | 1 | Water Depth rel... | Meters | True |
| DPT | 2 | Offset from tran... | Meters | False |
| DPT | 3 | Maximum range ... | Meters | False |
| MTW | 1 | Temperature | Degrees(C) | True |
| MTW | 2 | C(Celsius) | None | False |
| MWD | 1 | Wind direction | Degrees(T) | True |
| MWD | 2 | T(True) | None | False |
| MWD | 3 | Wind direction | Degrees(M) | True |
| MWD | 4 | M(Magnetic) | None | False |
| MWD | 5 | Wind speed | Knots(N) | True |
| MWD | 6 | N(knots) | None | False |
| MWD | 7 | Wind Speed | m/s(M) | True |
| MWD | 8 | M(meters/second) | None | False |
| RMC | 1 | UTC of position fix | None | False |
| RMC | 2 | Status | None | False |
| RMC | 3 | Latitude | Degrees(T) | True |
| RMC | 4 | Latitude N/S | None | False |
| RMC | 5 | Longitude | Degrees(T) | True |
| RMC | 6 | Longitude E/W | None | False |
| RMC | 7 | Speed Over Gro... | Knots(N) | True |
| RMC | 8 | Course Over Gr... | Degrees(T) | True |
| RMC | 9 | Date : ddmmyy | None | False |
| RMC | 10 | Magnetic variati... | None | False |
| RMC | 11 | Magnetic variati... | None | False |
| RMC | 12 | Mode Indicator | None | False |
| TXT | 1 | Total number of ... | None | False |
| TXT | 2 | Sentence number | None | False |
| TXT | 3 | Text identifier | None | False |
| TXT | 4 | Text message | None | False |

그림 4.1 시험용 *Field_info* 테이블

Fig 4.1 *Field_info* table for test

| talker_id | formatter | interval |
|---|---|---|
| GP | RMC | 30 Minutes |
| HC | VHW | 10 Minutes |
| SD | DPT | 5 Minutes |
| WM | VTG | 30 Minutes |
| YX | MTW | 1 Minute |
| YX | MWD | 10 Minutes |
| YX | XDR_C | 1 Minute |
| YX | XDR_F | 5 Minutes |
| YX | XDR_H | 10 Minutes |
| YX | XDR_P | 10 Minutes |

그림 4.2 시험용 *Sampling_info* 테이블

Fig 4.2 *Sampling_info* table for test

## 4.2 데이터베이스 모듈의 구현 및 시험

### 4.2.1 데이터 저장 모듈

그림 4.3은 임의로 데이터를 발생시킨 후 데이터 저장 모듈을 통하여 데이터베이스에 저장되는 과정을 나타낸 것이며 데이터가 저장된 결과는 그림 4.4에서 확인할 수 있다.

```
[2010-12-20 12:41:59.485] Talker ID : YX / Formatter : XDR
[2010-12-20 12:41:59.485] Talker ID : YX / Formatter : MTW
[2010-12-20 12:41:59.485] Talker ID : GP / Formatter : RMC
[2010-12-20 12:41:59.500] Talker ID : YX / Formatter : XDR
[2010-12-20 12:41:59.516] Talker ID : SD / Formatter : DPT
[2010-12-20 12:41:59.516] Talker ID : GP / Formatter : RMC
[2010-12-20 12:41:59.532] Talker ID : YX / Formatter : XDR
[2010-12-20 12:41:59.532] Talker ID : YX / Formatter : MTW
[2010-12-20 12:41:59.547] Talker ID : GP / Formatter : RMC
[2010-12-20 12:41:59.547] Talker ID : YX / Formatter : XDR
[2010-12-20 12:41:59.563] Talker ID : YX / Formatter : MTW
[2010-12-20 12:41:59.563] Talker ID : GP / Formatter : RMC
[2010-12-20 12:41:59.578] Talker ID : YX / Formatter : XDR
[2010-12-20 12:41:59.594] Talker ID : YX / Formatter : MWD
[2010-12-20 12:41:59.610] Talker ID : HC / Formatter : VHW
[2010-12-20 12:41:59.625] Talker ID : YX / Formatter : XDR
[2010-12-20 12:41:59.641] Talker ID : YX / Formatter : MWD
[2010-12-20 12:41:59.641] Talker ID : YX / Formatter : XDR
[2010-12-20 12:41:59.656] Talker ID : SD / Formatter : DPT
[2010-12-20 12:41:59.656] Talker ID : GP / Formatter : RMC
[2010-12-20 12:41:59.672] Talker ID : YX / Formatter : XDR
[2010-12-20 12:41:59.688] Talker ID : YX / Formatter : MTW
[2010-12-20 12:41:59.688] Talker ID : YX / Formatter : MWD
[2010-12-20 12:41:59.703] Talker ID : WM / Formatter : VTG
```

그림 4.3 데이터 저장 과정

Fig 4.3 Process of data storage

| | sen_no | time | talker_id | formatter | data |
|---|---|---|---|---|---|
| 1 | 494 | 2010-12-20 12:41:59.487 | YX | XDR_F | F,133.17,H,Frequency |
| 2 | 495 | 2010-12-20 12:41:59.487 | YX | MTW | 74.784,C |
| 3 | 496 | 2010-12-20 12:41:59.487 | GP | RMC | 123036,A,254.539,S,76.461,W,109.076,366.46,20100... |
| 4 | 497 | 2010-12-20 12:41:59.500 | YX | XDR_F | F,255.645,H,Frequency |
| 5 | 498 | 2010-12-20 12:41:59.517 | SD | DPT | 243.005,19.359,93.13 |
| 6 | 499 | 2010-12-20 12:41:59.517 | GP | RMC | 123036,A,40.051,S,102.795,W,70.082,177.39,201007... |
| 7 | 500 | 2010-12-20 12:41:59.533 | YX | XDR_F | F,22.77,H,Frequency |
| 8 | 501 | 2010-12-20 12:41:59.533 | YX | MTW | 85.116,C |
| 9 | 502 | 2010-12-20 12:41:59.547 | GP | RMC | 123036,A,172.782,S,34.419,W,37.118,602.98,201007... |
| 10 | 503 | 2010-12-20 12:41:59.547 | YX | XDR_F | F,220.8,H,Frequency |
| 11 | 504 | 2010-12-20 12:41:59.563 | YX | MTW | 111.561,C |
| 12 | 505 | 2010-12-20 12:41:59.563 | GP | RMC | 123036,A,2.979,S,93.093,W,77.184,394.2,20100722,1... |
| 13 | 506 | 2010-12-20 12:41:59.577 | YX | XDR_H | H,304.98,P,Humidity |
| 14 | 507 | 2010-12-20 12:41:59.593 | YX | MWD | 293.461,T,100.858,M,1.34,N,721.497,M |
| 15 | 508 | 2010-12-20 12:41:59.610 | HC | VHW | 230.02,T,47.083,M,88.708,N,93.568,K |

[ *Sentence* ]

| | sen_no | field_no | value |
|---|---|---|---|
| 1 | 494 | 1 | F |
| 2 | 494 | 2 | 133.17 |
| 3 | 494 | 3 | H |
| 4 | 494 | 4 | Frequency |
| 5 | 495 | 1 | 74.784 |
| 6 | 495 | 2 | C |
| 7 | 496 | 1 | 123036 |
| 8 | 496 | 2 | A |
| 9 | 496 | 3 | 254.539 |
| 10 | 496 | 4 | S |
| 11 | 496 | 5 | 76.461 |
| 12 | 496 | 6 | W |
| 13 | 496 | 7 | 109.076 |
| 14 | 496 | 8 | 366.46 |
| 15 | 496 | 9 | 20100722 |
| 16 | 496 | 10 | 708.1 |
| 17 | 496 | 11 | W |
| 18 | 496 | 12 | S |
| 19 | 497 | 1 | F |
| 20 | 497 | 2 | 255.645 |
| 21 | 497 | 3 | H |
| 22 | 497 | 4 | Frequency |

[ *Field* ]

그림 4.4 데이터 저장 결과

Fig 4.4 Result of data storage

그림 4.3과 그림 4.4의 시간을 비교하면 몇몇 데이터에 대해서 조금씩의 차이가 나는 것을 확인 할 수 있다. 이는 데이터가 데이터베이스에 저장되는 시간과 시스템 시간의 차이이며, 모듈 내에서 시스템 시간을 저장하더라도 데이터베이스에 저장되는 동안에 약간의 차이가 발생하게 된다. 그러나 데이터 검색 모듈에서의 검색 결과에는 영향을 미치지 않는다.

또한, 그림 4.3에서의 XDR은 그림 4.4에서는 XDR_C, XDR_F와 같은 형태로 저장이 되어 있다. 센텐스가 수신될 때는 그림 2.2에서와 같이 $__XDR로 전달된다. 그러나 그림 2.2에서 확인했듯 XDR은 여러 변환기의 정보를 한 번에 가지고 있기 때문에 XDR 형태 그대로 저장하게 되면 검색이 어렵게 된다. 따라서 검색의 효율성을 고려하여 이를 분리 후 저장하였으며, 저장된 변환기의 구별을 위하여 XDR 뒤에 각 변환기의 타입을 붙인 것이다.

한편, 임의로 발생시킨 데이터뿐만 아니라, PowerEdge 서버에 실제로 수심-수온 측정 장비(DST100) 및 풍향-풍속 측정 장비(PB200)을 연결하여 시험한 결과도 정상적으로 저장되는 것을 확인하였다.

### 4.2.2 데이터 샘플링 모듈

데이터 샘플링 모듈은 그림 4.5와 같은 사용자 인터페이스를 갖는다.

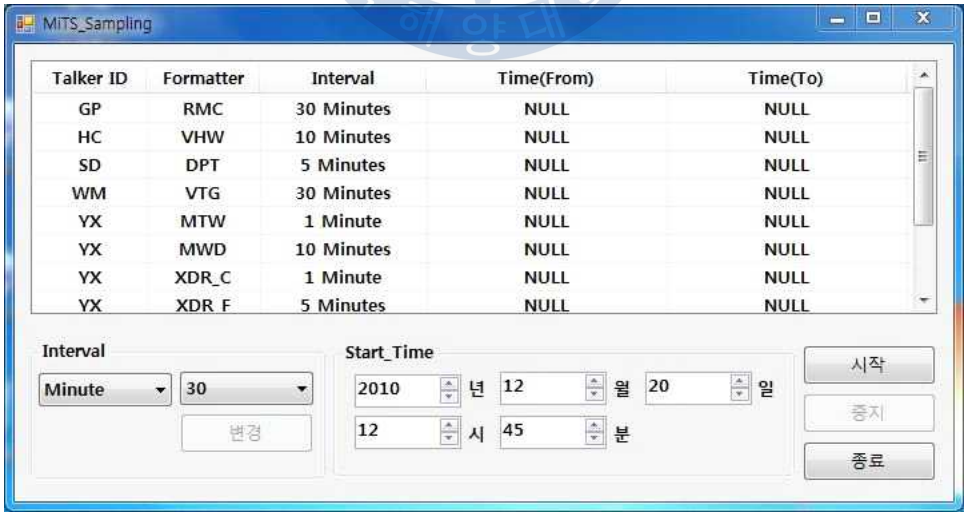


그림 4.5 데이터 샘플링 모듈의 사용자 인터페이스

Fig 4.5 User interface of data sampling module

샘플링을 하기 이전에 사용자가 각 센텐스 포맷터별로 간격을 조정 가능하며, 샘플링 수행 시작 시간도 지정 가능하다.

또한, 그림 4.5에서 샘플링을 시작하면 Time(From)은 샘플링 시작 설정 시간으로, Time(To)는 Time(From)의 값에 각각의 간격을 더한 시간으로 초기화되며, Time(To)의 시간보다 이후에 샘플링 한다. 그리고 샘플링이 진행된 시간으로 Time(From)을 바꾸고 Time(From)에 간격을 다시 더하여 Time(To)로 초기화한다. 이후 다음 센텐스 포맷터에 대하여 데이터 샘플링이 가능한지 확인한다.

그림 4.6은 데이터 샘플링이 일정 시간 동안 진행된 후의 결과를 나타내고 있다.

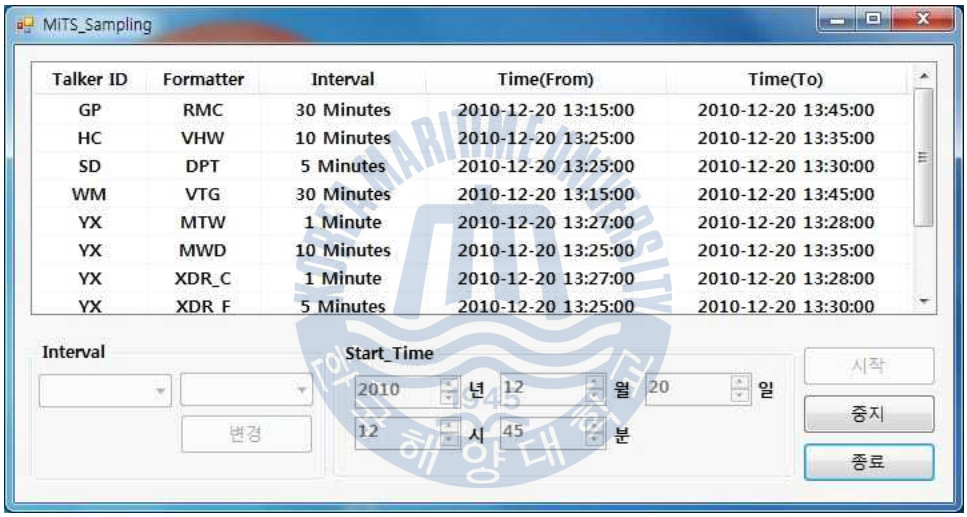


그림 4.6 데이터 샘플링 모듈의 동작

Fig 4.6 Process of data sampling module

한편, 이렇게 샘플링 된 결과는 데이터베이스에 저장된다. 1차 데이터베이스에서 샘플링 한 데이터는 2차 데이터베이스에 저장되며, 2차 데이터베이스의 구조는 1차 데이터베이스의 구조와 일치한다. 그림 4.7은 2차 데이터베이스의 *Sentence* 테이블을 나타낸 것으로서 그림 4.6에서 설정한 간격으로 샘플링을 한 것을 확인 가능하다.

| Formatter | Interval |
|---|---|
| RMC | 30 Minutes |
| VHW | 10 Minutes |
| DPT | 5 Minutes |
| VTG | 30 Minutes |
| MTW | 1 Minute |
| MWD | 10 Minutes |
| XDR_C | 1 Minute |
| XDR_F | 5 Minutes |

| | sen_no | time | talker_id | formatter | data |
|---|---|---|---|---|---|
| 1 | 1 | 2010-12-20 12:45:00.000 | YX | MTW | 120.048,C |
| 2 | 2 | 2010-12-20 12:45:00.203 | YX | XDR_C | C,183.885,C,tempeerature |
| 3 | 3 | 2010-12-20 12:46:00.013 | YX | MTW | 33.825,C |
| 4 | 4 | 2010-12-20 12:46:00.043 | YX | XDR_C | C,149.73,C,tempeerature |
| 5 | 5 | 2010-12-20 12:47:00.010 | YX | MTW | 58.917,C |
| 6 | 6 | 2010-12-20 12:47:00.103 | YX | XDR_C | C,264.96,C,tempeerature |
| 7 | 7 | 2010-12-20 12:48:00.023 | YX | MTW | 42.804,C |
| 8 | 8 | 2010-12-20 12:48:00.023 | YX | XDR_C | C,194.925,C,tempeerature |
| 9 | 9 | 2010-12-20 12:45:00.063 | SD | DPT | 236.698,124.28,28.006 |
| 10 | 10 | 2010-12-20 12:49:00.050 | YX | MTW | 114.267,C |
| 11 | 11 | 2010-12-20 12:49:00.050 | YX | XDR_C | C,95.565,C,tempeerature |
| 12 | 12 | 2010-12-20 12:45:00.047 | YX | XDR_F | F,151.455,H,Frequency |
| 13 | 13 | 2010-12-20 12:50:00.003 | YX | MTW | 28.29,C |
| 14 | 14 | 2010-12-20 12:50:00.517 | YX | XDR_C | C,119.715,C,tempeerature |
| 15 | 15 | 2010-12-20 12:51:00.077 | YX | MTW | 7.872,C |
| 16 | 16 | 2010-12-20 12:51:00.017 | YX | XDR_C | C,186.99,C,tempeerature |
| 17 | 17 | 2010-12-20 12:52:00.047 | YX | MTW | 89.913,C |
| 18 | 18 | 2010-12-20 12:52:00.030 | YX | XDR_C | C,28.98,C,tempeerature |
| 19 | 19 | 2010-12-20 12:53:00.073 | YX | MTW | 67.65,C |
| 20 | 20 | 2010-12-20 12:53:00.137 | YX | XDR_C | C,97.29,C,tempeerature |
| 21 | 21 | 2010-12-20 12:45:00.063 | HC | VHW | 323.512,T,197.653,M,12.328,N,323.833,K |
| 22 | 22 | 2010-12-20 12:50:00.207 | SD | DPT | 130.963,58.794,36.716 |
| 23 | 23 | 2010-12-20 12:54:00.023 | YX | MTW | 18.696,C |
| 24 | 24 | 2010-12-20 12:45:00.017 | YX | MWD | 245.231,T,100.619,M,60.568,N,10.234,M |
| 25 | 25 | 2010-12-20 12:54:00.010 | YX | XDR_C | C,211.83,C,tempeerature |
| 26 | 26 | 2010-12-20 12:50:00.207 | YX | XDR_F | F,239.43,H,Frequency |
| 27 | 27 | 2010-12-20 12:45:00.000 | YX | XDR_H | H,238.05,P,Humidity |
| 28 | 28 | 2010-12-20 12:45:00.017 | YX | XDR_P | P,20.355,P,Pressure |
| 29 | 29 | 2010-12-20 12:55:00.053 | YX | MTW | 26.199,C |
| 30 | 30 | 2010-12-20 12:55:00.850 | YX | XDR_C | C,144.21,C,tempeerature |

그림 4.7 데이터 샘플링 결과

Fig 4.7 Result of data sampling

그림 4.7은 데이터 샘플링이 된 2차 데이터베이스의 *Sentence* 테이블 결과를 보여준다. 그림 4.6의 센텐스 포맷터 및 간격과 비교하였을 때 각 포맷터의 간격마다 샘플링 되고 있음을 확인할 수 있다.

### 4.2.3 데이터 삭제 모듈

데이터베이스의 용량 초과를 방지하기 위하여 설계된 데이터베이스 삭제 모듈은 데이터베이스의 용량이 일정량 증가할 때마다 로그를 기록하도록 한다. 모듈이 실행되면 로그 파일에는 데이터베이스의 최대 용량과 현재 용량이 기록되며, 계속 용량을 확인하여 일정량이 증가할 때마다 시간과 함께 표시한다. 그리고 데이터베이스에 일정량의 데이터가 저장된 후 삭제할 때에도 삭제 대상이 되는 데이터의 최초 시간과 최종 시간을 기록한다.

로그 파일의 정보는 그림 4.8을 통하여 확인 가능하다.

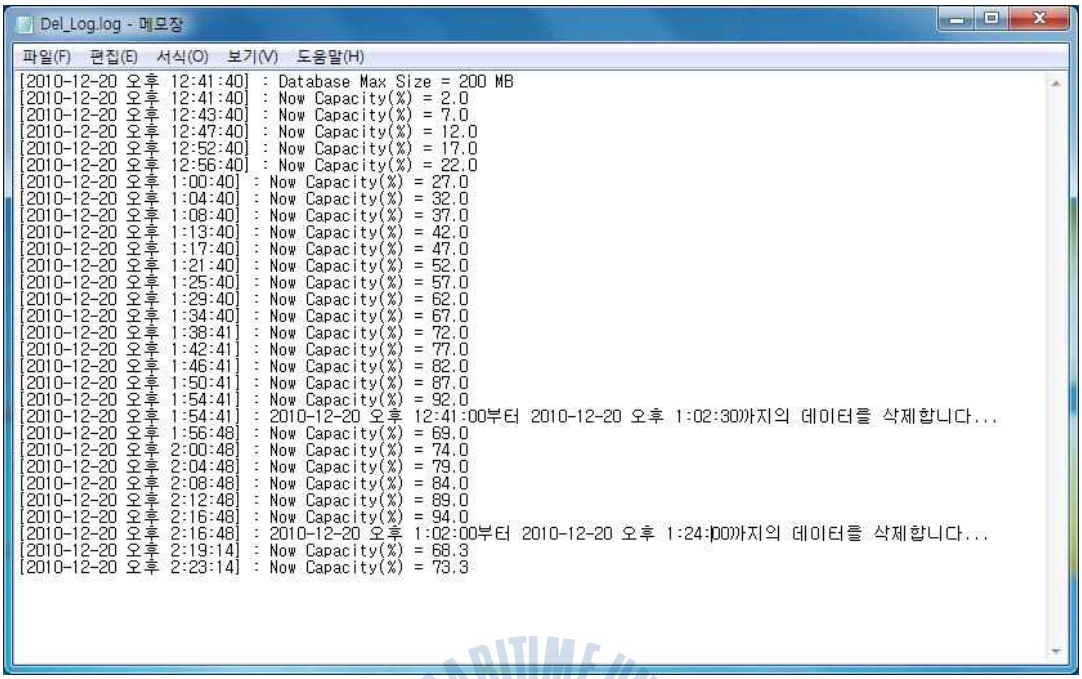


그림 4.8 데이터 삭제 모듈의 로그 파일

Fig 4.8 Log file of data deletion module

로그 파일의 가장 상단에는 데이터베이스 현재 최대 용량이 기록되어 있으며, 10MB, 즉 최대 용량이 200MB이므로 5%씩의 데이터가 저장 될 때마다 현재 시간 및 데이터베이스의 용량이 기록되도록 설정하였다.

한편, 데이터베이스의 용량이 90% 이상이 되면 데이터는 삭제되며, 데이터가 삭제되는 과정은 그림 4.9 및 표 4.1과 같다. 그림 4.9는 데이터가 삭제되는 과정을 흐름도로 나타낸 것이다.

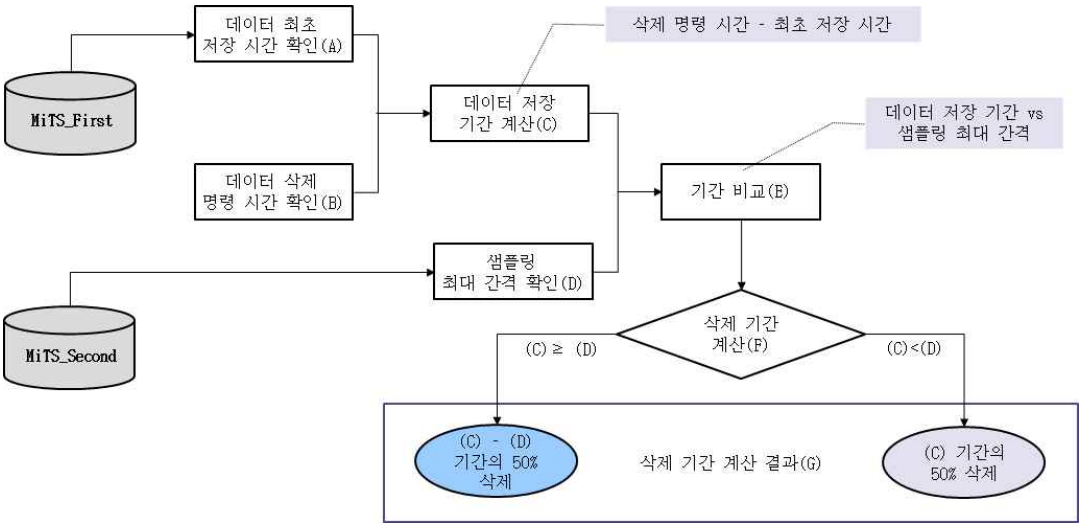


그림 4.9 삭제되는 데이터 기간 계산의 흐름도

Fig 4.9 Flow chart of deleted data scale calculation

표 4.1 삭제되는 데이터 기간의 계산

Table 4.1 Calculation of deleted data scale

| 항목 | 예시 | 용도 |
|---|---|---|
| 데이터 최초 저장 시간(A) | 2010년 12월 20일 오후 12시 41분 00초 | |
| 데이터 삭제 명령 시간(B) | 2010년 12월 20일 오후 1시 54분 00초 | |
| 데이터 저장 기간 계산(C) | 1시간 13분 00초 | |
| 샘플링 최대 간격(D) | 30분 | |
| 기간 비교(E) | 데이터 저장 기간 > 샘플링 최대 간격 | |
| 삭제 기간 계산(F) | 43분 * 0.5 = 21분 30초 | 50% 삭제 |
| 삭제 기간 계산 결과 | 2010년 12월 20일 오후 12시 41분 00초<br>~ 2010년 12월 20일 오후 1시 2분 30초 | |

그림 4.9에서 데이터베이스의 용량이 전체의 90% 이상이 되면 데이터 삭제 모듈은 1차 데이터베이스에 저장된 데이터 중 최초 저장 시간과 삭제 명령이 발생한 시간을 계산하여 총 데이터가 저장된 시간을 계산한다. 그리고 이 시간은 샘플링 최대 간격과 비교하는데, 데이터 저장 전체 기간이 샘플링 기간보다

크면 샘플링 기간 외의 데이터 중 50%를 삭제하고, 그렇지 않으면 전체 기간의 50%를 삭제한다. 표 4.1은 데이터가 삭제되는 기간을 계산하는 예를 그림 4.8과 그림 4.9를 참조하여 나타낸 것으로서, 데이터 저장 전체 기간이 샘플링 간격보다 크기 때문에 샘플링 간격을 제외한 나머지 기간에서 50%를 삭제한 결과를 나타낸 것이다.

### 4.2.4 데이터 검색 모듈

장비들을 통하여 발생된 데이터가 데이터베이스에 저장되는 것은 4.2.1절에서 확인하였으며, 이 데이터는 특정 조건에 따라 검색되어야 한다. 이를 위하여 3.2.4절에서 설계한 것을 바탕으로 구현된 사용자 인터페이스는 그림 4.10과 같다.

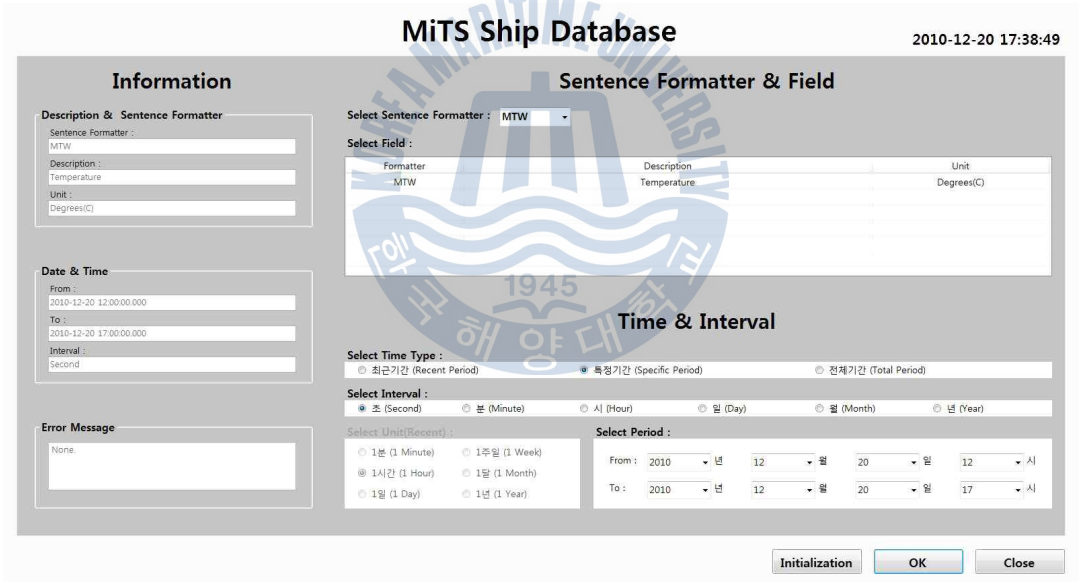


그림 4.10 데이터 검색 모듈의 사용자 인터페이스
Fig 4.10 User interface of data retrieval module

다음은 그림 4.10의 인터페이스를 각 기능에 따라 세부적으로 나눈 것을 나타낸다.

1) 센텐스 포맷터 및 타입 필드 설정

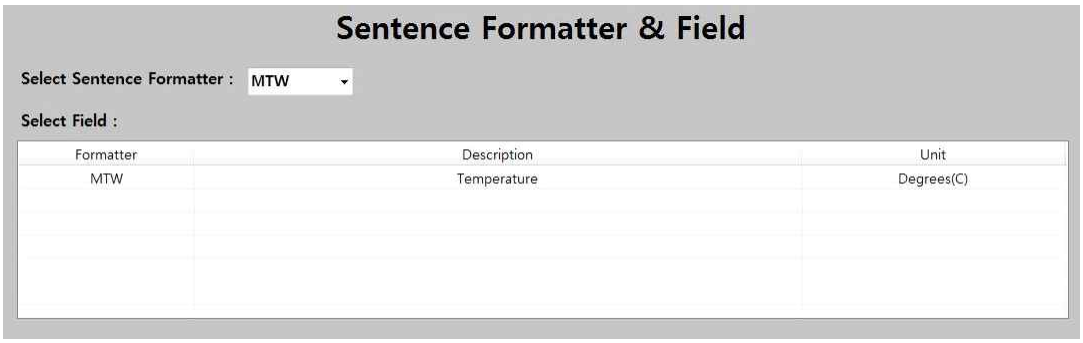


그림 4.11 센텐스 포맷터 및 타입 필드의 설정
Fig 4.11 Setting of sentence formatter & type field

사용자는 센텐스 포맷터 및 타입 필드를 선택하여 어떤 데이터를 검색할 것인가에 대한 선택을 할 수 있는데, 그림 4.11은 MTW를 선택하여 수온을 검색하도록 설정한 것이다.

2) 시간 및 간격 설정

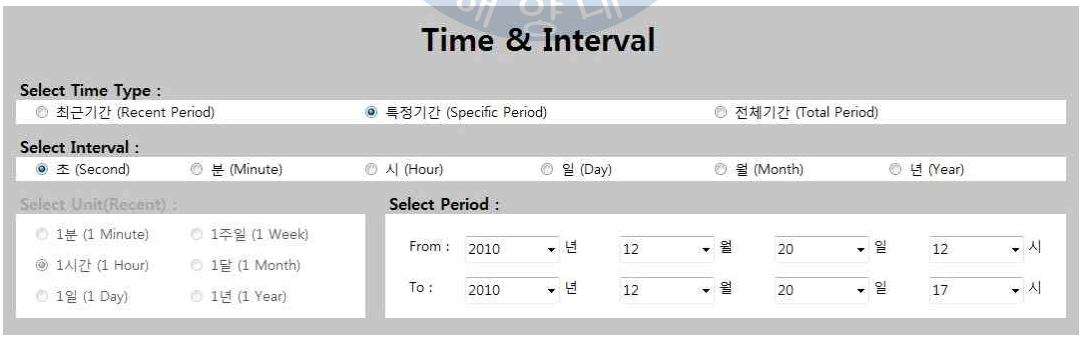


그림 4.12 시간 및 간격의 설정
Fig 4.12 Setting of time & interval

1)에서 센텐스 포맷터 및 타입 필드 설정과는 별도로, 검색 기간에 대한 설정이 가능하며 검색 기간 및 데이터의 간격을 설정 가능하다. 검색 기간의 경

우 최근 기간, 특정 기간, 전체 기간에 대하여 설정이 가능하며 간격의 단위는 초, 분, 시, 일, 월, 년 등의 단위로 검색 가능하다.

3) 정보 확인

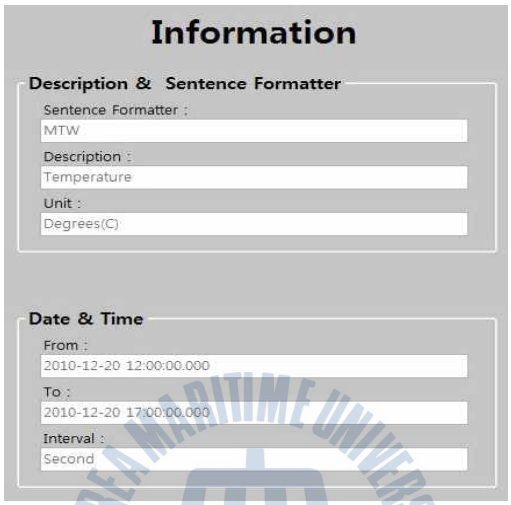


그림 4.13 사용자에 의하여 선택된 정보
Fig 4.13 Information selected by user

앞의 1)과 2)에서 선택한 정보에 대해서 확인 가능한 부분이다. 사용자는 자신이 어떤 값을 선택했는지, 최근 기간 및 전체 기간의 경우 언제부터 언제까지 데이터가 검색되는지에 대한 정보를 얻을 수 있다.

특정 조건을 주어 조건을 검색하였을 때 데이터의 시간적 변화는 그림 4.14와 같은 그래프로 그려지는 것을 확인할 수 있다. 또한 컴퓨터의 사양에 따라 약간의 차이는 있으나 전반적으로는 큰 지연 없이 그려지는 것을 확인하였다.

그러나 한 번의 그래프에 그려지는 데이터의 양이 2만 개 이상이 될 경우 컴포넌트의 문제로 인하여 그래프가 쉽게 그려지지 않는 것도 확인하였다.

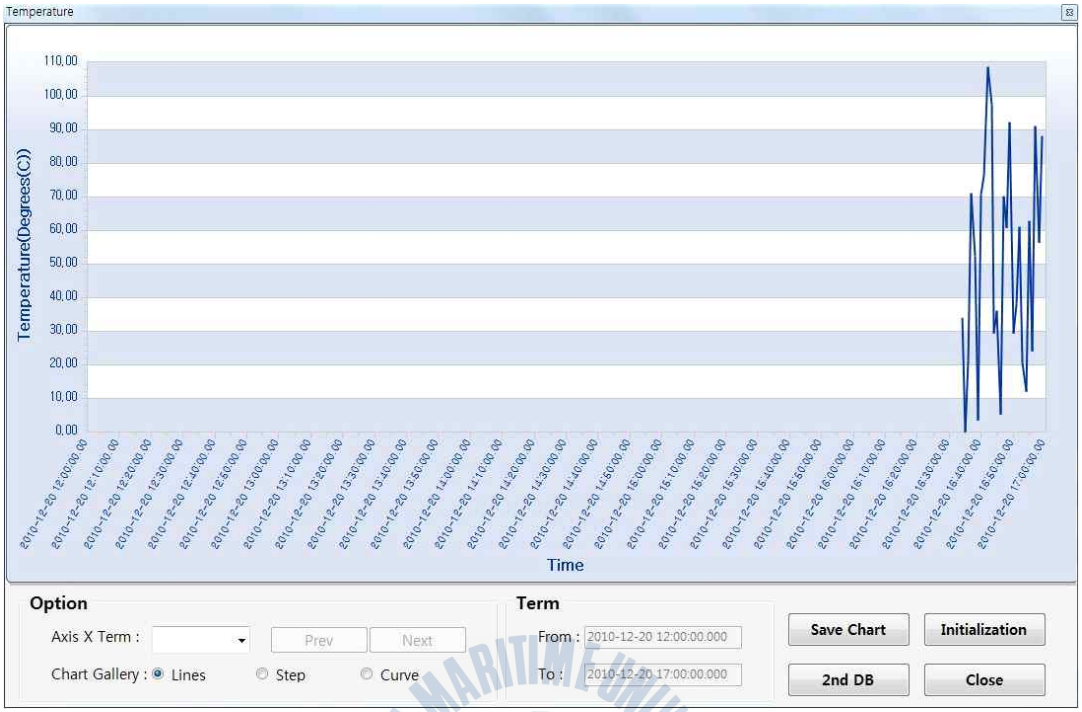


그림 4.14 1차 데이터베이스의 검색 결과

Fig 4.14 Retrieval result from 1st database

그림 4.14에서 5시간을 검색했음에도 불구하고 최근 30분 정도를 제외한 나머지 시간에 그래프가 출력되지 않는 이유는 1차 데이터베이스에 데이터가 존재하지 않기 때문이다. 따라서 이러한 경우에 사용자는 2차 데이터베이스를 통하여 더 많은 데이터를 검색 가능하며, 2차 데이터베이스를 검색한 결과는 그림 4.15와 같다.

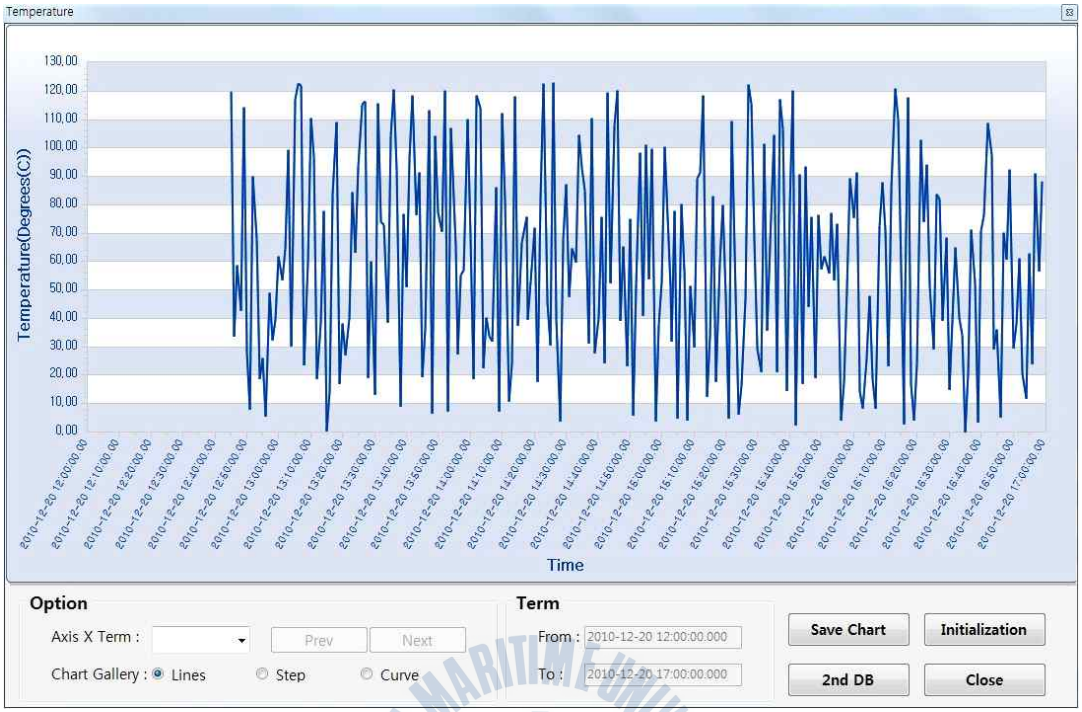


그림 4.15 2차 데이터베이스의 검색 결과

Fig 4.15 Retrieval result from 2nd database

그림 4.14와 그림 4.15를 비교한 결과, 1차 데이터베이스에서 존재하지 않던 부분의 데이터가 출력 된 것을 확인할 수 있다. 그러나 12시 40분 이전의 데이터는 그림 4.15에서도 출력되지 않았으며, 이러한 경우 처음부터 1차 데이터베이스에 데이터가 존재하지 않았거나 혹은 데이터 샘플링 모듈이 동작하지 않아 샘플링이 되지 않은 채 삭제되었음을 의미한다.

한편, 그림 4.14 및 그림 4.15의 그래프로는 각 시간의 상세한 값을 알기 어렵다. 따라서 X축의 검색 간격을 조절하면 그림 4.16과 같은 그래프 검색이 가능하다.

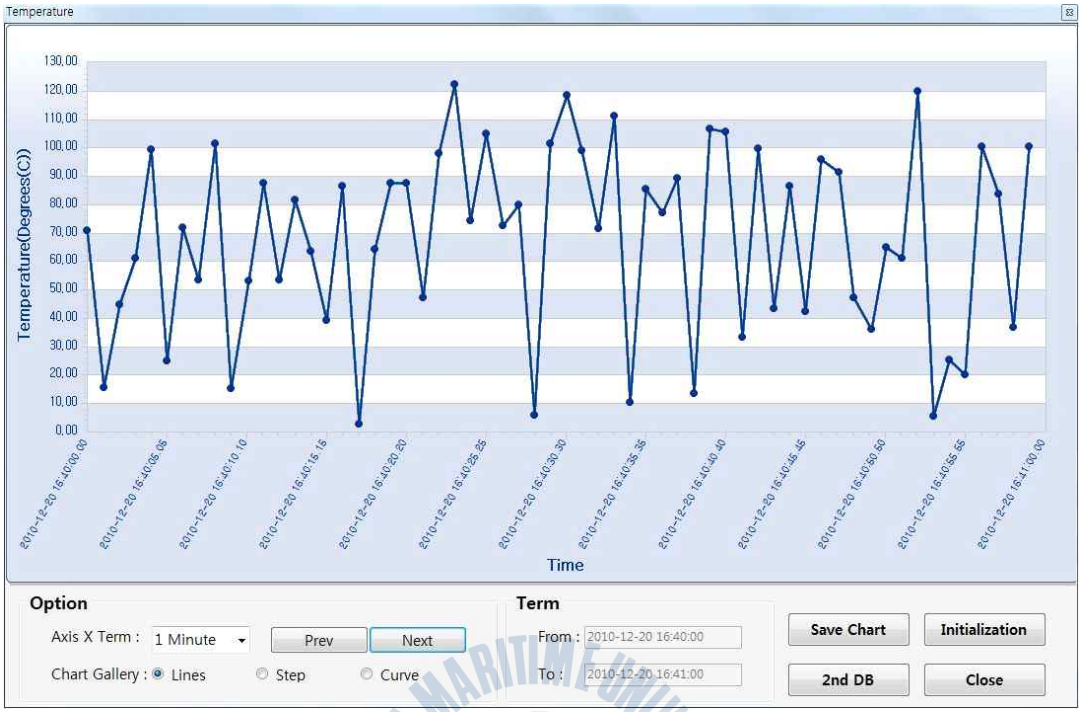


그림 4.16 시간 간격 조절 결과

Fig 4.16 Result of time scale regulation

그림 4.14 및 그림 4.15에서 X축의 전체 간격은 5시간이었다. 그러나 옵션에서 X축의 간격을 1분 단위로 조절한 결과 그림 4.16에서는 좀 더 상세하게 값을 표시할 수 있는 것을 확인 할 수 있다.

# 제 5 장 결론 및 향후 과제

본 논문에서는 NMEA 0183 센텐스 형식으로 변환된 데이터를 통합 관리하기 위한 다단 데이터베이스를 설계하였다. 또한 이 설계를 바탕으로 구축된 다단 데이터베이스는 급격한 데이터의 발생 빈도에도 문제없이 동작하였으며, NMEA 0183 센텐스로 변환된 여러 데이터가 통합 관리되는 것을 확인하였다. 또한 이를 시험하기 위한 각종 모듈을 구현하였으며 장비를 직접 서버에 연결하여 얻은 데이터와 임의로 생성한 데이터를 가지고 시험하였다. 결과적으로 두 가지 경우 모두 정상적으로 데이터가 저장되는 것을 확인하였으며 샘플링 및 삭제 모듈을 통하여 데이터베이스의 용량이 초과되어 문제가 발생하기 전에 데이터를 샘플링 및 삭제하여 용량을 확보하는 것을 확인할 수 있었다. 한편, 데이터베이스에 저장된 데이터를 검색 모듈을 통하여 사용자가 원하는 정보를 검색할 수 있고 샘플링이 된 2차 데이터베이스를 검색한 결과도 출력 가능함을 확인하였다.

데이터 통합 관리를 위한 다단 데이터베이스를 설계하고 이를 관리하기 위한 각종 모듈을 구현한 결과, 현재의 시스템에 비하여 비용 측면에서 많은 절감을 할 수 있고 데이터 관리 측면에서는 특정 데이터 이외에도 다수의 데이터를 통합적으로 관리할 수 있는 것을 확인하였다.

향후 연구과제로는, 데이터의 샘플링 이외에도 사용자가 원하는 정보를 쉽게 얻을 수 있도록 데이터 요약을 하는 방법에 대한 연구가 필요하다. 또한, 본 논문에서 사용한 센텐스 포맷터 이외에도 여러 센텐스 포맷터를 통해서 데이터가 통합 관리가 가능한지에 대한 시험을 할 필요가 있다고 본다.

# 참고 문헌


[1] 최항섭, 박동호, 김기철, 진민정, “항해데이터의 유효성 및 무결성 검증을 위한 평가방법”, 2006년도 대한전기학회 하계학술대회 논문집, 대한전기학회, pp.1136-1137, 2006.

[2] 이창의, 김달용, 유영호, 신옥근, “NMEA 2000을 이용한 임베디드 선박 모니터링 시스템의 개발”, 한국마린엔지니어링학회지, 제33권, 제5호, pp. 746-755, 2009.

[3] 김기영, 송병호, 나승유, 류상진, 정민아, 이성로, “상황 인식 기반 디지털 선박 모니터링 시스템 구현”, 대한전자공학회 학술대회 논문집, 제33권, 제1호, 대한전자공학회, pp. 1533-1536, 2010.

[4] 임용곤, “IT 기반 ‘디지털 선박’ 국내 개발 본격화”, 월간해양한국, pp. 104-107, 2002.

[5] 박정호 , 진광자, 김재명, 유대승, 오문균, 임동선, “조선 IT 현황과 전망”,전자통신동향분석, 제25권, 제4호, 한국전자통신연구원, pp. 19-26, 2010.

[6] 김재동, 박수한, 김형진, 고성위, 정해종, “실습조사선의 종합정보통신망 시스템 구축에 관한 연구”, 한국해양공학회지, 제18권, 제6호, pp. 44-50, 2004.

[7] 황훈규, 김태종, 윤진식, 서정민, 박휴찬, 장길웅, 이장세, “선박 내 통합정보 서비스를 위한 미들웨어 서버의 서비스 모듈 설계 및 구현”, 한국마린엔지니어링학회지, 제34권, 제1호, pp. 141-146, 2010.

[8] 홍기용, 신승호, 송부석, “항해지원을 위한 해양환경정보 실시간 예보시스템 개발”,한국해양환경공학회지, 제8권, 제1호, pp. 45-52, 2005.

[9] 해양수산부, 국제해사기구(IMO) 제53차 항해안전전문위원회(NAV) 회의 결과, 2007.

[10] 이장세, 박휴찬, 장길웅, 이주형, 장남주, 이주영, 이부형, “선박 내 정보의 통합관리를 위한 정보 아키텍처”, 2009년도 공동학술대회논문집, 한국마린엔지니어링학회, 2009.

[11] 박휴찬, 이장세, 장길웅, 이정우, 정희섭, 박중현, 강순열, “선박에서의 통합 정보처리를 위한 시스템 아키텍처”, 2009년도 한국마린엔지니어링학회 공동학술대회 논문집, 한국마린엔지니어링학회, 2009.

[12] IEC 61162-4 : Maritime Navigation and Radio-communication Equipment and Systems - Digital Interfaces - Multiple Talkers and Multiple Listeners - Ship Systems Interconnection, 2001.

[13] 김태종, “선박에서의 통합 정보처리를 위한 메시지처리 시스템 설계 및 구현”, 한국해양대학교 대학원 컴퓨터공학과 공학석사 학위논문, 2010.

[14] IEC 61162-450 Ed.1 : Maritime Navigation and Radio-communication Equipment and Systems - Digital Interfaces - Part 450: Multiple Talkers and Multiple Listeners - Light-Weight Ship Systems Interconnection, 2010.

[15] National Marine Electronics Association, NMEA 0183 Version 4.0 : Standard for Interfacing Marine Electronic Devices, 2008.

[16] 서정민, 황훈규, 윤진식, 이성대, 박휴찬, 이장세, 장길웅, “선박에서 데이터의 통합 관리를 위한 데이터베이스 설계 및 구현”, 한국마린엔지니어링학회지, 제34권, 제8호, pp. 118-1194, 2010.

[17] National Marine Electronics Association, NMEA 2000 : Standard for Serial-Data Networking of Marine Electronic Devices, Version 1.2, 2004.

[18] 황훈규, 윤진식, 서정민, 이성대, 장길웅, 이장세, 박휴찬, “Light-Weight Ethernet 기반 MiTS 서비스 모듈 개발”, 한국마린엔지니어링학회지, 제34권, 제8호, pp. 1180-1187, 2010.

[19] 정성훈, 김병찬, 양규식, “해상환경에서 IEEE 802.16e의 RSSI 및 CINR 측정 분석”, 한국항해학회논문지, 제13권, 제6호, 2009.

# 감사의 글

제가 대학원에 입학한 것이 엊그제 같은데 벌써 2년이 지나 졸업을 앞두고 논문을 쓰고 있는 것에 대하여 정말 시간은 참 빨리 간다는 생각이 듭니다. 논문을 쓰면서 제가 얼마나 부족한지 많이 느꼈으며 그 부족함을 채우기 위하여 많은 분들의 도움을 받아 이렇게 부족하나마 논문을 마칠 수 있었던 것 같습니다. 이 논문이 작성되기까지 많은 도움을 주신 분들께 정말 감사드립니다.

먼저, 제가 대학원에 입학하기부터 이 논문을 쓰고 졸업하기까지, 편찮으신 중에도 많은 것을 가르쳐 주시고 논문 지도에도 신경 써서 해 주신 지도 교수님 박휴찬 교수님께 감사드립니다. 그리고 언제나 웃으며 인사를 받아주시고, 고민이 있을 때 많은 조언을 해 주신 류길수 교수님과 신옥근 교수님, 좀 더 많은 지식을 얻을 수 있도록 열정적으로 수업을 해 주셨던 김재훈 교수님, 바쁘신 와중에도 논문 지도에 많은 도움을 주셨던 이장세 교수님과 이서정 교수님, 손주영 교수님, 유영호 교수님께도 감사드립니다.

그리고 이 논문이 작성되기까지 많은 조언을 주셨던 이성대 박사님, 제가 6년간 쉬지 않고 학교를 다니면서 어려움이 없도록 여러 가지로 많은 도움을 주셨던 강군호 조교님과 김경언 조교님께도 감사드립니다.

또한 제가 데이터베이스 연구실에 있는 동안 다방면으로 도움을 준 김태종 형님, 홍흥락 형님, 김현 형님과, 같이 연구실에서 지내며 많은 도움이 된 신봉섭님, 김인규님, 박민진님, 이재승님께도 감사드립니다. 그리고 가까이 지냈던 자연어처리 연구실의 서형원 형님, 김형철 형님, 전길호 형님, 최명길 형님, 네트워크 연구실의 문성미 누님, 이수환 형님, 보안시뮬레이션 연구실의 박근우 형님, 윤진식 형님, 황훈규 형님, 지금은 학교에 안 계시지만 인공지능 연구실의 박종일 형님, 박상우 형님께도 감사드립니다.

이외에도 6년 동안 학교를 다니며 정말 많은 의지를 할 수 있었던 최규진님, 김혜연님, 남유림님, 이내리님, 김수진님, 최영인님, 최은지님, 조민희님, 권홍석님을 비롯한 여러 동기님들과, 같이 학부 때 수업을 듣고 졸업한 홍내영 형님, 옥인표 형님, 이현화 형님, 김길용 형님, 구태우 형님, 손호용 형님, 김법연 형님, 김민지 누님을 비롯한 여러 선후배님들께 감사드립니다. 그리고 시험 와중에도 시간

을 내서 논문 검토를 도와 준 김수현님, 한준희님, 전용규님과 학교를 다니는 동안 잘 따라주고 많은 도움이 되어 주었던 강성화님, 김민지님, 박현님, 최세나님, 이하나님, 홍은주님, 조아라님 그리고 제가 학교 다니던 6년 동안 많은 버팀목이 되어 주었던 아치셈틀의 현 회장 이성민님을 비롯한 여러 선후배님들께도 감사드립니다.

마지막으로 제게 언제나 신경 써 주시고 보살펴 주시지만 저는 아직 아무 것도 해 드리지 못한 부모님께 이 지면을 빌려 죄송스러움을 표합니다.