KRISTAL-III 사용자 매뉴얼

(Ver.3.1.1)

KNOWLEDGE RETRIEVAL IN SCIENCE & TECHNOLOGY AFFILIATED LITERATURES

GIIS/KISTI

KiSTi www.kisti.re.kr 한국과학기술정보연구원 Korea Institute of Science and Technology Information

본 문서는 KRISTAL-IRMS 에 대한 범용 사용자 매뉴얼이다.

If you have any question or comment for this manual, please mail to giis@kisti.re.kr.

목 차

5.5 데이터베이스 관리 예제 80

제 1 장 서론

KRISTAL[1]-III 는 한국과학기술정보연구원(Korea Institute of Science & Technology Information, KISTI)에서 개발한 정보검색관리시스템(Information Retrieval & Management System, IRMS)으로, 유닉스 기반에서 정보 검색 및 관리를 수행한다. KRISTAL-III IRMS 는 DBMS 를 별도로 사용하지 않고 KRISTAL-III 만으로 문헌정보의 관리 및 서비스를 제공할 수 있는 정보서비스시스템(Information Service System)을 지향한다.

1.1 KRISTAL-III 특징

KRISTAL-III 정보검색관리시스템(IRMS)의 주요 특징은 다음과 같다.

- 유니코드 기반

KRISTAL-III IRMS 는 유니코드 기반으로 데이터를 저장 및 관리하므로, 다국어 및 한글 고어, 중국어 등 국내, 국외의 다양한 분야에 이용될 수 있다.

- 서비스 데몬화

KRISTAL-III IRMS 는 서버-클라이언트 구조를 채택하고 있으며, 서비스 데몬을 통한 시스템 자원관리 및 다수사용자의 동시성을 지원한다.

- 일반 정보와 XML 문서에 적용 가능

일반 문서(Plain Document)와 더불어 XML 문서(Structured Document)도 수용한다. 일반 문서에 있어서, 정형 데이터, 비정형 데이터 및 대용량 바이너리 데이터(BLOB)를 모두 지원합니다. 특히, XML 문서의 경우 구조적 정보와 함께 전문을 처리함으로써 XML 데이터의 편집 및 관리기능을 지원하고, 특정 엘리먼트에 대한 색인정보를 이용하여 검색기능을 지원한다.

[1] KRISTAL = Knowledge Retrieval In Science&Technology Affilicated Literatures

- 다양한 색인 방법 지원

다양한 색인 방법을 지원함으로써, 사용자는 자신이 처리하고자 하는 데이터에 가장 적합한 색인 방식을 선택할 수 있고 가장 적합한 색인 결과를 생성함으로써 검색의 정확도와 재현율을 개선하였다. 한적(漢籍) 자료와 관련하여 한자의 이체자 색인, 독음 색인 등을 기본적으로 지원하므로 고문서 관리와 검색에 탁월한 성능을 발휘한다.

- 안정된 데이터베이스 관리

온라인 데이터 관리 기능 수행시 트랜잭션 처리를 지원함으로써 데이터관리의 안정성을 보장한다. 또한 백업 및 Migration 을 지원함으로써 데이터의 유동성을 높일 수 있다.

- 다양한 검색 방법

검색에 많은 지식을 갖고 있지 않은 일반 유저와 전문 검색을 원하는 전문 유저들을 모두 수용하기 위해서 벡터검색과 불리언 검색을 제공한다. 이러한 검색 방법은 일반 문서(Plain Document) 뿐만 아니라 구조문서(Structured Document)에서도 사용할 수 있다.

- 내장된 자동문서분류엔진 제공

KRISTAL-III 정보검색관리시스템은 예제기반 알고리즘을 수용한 문서분류엔진을 내장하고 있다. KRISTAL-III 의 문서분류엔진은 후학습에 기반한 문서범주화 결과를 제공하므로 문서관리와 병행하여 서비스를 수행할 수 있는 장점이 있다.

- 생물서열정보 지원 (Bio-KRISTAL)

KRISTAL-III 정보검색관리시스템은 유전자 염기서열과 단백질 아미노산 서열정보로부터 색인을 추출하고 검색할 수 있는 생물정보검색 기법을 제공한다. 기존의 BLAST 와는 달리 정보검색 알고리즘에 기반한 검색을 수행하므로 수십배 이상의 빠른 속도로 생물서열정보를 검색할 수 있다. 또한 KRISTAL 문서분류 엔진을 이용하여 단백질 서열의 분류에 활용할 수도 있다.

- 개발자 라이브러리 및 온라인 Q&A 지원

C++, JAVA 라이브러리와 함께 사용 예제를 제공하며, 온라인 게시판 및 Q&A 를 통한 개발자 교육 및 기술지원을 하고 있어 개발이 한층 쉬워진다.

1.2 사용자 매뉴얼 구성

본 매뉴얼은 다음과 같이 구성되어 있습니다.

1 장에서는 KRISTAL-III 에 대한 개괄적인 내용에 대해 설명합니다.

2 장에서는 KRISTAL-III 바이너리의 설치 및 운영 환경에 대하여 설명합니다.

3 장에서는 KRISTAL-III 데이터베이스의 물리적, 논리적 구조와 함께, 사용되는 용어들에 대해 설명합니다.

4 장에서는 KRISTAL-III 가 제공하는 색인형식 및 색인기가 사용하는 사용자 사전 등에 관하여 설명합니다.

5 장에서는 데이터베이스 생성 방법, 적재 및 관리에 대하여 설명합니다.

6 장에서는 KRISTAL-III 가 제공하는 검색 모델과 그것의 사용방법에 대하여 설명합니다.

7 장에서는 KRISTAL-III 데몬을 이용하여 데이터베이스를 운영하는 방법을 설명합니다.

8 장에서는 KRISTAL-III 에서 XML 을 처리하는 방법에 대해서 설명합니다.

9 장에서는 KRISTAL 데이터베이스 설계에 대해서 설명합니다.

제 2 장 설치 및 운영환경

본 장에서는 KRISTAL-III 설치를 위한 기본 환경, 바이너리 패키지의 설치방법 및 구성, 필요한 환경 설정에 대해서 설명한다.

2.1 기본 환경

다음은 KRISTAL-III 설치 및 사용을 위한 기본 환경으로서, KRISTAL-III 에서 기본적으로 소요 예상되는 자원사항을 명세한 것이다.

구분	설명
운영체제(OS)	SUN, Solaris, IBM, AIX, Linux
컴파일러(Compiler)	GNU G++ 등
최소 메모리(Memory)	512 MB
최소 디스크공간	500 MB

정확한 자원사항은 데이터의 크기 및 특성에 따라 다를 수 있다. KRISTAL-III 의 성능을 제대로 활용하기 위해서는 데이터의 특성에 맞는 시스템 구성을 해야 한다.

2.2 바이너리 패키지 설치

먼저 바이너리(Binary) 아카이브 파일의 압축을 푼다.

```
% zcat K2002.binary.tar.Z | tar xvf -
% cd K2002
% ls -F
      bin/   lib/        include/
```

KRISTAL-III 바이너리 배포 배키지가 포함하는 디렉토리와 파일들을 확인할 수 있다. 기본적으로 3 개의 디렉토리와 수십개의 파일들을 포함하는데, 각 디렉토리와 파일들에 대한 것은 다음에 설명한다.

- bin : KRISTAL-III 데이터베이스를 구축하고 서비스를 수행하기 위한 실행 파일들을 포함
- lib : KRISTAL-III 응용 프로그램 작성을 위한 각종 라이브러리를 포함
- include : KRISTAL-III 라이브러리에서 필요로 하는 헤더 파일들을 포함

다음은 각 디렉토리에서 포함된 기본 파일들에 대한 것이다.

디렉토리	파일	설명
bin	kristald	검색 및 온라인 문서관리를 위한 데몬 실행파일
	kristal_dbadmin	KRISTAL-III 데이터베이스 생성, 삭제, 적재를 위한 실행 파일
	kristal_stop	구동된 데몬을 종료
lib	libcom.a libclient.a	통신 라이브러리 클라이언트 API 를 포함하는 라이브러리
	libiconv.a libshare.a	인코딩 형태 변환 라이브러리 기본 공유 라이브러리
	libidx.a	색인어 추출을 위한 라이브러리
	libxerces-c.a	Xml 문서 파싱 라이브러리
	k_dic/*	색인어를 추출할 때 필요한 사전 파일들을 포함하는 디렉토리
include	client/*.h	KRISTAL-III 서버와 통신할 수 있는 API 정의

2.3 환경 설정

KRISTAL-III 는 공유 라이브러리를 이용한 바이너리 형태로 배포되기 때문에 제공하는 각종 실행 파일을 올바르기 작동시키기 위해서는 다음과 같은 환경 설정이 필요하다.

■ 공유 라이브러리에 대한 환경 설정

1) bash 의 경우

$ export KRISTAL_HOME=*KRISTAL_INSTALL_ROOT*

$ export LD_LIBRARY_PATH=$KRISTAL_HOME/lib

2) csh, tcsh 의 경우

$ setenv KRISTAL_HOME *KRISTAL_INSTALL_ROOT*

$ setenv LD_LIBRARY_PATH $KRISTAL_HOME/lib

■ 실행 파일 패스에 대한 환경 설정

1) bash 의 경우

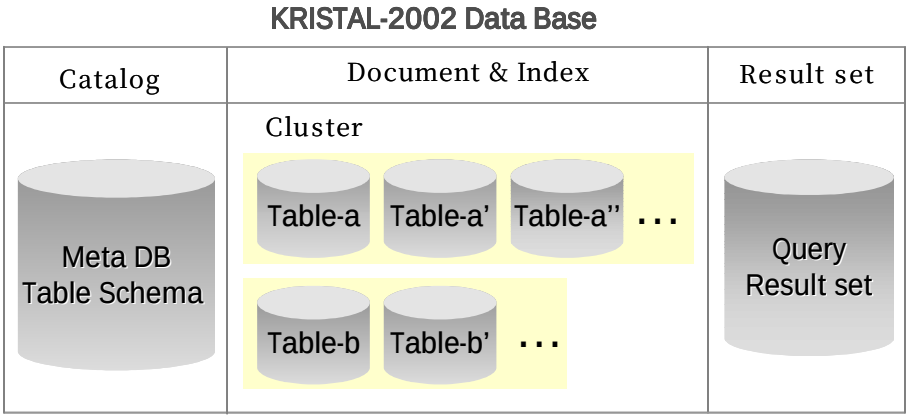
$ export PATH=$KRISTAL_HOME:$PATH

2) csh, tcsh 의 경우

$ set PATH=$KRISTAL_HOME:$PATH

제 3 장 데이터베이스 구조

KRISTAL-III 데이터베이스는 다수 사용자의 동시 검색(Concurrency) 및 온라인/오프라인 데이터 관리, 백업 그리고 회복(Recovery) 기능을 안정적으로 수행한다. 또한 검색엔진의 필수 조건인 대용량 데이터베이스에 대한 빠른 검색속도를 보장하기 위해 멀티쓰레드를 이용한 분산 검색을 수행한다. 아래 [그림 3-1]은 KRISTAL-III 데이터베이스의 구조를 보여준다.



[그림 3-1] KRISTAL-III 데이터베이스 구조

3.1 데이터베이스 구조

KRISTAL-III 데이터베이스는 기본적으로 목록(Catalog), 문서 및 색인(Document & Index) 그리고 결과 셋(Result Set) 데이터베이스로 구성된다. 이와 같이 구성된 데이터베이스는 KRISTAL-III 데이터베이스 시스템이 제공하는 기능을 수행할 수 있는 데이터 관리 서비스(Data Management Service)의 대상이 된다.

*목록 데이터베이스*는 테이블 속성에 관한 스키마(Schema)정보를 저장한다. 즉, 테이블 구조, 색인 방법, 주키(Primary Key), 불용어 등 데이터베이스에 관리에 필요한 전반적인 정보가 저장된다.

*문서 및 색인 데이터베이스*는 하나 또는 여러 개의 클러스터(Cluster)로 구성되며, 각각의 클러스터는 같은 스키마를 사용하는 한 개 이상의 테이블로 구성된다. 클러스터는 검색결과의 랭킹이 적용되는 단위이며, 동시 검색의 대상으로 사용될 수 있다. 문서 및 색인 데이터베이스 테이블은 문서, 주요키, 색인 데이터베이스로 구성되며 목록 데이터베이스에서 정의한 테이블 스키마 구조에 따라 구성된다.

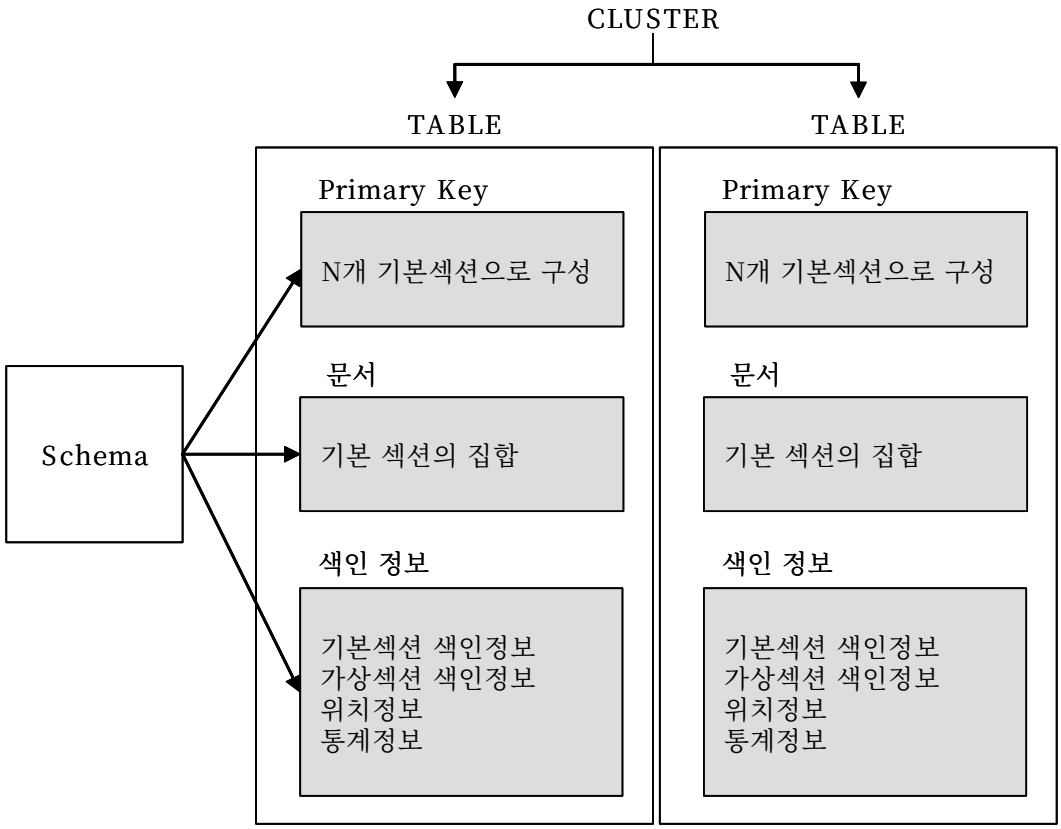
*결과 집합 데이터베이스*는 검색 질의에 대한 빠른 응답을 위해서 검색된 결과를 질의와 함께 저장하고 관리한다.

이후에 다시 설명하겠지만, KRISTAL-III 의 데이터베이스 구조는 테이블스키마(Table Schema)와 데이터베이스[2]가 일대다(1-n) 방식으로 맺어지는 관계를 취하고 있다. 또, 위에서 클러스터라는 용어를 사용했으나 KRISTAL-III 에서는 클러스터를 물리적으로 정의하지는 않는다. 다만 동일한 테이블 스키마를 가지는 2 개 이상의 테이블을 논리적으로 테이블 클러스터 또는 클러스터라고 부르는 점에 주의하기 바란다.

3.2 테이블 구조

KRISTAL-20002 데이터베이스는 문서 개체들의 집합인 테이블로 구성되어 있으며, 각 테이블은 주요키, 문서 그리고 색인 부분으로 나누어 진다.

[2] KRISTAL-III 에서 데이터베이스라는 용어는 광범위하게 사용되는 정의이다. 이 경우에는 데이터베이스라는 용어 대신 테이블이라는 용어를 사용하는 것이 정확하다. 데이터베이스는 때에 따라서는 하나의 기계에서 관리되는 KRISTAL-III 데이터 전체를 이를 수도 있고, 테이블 클러스터를 데이터베이스라 할 수도 있으며, 테이블 하나를 데이터베이스라 부를 수도 있다.



[그림 3-2] KRISTAL-III 테이블 구조

그 중에서 문서부분은 KRISTAL-III 데이터베이스의 최소검색 단위인 섹션들의 집합으로 구성 되며 원시 데이터에 해당한다. 그리고 주요키 부분은 하나이상의 기본섹션으로 구성 되며, 구성된 키들은 테이블에서 각 문서들을 구분할 수 있는 키로 사용될 수 있다. 마지막으로 색인 부분은 기본섹션과 가상섹션에 대한 색인정보, 위치정보 그리고 색인어 들에 대한 통계정보를 가지고 있다. 동일한 스키마를 갖는 테이블들은 하나의 클러스터로 관리되며, KRISTAL-III 의 최상위 검색대상이 된다. [그림 3-2]는 KRISTAL-III 테이블 구조를 나타낸다.

3.3 문서 구조

KRISTAL-III 에서는 처리대상의 기본 단위를 '문서(Document)'라 한다. 하나의 문서는 DBMS 의 레코드(record)에 해당하며, 각 문서는 하나 이상의 섹션들의 집합으로 이루어져있다. 섹션(Section)은 DBMS 레코드의 필드(Field)에 해당한다. KRISTAL-III 의 섹션은 섹션값의 처리 방식에 따라서 *기본섹션*(Basic Section), *가상섹션*(Virtual Section), *통합섹션*(Union Section)으로 분류된다.

1) 기본 섹션(Basic Section)

기본 섹션은 하나의 문서를 구성하는 최소 단위이며 또한 검색 과정에서 문서 검색의 범위를 지정하기 위한 기본단위로 사용된다. 예를 들면, '제목', '저자명', '저자의 소속 기관', '초록', '본문' 등은 모두 논문을 구성하는 기본 섹션이 될 수 있다. 그리고 사용자는 검색 범위를 '논문 제목'과 '초록'이라는 두 개의 기본 섹션에 한정시킴으로써 질의의 수행을 지시할 수 있다.

2) 가상 섹션(Virtual Section)

기본 섹션들에 대한 다양한 색인타입을 지원하기 위해 **물리적**으로 하나의 의미를 갖는 새로운 섹션을 구성할 수 있다. 이 섹션 형태를 가상 섹션이라고 한다. 예를 들어, '제목'의 기본 섹션에 대해 '제목 2', '제목 3' 의 가상섹션을 구성하고 각각을 부분 일치(INDEX_BY_MA), 완전 일치(INDEX_AS_IS)등 다양한 타입의 색인을 구성할 수 있다. 또한 다수의 섹션에 대한 빠른 검색 성능이 요구될 때 다수의 섹션을 마치 하나의 섹션처럼 통합하여 색인함으로서 검색속도를 향상시킬수 있다.

3) 통합 섹션(Union Section)

여러 개의 기본 섹션 및 가상섹션들을 결합하여 **논리적**으로 하나의 의미를 갖는 새로운 섹션을 구성할 수 있다. 이러한 섹션을 통합 섹션이라고 부르는데, 이것은 실제 색인 정보를 생성하지는 않지만 검색 관점에서 하나의 섹션 단위로 인식되는 가상의 섹션이다. 예를 들어, '주제목', '부제목', '소제목'의 기본 섹션 또는 가상섹션을 결합하여 '제목'이라는 결합 섹션을 구성할 수 있다. 통합섹션인 '제목' 섹션에 검색을 수행하면 '주제목','부제목','소제목'에서 수행된 결과를 통합하여 검색한다. KRISTAL-III 가 내

부적으로 여러 섹션의 묶음을 지원함으로써 서비스 개발자 및 검색 사용자는 편리하게 여러 섹션에 대한 검색을 동시에 사용할 수 있다.

3.4 섹션 데이터 형식

KRISTAL-III 의 기본 섹션의 내용은 고정길이 문자열, 가변길이 문자열, 숫자, 불리언 형식을 지원한다.

1) KSTRING

가변길이 문자열에 사용되며 최대크기는 시스템의 가용 메모리까지 가능하다. 대부분의 섹션에 대해서 적용가능한 데이터 타입이다.

2) KINT, KUINT, KFLOAT

숫자 데이터에 사용되며 각각 integer, unsigned int, float 를 의미한다. 숫자로만 이루어진 섹션에 대해서만 적용하여야 한다.

3) KBOOL

불리언 값에 사용되며 TRUE, FALSE 값을 가진다. 참/거짓의 두 가지 경우의 값만을 가지는 정보에 대해서 사용할 수 있다. 예를 들어 원문의 유무, URL 링크의 작동 유무 등과 같은 이진 정보를 표현할 수 있다.

4) KCHAR[N]

고정길이 문자열에 사용되며 최대 N 바이트 크기이다. 숫자, 문서의 제어번호 등과 같이 고정된 길이를 가지는 정보에 대해서 사용할 수 있다. KCHAR 형은 KSTRING 에 비해 문서의 수정 및 삽입 속도가 빠른 장점이 있다.

5) KBLOB

바이너리 데이터에 사용되며 최대크기는 시스템의 가용 메모리까지 가능하다.

3.5 데이터베이스 제한사항

KRISTAL-III 은 하나의 데이터베이스가 최대 65536 개의 테이블을 가질 수 있고 하나의 테이블은 2^{32} / (가변길이 섹션 수 + 1)개의 레코드를 가질 수 있다. 레코드의 섹션 개수나 크기에는 제한이 없고, 고정길이 및 가변길이 섹션을 지원한다. 그리고 하나의 테이블 섹션의 최대 유일한 색인어 수는 2^{32} 개이다.

제 4 장 색인(Indexing)

문서 색인(document indexing)이란 입력 문서를 분석하여 그 문서를 효과적으로 표현하는 색인어를 추출하는 과정이다. 입력 문서를 구성하는 다양한 유형의 어절 및 문자열을 그 형태에 따라 구분하여 그 유형에 맞는 분석 방법을 선택하게 된다. 따라서 KRISTAL-III 의 색인 시스템은 다양한 분석 기능을 포함하고 있으며 이는 통합적으로 운영된다.

KRISTAL-III 에 탑재된 색인 시스템의 기능은 (1) 한글 형태소 분석 기능, (2) 영어 스테밍(Stemming) 기능, (3) 한자 변환 기능, (4) 불용어 관리 기능, (5) 사용자 사전 기능 등이 있다. 또한 다양한 색인형태를 제공함으로써 사용자가 보다 유연하게 색인 기능을 활용할 수 있도록 하였다. 더불어 색인 시스템은 독립된 라이브러리 형태로 KRISTAL-III 배포판에 이식되어 있으므로 사용자가 원한다면 색인 API 를 직접 호출하여 사용할 수도 있다.

4.1 색인 형식

사용자의 편이성과 유연성을 위하여 다양한 형태의 색인 형식을 제공한다. 또한 한자변환 기능을 옵션으로 처리하여 각 색인 타입마다 한자에 대한 처리를 지정할 수 있도록 하였다. 영어에 대한 스테밍 여부도 옵션으로 처리하였다.

4.1.1 INDEX_AS_IS

INDEX_AS_IS 는 문서에서 섹션의 내용 또는 값 전체를 하나의 색인어로 추출한다. 그렇게 함으로써 그 섹션에 대한 완전 일치(Exact Matching)의 검색을 지원한다. INDEX_AS_IS 로 색인된 섹션에 대해 단지 문자열 값의 일치(Character String equality)여부만을 확인하는 수준에서의 검색을 수행한다. 관계연산자(<, <=, >, >=, -)를 이용한 문자열 값의 비교는 가능하다. INDEX_AS_IS 색인 방식은 '제어번호'와 같이 레코드를 구별할 수 있도록 유일하게 부여된 기본 섹션에 주로 사용된다.

예 1) 제 목 : “정보검색에 관한 연구”

색인어 : “정보검색에 관한 연구”

예 2) 제 목 : “情報檢索에 관한 硏究”

색인어 : “情報檢索에 관한 硏究”

예 3) 제어번호 : “AN00012”

색인어 : “AN00012”

앞서 설명한 바와 같이 INDEX_AS_IS 는 섹션값 전체를 하나의 색인어로 취급하므로 데이터베이스 관리를 위한 제어번호 등과 같이 섹션값이 단일값을 가지는 경우에 대해서만 사용할 것을 권장한다. 예 1 과 예 2 의 경우에는 완전일치로 검색을 수행해야 하므로 제목을 INDEX_AS_IS 로 색인할 경우 검색이 매우 불편해지는 단점이 있다. 따라서 한글 제목과 같은 경우에는 향후 설명할 INDEX_BY_MA 를 사용할 것을 권장한다. 영어 제목에 대해서는 INDEX_BY_TOKEN 을 사용할 수 있다.

4.1.2 INDEX_BY_TOKEN

INDEX_BY_TOKEN 은 텍스트 검색과 같이 내용 기반의 부분 일치(Partial matching) 검색을 지원해야 하는 섹션에 적용 가능하다. INDEX_BY_TOKEN 색인 방식은 섹션의 내용 또는 값 전체를 색인어로 사용하는 것이 아니라, 섹션내의 어절 또는 단어들 중에서 색인어를 선정한다. INDEX_BY_TOKEN 은 섹션에서 불용어를 제외한 어절 또는 단어들을 색인어로 추출하는 초보적인 색인 방식으로, 별도의 후처리를 수행하지 않고

원문에 나타난 형태 그대로를 색인어로 사용한다. 따라서 이 방식은 영어로만 작성된 섹션, ‘사람 이름’이나 ‘지명’과 같은 고유명사들을 주로 포함하고 있는 섹션이나 논문의 ‘키워드 리스트’와 같이 별다른 후처리가 필요 없는 섹션들에 적합하다.

HANJA2HANGUL (FALSE)

예) 제 목 : “정보검색에 관한 연구”

색인어 : “정보검색에”, “관한”, “연구”

예) 제 목 : “情報檢索에 관한 硏究”

색인어 : “情報檢索”, “에”, “관한”, “硏究”

HANJA 2HANGUL(TRUE)

예) 제 목 : “정보검색에 관한 연구”

색인어 : “정보검색에”, “관한”, “연구”

예) 제 목 : “情報檢索에 관한 硏究”

색인어 : “정보검색에”, “관한”, “연구”

KRISTAL 의 INDEX_BY_TOKEN 방식 색인에서는 구분자를 지원한다. 즉 특수한 구

분자를 지정하여 원하는 값을 색인할 수 있다. 분야별 분류, 저자 섹션 등에 유용하게 사용할 수 있다. 다음은 그 예를 보여주고 있다.

DELIMIT-CHARS(“;”)

예) 분 야 : 전산학/정보검색;전산학/문서분류;

색인어 : “전산학/정보검색”, “전산학/문서분류”

설명: “;” 문자로 구분하여 색인을 추출한다.

DELIMIT-CHARS (“/”), DELETE-CHARS (“,. ”)

예) 저 자 : 홍길동/Smith, J./허 균

색인어 : “홍길동”, “SmithJ”, “허균”

설명 : “/” 단위로 토큰 추출후, 마침표(.), 쉼표(,), 공백( )을 제거한다.

구분자 및 삭제 문자 지정은 기본 섹션이나 가상 섹션 정의시에 옵션으로 지정할 수 있다. 위의 예에서 보여주는 것처럼 구분자 속성인 delimit-chars 를 이용하면 사용자가 지정하는 단위로 토큰을 추출하여 색인할 수 있다. 또한 추출된 토큰 내의 특수한 문자를 제거하고자 할 경우에는 삭제문자 지정자 속성인 delete-chars 를 사용할 수 있다.

4.1.3 INDEX_BY_MA

INDEX_BY_MA 는 INDEX_BY_TOKEN 의 색인 방식에서 한 단계 더 나아가 한글 텍

스트의 색인을 위해 한글 형태소 해석기(Morphological analyzer) 를 사용하며 영어 텍스트의 경우에는 옵션으로 스테머(stemmer)를 이용한다. 한글 텍스트의 각 어절에 대해 형태소 해석을 수행함으로써 명사, 조사, 접미사, 동사, 형용사 등의 최소 형태소 단위를 구분한 후, 섹션의 내용을 대표할 수 있는 단순 명사를 색인어로 추출한다. 그리고 영어의 경우에는 규칙적인 복수형 및 동사의 시제 변화 등을 처리하여 어간을 추출한다. 따라서 이 방식은 단순한 INDEX_BY_TOKEN 의 어절 단위 색인보다 양질의 색인을 수행한다. 따라서 INDEX_BY_MA 는 '논문 제목'이나 '초록'과 같은 기본 섹션에 적합하다.

HANJA2HANGUL (FALSE)

예) 제 목 : "정보검색에 관한 연구"

색인어 : "정보", "검색", "정보검색", "연구"

예) 제 목 : "情報檢索에 관한 硏究"

색인어 : "情報檢索", "硏究"

HANJA 2HANGUL(TRUE)

예) 제 목 : "정보검색에 관한 연구"

색인어 : "정보", "검색", "정보검색", "연구"

예) 제 목 : "情報檢索에 관한 硏究"

색인어 : "정보", "검색", "정보검색", "연구"

STEMMING (FALSE)

예) 제 목 : “Information Systems”

색인어 : “information”, “systems”

STEMMING (TRUE)

예) 제 목 : “Information Systems”

색인어 : “informat”, “system”

INDEX_BY_MA 에서의 스테밍(Stemming) 옵션은 재현율을 높이기 위한 방편으로 제공된다. 재현율이 중요한 검색서비스에서는 스테밍을 사용할 수 있으나 일반적으로 스테밍은 정확도를 떨어뜨리는 단점이 있음을 상기하기 바란다.

섹션의 내용이 영어로만 구성된 문서에 대해서는 INDEX_BY_MA 보다는 INDEX_BY_TOKEN 형식의 사용을 권장한다. INDEX_BY_MA 는 한글 형태소를 분석하기 위한 작업을 수행하기 때문에 INDEX_BY_TOKEN 방식에 비해서 색인 속도가 느리다는 단점이 있다.

4.1.4 INDEX_BY_CHAR

INDEX_BY_CHAR 는 기본 섹션에서 영어인 경우에는 한 음절씩, 한글인 경우에는 두 음절씩을 추출하여 색인어를 생성한다. 이 방식은 한글 내용이 담긴 기본 섹션중에서 ‘사람 이름’과 같은 섹션에 주로 사용한다. 즉, 사람 이름이 ‘홍 길동’, ‘홍길동’, ‘홍 길 동’등과 같이 한글 띄어쓰기가 자유롭게 되어 있는 경우 ‘홍’, ‘길’, ‘동’과 같이 음절

단위로 색인하여 저장하면, 사용자가 사람 이름을 검색할 경우 띄어쓰기에 관계없이 검색이 가능하도록 한다.

HANJA 2HANGUL(FALSE)

예) 이 름 : “홍 길동”

색인어 : “홍”, “길”, “동”

예) 이 름 : “洪 吉童”

색인어 : “洪”, “吉”, “童”

HANJA2HANGUL (TRUE)

예) 이 름 : “홍 길동”

색인어 : “홍”, “길”, “동”

예) 이 름 : “洪 吉童”

색인어 : “홍”, “길”, “동”

INDEX_BY_CHAR 는 모든 낱자에 대한 색인을 수행하므로 INDEX_BY_MA 나 INDEX_BY_TOKEN 등의 색인방식에 비해 수배 이상의 과도한 색인을 만들어낸다. 따라서 고문서 등과 같은 특수한 경우가 아니면 INDEX_BY_CHAR 의 사용을 피할 것을 권고한다. 특히 기본 글자의 숫자 적은 영어권 문서에서는 INDEX_BY_CHAR 를 사용해서는 안된다.

4.1.5 INDEX_AS_NUMERIC

INDEX_AS_NUMERIC 은 숫자로 구성된 기본 섹션에 대하여 색인어를 추출한다. INDEX_AS_NUMERIC 로 지정된 섹션은 단일의 값(atomic value)만을 갖는 것으로 가정하며, 이들 섹션에 대해서는 관계연산(<, <=, >, >=, =, -)이 가능하다. 이 색인 방식은 '날짜'와 같이 년월일 형태로 일정한 형식이 갖추어지고 숫자로 이루어진 섹션에 대해 주로 사용한다.

예) 날 짜 : "19961214"

색인어 : 19961214

4.1.6 INDEX_AS_IS_MA

INDEX_AS_IS_MA 는 앞서 설명한 INDEX_AS_IS 색인방식을 다소 변형시킨 방법과 INDEX_BY_MA 색인방식을 동시에 만족하는 색인 방식이다. 즉, 문서에서 섹션의 내용 전체를 색인어로 추출하고, 동시에 한글 텍스트의 각 어절에 대해 형태소 해석을 수행함으로써 명사, 조사, 접미사, 동사, 형용사 등의 최소 형태소 단위를 구분한 후, 섹션의 내용을 대표할 수 있는 단순 명사를 색인어로 추출한다. INDEX_AS_IS_MA 는 단행본의 제목과 같은 짧은 섹션에 사용해야 함에 주의하기 바란다.

HANJA2HANGUL (FALSE)

예) 제 목 : "정보검색에 관한 연구"

색인어 : "정보검색에 관한 연구", "정보", "검색", "정보검색", "연구"

예) 제 목 : "情報檢索에 관한 硏究"

색인어 : “情報檢索에 관한 硏究”, ”情報檢索”, “硏究”

HANJA2HANGUL (TRUE)

예) 제 목 : “정보검색에 관한 연구”

색인어 : “정보검색에 관한 연구”, “정보”, “검색”, “정보검색”, “연구”

예) 제 목 : “情報檢索에 관한 硏究”

색인어 : “정보검색에 관한 연구”, “정보”, “검색”, ”정보검색”, “연구”

INDEX_AS_IS_MA 색인방식은 단행본 제목 등과 같이 길이가 짧은 섹션의 완전일치 및 부분일치를 동시에 지원하기 위한 방안으로 사용할 수 있다. 그러나 완전일치를 통한 제목의 검색은 사용자의 입력으로는 어려운 일이다. 따라서 시스템이 자동으로 섹션값을 읽어서 완전일치를 통해 검색을 수행하는 경우가 아니라면 INDEX_AS_IS_MA 의 사용을 권장하지 않는다. 또한 INDEX_AS_IS_MA 를 요약, 설명 등과 같이 길이가 긴 섹션에서는 저장용량의 부하를 주기 때문에 사용하지 말아야 한다.

4.1.7 INDEX_BY_MIX_CHAR

INDEX_BY_MIX_CHAR 는 한자 변환 기능을 확장한 색인 기능으로서, 섹션 값 중에서 한자는 대응하는 모든 한글 음으로 변환하여 이를 색인어 집합에 추가하고 한글은 INDEX_BY_CHAR 와 같이 낱자 색인을 수행한다. 이것은 한자 한 음에 대해서 여러 개의 한글 음이 있을 수 있으므로 이를 처리하기 위한 기능이다(예: 樂 → 악, 락, 요). 또한 한자에 대해서 두음법칙을 적용하여 가능한 모든 한글 음을 추출하여 색인어로 추출한다.

예) 제 목 : “樂器에 관한 硏究”

색인어 : “樂”, “악”, “락”, “요”, “器”, “기”, “에”, “관”, “한”, “硏”, “연”, “究”, “구”

한적자료 등에서와 같이 재현율을 높이기 위한 경우에 INDEX_BY_MIX_CHAR 를 사용할 수 있으나, 낱자 색인을 하므로 색인양의 매우 커지는 단점이 있다. 일반적인 문서에서는 사용을 권장하지 않는다.

4.1.8 INDEX_BY_MIX_MA

INDEX_BY_MIX_MA 는 한자 변환 기능을 확장한 색인 기능으로서, 섹션 값 중에서 한자는 대응하는 모든 한글 음으로 변환하여 이를 색인어 집합에 추가하고 한글은 INDEX_BY_MA 와 같이 형태소 분석 색인을 수행한다. 이것은 한자 한 음에 대해서 여러 개의 한글 음이 있을 수 있으므로 이를 처리하기 위한 기능이다(예: 樂 → 악, 락, 요). 또한 한자에 대해서 두음법칙을 적용하여 가능한 모든 한글 음을 추출하여 색인어로 추출한다.

예) 제 목 : “樂器의 발전에 관한 硏究”

색인어 : “樂”, “악”, “락”, “요”, “器”, “기”, “硏”, “발전”, “연”, “究”, “구”

4.1.9 INDEX_DNA

INDEX_DNA 는 생물서열 중 유전자의 염기서열을 색인하기 위한 방식이다. DNA 염기 서열은 A, G, C, T 네개의 문자로 구성되며 공백없이 이어지는 문자열이다. KRISTAL-III 는 이러한 DNA 염기서열을 중첩된 11-gram 으로 색인한다.

예) 서 열 : “AGCTAAGCTAAGCTA”

색인어 : “AGCTAAGCTAA”, “GCTAAGCTAAG”, “CTAAGCTAAGC”,

“TAAGCTAAGCT”, “AAGCTAAGCTA”

4.1.10 INDEX_PROTEIN

INDEX_PROTEIN 은 생물서열 중 단백질의 아미노산 서열을 색인하기 위한 방식이다. 단백질 서열은 20 개 아미노산을 지칭하는 문자로 구성되며 DNA 와 마찬가지로 공백 없이 이어지는 문자열이다. KRISTAL-III 는 이러한 단백질 아미노산 서열을 중첩된 5-gram 으로 색인한다.

예) 서 열 : “ACDEFMALYI”

색인어 : “ACDEF”, “CDEFM”, “DEFMA”, “EFMAL”, “FMALY”, “MALYI”

4.2 색인 형식과 데이터 형식과의 관계

색인 형식은 각 데이터 형식에 연결하여 사용하는데, 검색을 제공하고 싶은 섹션이 있다면 각 데이터 형식에 하나의 색인 형식을 지정하여야 한다. 이때 모든 데이터 형식과 색인 형식이 조합되어 사용될 수 있는 것은 아닌데, <표 4-1>에 그들의 관계에 연결 관계에 대해서 설명하였다.

[표 4-1] 데이터 형식에 적용 가능한 색인 형식

<table>
<tr><th>구분</th><th>데이터 형식</th><th>적용 가능한 색인 형식</th></tr>
<tr><td rowspan="4">숫자형</td><td>KINT</td><td rowspan="4">INDEX_AS_NUMERIC</td></tr>
<tr><td>KUINT</td></tr>
<tr><td>KFLOAT</td></tr>
<tr><td>KBOOL</td></tr>
<tr><td rowspan="2">문자형</td><td>KCHAR</td><td rowspan="2">INDEX_AS_IS
INDEX_BY_TOKEN
INDEX_BY_MA
INDEX_BY_CHAR
INDEX_AS_IS_MA
INDEX_BY_MIX_CHAR
INDEX_BY_MIX_MA</td></tr>
<tr><td>KSTRING</td></tr>
<tr><td>바이너리</td><td>KBLOB</td><td>색인 불가능</td></tr>
</table>

<표 4-1>에 따르면 모든 숫자형의 데이터 형식은 숫자형의 색인 형식으로만 색인할 수 있고, 문자형의 데이터 형식은 문자형의 색인 형식으로만 색인 할 수 있음을 알 수 있다. 반면에, 바이너리 타입의 데이터 형식은 색인할 수 없다.

이러한 규칙들은 데이터베이스의 생성을 위한 스키마 생성시 그대로 반영된다.

4.3 색인 API 사용

KRISTAL-III 가 설치된 기계에서는 색인기 라이브러리를 이용하여 색인 API 를 사용할 수 있다. 색인기 라이브러리는 KRISTAL-III 의 lib 디렉토리에 존재하며 이름은 libidx.a 이다. 또한 lib 디렉토리 아래의 k_dic 디렉토리에 존재하는 파일들이 사전 파일이다. 색인 시스템을 사용하기 위한 API 와 그 설명은 다음과 같다.

4.3.1 색인기 초기화 / 종료

❒ 사전 적재 및 기타 초기화 작업

함수명

int IDX_InitIDX(char *KristalDir)

인자들

KristalDir : KRISTAL - III 홈 디렉토리 (이 디렉토리 아래의 lib/k_dic/ 디렉토리에 사전이 들어가야 함)

리턴값

성공 - 1, 실패 - 0

❒ 사전 메모리 해제

함수명

int IDX_CloseIDX()

리턴값

성공-1, 실패-0

4.3.2 색인 수행

색인유형에 따라서 색인작업 수행

일반적인 색인은 106(INDEX_BY_MA)을 인수로 넘김

함수명

int IDX_MainProc(unsigned char req, char *data, POSTINFO *PostInfo, int startWordNum, int stem_opt, int hanja_opt)

인자들

req : 색인 유형(106)

data : 색인 대상 데이터(UTF-8)

PostInfo : 색인결과 저장 데이터

```
typedef struct {
        char            key[MAXKEYLEN];
        unsigned int    keyLen;
        unsigned int    psgNum;
        unsigned int    wordNum;
} POSTINFO;
```

startWordNum : 버퍼링을 하기 위해서 PostInfo의 시작위치를 알림 (default:0)

stem_opt : 영어에 대한 스태밍 여부(1 --> 스태밍, 0 --> 노 스태밍)

hanja_opt : 한자를 한글로 변환 여부(0 --> 변환하지 않음, 1 --> 변환)

리턴값

PostInfo 개수(색인어 개수)

특이 사항

복합명사 분석결과는 일반 색인기와는 조금 다름. 사전에 "정보", "검색", "시스템", "정보검색", "검색시스템"이 있다고 가정하면 정보검색시스템의 명사추출결과는 다음과 같음

- 정보+검색+시스템 ===> 모든 psgNum = 0
- 정보검색+시스템 ===> 모든 psgNum = 1
- 정보+검색시스템 ===> 모든 psgNum = 2
- 정보검색시스템 ===> 모든 psgNum = 100

다시 말해서 PostInfo 에 단위명사가 중복해서 나올 수 있음

이를 판정하는 기준은 PostInfo 의 psgNum 을 기준으로 판정할 수 있음

4.3.3 색인기 불용어 관리

불용어 사전을 등록하고 불용어 제거에 관련된 작업 수행

색인기 초기화를 하고 불용어를 등록하고 활성화시켜야만 불용어 제거기능을 사용할 수 있음

❒ 불용어 사전 등록

함수명

int RegisterSWSet(char *SWID, char *SWSet[], int SWNum)

각 불용어는 UTF-8 로 인코딩되어 있어야 함.

인자

SWID : 불용어 사전 식별자

SWSet : 불용어 사전

SWNum : 불용어 사전 엔트리 개수

리턴값

성공-1, 실패-0

❒ 해당 불용어 사전 식별자에 해당하는 불용어 사전을 활성화

함수명

int ActivateSWSet(char *SWIDs[], int SWIDNum)

불용어 사전이 활성화되면 색인 과정에서 자동적으로 불용어 제거기능 수행함

한꺼번에 여러개의 불용어 사전을 활성화할 수 있음

인자들

SWIDs : 불용어 사전 식별자들

SWIDNum : 불용어 사전 식별자 개수(활성화할 불용어 사전 개수)

리턴값

성공-1, 실패-0

❒ 활성화된 모든 불용어 사전을 메모리에서 제거

함수명

int DeActivateSWSet()

리턴값

성공-1, 실패-0

4.3.4 색인기 사용 예제

```
#include <stdio.h>
#include <cstring>
#include <string>
#include <IDX_common.h>
#include <IDX_user.h>
#include <iconv_str.h>

#define KRISTAL_DIR     "/raid1/k2000/K2000"

POSTINFO postInfo[MAXPOSTINFOSIZE];

void main(int argc, char **argv)
{
        FILE *fd_in;
        char line[10000];
        string src, dest;
        int PostInfoNum;

        if (argc != 2)
                printf("CheckData <in_file>\n");

        fd_in = fopen(argv[1], "rt");
        if (fd_in == NULL)
        {
                exit(1);
        }

        int ret_val = IDX_InitIDX(KRISTAL_DIR);
```

```
	if (!ret_val) exit(1);

	// 이 중간에 불용어 관련 함수를 호출하면 됨.
	// RegisterSWSet
	// ActivateSWSet

	while (fgets(line, 10000, fd_in))
	{
		src = (const char *) line;
		EUCKR_TO_UTF8(src, dest);			// utf-8 로 변환
		PostInfoNum = IDX_MainProc((char)106, (char *) dest.c_str(), postInfo, 0, 1, 1);
		for (int i = 0; i < PostInfoNum; i++)
		{
			src = (const char *)postInfo[i].key;
			UTF8_TO_EUCKR(src, dest);	// 결과를 다시 euc-kr 로 변환
			cout << dest << "\n";
		}
	}
	fclose(fd_in);

	// DeActivateSWSet();
	IDX_CloseIDX();
}
```

4.4 사용자 사전

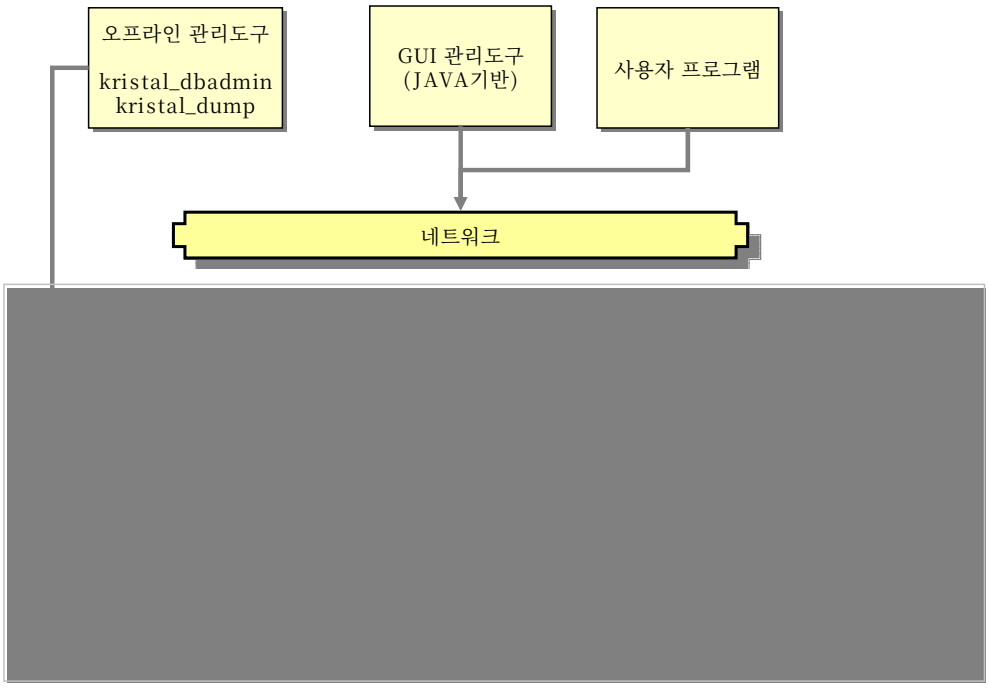
사용자 사전이란 색인 시스템의 기본 사전에 존재하지 않는 미등록어나 고유명사 등의 단어를 사용자가 색인어로 추출하고 싶을 때 직접 추가하여 색인할 수 있는 기능이다. 사용자는 $(KRISTAL_HOME)/lib/k_dic 디렉토리에 “USERDIC.DIC”이라는 파일을 만들고 한 라인에 하나씩 단어를 추가하면 된다. 본 시스템에서는 입력 단어에 대한 형태소 분석을 수행한 후에 분석 실패한 어절에 대해서만 사용자 사전 탐색을 수행하고, 조사, 어미와 같은 형식형태소 등을 분리하여 분석할 수 있다. 예를 들어, 기본 사전에 “부시”라는 단어가 존재하지 않아서 사용자가 사용자 사전에 “부시”라는 단어를 추가하였다고 하자. 그러면 색인 대상 어절 “부시입니다”는 색인기를 거쳐 “부시”가 색인어로 추출된다. 색인 시스템의 입력 문자 코드는 기본적으로 UTF-8 이지만 사용자 사전에 추가되는 어절의 코드는 사용자의 편의를 위해서 EUC-KR 로 지정하게 하였다.

제 5 장 데이터베이스 관리

KRISTAL-III 은 데몬의 구동 여부에 따라 여러 방식 데이터베이스 관리 방법을 제공한다. 5 장에서는 그중 데몬이 구동되지 않은 오프라인 상태를 가정으로하여 데이터베이스를 관리하기 위한 제반 내용을 설명한다. 5.1 절에서는 KRISTAL 이 제공하는 모든 데이터베이스 관리 방법에 대해서 설명하고, 5.2 절에서는 KRISTAL 데이터베이스 스키마(Schema)에 대해서, 5.3 정에서는 KRISTAL 적재 스키마에 대해서 설명하고, 5.4 절에서는 데이터베이스 관리도구에 대해서 설명한다.

5.1 KRISTAL 의 데이터베이스 관리 방법

KRISTAL-III 에서 제공하는 데이터베이스 관리는 관리 프로그램의 연결 방법에 따라 세가지 방식으로 구분될 수 있다. (그림 5-1 참조)



[그림 5-1] KRISTAL-III 에서 제공하는 데이터베이스 관리 방법

■ **방식 1 : 오프라인 기반의 관리 도구**

가장 기본적인 방식의 관리도구로서, KRISTAL 데몬이 구동되지 않은 상황에서 데이터베이스 관리 기능을 수행한다. 이를 지원하기 위해 제공하는 실행 프로그램은 kristal_dbadmin, kristal_dump, kristal_backup 이 있는데, 이 중 kristal_dbadmin 은 KRISTAL 스키마를 이용한다. KRISTAL 스키마는 오프라인 방식의 관리자(kristal_dbadmin)를 이용하여 데이터베이스를 관리하기 위해 필요한 핵심 요소[3]이다. 데이터베이스 생성 및 데이터 적재/색인 등의 다양한 기능을 수행하기 위해서는 먼저 KRISTAL 스키마를 작성해야 하는데, 데이터베이스의 생성 과정에서는 *데이터베이스 스키마*를, 데이터의 적재를 위해서는 *적재 스키마*를 사용한다. 5 장의 데이터베이스 관리편은 스키마를 기반으로 데이터베이스를 관리하는 방법을 설명한 것으로써, 방식 1 에 초점이 맞추어져 있다는 것을 기억하기 바란다.

■ **방식 2 : GUI 관리 도구**

관리 프로그램은 GUI 기반으로 작성되어 있고 KRISTAL 데몬과 네트워크로 연결하여 관리기능을 수행한다. GUI 프로그램은 사용자가 보다 편리하게 사용할수 있어서 관리의 능률을 향상시킬수 있다. 이러한 관리 작업을 수행하기 위해서, GUI 프로그램은 KRISTAL 데몬에서 제공하는 관리 API 와 사용자 API 를 사용한다. 이때, 일부 관리 API 들은 관리도구에서만 사용할 수 있도록 제약을 가질 수 있다.

■ **방식 3 : 사용자 프로그램**

KRISTAL 데몬에서 제공하는 API 중 사용자 API 라고 명명된 함수들을 이용함으로서, 사용자가 자신의 고유한 관리 프로그램을 작성하여 사용할 수 있다. 제공하는 함수에 대한 설명은 API 명세서를 참조하기 바란다.

[3] KRISTAL 스키마는 온라인 데이터베이스 관리에서도 동일하게 사용한다.

5.2 KRISTAL 데이터베이스 스키마

데이터베이스 스키마는 KRISTAL 의 두 스키마 중의 하나로서, 데이터베이스의 구조적인 관리 기능을 수행하기 위해서는 먼저 **데이터베이스 스키마**를 정의한다.

5.2.1 데이터베이스 스키마 개요

데이터베이스 스키마는 데이터베이스 및 테이블의 관리(생성/삭제)에 필요한 각종 정보를 담고 있는 텍스트 파일로서 XML 형태[4]로 작성된다. 스키마내에서 사용하는 모든 값들은 XML 문서 속성을 그대로 수용하기 때문에 각 값들의 대소문자에 유의하여 작성하여야 한다. 데이터베이스 스키마의 구조는 그림 5-2 와 같이 주요 세 부분으로 구성된다.

```
<?xml version="1.0"?>

<!DOCTYPE DatabaseSchema SYSTEM "http://giis.kisti.re.kr/kristal-schema/database_schema.dtd" []>
<DatabaseSchema>
    Part 1 : 데이터베이스 정의
    Part 2 : 사용자 영역 정의
    Part 3 : 테이블 스키마 정의
    Part 4 : 테이블 정의
</DatabaseSchema>
```

[그림 5-2] 데이터베이스 스키마 구조

[4] KRISTAL 스키마는 XML 문서로 구성되기 때문에, 스키마에 대한 이해를 위해서 기본적인 XML 문법 용어들에 대한 이해가 필요하다. 대표적인 예로서 엘리먼트(Element)와 속성(Attribute)이 있다.

네 부분들은 각각 데이터베이스 정의(Part 1), 사용자 영역 정의(Part 2), 테이블 스키마 정의(Part 3)와 테이블 생성 정의(Part 4)로 구성되는데, 수행하고자 하는 기능에 따라 네 부분의 일부 혹은 조합이 사용될 수 있다. 네 부분들의 연관 관계는 5.5 의 예제를 참고하기 바란다. 각 부분들은 순차적으로 처리되는데, 스키마의 가독성을 고려하여 Part1, Part2, Part3, Part4 의 순으로 정의할 것을 권장한다. 사용자의 이해를 돕기 위해, 각 부분들에 사용되는 모든 엘리먼트들 태그를 표 5-1 에 정의하였다.

표 5-1 데이터베이스 스키마에서 정의 할수 있는 엘리먼트 태그들

부분	기 능		엘 리 먼 트
1	데이터베이스 생성		<CreateDatabase database-name=”…” volume-dir=”…” />
	데이터베이스 사용		<UseDatabase database-name=”…” volume-dir=”…” />
	데이터베이스 삭제		<DeleteDatabase database-name=”…” volume-dir=”…” />
2	사용자 영역 생성		<CreateUserArea> … </CreateUserArea>
3	테이블 스키마 생성	상위 엘리먼트	<CreateTableSchema name=”…” doc-type=”…” alias=”…”> </CreateTableSchema>
		기본 키 정의	<PrimaryKey sections=”…” />
		불용어 정의	<Stopword file=”…” />
		기본 섹션 정의	<BasicSection name=”…” data-type=”…” index-type=”…” index-remove-element-tag=”…” use-stemming=”…” remove-stopword=”…” hanja-to-hangul=”…” use-index-bigram=”…” default-value=”…” alias=”…”/>
		가상 섹션 정의	<VirtualSection name=”…” source-section=”…” index-type=”…” index-remove-element-tag=”…” use-stemming=”…” remove-stopword=”…” hanja-to-hangul=”…” use-index-bigram=”…” alias=”…”/>
		통합 섹션 정의	<UnionSection name=”…” include-sections=”…” alias=”…” />
	테이블 스키마 삭제		<DeleteTableSchema name=”…” />
4	테이블 생성		<CreateTable table-name=”…” with-schema=”…” alias=”…” />
	테이블 삭제		<DeleteTable table-name=”…” />

KRISTAL-III 데이터베이스 스키마의 유효성(Validation)을 확인하는 과정에서 KRISTAL-III 데이터베이스 스키마 DTD 를 사용한다. 사용자의 편의를 위해 최신 DTD 를 http://giis.kisti.re.kr/kristal-schema/database_schema.dtd 에 배포하고 있다. 사용자는 그림 5-2 의 두번째 라인에서와 같이 스키마 내부에 해당 DTD 를 명시해서 사용하면 되고, 유효성은 KRISTAL-III 데이터베이스 스키마를 처리하는 도구에 의해서 자동으로 처리된다. 다만 이과정에서 네트워크 연결이 필요한데, 네트워크가 연결되지 않은 호스트에서 데이터베이스 스키마를 처리하고자 한다면 DTD 를 다운받아 로컬에 저장한 후 DTD 경로를 변경하면 된다.

데이터베이스 스키마 작성에 사용되는 각 엘리먼트 및 속성들에 대한 내용은 하위 절에서 설명한다.

5.2.2 데이터베이스 정의

KRISTAL-III 데이터베이스에 대한 처리를 정의한다. 여기서 데이터베이스라 함은 단일 테이블을 의미한다. 다음 세 엘리먼트를 이용하여 KRISTAL 테이블을 생성, 재사용, 또는 삭제할 수 있다.

관련 엘리먼트: CreateDatabase, UseDatabase, DeleteDatabase

데이터베이스는 두 가지 형태 중의 하나로 정의되는데, 데이터베이스를 생성하는가 또는 이전의 데이터베이스를 그대로 사용하는 가에 따라 다른 형태의 엘리먼트 태그를 사용한다.

- 데이터베이스 생성 시: *CreateDatabase* 를 이용하여 정의한다.
- 기존 데이터베이스 사용 시: *UseDatabase* 를 이용하여 정의한다.
- 데이터베이스 삭제 시: *DeleteDatabase* 를 이용하여 정의한다.

관련 속성: database-name, volume-dir

Database-name	데이터베이스 이름을 지정한다.
volume-dir	데이터베이스 볼륨 위치를 지정한다.

Db-type	CreateDatabase 엘리먼트에만 적용 데이터베이스에 저장할 문서의 종류(XML|PLAIN, default=PLAIN)

사용 예:

⇨ 데이터베이스 생성

<CreateDatabase database-name="BLUE-DB" volume-dir="/home/BLUE/volume" />

BLUE-DB 라는 새로운 데이터베이스를 생성한다.

⇨ 기존 데이터베이스 사용

<UseDatabase database-name="BLUE-DB" volume-dir="/home/BLUE/volume" />

이전에 이미 생성되어 있는 BLUE-DB 를 사용한다.

⇨ 데이터베이스 삭제

<DeleteDatabase database-name="BLUE-DB" volume-dir="/home/BLUE/volume" />

BLUE-DB 데이터베이스를 영구 삭제한다.

⇨ XML 문서를 다루기 위한 데이터베이스 생성

<CreateDatabase database-name="BLUE-DB" volume-dir="/home/BLUE/volume" db-type =”XML”/>

새로 생성되는 BLUE-DB 는 XML 문서를 저장하고 관리하기 위한 데이터베이스이다. 속성 db-type 은 CreateDatabase 엘리먼트에만 종속된 속성이다.

5.2.3 사용자 영역 정의

사용자 영역은 데이터베이스에 관련된 일종의 메타 정보를 사용자 자신의 말로 기술

하여 유지함으로써 데이터베이스 관리의 편의성을 증진시키기 위해 사용한다. 특정 형식이 정해져 있지 않기 때문에, 스키마를 이용하여 임의 문자열로 구성된 정보 영역을 생성하고 클라이언트 라이브러리 API 를 이용하여 사용자 영역을 획득한다.

사용자 영역은 데이터베이스 관리자의 편의를 위하여 생성하는 것으로 데이터베이스 설명, 정책 등 관리자가 필요한 사항을 자유로이 기술하여 KRISTAL 데이터베이스에 수록할 수 있는 영역이다. 사용자 영역은 클라이언트 API 를 통하여 읽어들일 수 있으므로 메타 검색에서의 데이터베이스 정보 공유 등에 유용하게 사용할 수 있다.

관련 엘리먼트: CreateUserArea

사용자는 여는 엘리먼트 태그와 닫는 엘리먼트 태그의 쌍으로 형태를 구성하고, 내용은 엘리먼트 내의 TEXT 파트나 CDATA 형식의 혼합으로 작성할 수 있다. TEXT 파트 혹은 CDATA 형식 중 하나만으로도 기술될 수 있다. 다만 텍스트 부분의 양끝 공백문자(스페이스, 탭)와 개행문자(\r, \n)는 제거된다.

관련 속성: 없음

사용 예:

⇨ 사용자 영역을 생성하는 스키마 예제로서, 반전된 부분이 실제로 추출되는 데이터이다.

```
<CreateUserArea>
        Descriptions on the text
        <![CDATA[
        blur blur ...
        ]]>
</CreateUserArea>
```

5.2.4 테이블 스키마 정의

KRISTAL-III 는 여러 개의 스키마를 관리할 수 있다. 다음 두 엘리먼트는 테이블 정의시 사용할 테이블 스키마(Table Schema)의 원형을 생성하거나 삭제한다. 테이블스키마에는 주요키(Primary Key), 기본섹션(Basic Section), 가상섹션(Virtual Section), 통합섹션(Union Section) 등 테이블의 기본적인 구조를 기술한다.

관련 엘리먼트 : CreateTableSchema, DeleteTableSchema

CreateTableSchema 엘리먼트를 이용하여 1 개 이상의 스키마를 생성한다.

DeleteTableSchema 엘리먼트를 이용하여 1 개 이상의 스키마를 삭제한다.

관련 속성 : **name, doc-type, alias**

Name	해당 엘리먼트는 속성으로서 name 을 사용하는데, name 은 각 스키마들을 구분할 수 있도록 해주고 실제 테이블 생성 시 사용될 스키마로서 주어진다.
doc-type	테이블 스키마의 타입을 선택한다. 선택한 타입에 따라 "*PLAIN*"과 "*XML*"의 두가지 값 중 하나를 가질 수 있고, 생략된 경우 기본값으로 *PLAIN* 으로 지정된다. XML 타입을 선택하였다면 8 장의 XML 처리편을 참고하기 바란다.
Alias	테이블 스키마를 구분하기 위한 대체 이름을 지정한다.

사용 예 :

⇨ 테이블 스키마 schema01 를 생성한다.

<CreateTableSchema name="schema01"> …. </CreateTableSchema>

⇨ 테이블 스키마 schema02 를 삭제한다.

<DeleteTableSchema name="schema02">

5.2.4.1 주요 키 정의

데이터 관리시 데이터들 간의 중복 관계를 확인함으로써 효율적인 데이터 관리를 도모하기 위해서 정의하는 부분이다. 정의가 생략되었을 경우에는, 주요 키를 이용한 데이터 관리를 수행하지 않는다. 따라서 데이터의 무제한적인 중복이 발생 할 수 있다는 것을 기억해야 한다.

관련 엘리먼트: **PrimaryKey**

관련 속성: **sections**

sections	기본 키로 사용 될 섹션들을 지정하기 위한 것으로 공백이나 ‘,’에 의해 각 섹션을 구분하여 나열 할 수 있다. PrimaryKey 엘리먼트를 사용하고자 할 경우에는 반드시 1 개 이상의 섹션이 지정되어야 한다.

사용 예 :

⇨ 섹션 AN, TIK 를 기본 키로서 설정한다.

```
<PrimaryKey sections="AN TIK"/>
```

주요키는 제어번호 등과 같이 각 문서에 고유한 값을 가지는 섹션을 지정하면 된다. 이러한 경우에는 단일 섹션으로 주요키를 사용할 수 있다. 그러나 문서마다 고유한 값을 가지는 섹션이 없을 경우에는 두개 이상의 섹션값을 조합하여 주요키를 생성할 수 있다. 이러한 경우에는 sections 속성에 두 개 이상의 섹션을 지정할 수 있다. 예를 들어 하나의 테이블이 “저자(AU)”, “제목(TI)”, “출판일(PY)”, “출판사(PC)”, … 등의 섹션으로 구성될 경우에는 sections=”AU,TI,PY”의 조합으로 지정하면 모든 문서에 고유한 주요 키를 생성할 수 있다.

데이터 관리가 중요한 서비스에서는 주요키를 정의하여 사용하는 것이 데이터의 중복을 피할 수 있는 좋은 방법이다. 그러나 중복 제거에 대한 필요성이 없거나, 검색을

위주로 하는 서비스에서는 굳이 주요 키를 정의할 필요는 없다. (이 경우에는 KRISTAL 이 내부적으로 고유한 키값을 생성하여 주요키를 관리한다.)

5.2.4.2 불용어 정의

불용어(Stopword)란 검색에서 불필요하다고 정의될 수 있는 단어들을 의미한다. 한국에서는 주로 부사나 형용사 등을 불용어로 취급할 수 있고 영어 문서에서는 관사, 형용사, 부사 등을 불용어로 취급할 수 있다.

불용어를 정의함으로써 검색할 때 검색 대상에 포함되지 않도록 한다. 색인 측면에서는 불필요한 색인어를 제거함으로써 색인 및 검색 시간의 개선 효과를 가져온다. 그러나 불용어를 사용하면 불용어로 검색했을 때 해당 문서를 가져오지 못하여 재현율을 떨어뜨리는 단점이 발생할 수 있다. 따라서 불용어를 사용할 경우에는 데이터베이스 특성에 따라 불용어 선택에 주의를 기울여야 한다.

관련 엘리먼트: **Stopword**

여러 개의 불용어 파일을 동시에 정의할 수 있다. 정의된 파일들은 전체가 하나의 셋으로 결합된 후 스키마 내 유일한 불용어 리스트를 생성한다.

관련 속성: **file**

File	불용어 파일 이름을 명시한다. 불용어 파일은 절대 패스 형태의 완전한 파일 이름을 사용하여야 한다. (와일드 카드 문자 비 허용)

사용 예:

⇨ 다음 예는 두 개의 파일을 해당 스키마의 유일한 불용어 리스트로 지정하는 방법을 보인다.

```
<Stopword file="/home/k2002/TEST/stopword/swords-eng"/>
<Stopword file="/home/k2002/TEST/stopword/swords-han"/>
```

사용자가 제공하여 할 불용어 파일의 구조는 다음과 같이 한 라인에 공백 없이 한 단어로 구성 된 형태를 가져야 한다. 사용자는 각 파일 안의 단어에 대해서 오름차순에 따른 관리를 함으로써 관리의 용이함을 꾀할 수 있으나, 정열이 되어 있지 않아도 무방하다. 그러나 인코딩 형태는 사용자의 편의를 위하여 EUC-KR 로 사용하도록 하고 있다.

```
…..
a
about
against
all
…
그것
그들
….
```

5.2.4.3 기본 섹션 정의

테이블 내의 기본 섹션을 정의한다. 앞서 설명한 바와 같이 기본섹션은 하나의 문서를 구성하는 가장 기본적인 단위이다.

관련 엘리먼트: BasicSection

관련 속성: name, data-type, index-type, use-stemming, remove-stopword, hanja-to-hangul, use-index-bigram, default-value, index-remove-element-tag, alias

name	기본 섹션의 이름을 지정한다. 영문을 사용할 것을 권장하며, 향후 관리의 편의성을 위해 짧게 지정하는 것이 좋다. 대소

	문자를 구분한다.
data-type	기본 섹션의 데이터 타입을 지정한다. KSTRING, KCHAR, KINT, KUNIT 등 데이터 값의 형태에 따라 선택하여 사용할 수 있다.
index-type	기본 섹션의 색인 타입을 지정한다. (KRISTAL-III 의 데이터 타입 및 색인 타입에 관한 것은 4 장 색인형식을 참조하기 바란다.)
index-remove-element-tag	XML 문서의 경우 색인시 엘리먼트와 에트리뷰트 값의 고려 유무를 선택하도록 한다. “YES”값의 경우 엘리먼트와 에트리뷰트를 모두 제거한 후 색인을 한다. 기본값은 “NO”이다.
use-stemming	색인 시 스테밍(Stemming)을 사용할 지 여부를 결정한다. 스테밍은 동일한 의미를 지니고 있는 서로 다른 형태의 단어들을 단일 형태로 통일한다. 예를 들면 문서에 출현한 represent, represents, represented 라는 단어들은 represent 라는 원형으로 변환된다. KRISTAL 의 스테밍은 영어권을 대상으로 한다. 해당 속성 값으로는 “YES” 또는 “NO”를 사용할 수 있는데, “YES” 값이 지정되면 스테밍 처리를 수행함을 의미하고, “NO” 값은 스테밍 처리를 수행하지 않음을 의미한다. 기본값은 “NO”이다.
remove-stopword	색인 시 불용어 처리의 여부를 정의하는 것으로, “YES” 또는 “NO”의 값을 가질 수 있다. “YES” 값이 지정되면 불용어 처리를 수행함을 의미하고, “NO” 값은 불용어를 제거하지 않음을 의미한다. 기본값은 “no”이다.
hanja-to-hangul	색인 시 문서에 출현하는 한자어를 한글로 변환하여 색인 할지 여부를 결정한다. “YES”값이 지정되면 한자어를 한글로 변환하여 색인 함을 의미하고, “NO”는 변환을 하지 않은 채로 색인 함을 의미한다. 기본값은 “NO”이다.
use-index-bigram	이미 지정된 색인 방법이외에 단어 단위 바이그램을 적용할지 여부를 지정함. “YES”값이 지정되면 추가적으로 바이그램 방식의 색인어가 추자적으로 생성된다. 기본값은 “NO”이다.
default-value	해당 섹션의 기본 값을 지정하기 위한 속성으로, 데이터 적

	재 시 해당 섹션 값이 명시되지 않은 경우 이 속성 값을 해당 문서의 섹션 값으로 정의한다. 섹션값이 존재하지 않는 문서를 관리하기 위한 방편으로 사용할 수 있다. 다만 이 값은 특수한 값을 사용하여 검색에 영향을 미치지 않도록 주의하여 한다.
alias	테이블 스키마를 구분하기 위한 대체 이름을 지정한다. 섹션의 이름이 어려울 경우에는 alias 를 이용하면 관리자가 알기 쉬운 별칭으로 섹션을 지정할 수 있는 장점이 있다.

사용 예 :

⇨ 가변 길이 형태의 AN 섹션을 정의한다.

<BasicSection name="AN" data-type="KSTRING" index-type="INDEX_AS_IS" />

⇨ 가변 길이 형태의 AN 섹션을 정의하는데, .데이터 적재 시 섹션 값이 명시되지 않을 경우 “0000_0100”을 기본 섹션 값으로 사용한다.

<BasicSection name="AN" data-type="KSTRING" index-type="INDEX_AS_IS" default-value=”0000_0100” />

⇨ 정수 형태의 ANI 섹션을 정의한다.

<BasicSection name="ANI" data-type="KINT" index-type="INDEX_AS_NUMERIC" />

⇨ 가변길이 문자열 형태의 TITLE 섹션을 정의한다.

<BasicSection name="TITLE" data-type="KSTRING" index-type="INDEX_BY_MA" />

⇨ 가변길이 문자열 형태의 ETITLE 섹션을 정의한다.

<BasicSection name="ETITLE" data-type="KSTRING" index-type="INDEX_BY_TOKEN" />

⇨ 고정 길이 형태의 AN(제어번호) 섹션을 정의한다. 크기는 10 바이트 형태로서 KCHAR 타입을 사용한다.

<BasicSection name="AN" data-type="KCHAR[10]" index-type=”INDEX_AS_IS" />

기본섹션의 index-type 이 INDEX_BY_TOKEN 일 경우에 한하여 delimit-char 와 delete-chars 의 두가지 속성을 추가적으로 사용할 수 있다. 두 속성은 각각 토큰 추출시의 구분자와 추출된 토큰의 삭제문자 지정자이다. INDEX_BY_TOKEN 색인방식에 대해서는 4.1.2 절을 참조하기 바란다.

delimit-chars	토큰 추출시에 사용할 구분자를 의미한다. 낱자단위로 부여할 수 있다. 저자, 분야 등과 같이 특수한 구분자를 통해 분리되는 문자열을 색인하고자 할 경우에 사용할 수 있다.
delete-chars	위의 delimit-chars 를 지정하여 추출된 토큰에서 제거할 문자를 지정한다. 따라서 delimit-chars 가 지정되었을 경우에만 효력을 발생한다.

사용 예 :

⇨ 가변 길이 형태의 AUTHOR 섹션을 정의하고 INDEX_BY_TOKEN 으로 색인하며 ; 문자로 구분되는 토큰들을 색인한다.

<BasicSection name="AUTHOR" data-type="KSTRING" index-type="INDEX_BY_TOKEN" delimit-chars=";" />

⇨ 가변 길이 형태의 AUTHOR 섹션을 정의한다. 나머지 부분은 위와 같으나 delete-chars 속성에 공백문자를 주어 추출된 토큰내의 공백문자를 제거한다.

<BasicSection name="AUTHOR" data-type="KSTRING" index-type="INDEX_BY_TOKEN delimit-chars=";" delete-chars=" " />

삭제 문자 지정자인 delete-chars 속성은 delimit-chars 속성과 함께 정의되어야 한다.

5.2.4.4 가상 섹션 정의

가상섹션(Virtual Section)은 기본 섹션에 대해서 추가적인 색인 타입 정의를 위해서 사용하는 가상적인 형태의 섹션으로서, 검색시 사용하는 섹션이다. KRISTAL 은 가상 섹션에 대해서 실제 문서는 저장하지 않고, 지정한 섹션들에 대한 색인기능만을 제공하므로 가상섹션이라 부른다.

관련 엘리먼트: VirtualSection

관련 속성: name, source-section, index-type, use-stemming, remove-stopword, hanja-to-hangul, index-remove-element-tag, alias

name	가상 섹션의 이름을 지정한다.
source-section	가상 섹션 정의 대상이 되는 기본 섹션(들)을 지정한다.
index-type	가상 섹션의 색인 타입을 지정한다. (기본 섹션과 동일하게 정의한다.)
index-remove-element-tag	*기본 섹션과 동일*
use-stemming	*기본 섹션과 동일*
remove-stopword	*기본 섹션과 동일*
hanja-to-hangul	*기본 섹션과 동일*
use-index-bigram	*기본 섹션과 동일*
alias	*기본 섹션과 동일*

사용 예 :

⇨ 기본 섹션 TIK, TIO 에 INDEX_BY_MA 의 색인 타입을 적용시켜 가상 섹션 VIR_TIK 를 생성한다.

```
<VirtualSection name="VIR_TIK" source-section="TIK TIO" index-type="INDEX_BY_MA" />
```

가상섹션은 기본 섹션에 대해서 두 가지 이상의 색인형태를 제공하기 위한 목적으로

사용할 수 있으며, 이와 더불어 가장 큰 목적은 두 개 이상의 기본 섹션을 통합하여 색인을 함으로써 검색 속도를 크게 향상시키는 데 있다. 예를 들어 하나의 테이블이 “국문제목(TIK)”, “영문제목(TIE)”, “국문요약(ABK)”, “영문요약(ABE)” 등을 포함하고 있다면 일반적으로 네 개의 섹션을 모두 검색대상으로 하는 것이 타당할 것이다. 이 경우에는 기본섹션과 더불어 네 개의 섹션을 모두 묶어서 가상의 검색 섹션을 생성할 수 있다. 가상 섹션은 색인의 경우, 기본섹션과 마찬가지로 물리적인 독립성을 가지기 때문에 독자적인 색인방식을 지정하여야 한다.

사용 예 :

⇨ 기본 섹션인 TIK, TIE, ABK, ABE 를 묶어서 TEXT 라는 가상 섹션을 생성하고 INDEX_BY_MA 색인방식을 적용한다. 색인시 단어 단위의 색인어 bi-gram 을 색인어로 추가 추출한다.

<VirtualSection name="TEXT" source-section="TIK TIE ABK ABE" index-type="INDEX_BY_MA" use-index-bigram=”YES” />

위에서 설명한 바와 같이 가상섹션은 독자적인 색인형태를 가지게 된다. 따라서 두 개이상의 섹션이 묶여지더라도 하나의 색인형태를 가지게 되므로, 두 개 이상의 기본섹션을 묶어서 가상섹션을 생성할 경우에는 묶여지는 기본섹션들의 형태가 유사한 경우여야 한다. 예를 들어 한글제목과 한글요약, 본문 등은 자연어로 기술되기 때문에 형태소분석이 필요하다. 따라서 이러한 경우에는 한글제목, 한글요약, 본문을 모두 하나의 가상섹션으로 묶어서 INDEX_BY_MA 색인방식을 적용하는 것이 자연스럽게 느껴진다. 그러나 저자나 영어키워드와 같은 섹션은 자연어로 기술되는 값이 아니라, 이름(저자) 및 주요 단어(영문 키워드[5])로 기술되는 경우가 많다. 이러한 경우에는 형태소분석

[5] 한글 키워드에 대해서는 다른 시각이 있을 수 있다. 한글 키워드의 경우에는 “정보검색”, “문서분류” 등과 같이 복합명사로 구성될 수 있으므로 필요에 따라서는 형태소분석을 수행하여 기본 단어로 분리하여 색인하는 것이 유리할 수 있다.

을 수행할 필요가 없을 것이다. 따라서 저자와 영문 키워드는 검색의 편의를 위하여 하나의 가상섹션으로 묶어도 좋을 것이다.

그러나 단순히 검색 속도를 향상시키기 위한 목적으로 “저자” 섹션과 “한글제목” 섹션을 묶어서 하나의 가상섹션으로 구성하는 것은 바람직하지 않다. 저자와 한글제목의 경우에는 서로 색인방식을 달리하여야 하기 때문이다. 이러한 경우에는 검색의 편의를 위하여 다음에 설명할 통합섹션으로 정의하는 것이 좋다. *서로 다른 색인 방식이 적용되어야 하는 상이한 섹션을 가상섹션으로 묶는 것은 권장하지 않음*을 유념해주기 바란다.

가상섹션은 기본섹션과 마찬가지로 독립적인 색인을 가지는 섹션이다. 따라서 가상섹션은 별도의 물리적 저장공간을 요구하기 때문에, 가상섹션의 사용은 서비스에 부담을 줄 수 있으므로 주의하여야 한다. 예를 들어 게시판과 같이 일상적인 문서의 수정과 삽입이 이루어지는 서비스의 경우에는 기본섹션과 더불어 가상섹션에 대한 색인 작업을 추가적으로 해야 하므로 글쓰기 기능이 가상섹션이 없을 경우에 비해 느려지게 되는 현상이 발생할 수 있다.

가상섹션을 INDEX_BY_TOKEN 으로 색인할 경우에는 기본섹션에서와 마찬가지로 delimit-char 옵션과 delete-char 옵션을 사용할 수 있다. 보다 자세한 사용법은 위의 기본섹션에 대한 설명을 참조하기 바란다.

5.2.4.5 통합 섹션 정의

통합 섹션(Union Section)은 기본 섹션들이나 가상 섹션들 또는 둘의 조합을 묶어 하나의 섹션 이름에 대응 시킨 형태의 논리적 섹션으로서, 검색의 편의성을 위하여 사용하는 일종의 대표 검색 섹션이라 할 수 있다. 통합섹션을 이용하면 사용자의 검색 질의를 매우 단순화할 수 있다. 가상섹션이 물리적인 형태의 색인을 독립적으로 가지는 반면, 통합섹션은 여러 개의 섹션을 묶어서 단일 섹션으로 지정만 하는 순수한 논리적인 섹션이며, 별도의 색인을 가지지 않는다. 통합섹션은 “논리적 통합 검색 섹션”으로 이해할 수 있다.

관련 엘리먼트: UnionSection

관련 속성: name, include-sections, alias

name	통합 섹션의 이름을 정의한다.
include-sections	대상이 되는 기본 섹션 및 가상 섹션들을 지정한다.
alias	*기본 섹션과 동일*

사용 예:

⇨ 다음 예제는 TIK, TIO, ABS 를 통합 섹션 BASIC 에 대응시키는 것으로서 검색 질의 생성 시 BASIC 이라는 대표 이름으로 세 개의 섹션 모두에 대해서 검색을 수행시킬 수 있도록 해준다.

<UnionSection name="BASIC" include-sections="TIK TIO ABS" />

통합 섹션(Union Section)은 검색의 편의를 위하여 제공되는 논리적 섹션이다. 예를 들어 저자(AUTH), 제목(TITL), 요약(ABST), 본문(BODY)의 기본섹션으로 이루어진 문서 데이터베이스가 있다고 하고, 기본적으로 검색은 4 개의 섹션 모두를 대상으로 하는 설계를 했다고 하면, 4 개의 섹션을 모두 하나의 통합섹션으로 설정하면 검색인터페이스를 매우 단순화시킬 수 있다. 새로운 통합섹션의 이름을 SEARCH 라고 한다면 <UnionSection name=”SEARCH” include-sections=”AUTH TITL ABST BODY” />로 스키마에 기술하면 된다[6].

[6] 여기서는 설명의 편의를 위하여 기본섹션으로만 통합섹션을 구성하였다. 보다 빠른 검색이 요구되는 경우에는 먼저 색인방식이 유사한 기본섹션들을 가상섹션으로 묶은 후 이를 통합섹션으로 묶는 방식을 사용할 수 있다. 예를 들어 이 경우에는 TITL, ABST, BODY 를 TEXT 라는 가상섹션으로 묶는다고 하면 검색을 위한 통합섹션은 AUTH 와 TEXT 의 두 개의 섹션으로 구성할 수 있다. <UnionSection name=TEXT include-sections=”AUTH TEXT” />

위의 경우에 대하여 통합섹션의 유무에 따른 검색의 복잡도를 살펴보자. 먼저 SEARCH 라는 통합섹션을 정의하지 않고 4 개의 섹션에 대해서 검색을 하기 위해서는 다음과 같은 복잡한 질의식이 요구된다.

(AUTH: 질의) | (TITL: 질의) | (ABST: 질의) | (BODY: 질의) - 질의 (1)

그러나 SEARCH 라는 통합섹션으로 검색 대상 섹션들을 묶으면 다음과 같이 질의가 매우 단순해진다.

(SEARCH: 질의) - 질의 (2)

질의 (2)는 질의 (1)에 비해 매우 단순하여 사용하기 편리하지만, 통합섹션은 KRISTAL 이 내부적으로 기본섹션 또는 가상섹션으로 확장하기 때문에 검색 결과는 질의 (1)과 동일하다.

5.2.5 테이블 정의

5.2.4 에서 정의한 테이블 스키마를 이용하여 실제적으로 데이터가 저장될 테이블[7]을 정의한다. KRISTAL-III 는 멀티 테이블을 지원하기 때문에 한 데이터베이스 내에서 1 개 이상의 테이블을 정의할 수 있다.

관련 엘리먼트: **CreateTable**, **DeleteTable**

CreateTable 을 이용하여 테이블을 생성한다.

---

[7] 테이블(Table)은 KRISTAL-III 데이터베이스의 가장 기본적인 단위이다. 문서들이 물리적으로 모여서 저장되는 하나의 저장소를 테이블이라 한다. 앞서 설명한 바와 같이 KRISTAL-III IRMS 는 동일한 테이블스키마를 가지는 다수의 테이블이 존재할 수 있다. 이와 같이 동일한 스키마를 가지는 테이블의 그룹을 (논리적인) 테이블 클러스터(Table Cluster) 또는 클러스터라고 부르기도 한다.

DeleteTable 을 이용하여 테이블을 삭제한다.

관련 속성: table-name, with-schema, alias

table-name	생성 할 테이블 이름을 지정한다. 대소문자를 구분한다. 길이에 제한은 없으나 향후 관리를 위해 명확하되 단순한 이름을 사용할 것을 권장한다.
with-schema	테이블 생성에 사용할 테이블 스키마 이름을 지정한다.
alias	테이블 스키마를 구분하기 위한 대체 이름을 지정한다.

사용 예:

⇨ 테이블 스키마 schema01 을 이용하여 두 개의 테이블(blue01, blue02)를 생성한다. 생성 된 테이블은 논리적으로 클러스터 테이블로 인식되어 해당 테이블들에 대한 통합 검색 및 랭킹을 사용할 수 있다.

```
<CreateTable table-name="blue01" with-schema="schema01"/>
<CreateTable table-name="blue02" with-schema="schema01"/>
```

5.3 KRISTAL 적재 스키마

5.3.1. KRISTAL 적재 스키마 개요

KRISTAL-III IRMS 는 데이터 적재 및 색인을 위해서 1) *벌크 처리 방식*과 2) *온라인 처리방식*의 두 방법을 제공한다. 벌크 처리 방식은 KRISTAL-III 데몬이 기동되지 않은 상황에서 시스템의 모든 자원을 효과적으로 활용하여 빠른 데이터 적재 및 색인을 제공하기 위한 방법이고, 온라인 처리 방식은 KRISTAL-III 데몬이 서비스 중에 데이터를 관리하기 위한 방법이다. 벌크 처리에서는 적재 스키마(Loader Schema)를 이용하여 데이터의 적재 및 색인을 수행하는 반면, 온라인 처리 방식에서는 KRISTAL 관리도구나 클라이언트 라이브러리의 API 를 사용하여 처리하기 때문에 적재 스키마를 별도로 필요로 하지는 않는다. 단, KRISTAL 2.0.3 에서는 온라인 처리 방식과 동일하게 오프라인에서 KRISTAL-III 데이터베이스에 대한 문서 작업을 벌크로 처리할 수 있는 기법을 지원한다. 이에 대해서는 5.4.2 의 오프라인 임포트 부분을 참고하기 바란다.

벌크 처리방식에서 사용하는 KRISTAL 적재스키마는 관리자가 가지고 있는 다수의 일반 문서 파일을 어떻게 KRISTAL-III 로 적재할 것인가를 지정한다. 예를 들어 원시문서가 시스템의 어느 곳에 위치해 있는지, 문서의 각 섹션은 어떤 구분자를 통해 뽑아낼 수 있는지, 각각의 문서는 어떤 테이블에 저장할 지 등의 정보를 기술하게 된다.

KRISTAL 적재 스키마는 데이터베이스 스키마와 동일하게 XML 문서로 작성되어 있고, 대소문자를 엄격하게 구분한다. 그림 5-2 는 적재 스키마의 구조를 표현하고 있는데, 각 부분들이 순차적으로 정의되도록 작성하는 것이 좋다.

```
<?xml version="1.0" ?>

<LoaderSchema … >
    Part 1 : 적재 문서 구조 정의
    Part 2 : 적재 맵 구조 정의
</LoaderSchema>
```

[그림 5-2] KRISTAL 적재 스키마 구조

그림 5-2 에서와 같이 *LoaderSchema* 태그 안쪽으로 적재 문서 구조와 적재 맵 구조를 정의한다. 전자는 적재 대상으로 삼는 문서의 구조를 정의함으로써 적재기가 문서 구조를 효과적으로 파악하기 위한 수단을 제공하며, 후자는 정의 된 문서 구조 형태를 갖는 대상 문서들을 지정된 테이블에 적재하여 색인이 수행되도록 한다.

관련 엘리먼트: LoaderSchema

관련 속성: database-name, volume-dir, kristal-root

database-name	사용할 데이터베이스의 이름을 지정한다.
volume-dir	사용할 데이터베이스 볼륨의 물리적인 디렉토리 위치를 지정한다.
kristal-root	KRISTAL-III 바이너리의 패키지의 루트 디렉토리를 지정한다.

사용 예

```
<LoaderSchema database-name = "BLUE-DB"
                    volume-dir = "/TEST/BLUE/volume"
                    kristal-root = "/K2002">
  ….
</LoaderSchema>
```

5.3.2 적재 문서 구조 정의

KRISTAL-III 에서 적재 대상으로 삼는 문서의 구조를 정의한다. KRISTAL-III 에 저장될 문서는 반드시 정형화된 규칙에 따라 작성된 구조를 가지는 문서여야 한다. 다만

정형화된 규칙 자체에 대한 제한은 거의 없으며 각각의 문서가 동일한 규칙에 의해 기술되어 있으면 된다. 이러한 문서구조를 KRISTAL 에서는 반정형문서(Semi-structured Document)라 한다.

DocStructure 엘리먼트는 문서구조 스키마의 이름을 지정하고 원시문서에서 각 문서를 구분하는 문서간 경계 구분자(border-string)를 지정하도록 하고 있다.

관련 엘리먼트: DocStructure

관련 속성: name, border-string

name	정의하는 문서 구조에 대한 이름을 지정한다. 구조가 정형화되어 있다고 하더라도 여러 종류의 문서를 데이터베이스로 구축할 수 있기 때문에 이름을 지정하여 구분한다.
border-string	문서를 분석(Parsing)하는 과정에서 문서들을 구분하는 문서 구분 문자열을 지정한다. 그림 5-3 의 “@Yellow_View” 참조.

사용 예

⇨ struct01 이라는 이름을 갖는 문서 구조를 정의하는데, 문서들간의 구분 문자열로서 “@Yellow_View”를 사용한다.

```
<DocStructure name="struct01" border-string="@Yellow_View">
…
</DocStructure>
```

5.3.2.1 KRISTAL-III 적재 대상 문서

KRISTAL 이 적재 대상으로 삼는 문서는 하나의 물리적인 파일 안에 문서 구분 문자열을 이용하여 여러 문서를 한 파일 안에 담은 형태를 가진다(물론 하나의 파일에 하

나의 문서가 담겨 있는 경우라고 하더라도 동일한 규칙이 적용된다). 그림 5-3 을 보면 두 개의 문서가 “@Yellow_View”에 의해 구분되어 한 파일에 담겨 있음을 볼 수 있다. 이러한 문서의 구분자는 앞서 설명한 DocStructure 엘리먼트의 속성인 border-string 을 이용하여 다음과 같이 정의할 수 있다.

<DocStructure name="struct01" border-string="@Yellow_View">

@Yellow_View
#AN=BLUE_00001
#ANI=00001
#TIK=이집트 기술개발 프로그램 홈페이지
#TIO=Technology Development Program hompage
#ABS= 12345 TDP 는 이집트에 있어서 새로운 기술을 기반으로 산업에 적용시키기 위한 분석, 촉진추진체로서 기술개발 프로젝트내용과 활동내용을 포함하고 있다
#LNG=eng
#CON=egy
#CLS=AA
#URL=http://163.121.10.41/tdp/
@Yellow_View
#AN=BLUE_00002
#ANI=00002
#TIK=뉴질랜드 과학의 문
#TIO=Gateway to New Zealand science
#ABS=뉴질랜드의 과학구조와 과학저널 발간물의 카달로그를 볼 수 있고 과학기술자들의 인명데이터를 볼 수 있으며 과학관련 소식을 접할 수 있다
#LNG=eng
#CON=nzl
#CLS=AA
#URL=http://www.rsnz.govt.nz/

[그림 5-3] KRISTAL-III 적재 문서 파일의 예

5.3.2.2 섹션 태그 정의

한 문서 안의 섹션들을 구분하기 위한 태그(Tag)를 정의한다. 태그는 각 섹션간의 구분자 역할을 하며, 그 내용을 저장할 기본섹션까지 지정하도록 되어 있다.

관련 엘리먼트: **Tag**

관련 속성: name, mapping-section, do-skip

name	섹션 구분을 위해 사용 할 태그 이름을 지정한다.
mapping-section	분리 된 해당 섹션 값을 어느 섹션에 지정할지를 결정한다.
do-skip	분리 된 해당 섹션을 무시할지 여부를 결정한다. 설정 값으로 “yes”, “no” 둘 중의 하나의 값을 가질 수 있다. “yes” 값을 설정할 경우, 해당 섹션은 무시되어 저장되지 않는다. 이 속성을 이용하지 않고 태그를 무시할 경우에는 태그 값이 바로 위에 출현하는 섹션의 값으로 저장될 수 있음에 주의하여야 한다.

사용 예

⇨ “#AN=”으로 시작하는 문자열을 섹션 “AN”에 지정한다.

```
<Tag name="#AN=" mapping-section="AN"/>
```

⇨ “#ANI=”로 시작하는 문자열을 가진 섹션을 무시한다.

```
<Tag name="#ANI=" do-skip="yes"/>
```

5.3.3 적재 맵 구조 정의

대상 문서와 대상 테이블을 매칭시킴으로써, 적재를 수행할 수 있도록 한다. 이 과정까지 완료되면 시스템 내의 원시문서와 원시문서가 작성된 구조를 이해할 수 있으므로

이제는 지정된 테이블로 원시문서를 적재할 수 있게 된다. LoaderMap 엘리먼트는 추가적으로 원시문서의 인코딩 방식을 지정하도록 되어 있다.

관련 엘리먼트: **LoaderMap**

관련 속성: table, doc-structure, file, encoding

table	문서를 적재할 테이블을 지정한다.
doc-structure	사용할 문서 구조 정의를 지정한다.
file	적재 대상 문서 파일을 지정한다. (문서 파일 지정 참조)
encoding	적재 대상 문서의 문서 인코딩 방식을 지정한다[8].

사용 예:

```
<LoaderMap table="blue01" doc-structure="struct01"
        file="/home/BLUE/data/blue0*" encoding="EUC-KR" />
<LoaderMap table="blue02" doc-structure="struct01"
        file="/home/BLUE/data/blue1*" encoding="EUC-KR" />
<LoaderMap table="BBS01" doc-structure="structBBS"
        file="/home/BBS/data/bbs*.txt" encoding="UTF-8" />
```

위의 사용예의 첫줄에서는 EUC-KR(한국 인코딩 방식) 방식으로 인코딩된 blue0*의 문서를 struct01 이라는 문서구조(DocStructure)를 이용하여 blue01 이라는 테이블로 적재하라는 것을 지시하고 있다. 두번째 줄도 유사하게 해석할 수 있다. 세번째 줄은 UTF-

[8] KRISTAL-III IRMS 는 유니코드를 기반으로 문서를 저장한다. 보다 정확하게는 UTF-8 인코딩 방식으로 문서를 저장한다. 나라마다, 운영체제마다, 또는 개인별로도 서로 다른 문자 인코딩 방식을 사용할 수 있기 때문에 원시문서를 해석할 때 KRISTAL 시스템은 원시문서의 인코딩 방식을 알아야만 오류없이 문서를 해석할 수 있다.

8 문서인 bbs*.txt 라는 문서를 structBBS 라는 문서구조를 이용하여 BBS01 이라는 테이블로 적재할 것을 명식하고 있다.

5.3.3.1 원시문서 파일 지정

적재할 원시문서를 담고 있는 파일이름(LoaderMap 엘리먼트의 file 속성)에는 절대 경로/상대 경로 및 와일드카드 문자를 허용한다. 사용 가능한 패턴 형태는 아래 목록에 보인다.

지정 패턴	매 칭 파 일
/home/file.txt	/home/file.txt 만을 허용한다.
file.txt	현재 디렉토리의 file.txt 만을 허용한다.
/home/test/data	data 디렉토리 내의 모든 파일을 포함한다. 만약 하위 디렉토리가 존재한다면 모든 하위 디렉토리가 처리의 대상이 된다.
/home/file*	/home 디렉토리의 file 로 시작하는 모든 파일을 허용한다. 만약 file 로 시작하는 하위 디렉토리가 존재한다면 해당 디렉토리도 포함한다.
File[1-9].txt	File1.txt, … File9.txt
File[a-z][A-Z]	Filea, … ,Filez, FileA, … FileZ
/home/blue/data/BLUE*.txt	/home/blue/data 디렉토리에 있는 BLUE 로 시작하고 확장자가 txt 인 모든 파일을 적재 대상으로 한다. (권고)

KRISTAL-III 에서는 원시문서의 파일이름을 가능한 명확하게 사용할 것을 권고한다. 상대적 경로를 사용하게 되면 적재할 수 있는 디렉토리 위치가 한 곳으로 제한되는 단점이 있다. 또한 확장자를 사용하지 않으면 하위 디렉토리가 포함되어 종종 원하지 않는 원시문서들이 적재되는 혼란을 야기하는 경우도 발생한다. 즉, LoaderMap 의 file 속성을 기술할 경우에는 절대 경로를 쓰고, 원시 파일의 확장자를 명시하기를 권고한다. 위 표

의 마지막 줄의 예를 참조하기 바란다.

5.3.3.2 인코딩 방식 지정

KRISTAL 은 다양한 문자셋에 대한 처리를 위해 내부적으로 모든 문서를 유니코드 인코딩 방식(UTF-8)으로 처리하여 저장한다. 따라서 입력 문서의 인코딩 방식을 올바르게 지정하여야만 바른 결과를 얻을 수 있다. KRISTAL 에서 인코딩 방식의 지정을 위해 사용하는 문자셋 이름은 IANA 에 등록된 것[9]을 따른다. 대표적인 인코딩 방식에 대한 예는 다음 목록을 참고하기 바란다.

EUC-KR
EUC-JP
ISO-8859-1
ISO-2022-KR
UTF-8
UTF-16
UTF-32

[9] http://www.iana.org/assignments/character-sets, 국제 표준 기관(ISO)에서 정의한 것은 아니지만 실사용면에서 표준으로 인식하여 사용하고 있다.

5.4 데이터베이스 관리 도구

KRISTAL-III 는 데이터베이스의 관리를 위해서 다양한 관리 프로그램들을 제공한다. 관리 프로그램들은 명령어 방식으로 일반적인 터미널에서 사용할 수 있도록 제공된다. 그래픽 환경에서의 관리를 필요로 하는 사용자는 향후 제공될 <KRISTAL-III GUI 관리 도구>를 참조하기 바란다.

5.4.1 kristal_dbadmin

KRISTAL-III 데이터베이스에 대한 배치 방식의 데이터 관리 작업을 수행하기 위한 도구로서 다음의 기능을 수행할 수 있다.

데이터베이스 스키마 처리
문서(데이터) 적재
적재 문서(데이터) 색인

❒ 데이터베이스 스키마 처리

데이터베이스 스키마에 기반하여 데이터베이스 스키마를 처리한다.

명령어 :

kristal_dbadmin –c –s [데이터베이스 스키마 파일]
kristal_dbadmin –c –f –s [데이터베이스 스키마 파일]

사용 예 :

kristal_dbadmin –c –f –s blue.database.xml

옵션 설명 :

-c : 데이터베이스를 처리할 것을 지시한다.
-f : 데이터베이스가 존재할 경우 이를 제거한 후 생성한다.
-s : 데이터베이스 스키마를 지정한다.

❒ 데이터 적재

적재 스키마에 기반하여 문서를 적재한다.

명령어 :

kristal_dbadmin –l –s [적재 스키마 파일]
kristal_dbadmin –l -f –s [적재 스키마 파일]

사용 예 :

kristal_dbadmin –l –f –s blue.load.xml

옵션 설명 :

-l : 문서를 적재할 것을 지시한다.
-f : 중복 된 문서의 경우 해당 문서 만을 skip 한 후, 적재를 계속 진행한다.
-s : 적재 스키마를 지정한다.

❒ 적재 데이터 색인

적재 스키마에 기반하여 색인되지 않은 문서 만을 색인 한다.

명령어 :

kristal_dbadmin –i –s [적재 스키마 파일]

사용 예 :

kristal_dbadmin –i –s blue.load.xml

옵션 설명 :

-i : 적재 된 문서 중 색인 되지 않은 문서들을 색인 할 것을 지시한다.
-s : 적재 스키마를 지정한다.
-m : 색인 시 사용할 메모리의 크기를 Mbytes 단위로 지정한다. 사용할 수 있는 최소 메모리는 100MB 이며, 최대 메모리는 2GB 이다.

KRISTAL-III 배치 적재도구인 kristal_dbadmin 의 적재 절차는 일반적을 데이터베이스 생성, 데이터 적재, 문서 색인의 3 단계를 거친다. 테이블의 추가 생성 및 적재에서도 마찬가지로 추가 테이블 생성, 추가 데이터 적재, 추가 문서의 색인의 3 단계를 거쳐야 한다. BLUE-DB 라는 데이터베이스를 적재하기 위한 테이블 스키마를 blue.database.xml 이라 하고 적재 스키마를 blue.load.xml 이라 하자. 이 경우 kristal_dbadmin 을 이용한 적재 절차는 다음과 같다.

```
kristal_dbadmin –c –f –s blue.database.xml  [데이터베이스 생성]
kristal_dbadmin –l –f –s blue.load.xml      [데이터 적재]
kristal_dbadmin –i –s blue.load.xml         [문서 색인]
```

위의 예에서 –f 옵션을 이용하면 기존의 테이블이 존재하면 지우는 것에 주의하라. 또한 두 번째 줄에서 중복되는 문서를 제거하기 위해서는 –f 옵션을 사용해야 한다. 데이터 적재시 –f 옵션을 사용하면. 주요키를 검사한 후 중복되는 문서의 적재는 생략하

고 건너뛴다. 위의 과정은 데이터 적재과정과 문서색인 과정을 하나의 명령어로 처리하여 2 단계로 처리할 수 있다. 다음은 위의 예제와 동일하다.

```
kristal_dbadmin –c –f –s blue.database.xml  [데이터베이스 생성]
kristal_dbadmin –li –f –s blue.load.xml     [데이터 적재/색인]
```

적재된 데이터를 대상으로 색인 작업을 수행하여야만 검색 기능을 사용할 수 있다는 것을 기억하여야 한다. 적재시에는 데이터베이스를 대상으로 단순 문서 삽입 기능을 수행하고, 색인 명령시 해당 문서에 대한 색인을 실시한다. 만약 적재와 동시에 색인을 수행하고자 하는 경우에는 “-li”옵션을 사용한다.

KRISTAL-III 의 배치방식 데이터베이스 적재도구인 kristal_dbadmin 의 적재 방식은 데이터베이스의 크기에 큰 영향을 받는다. 보다 정확하게는 단일 테이블에 적재될 문서의 수, 내용이 가장 많은 섹션의 평균적인 크기, 가장 큰 섹션의 색인방식 등에 따라 적재 옵션의 최적화가 필요하다. 예를 들어 BLUE 데이터베이스를 기술하는 테이블 스키마를 blue.database.xml, 적재 스키마를 blue.load.xml 이라고 하자. 데이터베이스의 크기가 크지 않다고 하면, 일반적인 BLUE 데이터베이스의 적재방식은 다음과 같이 간단하게 적용할 수 있다.

일반적인 데이터베이스 적재도구 사용 예 :

kristal_dbadmin –c –f –s blue.database.xml

```
kristal_dbadmin –li –f –s blue.load.xml
```

위의 예에서 첫 줄에서는 blue.database.xml 에 기술된 테이블 스키마를 통해 테이블 구조를 저장하고, 적재에 사용할 테이블을 생성한다. 그리고 두번째 줄의 명령을 통해서 blue.load.xml 에 지정된 원시문서를 앞서 생성된 테이블로 적재한 후(-l 옵션), 색인 작업을 곧이어서 수행한다(-i 옵션).

KRISTAL-III 의 배치형 문서적재도구인 kristal_dbadmin 은 문서의 적재가 끝나면 테이블 단위로 색인작업을 수행한다. 또한 테이블 내에서는 섹션별로 색인작업을 수행한다. 문서의 적재 과정은 각 문서별로 원시문서를 읽어들여서 해당 테이블에 삽입하기 때문에 가장 큰 문서만큼의 메모리가 필요하다. 그러나 색인 작업은 문서단위로 수행하는 것이 아니라 테이블 내의 모든 문서에 대해서 한꺼번에 색인작업을 수행하게 된다(섹션별로 수행한다). 따라서 색인작업에는 매우 큰 메모리가 요구되기 때문에, 시스템이 제공하는 메모리의 크기가 제한되어 있을 경우에는 메모리의 크기에 따라서 테이블의 색인작업을 분할하여야 한다. 이러한 작업 분할을 지정할 수 있는 옵션이 –m 옵션이다.

대용량 데이터베이스 적재도구 사용 예 :

```
kristal_dbadmin –c –f –s blue.database.xml

kristal_dbadmin –li –f –m 1024 –s blue.load.xml
```

위의 예에서는 BLUE 테이블이 매우 용량이 커서 적재시 시스템 메모리를 1024MB 만을 사용하도록 지정한 것이다. 이렇게 1024MB 씩 메모리 분할작업을 지정하면, 해당 섹션에 대한 색인이 대략 1GB 내외의 메모리를 차지할 때까지만 색인작업을 수행하고 그때까지 색인된 데이터를 KRISTAL 색인으로 저장한다. 그리고 다시 색인작업을 재개하여 1GB 까지 메모리를 차지하면, 다시 색인을 쓰게 된다. 모든 문서에 대한 해당 섹션의 색인이 끝날 때까지 이 과정을 반복한다. 이렇게 분할 적재를 하게 되면 한번에

저장할 색인을 여러 번에 걸쳐서 저장해야하는 과정이 필요하기 때문에 추가적으로 많은 시간이 더 소요되는 단점이 있다.

일반적으로 시스템 메모리의 제한이 발생하는 경우는 다음과 같은 경우가 있다. 첫째, 시스템이 제공하는 메모리가 데이터의 전체 크기에 비해 매우 작을 경우이다. 이 경우에는 시스템의 메모리를 증설하기를 권장한다. 둘째, 시스템은 메모리를 충분하게 가지고 있으나 단일 테이블에서 생성되는 색인의 양이 2GB 를 넘는 경우이다. 현재 가장 널리 사용되고 있는 32 비트 계열의 리눅스 장비가 이러한 제한을 가진다. 그러나 64 비트 리눅스는 이러한 제한을 가지지 않는다. 이 경우에는 분할 적재가 반드시 필요하다. 셋째, 매우 큰 메모리를 장착한 시스템에서조차도, 단일 테이블의 단일 섹션의 색인량이 매우 큰 경우이다. 특히 INDEX_BY_CHAR, INDEX_BY_MIX_CHAR, INDEX_BY_MIX_MA, INDEX_DNA, INDEX_PROTEIN 등과 같은 색인 방식은 일반적인 INDEX_BY_MA 나 INDEX_BY_TOKEN 에 비해 색인의 양이 3~4 배 큰 경향을 보인다. 따라서 색인시에도 매우 큰 메모리를 요구하게 된다.

이 같은 메모리 제약에 의한 색인 분할 적재 방식은 데이터베이스의 배치적재속도를 크게 떨어뜨리는 단점이 있다. 이를 해결할 수 있는 방법은 대형 테이블을 보다 작은 여러 개의 테이블로 분할하는 방법이다. 색인분할을 하지 않을 정도로 적당한 테이블로 데이터를 분할하여 적재하면 배치 적재속도를 떨어뜨리지 않을 수 있다.[10]

[10] 그러나 이와 같이 대형 테이블을 여러 개의 작은 테이블로 분할하게 되면 검색속도가 어느정도 저하되는 단점이 있다. 따라서 데이터베이스의 양이 어느 정도 – 1,000 만건 이상 – 증가하면 장비를 물리적을 분산할 것을 권장한다.

5.4.2 kristal_dump

배치 방식의 데이터베이스 관리도구의 하나인 kristal_dump 는 KRISTAL-III 특정 데이터베이스 내의 각종 정보의 열람을 위한 도구로서 다음과 같은 기능들을 수행할 수 있다.

스키마(Table Schema) 정보 열람
카탈로그(Catalog) 정보 열람
문서 열람
색인어 열람
기본 키 열람

사용 예

kristal_dump -acdip -h <데이터베이스 볼륨 디렉토리> -n <데이터베이스명>
[-x 스키마명] [-t 테이블명] [-r 레코드 ID] [-s 섹션명] [-f 출력파일명]

가정) 제시되는 구체적인 예제는 다음 사항을 가정으로 한다.

1) 데이터베이스의 이름은 BLUE-DB 이다.

2) 데이터베이스의 볼륨 디렉토리는 /home/blue/volume 이다.

❒ 스키마 정보 열람

해당 데이터베이스 볼륨내의 i)모든 스키마 명을 열람하거나, ii)명시된 스키마내의 구체적인 정보를 열람한다.

명령어:

i) kristal_dump -a -h <볼륨명> -n <데이터베이스명>
ii) kristal_dump -a -h <볼륨명> -n <데이터베이스명> -x <스키마명>

사용 예:

i) kristal_dump -a -h /home/blue/volume -n BLUE-DB
ii) kristal_dump -a -h /home/blue/volume -n BLUE-DB –x SCHEMA01

❒ 카탈로그 정보 열람

특정 데이터베이스 내의 카탈로그 정보(테이블 정보와 구조 정보)를 열람한다.

명령어:

kristal_dump -c -h <볼륨명> -n <데이터베이스명>

사용 예:

kristal_dump –c -h /home/blue/volume -n BLUE-DB

결과:

****** Describe Tables *******
BLUE01:1:SCHEMA01
BLUE02:2:SCHEMA01
BLUE03:3:SCHEMA01
****

❒ 문서 열람

특정 데이터베이스내의 특정 테이블에 저장된 문서를 열람한다.

명령어:

kristal_dump –d -h <볼륨명> -n <데이터베이스명> -t <테이블명>

사용 예:

kristal_dump -d -h /home/blue/volume -n BLUE-DB -t BLUE01

❒ 색인어 열람

특정 섹션내의 색인어를 열람한다.

명령어:

kristal_dump –i -h <볼륨명> -n <데이터베이스명> -t <테이블명> -s <섹션명>

사용 예:

kristal_dump -i -h /home/blue/volume -n BLUE-DB -t BLUE01 -s AN

❒ 기본 키 열람

기본 키를 열람한다.

명령어:

kristal_dump –p -h <볼륨명> -n <데이터베이스명> -t <테이블명>

사용 예:

```
kristal_dump -p -h /home/blue/volume -n BLUE-DB -t BLUE01
```

5.4.3 kristal_import

KRISTAL-III 데이터베이스에 대한 온라인 문서작업을 수행하기 위한 명령이며, 데이터로 사용될 문서는 규정된 구조에 맞는 XML 문서로 작성되어야 한다.

입력 문서 예)

```
<?xml version="1.0" encoding="EUC-KR"?>

<DOCGRP>

<DOCUMENT action="insert" table="blue04">
<AN>BLUE_00001</AN>
<ANI>1</ANI>
<TIK>이집트 기술개발 프로그램 홈페이지</TIK>
<TIO>Technology Development Program hompage</TIO>
<ABS>이집트에 있어서 새로운 기술을 기반으로 산업에 적용시키기 위한 분석, 촉진추진체로서 기술개발 프로젝트내용과 활동내용을 포함하고 있다</ABS>
<CLS>A1 A2-A3</CLS>
<CON>egy</CON>
<URL>http://163.121.10.41/tdp/</URL>
</DOCUMENT>
…..
<DOCUMENT action="INSERT" table="blue04">
<AN>BLUE_00002</AN>
<ANI>2</ANI>
<TIK>뉴질랜드 과학기술 관련 홈페이지</TIK>
<TIO>Gateway to New Zealand science</TIO>
<ABS>뉴질랜드의 과학구조와 과학저널 발간물의 카달로그를 볼수있고 과학기술자들의 인명데이터를 볼 수 있으며 과학관련 소식을 접할
```

```
수 있다</ABS>
  <CLS>A1-A2-A3</CLS>
  <CON>nzl</CON>
  <URL>http://www.rsnz.govt.nz/</URL>
  </DOCUMENT>

  </DOCGRP>
```

입력문서는 “DOCGRP” 엘레멘트를 루트로 갖고, “DOCUMENT”는 처리될 하나의 문서에 해당된다. DOCUMENT 내의 서브엘레멘트들은 데이터베이스에서 사용되는 섹션의 이름을 사용하여 구성하면된다.

DOCUMENT 엘레멘트에는 action 과 table 속성을 사용할 수 있으며, action 은 문서에 대한 “삽입”,”삭제”,”수정”,”스킵”명령어를 지정한다. 대문자로 지정되면 지정된 action 대로 실행되고, 소문자로 지정되었을 경우에는 변경될 수 있다. table 속성에는 문서가 속한 테이블이름을 지정할 수 있다.

❒ 데이터베이스 온라인 임포트(online import)

입력된 문서를 분석하여 데이터베이스를 갱신(삽입,삭제,수정)한다.

명령어 :

kristal_import -ip<ip address> -port<port number> -f<입력문서>

사용 예 :

kristal_import –ip127.0.0.1 –port32002 –fblue04.xml

옵션 설명 :

-v : 버전 출력.

-ip : 데이터베이스 접근을 위한 IP 주소
-port : 데이터베이스 접근을 위한 포트번호
-f : 사용자 입력 파일
-execute : "사용자 입력 파일"을 체크한 뒤 실행
-EXECUTE : 체크없이 바로 실행
-p : import 작업 수행 시 나타나는 상세한 정보 출력
-o : import 작업 중 실패가 발생하였더라도, 건너 뛰고 다음 작업을 수행
-l : import 작업과 관련된 정보를 로그파일에 기록
-t : 테이블 이름을 지정
-a : 액션(삽입|삭제|수정|스킵)을 정의함
소문자(insert|delete|update|skip)인 경우에는 문서에 정의된 액션을 따름
대문자인 경우에는 문서에 정의된 액션이 소문자거나 정의되지 않은 경우에 적용됨

❒ 데이터베이스 오프라인 임포트(offine import)

KRISTAL-2.0.3 에서는 온라인 문서 작업을 오프라인 임포트 스키마와 kristal_dbadmin 도구를 이용하여 벌크로 처리하는 방식을 지원한다.

오프라인 임포트 스키마 구조 예)

```
<?xml version="1.0" ?>

<XMLBulkLoaderSchema … >
    <XMLBulkLoaderMap … />
    …
</XMLBulkLoaderSchema>
```

관련 엘리먼트: XMLBulkLoaderSchema

관련 속성: database-name, volume-dir, kristal-root

database-name	사용할 데이터베이스의 이름을 지정한다.
volume-dir	사용할 데이터베이스 볼륨의 물리적인 디렉토리 위치를 지정한다.

kristal-root	KRISTAL-III 바이너리의 패키지의 루트 디렉토리를 지정한다.

관련 엘리먼트: XMLBulkLoaderMap

관련 속성: table, action, file

table	XML 파일 내의 문서들이 임포트될 기본 테이블을 지정한다.
action	XML 파일 내의 각 문서에 대한 삽입/수정/삭제/생략에 대한 기본 행동을 지원한다
file	임포트될 문서가 있는 XML 파일을 지정한다.

사용 예:

```
<XMLBulkLoaderSchema database-name = "BLUE-DB"
                     volume-dir = "/TEST/BLUE/volume"
                     kristal-root = "/K2002">
    <XMLBulkLoaderMap table="BLUE01" action="INSERT"
                      file="/home/k2002/TEST/BLUE2/data/blue01.xml" /
    <XMLBulkLoaderMap table=" BLUE04" action="SKIP"
                      file="/home/k2002/TEST/BLUE2/data/blue04.xml " />
</XMLBulkLoaderSchema>
```

오프라인 임포트 스키마를 이용한 kristal_dbadmin 실행)

1. 임포트

명령어 : kristal_dbadmin –b –s [오프라인 임포트 스키마 파일]

kristal_dbadmin –b -f –s [오프라인 임포트 스키마 파일]

사용 예 :

kristal_dbadmin –b –f –s blue.xml_bulk_load.sch

옵션 설명 :

-b : 지정된 스키마 파일에 있는 XML 파일 내의 문서를 임포트 할 것을 지시한다.

-f : 중복된 문서의 경우 해당 문서 만을 skip 한 후, 임포트를 계속 진행한다.

-s : 오프라인 임포트 스키마를 지정한다.

2. 임포트된 문서에 대한 색인

명령어 :

kristal_dbadmin –x –s [오프라인 임포트 스키마 파일]

사용 예 :

kristal_dbadmin –x –s blue.xml_bulk_load.sch

옵션 설명 :

-x : 임포트된 XML 파일 내의 문서 중 색인 되지 않은 문서들을 색인할 것을 지시한다.

-s : 오프라인 임포트 스키마 파일을 지정한다.

-m : 색인 시 사용할 메모리의 크기를 Mbytes 단위로 지정한다. 사용할 수 있는 최소 메모리는 100MB 이며, 최대 메모리는 2GB 이다.

5.4.4 log_analyzer

KRISTAL-III 데몬이 수행한 명령어에 대한 로그파일을 분석하고, 재실행하는 도구이다..

로그파일 예)

```
[Child-Pid] : 16313
Wed Sep 28 10:07:25 2005
<Message>
<Header>
   <Version>2002.2.0.4</Version>
</Header>
<Body>
   <Process>RETRIEVE</Process>
   <Object>
      <Table_Name>X0tSSVNUQUxfQUxMX1RBQkxFU18=</Table_Name>
      <Space_Operator>0</Space_Operator>
      <Method>0</Method>
      <Query>KFRJSyA6IOydtOynke2KuCAmIO2ZiO2OmOydtOyngCk=</Query>
      <Term_Expansion>2</Term_Expansion>
      <Remove_Chars_Word>1</Remove_Chars_Word>
      <Order>FALSE</Order>
      <Integrate>
          <Is_Used>FALSE</Is_Used>
         <Sorting_Key_Type>-1</Sorting_Key_Type>
         <Order>FALSE</Order>
         <Group_Type>1</Group_Type>
         <Counter>0</Counter>
      </Integrate>
   </Object>
```

```
</Body>
</Message>
```

위의 로그파일은 검색을 수행하였을 때, 통신상으로 전달되는 입력메시지이며, 문서는 XML 로 구성되어 있고, 문자열이 입력되는 부분은 base64 로 인코딩되어 있다. 로그파일을 생성하려면 데몬 수행시(kristald) –L(검색관련 로그만 생성), -l(모든 로그를 생성) 옵션을 사용하면 된다.

❒ 로그파일 분석

입력된 로그파일을 분석하여 내용 확인을 하거나, 로그파일에서 수행된 명령어들을 재실행 한다..

명령어 :

log_analyzer [-eup] -l <Log filename> -c [config filename]

사용 예 :

log_analyzer -e -l sample.log

옵션 설명 :

-e : 출력을 EUC-KR 로 변경(디폴트는 UTF-8 로 출력됨).
-u : 환경화일(데몬 수행시 사용하는)을 입력받아 업데이트도 수행함
환경화일만 입력받으면 검색만 수행함(업데이트 연산 수행 안함)
-s : 수행 중 에러가 발생하면 일단 멈춤
-l : 로그 파일 입력 받음

-c : 환경화일 입력 받음

5.5 데이터베이스 관리 예제

다음은 특정 작업을 위해서 데이터베이스 스키마와 적재 스키마를 작성하는 예제로서, 5.3.1 에서 설명한 kristal_dbadmin 이라는 KRISTAL-III 관리 도구를 이용하여 처리할 수 있다.

5.5.1 데이터베이스 관련 예제

KRISTAL 배치 관리도구인 kristal_dbadmin 을 이용한 데이터베이스 생성과 삭제의 두 예제를 설명한다.

5.5.1.1 데이터베이스 생성 예제

```
<?xml version 1.0"?>
<DatabaseSchema>
        <CreateDatabase database-name="BLUE-DB" volume-dir="/TEST/BLUE/volume" />
</DatabaseSchema>
```

위의 예제 XML(파일이름을 blue.create.xml 이라고 하자)과 kristal_dbadmin 을 사용하면 /TEST/BLUE/volume 디렉토리에 BLUE-DB 데이터베이스를 생성할 수 있다.

사용 예:

```
kristal_dbadmin –c –f –s blue.create.xml
```

5.5.1.2 데이터베이스 삭제 예제

```
<?xml version="1.0"?>
<DatabaseSchema>
        <DeleteDatabase database-name="KTIMS-DB" volume="/KTIMS/volume" />
</DatabaseSchema>
```

위의 예제 XML(파일이름을 ktims.delete.xml 이라고 하자)과 kristal_dbadmin 을 사용하면 /KTIMS/ volume 디렉토리에 이미 생성되어 있는 KTIMS-DB 데이터베이스를 삭제할 수 있다. 아래의 사용 예에서 생성에서와 동일한 –c 옵션을 사용하는 것에 주의하기 바란다.

사용 예:

```
kristal_dbadmin –c –f –s ktims.create.xml
```

5.5.2 테이블 스키마 관련 예제

데이터베이스 스키마 생성과 삭제의 두 예제를 설명한다. 아래 두 예제는 모두 kristal_dbadmin 관리도구를 이용하여 5.5.1 의 테이블 생성 및 삭제 예제와 동이한 방식으로 처리할 수 있다.

5.5.2.1 테이블 스키마 생성 예제

```
<?xml version="1.0"?>
<DatabaseSchema>
    < UseDatabase database-name="BLUE-DB" volume-dir="/TEST/BLUE/volume" />

    <!-- Table Schema Definitions -->
    <CreateTableSchema name="schema01" alias=”schema example”>
        <!-- Primary Key Definition -->
        <PrimaryKey sections="AN"/>

        <!-- Stopword File Definition -->
        <Stopword file="/TEST/stopword/swords-eng"/>
        <Stopword file="/TEST/stopword/swords-han"/>

        <!-- Basic Section Definitions -->
        <BasicSection name="AN" data-type="KCHAR[10]" index-type="INDEX_AS_IS" />
        <BasicSection name="ANI" data-type="KINT" index-type="INDEX_AS_NUMERIC" />
        <BasicSection name="TIK" data-type="KSTRING" index-type="INDEX_BY_MA" />
        <BasicSection name="TIO" data-type="KSTRING" index-type="INDEX_BY_MA" />
        <BasicSection name="ABS" data-type="KSTRING" index-type="INDEX_BY_MA" />
        <BasicSection name="URL" data-type="KSTRING" index-type="INDEX_AS_IS" />

        <!-- Virtual Section Definitions -->
        <VirtualSection name="VTIK" source-section="TIK" index-type="INDEX_BY_CHAR"/>
```

```
        <!-- Union Section Definitions -->
        <UnionSection name="BASIC" include-sections="TIK TIO ABS" />

    </CreateTableSchema>
</DatabaseSchema>
```

5.5.2.2 테이블 스키마 삭제 예제

```
<?xml version="1.0"?>
<DatabaseSchema>
        <UseDatabase database-name="KTIMS-DB" volume="/KTIMS/volume" />
        <DeleteTableSchema name="SCHEMA01"/>
        <DeleteTableSchema name="SCHEMA02"/>
</DatabaseSchema>
```

5.5.3 테이블 관련 예제

테이블 생성과 삭제의 두 예제를 설명한다.

5.5.3.1 테이블 생성(추가) 예제

다음 예는 C1.TIT.002 와 C1.BIB.002 라는 테이블을 추가 생성하는 예제로서, 테이블 스키마 schema01 은 이미 생성되어 있다고 가정한다.

```
<?xml version="1.0"?>
<DatabaseSchema>
    <UseDatabase database-name="KTIMS-DB" volume="/KTIMS/volume" />
    <CreateTable table-name="C1.TIT.002" with-schema="schema01"/>
    <CreateTable table-name="C1.BIB.002" with-schema="schema01"/>
</DatabaseSchema>
```

5.5.3.2 테이블 삭제 예제

```
<?xml version="1.0"?>
<DatabaseSchema>
        <UseDatabase database-name="KTIMS-DB" volume="/KTIMS/volume" />
        <DeleteTable table-name="table01"/>
        <DeleteTable table-name="table02"/>
</DatabaseSchema>
```

5.5.4 데이터 적재 스키마 예제

본 절에서는 PLAIN TEXT[11]와 BLOB 데이터의 적재에 관한 사항만을 설명한다. XML 데이터의 처리에 관한 사항은 8 장을 참조한다.

5.5.4.1 통합 데이터베이스 스키마 예제

다음은 데이터베이스, 테이블 스키마, 테이블을 모두 생성하는 데이터베이스 스키마 예제로서, PLAIN TEXT 와 BLOB 데이터 적재 시 공통으로 사용된다. 다만 'FILE' 섹션은 BLOB 데이터에서만 사용되는 섹션이다.

```
<?xml version="1.0"?>
<DatabaseSchema>
    <CreateDatabase database-name="BLUE-DB" volume-dir="/TEST/BLUE/volume" />

    <!-- Table Schema Definitions -->
    <CreateTableSchema name="schema01">
        <!-- Primary Key Definition -->
        <PrimaryKey sections="AN"/>

        <!-- Stopword File Definition -->
        <Stopword file="/TEST/stopword/swords-eng"/>
        <Stopword file="/TEST/stopword/swords-han"/>

        <!-- Basic Section Definitions -->
        <BasicSection name="AN" data-type="KCHAR[10]" index-type="INDEX_AS_IS" />
        <BasicSection name="ANI" data-type="KINT" index-type="INDEX_AS_NUMERIC" />
        <BasicSection name="TIK" data-type="KSTRING"
```

[11] PLAIN TEXT 란 문서내의 각 섹션이 서로 독립적인 형태의 문서를 의미한다.

```
                    index-type="INDEX_BY_MA"
                    use-stemming=”YES”
                    remove-stopword=”YES”
                    hanja-to-hangul=”YES”
                    default-value=”” />
        <BasicSection name="TIO" data-type="KSTRING" index-type="INDEX_BY_MA" />
        <BasicSection name="ABS" data-type="KSTRING"
                    index-type="INDEX_BY_MA"
                    remove-stopword=”YES”
                    hanja-to-hangul=”YES” />
        <BasicSection name="URL" data-type="KSTRING" index-type="INDEX_AS_IS" />
        <BasicSection name="FILE" data-type="KBLOB" />

        <!-- Virtual Section Definitions -->
        <VirtualSection name="VIR_TIK" source-section="TIK"
                    index-type="INDEX_BY_TOKEN"
                    use-stemming=”YES”
                    remove-stopword=”YES”
                    hanja-to-hangul=”YES” />

        <!-- Union Section Definitions -->
        <UnionSection name="BASIC" include-sections="TIK TIO ABS" />

    </CreateTableSchema>

    <CreateTable table-name="blue01" with-schema="schema01"/>
    <CreateTable table-name="blue02" with-schema="schema01"/>
    <CreateTable table-name="blue03" with-schema="schema01"/>
</DatabaseSchema>
```

5.5.4.2 PLAIN TEXT 적재를 위한 적재 스키마 예제

다음은 위 절에서 생성된 데이터베이스를 대상으로 PLAIN TEXT 적재를 수행하기 위한 적재 스키마 예제이다. 문서의 구조는

```
<?xml version="1.0" ?>
<LoaderSchema database-name="BLUE-DB" volume-dir="/BLUE/volume" kristal-root="/k2002">
    <DocStructure name="struct01" border-string="@Yellow_View">
        <Tag name="#AN=" mapping-section="AN"/>
        <Tag name="#ANI=" mapping-section="ANI" do-skip="no"/>
        <Tag name="#TIK=" mapping-section="TIK"/>
        <Tag name="#TIO=" mapping-section="TIO"/>
        <Tag name="#ABS=" mapping-section="ABS"/>
        <Tag name="#REM=" mapping-section="" do-skip="yes"/>
        <Tag name="#LNG=" mapping-section="" do-skip="yes"/>
        <Tag name="#CON=" mapping-section="" do-skip="yes"/>
        <Tag name="#CLS=" mapping-section="" do-skip="yes"/>
        <Tag name="#URL=" mapping-section="URL"/>
    </DocStructure>

    <LoaderMap table="blue01" doc-structure="struct01" file="/data/blue0*" encoding="EUC-KR" />
    <LoaderMap table="blue02" doc-structure="struct01" file="/data/blue1*" encoding="EUC-KR" />
    <LoaderMap table="blue03" doc-structure="struct01" file="/data/blue2*" encoding="EUC-KR" />
</LoaderSchema>
```

위의 예제에서 사용한 원시문서의 예제는 아래와 같다.

```
@Yellow_View
#AN=BLUE_00001
#ANI=00001
#TIK=이집트 기술개발 프로그램 홈페이지
```

#TIO=Technology Development Program hompage

#ABS= 12345 TDP 는 이집트에 있어서 새로운 기술을 기반으로 산업에 적용시키기 위한 분석, 촉진추진체로서 기술개발 프로젝트내용과 활동내용을 포함하고 있다

#LNG=eng

#CON=egy

#CLS=AA

#URL=http://163.121.10.41/tdp/

@Yellow_View

#AN=BLUE_00002

#ANI=00002

#TIK=뉴질랜드 과학의 문

#TIO=Gateway to New Zealand science

#ABS=뉴질랜드의 과학구조와 과학저널 발간물의 카달로그를 볼 수 있고 과학기술자들의 인명데이터를 볼 수 있으며 과학관련 소식을 접할 수 있다

#LNG=eng

#CON=nzl

#CLS=AA

#URL=http://www.rsnz.govt.nz/

5.5.4.3 BLOB 데이터 적재 예제

다음은 BLOB 데이터 적재 시 사용하는 적재 스키마 예제로서, PLAIN TEXT 의 경우와 거의 같다. 문서구조(DocStructure) 정의시, 아래 밑줄친 부분과 같이 BLOB 섹션과 관련된 TAG 를 명시하여야 한다.

```
<?xml version="1.0" ?>
<LoaderSchema database-name="BLUE-DB" volume-dir="/BLUE/volume" kristal-root="/k2002">
    <DocStructure name="struct01" border-string="@Yellow_View">
        ……. (중략) ……
        <Tag name="#FILE=" mapping-section="FILE"/>
    </DocStructure>
```

```
    <LoaderMap table="blue01" doc-structure="struct01" file="/blue01.txt" encoding="EUC-KR" />
</LoaderSchema>
```

BLOB 데이터를 적재하기 위해서는 문서 구조뿐만 아니라 원시 데이터 파일에서 적재하고자 하는 BLOB 파일을 지정하여야 한다. 예를 들면, “blue01.txt” 파일이 아래 내용을 포함하고 있을 경우, 데이터베이스 관리기는 “/home/blue/binary.dat” 파일 안의 바이너리 데이터를 해당 BLOB 섹션에 적재하게 되는 것이다.

```
…… (생략) ……
#FILE=/home/blue/binary.dat
```

이상과 같이 두 가지 모두를 명시하여야만 올바른 BLOB 적재 결과를 취할 수 있다.

제 6 장 사용자 질의 및 검색

이 장에서는 KRISTAL 에서 사용할 수 있는 사용자 질의문과 몇가지의 검색 방법에 대해서 설명한다. 1 절에서는 개략적인 사용자 질의의 특성에 대해서 설명하고, 2 절에서는 지원 검색 방법의 특성과 함께 각 검색 방법에서의 질의 예제를 설명하고, 3 절에서는 각 검색방법에 대한 사용자의 이해를 돕기위한 각 검색방법과 지정할 수 있는 옵션들에 대해서 설명한다.

6.1 사용자 질의

사용자 질의에서 다루고자 하는 내용은 다음과 같다.

- 질의문의 기본 형식
- 색인 타입에 따른 탐색어 구성의 제약
- 지정된 검색 대상 섹션이 영향을 미치는 범위
- 연산자 우선 순위
- 문자열 지정자의 사용
- 질의 확장

6.1.1 질의문의 기본 형식

KRISTAL-III 데이터베이스 검색기에서 사용하는 질의 처리 엔진은 섹션[12]이라는 검색 대상 식별자를 검색 대상의 기본 단위로 삼는다. 이로부터 검색 대상 식별자(섹션), 검색범위 지시자(섹션 지정자[13], 관계 연산자[14], 범위 연산자[15]), 단어들로 구성된 탐색어는 실질적인 검색의 최소 단위가 된다. 이 탐색어들은 하나 이상의 불리언 연산자[16], 근

[12] 섹션은 일반적인 데이터베이스의 필드와 유사한 개념이다
[13] “:”에 의해 표기됨
[14] =, <, =<, >=, >
[15] ~
[16] “& AND” - AND 연산자, “| OR” - OR 연산자, “! NOT” - NOT 연산자. 불리언 연산자는 불리안

접도 연산자[17], 괄호에 의해 결합된다.

KRISTAL 질의문 작성에 대한 문법 수준의 기본형식은 [그림 6-1]에 기술되어 있다.

```
<질의어> =  ( <질의어> )
            <질의어> <불리언 연산자> <질의어>
            <질의어> <근접도 연산자> <질의어>
            <탐색어>
<탐색어> = <섹션리스트> : <검색단어 리스트>
            <섹션리스트> : ( <검색단어 리스트> )
            <섹션리스트> : " <검색단어 리스트> "
            <섹션리스트> <관계 연산자> <단어>
<섹션 리스트> = <섹션리스트> , <섹션>
                  <섹션>
<검색단어 리스트> = <검색단어 리스트> <공백> <검색단어>
                        <검색단어>
<검색단어> = <단어>
               <단어>*
```

[그림 6-1] 질의문법의 기본 형식

이러한 질의문법에 따라 다양한 형태의 질의가 작성될 수 있는데, 다음에 몇 가지

질의에 대해서만 효과가 있으며 검색모델을 벡터연산으로 지정하였을 경우에는 무시된다.

[17] "/N" - NEAR 연산자, "/W" - WITHIN 연산자: A /N2 B 는 A 와 B 의 순서에 관계없이 두 단어 이내에 출현하는 경우가 모두 참이며, A /W1 B 는 A 와 B 가 순서대로 연이어 출현하는 문서만이 참이다.

KRISTAL 질의 사용 예를 보인다.

❒ 기본질의

ORGANIZATION 이라는 섹션에서 “한국과학기술정보연구원”을 찾고자 할 경우에는 “:” 지정자를 사용한다.

ORGANIZATION: 한국과학기술정보연구원

❒ 검색 대상 섹션 다중 정의

검색 대상 섹션을 두 개 이상 지정할 때에는 “,” 지정자를 사용한다. (1)은 “정보 검색”이라는 검색어에 TITLE 과 CONTENT 의 두 가지 섹션을 지정한 것으로, 이것은 (2)와 같은 의미를 가진다.

(1) TITLE, CONTENT : 정보 검색
(2) TITLE : 정보 검색 | CONTENT : 정보 검색

이것은 KRISTAL 의 초기버전이었던 KRISTAL-II 와의 호환성을 제공하기 위한 것이며, 향후 KRISTAL-III 이후 버전의 질의에서는 지원하지 않을 계획이므로 가급적 사용을 자제해 줄 것을 부탁한다. 다중 섹션을 사용해야 할 경우에는 앞서 설명한 KRISTAL-III 의 통합섹션(Union Section)을 사용하여 동일한 효과를 낼 수 있다.

6.1.2 색인 타입에 따른 탐색어 구성의 제약

KRISTAL-III 데이터베이스의 섹션은 일반적인 데이터베이스의 필드와 유사하지만, KRISTAL 은 정보검색을 그 출발점으로 삼기 때문에 각 섹션은 하나의 데이터 타입과 필요에 따라 하나의 색인 타입을 가진다. 이때, 데이터 타입은 각 섹션의 데이터 유형

을 지정하기 위한 것이고, 색인 타입은 각 섹션에 대한 검색을 수행하기 위한 색인방식을 지정하는데 사용된다.

KRISTAL 에서는 데이터 타입과 색인 타입을 자유롭게 결합하여 사용할 수 있지만, 특정 연산의 경우 적용될 수 있는 데이터 타입과 색인 타입에 제약이 있다. [표 6-1]은 색인 타입에 따른 탐색어 구성의 제약을 설명하는 것으로써, 관계 연산자(=, <, >, >=, <=) 및 범위 연산자(~)의 경우에 한해 2 가지 색인타입만이 허용됨을 명시하고 있다.

[표 6-1] 색인 타입에 따른 탐색어 구성의 제약

연산자	데이터 타입	색인 타입	탐색어 구성 예
관계 연산자	문자열	INDEX_AS_IS	SECTION: WORD SECTION = WORD SECTION < WORD SECTION > WORD SECTION <= WORD SECTION >= WORD SECTION:WORD~WORD
	숫자형	INDEX_AS_NUMERIC	
섹션 연산자	문자열	INDEX_BY_MA INDEX_BY_TOKEN INDEX_BY_CHAR INDEX_BY_MIX_CHAR INDEX_BY_MIX_MA INDEX_PROTEIN INDEX_DNA	SECTION: WORD

[표 6-1]에 제시한 INDEX_AS_IS 와 INDEX_AS_NUMERIC 색인방식의 탐색어 구성예에서 “SECTION: WORD”와 “SECTION = WORD”의 의미는 동일하다. 관계연산자를 사용할 수 있는 섹션과 섹션연산자만을 사용할 수 있는 섹션의 차이는 색인방식에 있다. INDEX_AS_IS 와 INDEX_AS_NUMERIC 은 섹션 값 전체에서 하나의 색인어가 추출된다. 즉 섹션값 전체를 하나아의 색인어로 취급하는 것이다. 반면 INDEX_BY_MA,

INDEX_BY_TOKEN 등과 같은 색인 방식은 섹션값을 분석하여 다수의 색인 값이 추출되므로 관계 연산자가 의미가 없다. 따라서 이러한 색인타입에서는 관계 연산자를 지원하지 않는다.

다음은 색인 타입에 따른 질의 사용 예를 보인 것으로, TITLE(제목)은 형태소 분석을 하는 INDEX_BY_MA 색인 타입으로, AUTHOR 는 섹션 전체를 색인으로 삼는 INDEX_AS_IS 타입으로, DATE 는 섹션값을 숫자로 색인하는 INDEX_AS_NUMERIC 타입으로 색인되었다고 가정한다.

(1) TITLE : 유전자
(2) TITLE : 유전자 정보
(3) TITLE : (가정용 컴퓨터 시스템)
(4) AUTHOR : 홍길동
(5) AUTHOR > 홍길동
(6) DATE > 1990
(7) TITLE > 유전자
(8) TITLE <= 유전체
(9) **DATE : 1990 ~ 2005** (DATE>=1990 AND DATE<=2005)

[표 6-1]에서 설명한 바와 같이 위의 (1)부터 (3)까지는 INDEX_BY_MA 색인 타입이므로 올바른 질의 구성이며, (4)~ (5)까지는 INDEX_AS_IS 색인이므로 올바르다. 또한 (6)의 경우에도 INDEX_AS_NUMERIC 의 색인 방식을 채택하고 있으므로 올바른 탐색어 구성 예이다. 그러나, TITLE 섹션에 대한 (7)부터 (8)까지는 올바르지 않은 경우이다. TITLE 은 INDEX_BY_MA 색인 타입을 가지고, [표 6-1]에 따르면 관계연산자의 사용이 허용되지 않는 섹션이다. (9)은 범위 연산자로서 DATE>=1990 AND DATE<=2005 와 같은 의미로 사용되고 검색 처리하는 속도 측면에서는 범위 연산자(~)가 더 빠르다. 그러므로 범위 연사자를 사용하는 것을 권장한다.

6.1.3 지정된 검색 대상 섹션이 영향을 미치는 범위

원칙적으로, 모든 검색 단어에는 섹션이 명시되어야 하며 그렇지 않은 경우에는 질의 작성 오류로 처리된다. 그러나 이전 검색 단어의 섹션 지정을 그대로 계승하는 경우에 한해서 질의 문맥의 의미에 따른 섹션 생략을 부분적으로 허용한다. 단 섹션 지정자(:)가 사용된 경우에만 그 생략을 허용하고, 관계연산이 사용된 경우에는 반드시 섹션을 명시하여야 한다.

❒ 섹션의 생략을 허용하는 경우

사용자가 제목(TITLE)에서 ‘car audio system’을 찾고자 하는 것으로, [표 6-2]의 오른쪽 열과 같이 표현되어야 하지만, 사용의 편의성을 위해 왼쪽 열의 질의 형태를 허용한다. 왼쪽 열에서 괄호의 사용은 무방하고, ‘car’, ‘audio’, ‘system’은 모두 TITLE 이라는 섹션 지정이 된 것으로 간주한다.

[표 6-2] 허용 질의 vs 원질의 의미

허용 질의	원 질의 의미
TITLE : car & audio & system TITLE : (car & audio & system) TITLE : (car) & audio & system TITLE : ((car & audio) & system)	TITLE : car & TITLE : audio & TITLE : system

❒ 섹션의 생략을 허용하지 않는 경우

다음 예제에서 (1)과 같은 형태도 허용하는데, 이 경우 ‘system’은 이전 섹션인 ‘TITLE’을 기본적으로 계승하기 때문에 가능한 경우이다 – 다만 이러한 방식의 질의 구성은 추천하지 않는다. (2)의 경우 질의어 ‘car’는 이전 섹션이 존재하지 않기 때문에 질의 구성 에러로 처리 한다.

(1) (TITLE: car & audio) & system
(2) car & TITLE : audio & system

관계 연산자의 경우는 섹션 생략이 허용되지 않는데, 이 경우에 대한 예제는 [표 6-3]에 보인다. [표 6-3]의 모든 예제는 잘못된 질의 구성이다.

[표 6-3] 섹션의 생략이 허용되지 않는 잘못된 예제

관계 연산자의 사용
(1) DATE > 1990 | 2002
(2) DATE > (1990 & 2002)
(3) DATE = 1990 | 2002
(4) DATE = (1990 & 2002)

[표 6-3]에서 질의 (1)은 의미상 “DATE > 1990”으로 올바르게 변환하여 사용할 수 있다. 질의 (2)는 (1990 & 2002) = false 이므로 검색결과가 존재 하지 않는다. 질의 (3)은 “DATE = 1990 | DATE = 2002”로 올바르게 고칠 수 있다. 질의 (4)는 질의 (2)와 마찬가지로 질의 자체의 검색결과가 존재할 수 없다.

KRISTAL-III IRMS 가 제공하는 불리안 연산자(Boolean Operator)는 모두 이진 연산자(Binary Operator)이다. 이진 연산자의 의미는 연산자의 좌우에 피연산자가 존재해야 한다는 것을 의미한다. 따라서 전체 질의구성에서 섹션을 제외한 순수 질의에 불리안 연산자가 이용되는 경우에는 연산자의 좌우에 질의어 또는 질의식이 존재해야 한다. 연산자를 사용하지 않고 공백으로만 처리할 경우에는 공백처리에 대한 지정을 해주어야 한다. 다음은 올바른 질의의 예제이다. 예제에서 TITLE 섹션은 INDEX_BY_MA, AUTHOR 섹션은 INDEX_BY_TOKEN, ABSTRACT 섹션은 INDEX_BY_MA 방식으로 색

인되었다고 가정한다.

(1) TITLE: 정보검색
(2) TITLE: 정보 & 검색
(3) TITLE: 정보 AND 검색 AND 시스템
(4) ABSTRACT: 정보 /w1 검색 /w1 시스템
(5) (TITLE: Information Retrieval) & (AUTHOR: Gerard Salton)

위의 표에서 검색식 (1)은 TITLE 섹션에서 '정보검색'을 포함하고 있는 문서를 검색한다. '정보검색'이라는 질의는 질의 처리 옵션에 따라 두가지 방식으로 처리될 수 있다. 예를 들어 질의시 클라이언트의 변수를 – C++ API 에서는 Cparameter_t p_in 으로 선언됨 – p_in.term_expansion = FULL_QUERY_TERM_EXPANSION 으로 선언한다고 하자. 이 변수의 의미는 사용자가 입력하는 질의도 해당 섹션의 색인방식과 동일하게 색인을 추출한 후에 검색을 수행하라는 의미이다. TITLE 섹션은 INDEX_BY_MA 방식으로 색인이 되므로 이 경우에는 검색식 (1)은 (TITLE: 정보 /W1 검색)으로 확장된다. 따라서 TITLE 섹션에 "정보 검색"이 있는 문서도 검색 결과로 제공된다. 반면 p_in.term_expansion = NO_QUERY_TERM_EXPANSION 으로 선언하면, 사용자가 제공하는 질의를 그대로 검색에 사용하게 된다. 이 경우에는 섹션값의 색인결과가 '정보검색'을 포함하는 문서만을 검색결과로 제공하며 "정보 검색"이 포함된 문서는 결과로 제공되지 않는다.

검색식 (2)는 TITLE 에 '정보'와 '검색'이라는 단어가 위치에 상관없이 모두 포함된 문서를 검색결과로 제공한다. 검색식 (3)도 (2)와 마찬가지로 '정보', '검색', '시스템'이라는 단어를 모두 포함하는 문서를 제공한다. 검색식 (4)는 '정보', '검색', '시스템'이 순서대로 1 단어 이내에 출현하는 문서를 검색 대상으로 한다. 섹션값으로 "정보검색시스템", "정보 검색 시스템", "정보 검색시스템", "정보검색 시스템" 등의 문자열이 포함된 문서가 검색대상이 된다.

검색식 (5)에서는 공백문자의 처리를 AND 로 처리한다고 가정하자. 공백처리에 대한 지정은 p_in.space_operator 변수로 지정할 수 있다. 공백문자는 SPACE_AND, SPACE_OR,

SPACE_NOT, SPACE_WITHIN, SPACE_NEAR 로 지정할 수 있다. 이 경우에는 AND 로 처리하기로 하였으므로 검색 수행전에 p_in.space_operator = SPACE_AND 로 지정하여 검색을 수행하면 된다. 따라서 검색식 (5)는 제목(TITLE)에 “Inforamtion”과 “Retrieval”이 동시에 포함되어 있고 저자(AUTHOR) 섹션의 값은 “Gerard”와 “Salton”이 모두 출현하는 문서들을 검색 대상으로 한다.

(6) TITLE: 정보 NOT 검색
(7) ABSTRACT: 황금 NOT 백금
(8) (TITLE: 정보 검색) | (ABSTRACT: 정보 검색)
(9) ABSTRACT: information /n2 content

검색식 (6)과 (7)은 모두 올바른 형태의 질의이다. 앞서 설명한 바와 같이 KRISTAL 의 불리안 연산자는 모두 이항연산자로 연산자의 좌우에 피연산자를 필요로 한다. 따라서 NOT 의 경우에도 “정보 NOT 검색”은 “정보”를 포함하지만 “검색”은 포함하지 않는 문서를 검색한다. “황금 NOT 백금”의 경우에도 “황금”은 포함하지만 “백금”은 포함하지 않는 문서만을 검색대상으로 한다. 그러나 (TITLE: NOT 검색)과 같은 질의는 잘못된 질의이다. 질의의 의미는 “검색”을 포함하지 않는 모든 문서를 가져오겠다는 의미이지만 KRISTAL 질의에서는 시스템의 성능을 위하여 단항연산자로서의 NOT 연산을 지원하지 않는다.

검색식 (8)은 TITLE 이나 ABSTRACT 에 “정보 검색”을 포함하는 문서를 검색대상으로 한다. “정보 검색”에서의 공백의 의미는 앞서 설명한 바와 같이 클라이언트 변수인 p_in.sapce_operator 로 지정할 수 있다. 검색식 (9)는 주로 영문에서 활용할 수 있는 근접연산자인 NEAR 의 예를 보여주고 있다. “information /n2 content”의 의미는 ‘information’과 ‘content’의 두 단어가 두 단어 이내에 출현하는 문서를 검색할 것을 지정하는 것이다. 일반적으로 “information content”와 “conent of information”은 동일한 의미를 가진다고 가정하면, “information /w1 content”로는 “content of information”이라는 문자열을 가진 문서를 검색할 수 없다 이러한 경우에는 근접은 하더라도 단어 위치의 순서를 고려하지 않는 NEAR 연산자를 사용하여 (ABSTRACT: information /n2 content)와 같은 질

의를 구성할 수 있다.

6.1.4 연산자 우선 순위

질의문에 여러 가지 연산자를 혼합하여 사용할 경우에는 연산 우선순위를 적용하여 연산자 간의 연산 순위를 결정한다. [그림 6-2]는 KRISTAL-III 검색 엔진에서 제공하는 연산자들의 우선 순위를 제시하고 있다. 위쪽에 위치할수록 연산자의 우선순위는 높은 것으로 해석하면 된다.

우선순위		연산자
높다	문자열 지정자	“”
	괄호	( )
	섹션 지정자	:
	관계연산자	= > < >= <=
	근접도 연산자	/W /N
낮다	불리언 연산자	AND(&) OR(|) NOT(!)

[그림 6-2] 연산자 우선 순위

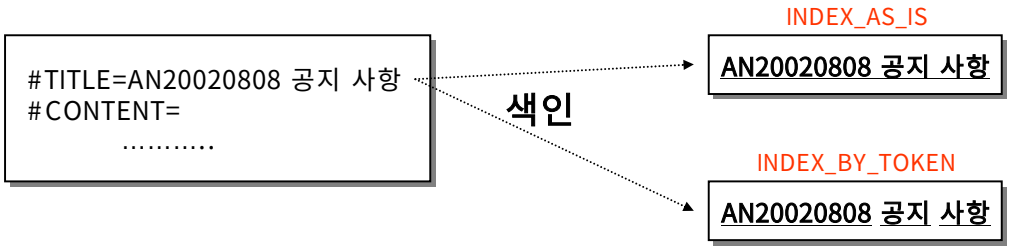
[그림 6-2]에 보여진 것처럼 연산자의 우선 순위는 높은 곳에서 낮은 쪽으로 [“”] > [( )] > [:] > [=] = [>] = [<] = [>=] = [<=] > [/W] = [/N] > [&] = [|] = [!]의 순서를 가진다[18].

[18] 앞서 설명한 바와 같이 다중 섹션 지정자인 쉼표(,)는 이전 버전과의 호환을 위하

6.1.5 문자열 지정자의 사용

문자열 지정자(“”)는 검색 단어의 구분 시 공백이나 특수 문자와 관계 없이 전체를 하나의 검색 단어로 구분하고자 하는 경우에 사용하는 기호로써, INDEX_AS_IS 색인 타입에서 유용하게 사용될 수 있다.

아래 그림은 문서의 TITLE 섹션을 두 가지의 색인 방식에 따라 색인 한 것으로, INDEX_AS_IS 방식에서는 공백이나 특수 문자를 고려하지 않기 때문에 총 1 개의 색인어가 생성되고, INDEX_BY_TOKEN 방식에서는 구분자를 이용하기 때문에 총 3 개의 색인어가 생성됨을 볼 수 있다.



이 경우 만약 사용자가 TITLE 섹션을 INDEX_AS_IS 와 같은 방식으로 색인 하였다면, 검색 시 (1)과 같이 문자열 지정자를 사용하여 질의를 구성하여야만 올바른 결과를 얻을 수 있다. 만약 (2)와 같이 문자열 지정자를 사용하지 않은 채 질의를 구성한다면, 질의 처리 과정에서 3 개의 단어로 분리 되기 때문에 올바른 검색 결과를 구할 수 없다.

여 제공하는 연산자이다. 본 절에서는 다중섹션 지정자를 생략하였으나 실제로 사용할 경우 우선 순위는 섹션지정자인 “:”와 동일하다. 그러나 다중 섹션 연산자의 사용은 권장하지 않는다.

(1) TITLE : “AN20020808 공지 사항”
(2) TITLE : AN20020808 공지 사항

이외에도, 문자열 지정자는 INDEX_AS_IS 이외의 색인 타입에서도 사용할 수 있다. 이 경우에는 질의 확장 여부에 따라 INDEX_AS_IS 와 동일하게 사용될 것인지의 여부가 결정된다. 만약 질의 확장을 실시하지 않는다면 INDEX_AS_IS 와 동일한 형태로 처리 될 것이고, 질의 확장을 실시하는 경우에는 확장된 형태의 질의 결과를 생성하게 된다. (자세한 내용은 “질의 확장”을 참고한다.)

문자열 지정자는 또한 다수 문자열로 구성된 질의에 사용할 수 있다. 특히 토큰별로 색인을 추출하는 INDEX_BY_MA, INDEX_BY_TOKEN 등의 색인방식에 use-index-bigram 옵션을 true 로 사용하는 경우에는 문자열 지정자를 사용하는 것이 보다 정확한 벡터 검색을 수행한다.

(3) ABSTRACT : “Information Retrieval”
(4) ABSTRACT : Information Retrieval

위에서 ABSTRACT 섹션은 INDEX_BY_MA 로 색인하고 use-index-bigram=”YES”로 하여 단어단위 색인어 bigram 을 추출하였다고 가정한다. 이러한 상황에서 벡터공간 검색 모델을 사용하여 질의 (3)과 질의 (4)를 수행하면 검색된 문서의 우선순위가 달라진다. 즉 질의 (3)에서는 “information retrieval”의 두 단어가 연이어 포함되는 문서가 가장 높은 순위로 제공되지만 질의 (4)에서는 “information”과 “retrieval”이 순서와 위치에 상관없이 모두 포함된 문서가 상위에 나오게 된다. 일반적으로 질의 (3)과 (4)에서 “information retrieval”이 연이어 나오는 문서가 보다 적합한 문서로 판정되는데 질의 (4)에서는 이러한 문서가 반드시 가장 상위에 제공된다는 보장을 할 수 없으나 질의 (3)에

서는 보장된다.

또한 IDNDEX_PROTEIN 과 INDEX_DNA 의 두 색인방식을 이용하는 경우에는 반드시 문자열 지정자를 이용하여 질의식을 구성하여야 한다. 생물 서열을 위한 이 2 개의 색인방식은 공백 문자를 무시하거나 내부적으로 제거하는 특성을 가지는 색인방식이다. 예를 들어 “ACD EFG”와 같은 단백질 서열이 있다고 하면 이 문자열은 공백을 무시하고 “ACDEF”, “CDEFG” 등으로 색인이 추출되어야 한다. 질의 확장에서 설명하겠지만 KRISTAL 의 질의처리기는 토큰 단위로 질의 분석을 미리 수행하게 되며 문자열 지정자 없이 공백이 들어간 문자열을 질의식으로 구성하게 되면 잘못된 검색을 수행할 수 있다.

(5) PROTEIN : ACDEF GHIJK
(6) PROTEIN : “ACDEF GHIJK”

KRISTAL 의 질의처리기는 토큰단위로 질의를 미리 분석한다. 따라서 검색식 (5)의 경우 질의 확장이 완료되면 KRISTAL 확장 질의식은 (PROTEIN: ACDEF) | (PROETIN: GHIJK)이 된다(벡터 검색에서는 공백문자가 OR 연산자와 유사하다). 그러나 일반적으로 생물서열에서 공백은 의미가 없는 문자열이어야 하므로 검색식 (5)는 형식에 있어서는 올바른 검색식이지만 의미에 있어서는 잘못된 검색식이라고 할 수 있다. 검색식 (5)는 검색식 (6)으로 문자열 지정자를 사용하여야만 올바른 질의가 된다.

검색식 (6)의 KRISTAL 확장질의는 (PROTEIN: ACDEF) | (PROTEIN: CDEFG) | (PROTEIN: DEFGH) | (PROTEIN: EFGHI) | (PROTEIN: FGHIJ) | (PROTEIN: GHIJK)이다 문자열 지정자에 따라 검색식 (5)와 검색식 (6)의 의미가 완전히 다른 것을 알 수 있다. 따라서 생물서열 다루는 시스템에서는 문자열 지정자를 기본적으로 사용하여야 한다.

6.1.6 질의 확장

질의확장은 사용자가 제시한 질의 단어를 각 색인 타입에 따라 후처리를 함으로써 검색의 재현율(recall)과 정확도(precision)를 향상시키기 위한 방법이다. 그러나 형태소 분석의 모호성으로 인해 발생할 수 있는 검색 결과의 부정확성과 검색에 보다 많은 시간을 소요하기 때문에 그 사용에 있어서 주의를 기울여야한다. 그러나 정확하게 사용한다면 아주 유용한 기능이다.

- 통합(유니온) 섹션 확장
- 공백 문자열 확장
- 색인 타입에 따른 질의 확장

질의 확장은 몇가지 형태로 구분해 볼 수 있는데, 위 3 가지의 확장 방법은 각 질의에 순차적으로 적용된다. 유니온 섹션 확장과 공백 문자열 확장은 기본적으로 시스템 내부에서 처리되는 반면, 색인 타입에 따른 질의 확장은 사용자에 의해 옵션으로 처리 여부가 결정된다.

위의 기본 질의확장과 별도로 KRISTAL 은 어절기반 질의어 확장 기능을 제공한다. 이는 일종의 사전기반 질의확장 개념이며 본절의 마지막에서 설명한다.

6.1.6.1 질의 확장 방식

KRISTAL 의 각 질의 확장 방식에 대해 구체적으로 살펴보자.

- 확장 1: 통합(유니온) 섹션 확장

통합섹션(Union Section)은 사용자가 검색시 여러 섹션을 명시해야 하는 불편을 제거하기 위해 검색 대상을 통합적으로 명시하는 방법을 제공하는 것으로서, 질의 처리시에는 기본 질의 형태로 분리되어 처리해야 올바른 검색 결과를 구할수 있다.

예를 들어, BASIC 이라는 통합 섹션에 TITLE, ETITLE, ABS 라는 세 섹션이 지정되어 있고, 사용자가 질의(1)을 제시하였다면 질의(1)은 시스템 내부적으로 질의(2)의 형태로

변환되어 처리된다. 이 경우 KRISTAL 확장질의는 섹션간의 OR 결과를 반환하게 되어 있으며, 섹션간의 AND 를 원하는 경우에는 사용할 수 없고 섹션간 AND 값을 얻고자 할 경우에는 기본섹션간의 AND 질의로 구성하여야 한다.

(1) BASIC:국민연금
(2) TITLE:국민연금 | ETITLE:국민연금 | ABS:국민연금

질의 (1)의 단순한 검색식은 질의 (2)의 복잡한 검색식과 동일하지만 검색식을 단순화하여 검색인터페이스를 매우 단순화할 수 있다는 장점이 있다[19]. 통합섹션은 표면적으로 단 하나의 섹션처럼 보이지만, 실제로는 여러 개의 섹션을 포함함으로써 검색 시간 복잡도를 야기시킨다는 점을 기억하여 올바로 사용하여야 한다. 예를 들어 기본 검색을 위한 웹 인터페이스에서 수십개에 달하는 모든 섹션을 통합섹션으로 묶어서 하나의 질의로 처리하는 경우가 있다. 이러한 기본 검색 인터페이스에서는 통합섹션의 사용에 주의를 기울여야 한다. 일반적으로 사용자들이 가장 선호하는 섹션 또는 반드시 기본적으로 검색 대상이 되어야 하는 섹션을 세밀하게 분석하여 기본 검색섹션의 대상을 최소화하는 것이 좋다. **과도하게 기본섹션의 수를 늘여서 통합섹션에 포함시키면 검색 속도가 크게 저하될 수 있다.** 기본적으로 검색해야 할 섹션의 수가 수십 개 이상일 경우에는 색인방식이 같은 섹션들을 VIRTUAL 섹션으로 묶은 후에 이를 UNION 섹션으로 묶을 것을 권장한다(다만 이 경우에, INDEX_AS_NUMERIC, INDEX_AS_IS 등의 섹션 전체를 하나의 색인으로 구성한 색인방식을 적용하는 섹션은 VIRTUAL 섹션으로

[19] 통합섹션(Union Section)은 도서관에서 사용된 초기 정보검색시스템들이 텍스트모드 기반으로 COMMAND LINE 에서 불리안 질의를 직접 입력받던 시절의 전통을 이어받은 것이다. 오늘날 주로 사용되는 정보서비스는 웹을 기반으로 하기 때문에 사용자가 불리안 질의를 직접 구성하여 입력하는 경우는 드물다. 따라서 통합섹션없이 기본섹션만을 이용하여도 최종 사용자에게는 큰 불편이 없을 수 있다. 다만 인터페이스 개발자에게는 통합섹션의 사용이 개발 복잡도를 어느 정도 감소시켜 줄 것이다.

묶을 수 없다).

UNION 섹션을 사용할 경우에는 연산자 우선순위에 대해서 주의하여야 한다[20]. 다음과 같은 예를 들어보자.

(1) (BI : 하늘) & (BI : 다람쥐)
(2) (BI : 하늘 & BI : 다람쥐)

위 예제에서 검색식 (1)과 검색식 (2)는 동치일까? UNION 섹션 BI 는 제목 섹션 BI 와 요약 섹션 AB 로 구성되어 있고 다음 문서들에 대한 검색을 한다고 하자.

구분	제목(TI)	요약(AB)
{문서 1}	하늘	다람쥐
{문서 2}	하늘 다람쥐	

검색식 (1)은 KRISTAL 질의분석기에 의해 (TI : 하늘 | AB : 하늘) & (TI : 다람쥐 | AB : 다람쥐)로 확장된다. 질의 (TI : 하늘 | AB : 하늘)은 문서 1 과 부합하고, (TI : 다람쥐 | AB : 다람쥐) 역시 문서 1 과 부합한다. 따라서 문서 1 은 검색식 (1)의 답이 된다. 또한 문서 2 의 경우에도 당연히 검색식 (1)과 부합하므로 검색식 (1)의 적합문서로는 문서 1 과 문서 2 가 반환된다.

검색식 (2)는 검색식 (1)과는 다르다. 전체 검색식에 대한 괄호연산자만 지정되어 있으므로 검색식 (2)는 **BI : (하늘 & 다람쥐)**로 해석된다. 따라서 검색식 (2)는 검색식 (1)과는 달리 **(TI : 하늘 & 다람쥐) | (AB : 하늘 & 다람쥐)**로 확장된다. 이 검색식에 의하면 문서 1 은 적합하지 않으며 문서 2 만이 적합하다. 따라서 검색식 (2)는 문서 2 만을 적합한 문서로 반환한다.

---

[20] 이러한 검색식의 모호성은 불리안 검색에서 UNION 섹션에 대한 AND, WITHIN, NEAR, NOT 등의 연산을 할 경우에 발생할 수 있다. 다만 OR 연산에서는 모호성이 발생하지 않는다.

위의 검색식(1)과 (2)는 서로 다른 결과를 반환한다는 것을 보여주었다. 이는 ( ) 연산자의 우선순위가 가장 상위에 위치하기 때문에 일어나는 현상이다. 따라서 ***UNION 섹션을 이용하는 경우에는 괄호 연산자의 사용에 주의하여야 한다.*** UNION 섹션에 대해서는 미리 괄호연산을 사용하지 않을 것을 권장한다(즉, 위의 예에서와 같이 검색식 (2)와 같은 방식으로 질의를 구성해야 한다).

- 확장 2: 공백 문자열 확장 (SPACE_OPERATOR)

공백 문자열 확장은 사용자가 공백으로 구분 된 여러 검색 단어를 입력할 경우에 해당 공백에 연산자를 적용하여 처리하는 방법이다. 기본적으로 사용되는 연산자는 '&' 이나, 사용자가 그 의미를 변경하여 사용하는 것을 허용한다.(의미의 변경은 검색 API의 입력변수의 하나인 space_operator 를 통해 변경할 수 있다. 검색을 수행하기 직전에 입력변수 p_in.space_operator = SPACE_AND | SPACE_OR | SPACE_NOT | SPACE_NEAR | SPACE_WITHIN 중의 하나로 지정하면 된다.)

(1) TI : 정보 검색 연구
(2) → TI : (정보 & 검색 & 연구)
(3) → TI : (정보 | 검색 | 연구)

질의 (1)은 검색 시스템 내부적으로 질의 (2)형태로 처리된다(p_in.space_operator 의 기본값은 SPACE_AND 이다. 그러나 보다 명확하게 할 수 있도록 p_in.space_operator 를 지정해줄 것을 권장한다). 만약 사용자의 공백 백자열 확장시 연산자를 SPACE_OR 로 지정하였다면(p_in.space_operator=SPACE_OR), 질의 (3)의 형태로 처리된다.

공백 문자열 확장은 벡터 검색 방법(모델)에서는 적용이 되지 않는다. 벡터 모델에서는 공백으로 구분된 문자열을 고유의 벡터 연산 방식으로 처리하기 때문이다. 질의식 (2)와 같이 사용자가 불리안 연산자를 명시하더라도 벡터 검색모델에서는 이러한 연산

자가 무시됨을 기억하기 바란다.

- 확장 3: 색인 타입에 따른 질의 확장

색인 타입에 따른 질의 확장은 사용자가 제시한 질의를 원 섹션의 색인방식과 동일하게 처리함으로써 정확도를 향상시키기 위한 방법이다. 예를 들어, TI 라는 섹션이 INDEX_BY_TOKEN 으로 색인 되었을 경우 원 섹션의 내용 “국민_연금” 은 “국민” “연금”의 색인어로 분리되어 저장된다. 이때 사용자가 “국민-연금”이라는 질의를 입력하였고 색인 타입에 따른 질의 처리를 해주지 않는다면, “국민-연금”이라는 색인어를 검색함으로써 올바른 검색 결과를 구할 수 없다.

색인 타입에 따른 질의 확장은 형태소 분석을 이용한 색인방법에서 가장 유용하게 사용된다. 이러한 질의 확장을 사용하면 불필요한 조사등을 분리하여 올바른 명사 색인어를 구할 수 있고 복합명사를 구성 명사 단위로 분리함으로써, 결과적으로 보다 정확한 검색을 수행할 수 있다.

예를 들어, 사용자가 다음 질의(1)을 제시하였고, TI 섹션은 INDEX_BY_MA 로 색인되었다고 가정하자. 검색 시스템은 ‘정보검색’을 색인기의 입력으로 주어 구성명사 ‘정보’, ‘검색’의 두 가지 분해 결과를 얻는다. 이러한 방식의 질의확장은 검색 모델에 따라 확장 결과의 차이를 보인다. 만약 불리언 모델을 사용한다면 동일 레벨의 구성 명사 사이에는 근접 연산자를 적용함으로써 (2)과 같은 질의를 생성해 낸다. 반면에 벡터 모델을 사용한다면 (3)과 같은 질의를 생성해 낸다.

(1) TI: 정보검색
(2) → TI: (정보 /W 1 검색)
(3) → TI: (정보 검색 정보검색)

질의 (2)의 검색 결과는 TI 섹션에서 “정보”와 “검색”이 순서대로 인접한 문서를 검색해낸다. 이것은 원문에 “정보검색”, “정보 검색”, “정보-검색”과 같이 다른 형태지만 동일한 의미를 지닌 내용을 검색할 수 있도록 한다.

한글에서와 마찬가지로 영문에서도 공백 이외의 단어 구분자로 연결된 단어를 정확하게 검색해 낼 수 있다. (질의 4 참조)

(4) TI: red-eared

단어별 질의 확장은 KRISTAL 검색 API 에서 사용하는 입력변수인 term_expansion 을 이용하여 지정할 수 있다. 단어별로 해당 섹션의 색인 타입에 따라 질의를 확장하고 잘 할 경우에는 p_in.term_expansion = FULL_QUERY_TERM_EXPANSION 으로 지정할 수 있으며, 단어별로 질의를 확장하지 않고자 할 경우에는 p_in.term_expansion = NO_QUERY_TERM_EXPANSION 로 선언하면 된다. 쌍따옴표내의 문자열만을 확장하지 않을 경우에는 PARTIAL_QUERY_TERM_EXPANSION 을 사용할 수 있다. 다음 표는 각 질의에 대한 질의확장 옵션 처리의 결과를 보여주고 있다. (기본 공백 옵션은 AND 로 가정하였다).

질의 \ 확장방식	No Expansion	Partial Expansion	Full Expansion
정보검색(벡터)	정보검색	정보 검색 정보검색	정보 검색 정보검색
“정보검색”(벡터)	정보검색	정보검색	정보 검색 정보검색
하늘다람쥐(불리안)	하늘다람쥐	(하늘 /W1 다람쥐) | 하늘다람쥐	(하늘 /W1 다람쥐) | 하늘다람쥐
“하늘다람쥐”(불리안)	하늘다람쥐	하늘다람쥐	(하늘 /W1 다람쥐) | 하늘다람쥐
Information Retrieval (벡터)	Information Retrieval	Information Retrieval	Information Retrieval
“Information Retrieval” (불리안)	Information /W1 Retrieval	Information /W1 Retrieval	Information /W1 Retrieval
Information Retrieval (불리안)	Information & Retrieval	Information & Retrieval	Information & Retrieval

한글 문서를 대상으로 하는 정보서비스에서는 벡터검색모델에서는 질의에 대해서 FULL_QUERY_TERM_EXPANSION 을 사용할 것을 권장한다. 불리안 모델에서도 특수한 경우를 제외하고는 질의를 확장할 것을 권장한다. 다음 예들에서 색인타입에 따른 질의 확장의 장점과 단점을 살펴보자.

(5) TITLE: 정보검색
(6) TITLE: 정보 /w1 검색

질의 (5)에서 색인방식에 따른 질의 확장을 할 경우에는 질의 (6)과 동일하게 확장된다. 그러나 색인방식에 따라 질의 확장을 하지 않을 경우에는 질의 (5)는 다른 의미를 가지게 되며 “정보검색”이라는 색인어가 추출되는 문서만을 검색하게 되므로 “정보 검색”, “정보-검색” 등으로 출현하는 문서는 검색되지 않는다.

다음으로 TITLE 섹션(INDEX_BY_MA 로 색인)과 AUTHOR 섹션(INDEX_BY_TOKEN 방식으로 색인)의 통합섹션인 BASIC 섹션에 대한 질의를 살펴보자.

(7) BASIC: 박동은
(8) (TITLE: 박동 | 박동은) | (AUTHOR: 박동은)
(9) (TITLE: 박동은) | (AUTHOR: 박동은)

질의 (7)은 색인방식에 따른 질의 확장을 지정할 경우 질의 (8)로 확장된다. 그러나 색인방식에 따른 질의 확장을 지정하지 않으면 질의 (9)로 확장된다. 이 경우 질의의 의미에 따라 검색의 결과의 만족도는 크게 달라진다. 만약 질의 (7)의 의미가 사람이름

인 “박동은”을 검색하는 것이라면 질의 (9)는 완벽한 결과를 제공할 것이다. 반면 질의 (8)은 “심장박동” 등의 문자열을 포함하는 문서를 여분으로 제공한다. 그러나 사용자의 질의가 “심장박동”에서의 “박동”과 동일한 의미의 단어를 조사인 “은”의 절단 없이 제공한 것이라면 질의 (9)는 “박동은”이라는 문자열이 포함된 문서만을 제공할 것이다. 이 경우에는 질의 (8)이 보다 적합한 질의식이라고 할 수 있다. 따라서 질의에 따라 검색결과의 만족도가 달라질 수 있음을 주지하기 바란다.

위의 질의 (8)의 경우에는 두 경우 모두에 대해서 비적합 문서가 포함되기는 하지만 원하는 문서를 모두 제공한다는 점에 주목하자. 질의 (8)은 해당 섹션의 색인방식에 따라 질의를 확장한 것이다. 이 경우에 있어서 질의 (9)보다는 질의 (8)이 보다 합리적임을 알 수 있으며 특수한 경우를 제외하고는 일반적으로 질의를 색인방식에 따라 확장을 하는 것이 올바른 검색 결과를 제공하므로 KRISTAL 질의를 구성할 경우에는 p_in.term_expansion = FULL_QUERY_TERM_EXPANSION 을 사용할 것을 권장한다.

6.1.6.2 질의 확장 예제

질의(1)은 불리안 검색에서 공백확장과 색인방식에 따라 질의 (2)로 확장(이 경우에는 복합명사 확장)이 동시에 발생하는 경우의 예제이다. 질의 (3)은 질의 (1)에 대한 벡터 공간 모델의 확장 예이다.

(1) TI : 정보검색 연구
(2) → TI : ((정보 /W 1 검색) & 연구)
(2) → TI : (정보 검색 정보검색 연구)

다음 표에서는 색인 방식에 따른 질의의 확장 결과를 보여주고 있다. 통합섹션에 의한 질의확장에 대해서는 위에서 별도로 설명하였으므로 여기서는 생략한다.

질의	공백	Term	색인	벡터모델 확장질의	불리안 확장 질의

	처리	Expansion	방식		
“李舜臣 장군”	(의미없음)	(의미없음)	AS_IS	“李舜臣 장군”	“李舜臣 장군”
李舜臣 장군	&	(의미없음)	CHAR	李 舜 臣 장 군	李 & 舜 & 臣 & 장 & 군
李舜臣 장군은	/W1	FULL	TOKEN	李舜臣 장군은	李舜臣 /W1 장군은
李舜臣 장군은	/W1	FULL	MA	李舜臣 장군	李舜臣 /W1 장군
李舜臣 장군	&	(의미없음)	MIX_CHAR	李 舜 臣 장 군	(李 /W1 舜 /W1 臣) & (장 /W1 군)
李舜臣 장군은	/W1	FULL	MIX_MA	李 舜 臣 장군	(李 /W1 舜 /W1 臣) /W4 ((장 /W1 군) | 장군)
李舜臣 장군은	&	NO	MIX_MA	李 舜 臣 장군은	(李 /W1 舜 /W1 臣) & ((장 /W1 군 /W1 은) | 장군은)

6.1.6.3 어절기반 동의어 확장

KRISTAL 은 어절 기반의 동의어 확장 기능을 제공한다. 즉 사전에 “싱가폴”과 동의어로 “싱가포르”를 지정하였고, 사용자의 질의가 “싱가폴”로 입력이 되면 사용자의 질의를 “싱가폴 | 싱가포르”로 확장할 수 있다. 다만 이러한 확장은 어절단위로 이루어지기 때문에 두어절 이상의 질의에 대해서는 확장이 이루어지 않는다. 예를 들어 “information retrieval”을 “정보검색”으로 사전에 정의할 경우에는 사용자의 질의가 “information retrieval”로 입력되더라도 “정보검색”으로 확장되지 않는다.

KRISTAL 의 어절기반 동의어 확장은 클라이언트 입력변수인 p_in.thesaurus_levels 라는 변수를 통해 이루어진다. 그리고 이를 위한 동의어 사전은 서버 환경 설정파일에서 지정된다. 사용자의 사전과 서버환경설정파일에서의 사전 파일 설정 및 동의어 확장 방법에 대해서 차례대로 알아보자.

동의어 사전은 XML 로 작성되어야 하며 문자 인코딩은 UTF-8 이어야 한다. 다음은 동의어 사전의 예이다.

```
<?xml version="1.0" encoding="UTF-8" ?>

<Dictionary>
    <Version> 1.0 </Version>
    <WordList>
        <Word name="정보검색">
            <Level no="1">
                <Synonym value="information retrieval"/>
            </Level>
        </Word>
        <Word name="李舜臣">
            <Level no="2">
                <Synonym value="이순신"/>
            </Level>
            <Level no="3">
                <Synonym value="충무공"/>
            </Level>
            <Level no="4">
                <Synonym value="임진왜란"/>
            </Level>
        </Word>
        <Word name="3.1 운동">
            <Level no="3">
                <Synonym value="삼일운동"/>
            </Level>
            <Level no="3">
                <Synonym value="삼일만세운동"/>
            </Level>
            <Level no="4">
                <Synonym value="삼일절"/>
            </Level>
        </Word>
    </WordList>
</Dictionary>
```

[동의어 사전 XML 파일 예제]

동의어 사전은 XML 로 작성되어야 하며, <Dictionary> 엘리먼트는 하위에 <Verion>과

<WordList> 엘리먼트를 가진다. <WordList> 엘리먼트의 하위에 단어와 동의어 및 레벨이 지정된다. <Word> 엘리먼트는 name 속성에 사전의 용어를 지정한다. 현재 이 용어는 반드시 단일어절이어야 한다. 즉 용어는 공백문자를 가지지 않아야 한다. 다시 <Word>의 하위 엘리먼트인 <Level> 엘리먼트는 주어진 단어에 대한 동의어의 레벨을 선언한다. 레벨에 대해서는 뒤에서 다시 설명한다. <Level>의 하위 엘리먼트인 <Synonym>에 해당 레벨의 동의어가 value 속성에 정의된다.

레벨 엘리먼트 <Level>은 각 <Word>에 대해서 1 개 이상이 작성되어야 하며, 레벨의 속성인 no 는 양수이어야 하며 인접한 <Level>의 no 속성과 독립적이다. 즉 동일한 레벨의 동의어를 계속 사용할 수 있다. 예를 들어 위의 예제에서 “3.1 운동”은 레벨 3 을 가지는 2 개의 동의어 “삼일운동”과 “삼일만세운동”을 지정하고 있음을 알 수 있다.

레벨 엘리먼트는 동의어의 레벨을 지정하기 위한 것으로 레벨의 지정은 서비스 개발자나 서비스 관리자의 해석이 동반되어야 한다. 예를 들어 위의 예제에서는 임의로 <Level no=”1”>인 경우는 한영 대역어, <Level no=”2”>인 경우는 한자의 음독, <Level no=”3”>인 경우는 동의어, <Level no=”4”>인 경우는 관련 용어(시소러스)로 정의하였다. 이러한 정의는 사전 작성자가 미리 인지하여야만 레벨에 따라서 향후 질의 확장에서 동의어 확장, 음독 확장, 대역어 확장 등의 별도 옵션으로 사용할 수 있다.

다음으로 서버 구동시의 사전 위치를 지정하는 방법에 대해 설명하고자 한다. 사전의 위치는 서버 환경 설정 파일의 <Retrieve> 엘리먼트의 속성에 정의된다. 이 속성의 이름은 query_thesaurus_files 이며 아래 예제와 같이 정의할 수 있다.

```
<Retrieve query_thesaurus_files="/home/trend/dict/synonym_dict.xml" />
<Retrieve
        query_thesaurus_files="/home/trend/dict/syndict1.xml
        /home/trend/dict/syndict2.xml
        /home/trend/dict/syndict3.xml" />
<Retrieve query_thesaurus_files="/home/trend/dict/syndict1.xml,
    /home/trend/dict/syndict2.xml, /home/trend/dict/syndict3.xml,
    /home/kristal/dict/hanja_dict.xml" />
```

시소러스 동의어 사전은 1 개 이상을 사용할 수 있다. 위의 첫번째 예제에서는 하나의 XML 파일을 동의어 사전으로 지정한 것이며, 두번째 예제는 공백을 이용하여 3 개의 사전 파일을 지정한 것이다. 세번째 예제에서는 쉼표를 이용하여 사전 파일 4 개를 사용할 것을 지정하고 있다. 속성에서 사전 파일의 지정을 위한 구분자는 1 개 이상의 공백문자(스페이스), 쉼표(,), 줄바꿈(,), 탭(TAP) 문자 등을 사용할 수 있다. 2 개 이상의 연속되는 구분자는 하나의 구분자로 취급한다.

어절 기반 질의어확장 기능을 사용할 때는 KRISTAL 데몬을 구동할 때마다 한번만 동의어 사전을 읽어들이므로, 사전이 변경되어서 변경된 내용을 질의 확장에 재적용하기 위해서는 KRISTAL 데몬을 다시 구동하여야 한다. KRISTAL 데몬은 query_thesaurus_files 를 한번만 읽어서 메모리에 유지하는 방식을 취하고 있다.

다음은 KRISTAL 클라이언트 프로그램 작성시의 어절단위 동의어 확장에 대해서 설명하고자 한다. 먼저 동의어 사전을 지정하여 KRISTAL 데몬이 구동되어 있는 상태라고 가정하자. 클라이언트 프로그래밍에서 어절단위 동의어 확장을 지정하는 방법은 간단하다. 입력 변수인 vector<int>형인 p_in.thesaurus_levels 를 채워서 질의를 서버로 보내면 동의어의 확장은 KRISTAL 서버가 자동으로 수행한다. 다음 표는 XML 로 작성된 동의어사전을 보기 쉽게 재구성한 것이다. 앞으로의 설명은 이 사전을 기반으로 한다.

용어 \ 레벨	1	2	3	4
李舜臣	General Lee	이순신	충무공	임진왜란
3.1 운동			삼일운동	유관순 삼일절
靑		청	맑음	
정보검색	Information retrieval			KRISTAL

[동의어 사전 구조 예제]

앞서 설명한 바와 같이 레벨은 양수로 정의되어야 하며, 레벨의 의미 해석은 XML

작성자에 의해 부여된다. 위의 예제에서는 레벨 1 은 한영 대역어, 2 는 음독, 3 은 동의어, 4 는 관련 용어로 설정하고 사전을 구성하였다고 가정한다.

INDEX_BY_MA, INDEX_BY_TOKEN 등의 색인방식을 사용하는 섹션에 대해서 사용자의 질의를 동의어와 음독으로 확장하고자 할 경우에는 p_in.thesaurus_levels 에 2 와 3 을 입력하고(KRISTAL 의 C++ 클라이언트 API 에서는 p_in.thesaurus_levels.push_back(2); p_in.thesaurus_levels.push_back(3);), RETRIEVE 등을 수행하면 된다. 예를 들어 TIT 라는 섹션이 INDEX_BY_MA 로 색인되었고 사용자의 질의가 아래의 (1)과 같이 "TIT: 李舜臣"이라면 2, 3 레벨로 확장할 경우 사용자의 질의는 (2)와 같이 "TIT: 李舜臣 | 이순신 | 충무공"으로 확장된다.

(1) TIT: 李舜臣
(2) TIT: 李舜臣 | 이순신 | 충무공
(3) WEATHER: 靑
(4) WEATHER: 靑 | 맑음
(5) ABS: 정보검색
(6) ABS: 정보 /W1 검색 | Information retrieval
(7) TIT: 3.1 운동
(8) TIT: 3 /W1 1 /W1 운동
(9) TIT: 3.1 운동 | 삼일운동

질의 (3)에서 WEATHER 섹션은 INDEX_BY_TOKEN 으로 색인되었으며 내용이 한자로 되어 있을 경우 이에 대한 한글 동의어를 확장하려고 한다. 이 경우에는 레벨을 3 으로 주어 확장하면 질의 (4)로 시스템이 질의를 확장한다. 질의 (5)의 경우 한영 대역어로 확장하고자 할 경우에는 레벨 1 을 주어 질의 (6)과 같이 확장할 수 있다.

이러한 동의어 확장은 어절을 기반으로 하며(즉 동의어는 두 어절 이상어도 상관없으나 사전의 용어는 단일 어절이어야 하며 공백을 포함할 수 없다), p_in.term_expansion

의 영향을 받는다. 질의 (7), (8), (9)에서 그 예를 볼 수 있다. TIT 가 INDEX_BY_MA 로 색인된 섹션이고 p_in.term_expansion=FULL_QUERY_TERM_EXPANSION 으로 설정되었으며 p_in.thesaurus_levels 에 3 을 입력하였을 경우에, 질의 (7)은 내부적으로 질의 (8)로 확장된다. 즉 동의어 확장이 되지 않는다. 이는 INDEX_BY_MA 색인 방식이 '.'을 공백과 같이 취급하기 때문에 일어나는 현상이다. 이 경우 올바르게 질의를 확장하기 위해서 p_in.term_expansion=NO_QUERY_TERM_EXPANSION 이라고 지정하였다면 질의 (7)은 질의 (9)로 확장할 수 있다. 그러나 질의 (9)의 경우에는 "3.1 운동"으로 색인된 문서가 없을 것이므로 "삼일운동"이 들어간 문서만을 검색하는 잘못된 결과를 반환하게 된다. 이러한 문제를 해결하기 위하여 KRISTAL 의 동의어 확장은 전확장(pre-expansion)과 후확장(post-expansion)으로 확장될 순서를 지정할 수 있도록 하였다. 그리고 현재까지는 후확장에 대해서 설명하였다.

동의어 전확장과 후확장의 기준은 KRISTAL 의 질의분석기를 거치기 전후의 차이를 의미한다. 즉, 질의분석기를 거치기 전에 동의어를 확장하는 것을 전확장, 질의분석기를 거친 후에 확장하는 것을 후확장이라고 한다. INDEX_BY_MA, INDEX_BY_TOKEN 등의 일반적인 색인 방식에서는 후확장을 사용하는 것이 일반적일 수 있다. 그러나 INDEX_BY_CHAR, INDEX_BY_MIX_CHAR, INDEX_BY_MIX_MA 와 같이 낱자 색인방식이 포함되어 있는 섹션에서는 후확장이 불가능한 경우가 발생한다. 예를 들어 "TIT: 李舜臣"이라는 질의에서 TIT 섹션의 색인 방식이 INDEX_BY_MIX_CHAR 일 경우 p_in.thesaurus_levels 에 어떠한 값을 주어도 질의는 "TIT: 李 /W1 舜 /W1 臣"으로 확장된다. 이것은 후확장 방식에서는 "李", "舜", "臣"에 대해서 동의어 사전을 검색하기 때문에 "李舜臣"의 동의어를 찾을 수 없게 되기 때문이다.

이러한 문제를 해결하기 위해 제공하는 동의어 전확장 기능을 사용할 수 있으며, 전확장 방법으로 p_in.thesaurus_levels 에 음수값을 주면 된다[21]. 이 음수값은 사전에 정의된 양수 레벨을 참조하여 KRISTAL 질의분석기에 들어가기 전에 질의를 확장하도록

[21] KRISTAL 동의어 확장에서 전확장과 후확장은 동시에 사용할 수 있다. 예를 들어 질의가 INDEX_BY_MA 방식 섹션에 대해서 "MA_SEC: 정보검색은"이라고 줄 경우에 thesaurus_levels={ -1, 1}로 주게 되면 "MA_SEC: 정보검색 | Information & retrieval"로 확장할 수 있다.

하는 것이다. 아래는 위에서 사용한 사전을 단순화하여 다시 실었다. 이 사전을 예제로 하여 질의의 전확장에 대해서 설명한다.

용어 \ 레벨	1	2	3
李舜臣	이순신	충무공	임진왜란
3.1 운동		삼일운동	유관순 삼일절
靑	청	맑음	
정보검색	IR		KRISTAL

[동의어 사전 구조 예제]

(1) “MIX_CHAR_SEC: 李舜臣” (thesaurus_levels={1})
(2) “MIX_CHAR_SEC: 李 /W1 舜 /W1 臣”
(3) “MIX_CHAR_SEC: 李舜臣” (thesaurus_levels={-1})
(4) “MIX_CHAR_SEC: 李舜臣 | 이순신”
(5) “MIX_CHAR_SEC: (李 /W1 舜 /W1 臣) | (이 /W1 순 /W1 신)”

INDEX_BY_MIX_CHAR 섹션에 대해서 질의 (1)을 위에서 주어진 thesaurus_levels 를 이용하여 확장하면 질의 (2)와 같이 확장된다. 즉 “이순신”이라는 음독이 확장되지 않은 것이다. 후확장으로 레벨 1 을 지정했기 때문에 주어진 용어에 대해서 동의어를 찾지 못한 것이다. 그러나 질의 (3)에서 처럼 thesaurus_levels 를 -1 로 주게 되면 후확장과 마찬가지로 레벨 1 의 사전을 참조하지만 질의분석기에 들어가기 전에 동의어를 확장하므로 레벨 1 의 “李舜臣”에 대한 동의어 “이순신”을 검색하여 질의 (4)와 같이 새로운 질의를 구성하게 되고 이 질의가 질의 분석기를 통하게 되면 질의 (5)와 같은 최종 시스템 질의로 확장된다.

6.2 KRISTAL 에서 지원하는 검색 방법

KRISTAL2002 검색 시스템에서는 불리언(Boolean), 벡터(Vector)과 벡터불리언(Vector Boolean) 모델 세 가지의 검색모델을 지원한다.

- KRISTAL 불리언 모델은 일반적인 불리언의 확장으로서, 불리언 연산자, 근접도 연산자, 관계연산자, 범위연산자, 그리고 절단 연산자[22]를 지원한다. 소규모 데이터베이스 및 낱자 단위로 색인(INDEX_BY_CHAR, INDEX_BY_MIX_CHAR, INDEX_BY_MIX_MA 등)한 섹션에 대한 검색에 사용할 것을 권장한다.
- KRISTAL 벡터 모델은 기본 벡터 모델[23]의 확장으로 일반 사용자들에게 빠른 검색 속도와 높은 정확도를 제공하기 위한 방법을 제공한다. 이 모델에서는 벡터 모델의 기능을 모두 수용하고 있다. INDEX_BY_MA, INDEX_BY_TOKEN 등의 토큰 단위 색인을 적용하는 대용량 데이터베이스에 사용할 것을 권장한다.
- KRISTAL 벡터 불리언 모델은 전문가들을 위한 모델로서, 불리언의 모든 연산자와 벡터의 랭킹 모델을 상속 받은 형태를 취한다. 이 모델에서는 랭킹 및 자연어 처리와 불리언 연산자를 모두 지원한다. 벡터 모델을 통한 문서 우선순위를 지원하고 불리안과 동일한 검색 결과를 제공한다는 장점이 있으나 불리안 모델과 벡터 모델의 이중 지원에 따른 검색속도의 저하가 현저하므로 중대형 데이터베이스에서의 사용은 권장하지 않는다.

6.2.1 불리언 검색

불리언(Boolean) 검색은 검색 시스템에서 가장 기본적으로 사용하는 검색 모델이다. 불리언 검색에서 지원하는 연산자는 [표 6-4]와 같은데, 연산자는 대문자로 표기되어야 함을 주의하기 바란다[24]. 불리언(Boolean) 검색은 연산자들과의 조합하여 정확한 색언

[22] *

[23] 사용자 질의와 문서간의 유사도 값의 계산에 따라 검색 결과를 내는 방법으로서, 사용자 질의 안의 검색 단어는 공백의 구분에 의한 단어 혹은 문장 형태로 주어진다.

[24] 단 현재 버전에서는 KRISTAL-II 와의 호환성을 위해 소문자의 사용도 허용한다.

어를 검색할 수 있는 장점이 있다.

[표 6-4] 불리언 검색 지원 연산자

불리안 연산자	관계 연산자	근접도 연산자	절단 연산자
&(AND) |(OR) !(NOT)	= < <= >= >	N(NEAR) W(WITHIN)	*

❒ 불리언 연산 검색

KRISTAL 에서 사용하는 불리언 연산자는 모두 이항 연산자(Binary Operator)이다. 다음은 불리언 연산 예제이다(질의 (4)는 잘못된 예제이다).

ABS: (정보 & 검색) ..... (1)
ABS: (정보 | 검색) ..... (2)
TI : (정보 ! 검색) ..... (3)
ABS : ! 검색 ..... (4)

질의 (1)의 검색 결과는 ABS 섹션에서 “정보”와 “검색”이 모두 포함하는 단어를 검색한다. 질의 (2)는 ABS 섹션에서 “정보” 또는 “검색” 중에 하나이상의 단어가 포함하는 단어를 검색한다. 질의 (3)은 TI 섹션에서 “정보”를 포함하는 문서 중에서 “검색” 단어가 포함하지 않는 문서를 검색한다. 질의(4)는 KRISTAL-III 에서 지원하지 않는 연산이다. KRISTAL 에서 ‘***!***’ 연산자는 “***AND NOT***”의 의미를 갖는다.

❒ 관계 연산 검색

DATE >= 20020101 …… (5)

질의 (5)에서 DATE 섹션은 INDEX_AS_NUMERIC 으로 색인되어 있다고 가정한다. 질의 (5)의 검색 결과는 DATE 섹션에서 날짜가 2002 년 1 월 1 일보다 크거나 같은 날짜를 포함하는 모든 문서를 검색한다. 관계 연산자 >, <, >=, <=는 각각 ‘크다’, ‘작다’, ‘크거나 같다’, ‘작거나 같다’ 등의 의미를 가진다.

❒ 근접도 연산 검색

ABS: (정보 /W2 검색) …… (6)
ABS: (정보 /N2 검색) …… (7)

질의 (6)의 검색 결과는 ABS 섹션에서 단어간의 거리가 2 이하이고 인접한 단어간의 순서대로 “정보”와 “검색”이 모두 포함하는 단어를 검색한다. 질의 (7)의 검색 결과는 ABS 섹션에서 단어간의 거리가 2 이하이고 인접한 단어간에 순서에 상관 없이 “정보”와 “검색”이 모두 포함하는 단어를 검색한다.

❒ 부정 연산 검색

ABS: (정보 ! 검색) …… (8)

질의 (8)의 검색 결과는 ABS 섹션에서 “정보”를 포함하지만 “검색”은 포함되지 않는 모든 문서를 검색한다.

❒ 절단 연산 검색

KRISTAL-III 에서 지원하는 절단 연산자는 '*' 이고, 우측 절단에 한해서만 활용될 수 있다. 질의 (10)과 (11)은 잘못된 질의 예제이다. 벡터 모델은 3 음절이상 단어에서만 지원을 하고 불리언 모델은 제한이 없다.

ABS: 정보* …… (9)

질의 (9)의 검색 결과는 ABS 섹션에서 "정보", "정보검색", "정보시스템", "정보처리" 등 "정보"를 포함하는 모든 문서를 검색한다.

6.2.2 벡터 검색

KRISTAL-III 시스템에서 벡터 검색은 일반적으로 알려진 벡터 모델과 유사하다. 일반 사용자에게 문서의 랭킹 기능 과 빠른 검색을 지원하기 위한 모델이다. KRISTAL 벡터 모델은 우절단 연산자만을 지원한다. KRISTAL 벡터 모델에서 지원하는 연산자는 [표 6-5]과 같다.

[표 6-5] 확장 벡터 검색 지원 연산자

불리안 연산자	관계 연산자	근접도 연산자	절단 연산자
지원하지 않음	지원하지 않음	지원하지 않음	*

벡터 모델에서는 불리언, 관계, 근접도 연산자를 지원하지 않고 검색 대상 섹션에 대해서 종합적인 가중치를 부여한다. 가장 높게 가중치를 받은 문서가 상위로 순위화가 되는 검색 모델이다.

❒ 기본적인 벡터 검색

TI: 정보 검색 시스템 ..... (1)
TI: 정보검색 ..... (2)

질의 (1)의 검색 결과는 TI 섹션에서 “정보 검색 시스템”에 대한 벡터 가중치를 적용하여 랭킹(ranking) 처리된 문서들을 검색한다. 불리안 연산자를 지원하지 않으므로 불리언 모델과는 달리 공백 문자에 대한 연산자 확장을 실시하지 않는다. 그러나 색인방식에 따른 단어별 질의확장은 동일하게 수행한다. 질의 (2)의 경우에는 섹션 색인방식에 따른 질의확장을 지정할 경우 (TI: 정보 검색 정보검색)으로 확장된다.

❒ 절단 연산 검색

ABS: 정보* ..... (3)
ABS: 정* ..... (4)
ABS: inf* ..... (5)

질의 (3)의 검색 결과는 ABS 섹션에서 “정보”, “정보검색”, “정보시스템”, “정보처리” 등 “정보”를 포함하는 모든 단어를 검색한다. 우절단 연산자는 시스템의 자원을 크게 소비하는 연산을 수행할 수 있다. 따라서 KRISTAL 의 벡터 검색모델에서는 영어의 경우 3 자 이하, 한글의 경우 1 자인 우절단 연산은 지원하지 않는다. 따라서 질의 (4)과 질의 (5)에 대해서는 질의 오류로 처리된다. 다만 불리안 검색모델에서는 이러한 제한이 없다.

❒ 복합 명사 확장 검색

TI: 정보검색 ..... (7)
→ TI: 정보 검색 정보검색.....(8)

질의 (7)에서 복합명사는 질의 (8)로 질의가 확장 된다. 질의 (8)의 검색 결과는 TI 섹션에서 “정보 검색”에 대한 벡터 가중치와 거리 가중치를 적용하여 랭킹 처리된 문서들 검색한다. 벡터 모델에서의 복합명사 확장은 불리언 검색에서의 확장과 차이가 있다.

6.2.3 벡터 불리언 검색

불리언과 벡터 검색의 장점을 모두 상속 받은 것으로서 구체적인 내용은 각각 불리언과 벡터 검색을 참조한다. KRISTAL 벡터 불리언 모델에서 지원하는 연산자는 [표 6-6]과 같다. 다만 벡터불리안 검색은 불리안과 벡터검색의 모든 기능을 내부적으로 지원하기 때문에 불리안 검색에 비해 속도가 약간 느리다(검색 모델의 속도는 벡터모델이 가장 빠르고, 불리안, 벡터불리안 모델의 순이다; 벡터 >> 불리안 > 벡터불리안).

[표 6-6] 벡터 불리언 검색에서 지원하는 연산자

불리안 연산자	관계 연산자	근접도 연산자	절단 연산자
&(AND)	=	N(NEAR)	*
|(OR)	<	W(WITHIN)	
!(NOT)	<=		
	>=		
	>		

6.3 검색 시스템의 특징 및 사용자 설정

본 절에서는 KRISTAL-III 시스템에서 특화된 모델에 대한 설명을 제공한다. 어떤 검색 모델을 사용하는가에 따라 많은 차이를 보여준다는 점을 상기하기 바란다.

웹, 포탈 사이트와 같이 대용량의 데이터에 대해서 빠른 검색을 위해서는 백터 모델을 사용해야한다. 그리고 정확한 통계 및 건수 등을 요구하는 전문적인 검색에서는 블리언 모델을 사용해야한다.

❒ 검색 모델 특징

검색 모델의 각각의 특징을 간략하게 설명은 [표 6-7]와 같다.

[표 6-7] 각 검색모델의 간략한 특징

특징	블리언 모델	벡터 모델	벡터 블리언 모델
검색 속도	느림	빠름	아주 느림
Recall(재현율)	높음	낮음	높음
Precision(정확율)	낮음	높음	높음
지원 연산자	AND, OR, NOT, WITHIN/NEAR, *	*	AND, OR, NOT, WITHIN/NEAR, *
검색 대상	전문가	일반사용자	전문가
시스템 점유율 (Memory, CPU)	높음	낮음	아주 높음
Memory DB 지원	O	O	O
공백문자 사용자 정의	O	X	O
섹션들 간의 가중치	없음	OR	OR

검색모델의 선택은 클라이언트 입력 변수에 의해 결정된다. 입력변수를 p_in 이라고 하면 p_in.method = BOOLEAN_MODEL 로 설정하면 불리안 검색, p_in.method =

VECTOR_MODEL 이면 벡터 검색모델, p_in.method = VECTOR_BOOLEAN_MODEL 이면 벡터 불리안 검색 모델을 선택한다.

❒ 검색 시스템 사용자 설정

KRISTAL-III 에서는 검색 시스템에 저사양 시스템 및 대용량의 데이터 처리 할 때 등의 시스템적인 제한 설정을 지원한다. 다양한 설정방법은 kristald 데몬을 띄울 때 설정할 수 있다. 설정은 configure 파일에 설정을 한다.

[표 6-8] 검색 시스템 사용자 설정

설정	내용	비고
<Retrieve max_sorting_size="2000"/>	모든 모델에 적용되며 특정 섹션을 정렬하고자 할 때 정렬할 결과 크기를 설정할 수 있다. 설정을 하지 않으면 시스템에서 **Default 값(100,000 건)**을 사용하며 **그 이상의 건수에 대해서는 오류를 반환한다.** 그 이상의 건수로 정렬하고자 할 때는 max_sorting_size 의 값을 증가시켜야 한다. **(unlimited size : -1)**	결과 정렬
<Retrieve max_query_size="20"/>	검색할 질의어 개수를 제한하는 옵션을 설정한다. 설정하지 않으면 제한하지 않는다**(default: unlimited query size)**	질의 제한
<Retrieve max_asterisk_query_size="50"/>	* 연산에 의해 확장되는 최종 질의어 개수를 제한하는 옵션이다 **(default: 20 개).** 예를 들어 “한국*”을 질의로 입력했을 경우 “한국”으로 시작하는 단어 50 개까지만 확장한다. (벡터 모델에서는 3 음절이상부터 지원)	우절단
<Retrieve max_document_list_size="3000"/>	한번에 가지고 올 문서 수를 제한하는 옵션이다. **(default: 2000 개)**	

<Retrieve document_length_normalization="middle"/>	벡터 모델에서 검색한 결과 중에서 긴 문서를 우선으로 할지 여부를 결정하는 옵션이다. Long, short, middle 중 하나를 선택할 수 있다 **(default: middle).**	벡터
<MemoryDB memory_filtering_mode=”true” additional_memory_size=”1000” > <MemorySection name="AN" target_schemas="schema01"/> </MemoryDB>	**빠른 정렬을 위한 설정** 대용량에 자료에 대해서 DB 에서 특정 섹션을 읽어서 정렬을 할 때는 속도가 느리게 된다. 보다 빠른 속도로 검색결과를 정렬하기 위해서 메모리 DB 를 설정한다. 특정 섹션을 메모리에 올려서 빠른 속도로 정렬을 지원하기 위한 사용자 설정이다. 메모리 DB 에 없을 때는 실제 DB 에서 검색을 한다. 정형섹션에 대해서만 적용이 되고, 비정형섹션에 대해서는 256byte 로 절단하여 사용한다. 여러 개의 섹션을 메모리 DB 에 올릴 수 가 있다. **Memory_filtering_mode 는 불리언/벡터 모델에만 적용**된다. 설정을 true 할 때 메모리에 올린 섹션을 Filtering(AND 연산)를 처리하게 된다. 제한 적인 기능이므로 관리자와 협의 필요함. (default 값은 false) 또한 메모리 DB 에 적재되는 섹션값은 정적이기 때문에, 삽입/수정/삭제 등과 같은 관리작업을 지원하지 않는다. DB 관리작업이 발생하는 서비스에서는 사용해서는 안된다.	Memory-DB

<ResultSet reuse="true" max_result_size="0" set_size="5000000"/>	KRISTAL 시스템은 Set-Cache 를 사용한다. Resue=”true” 이전의 검색한 셋을 재사용을 설정한다. 이전의 검색 결과가 유지되므로 다른 사용자가 똑 같은 검색을 하면 다시 검색을 하지 않고 바로 그 결과를 제공하게 된다. Max_result_size 는 대용량의 데이터를 검색 했을 때 모든 검색한 결과를 저장하면 검색 속도가 느려 질 수 있다. 이런한 경우에 지정 건수만큼만 저장을 할 수 있도록 설정할 수 있다. Set_size 는 지정된 값만큼의 저장하고 값이 넘어 갈 경우에 refresh 를 한다. (Defaut 값은 false)	Set-Cache
<ResultSet do_not_keep_document_id=”true” />	KRISTAL Version 2.x.x 버전에 문서의 빠른 수정 처리 하기위해서 기존의 Document ID 를 새로운 Document ID 를 생성하게 된다.(삭제/삽입 방법 사용) 불리언 모델에서 기존의 Document ID 순서되도 문서를 보존 하지 않기위해서는 위의 옵션을 설정해야 한다.	Version 2.x.x 이상 3.1 사용하지 않음

[표 6-8]에서와 같이 검색 시스템의 위와 같은 사용자 제한적인 설정을 적용한다. 이러한 제약을 통한 서버의 기동에 대해서는 7 장을 참조하기 바란다.

❒ 요약 DB 사용자 설정

요약 DB 는 벡터 모델에서만 사용되는 검색 요소 기술이다. 요약 DB 를 구축하는 이유는 벡터 모델로 검색을 할 때 색인어의 고빈도 단어들에 대해서 미리 정규화 작업을 처리한다. 이렇게 정규화 처리된 색인어들에 대해서 가중치를 선별적으로 미

리 계산하여 저장한다. 벡터 모델로 검색을 할 때 보다 빠르게 검색을 가능하도록 하기위한 특별한 기능 중에 한 방법이다. 또한 검색 성능도 향상 시키는 한 방법이다. 구축하지 않은 것 보다 구축하는 것이 더 좋은 성능을 보여 준다. 요약 DB 를 설정할 때는 보통 50,000 <= DF <= 100,000 으로 설정하는 것이 좋다. 보통 50,000 정도를 권장한다.(KRISTAL 기술 문서 요약 DB 구축 실험 참조)

요약 DB 를 구축은 처음 DB 를 구축했을 때 하고 또한 많은 업데이트 처리가 된 경에 다시 구축할 필요가 있다. 단순히 몇개 업데이트 되었다고 재구성할 필요성은 없다.

요약 DB 구축 방법은 kristal_firefox 툴을 이용하여 구축을 한다. kristal_firefox 사용법은 아래와 같다.

```
************************************************************
kristal_firefox version 0.1.2 - GIIS(tm)
************************************************************
------------------------------------------------------------
SYNOPSIS : KRISTALDir DBName DBDir [-tsfqb] ...
------------------------------------------------------------
KRISTALDIR : Directory of the installed KRISTAL-III (KRIRSTAL 디렉토리)
DBName : DataBase Name(데이터베이스 이름)
DBDir : DataBase Directory(데이터베이스 디렉토리)
------------------------------------------------------------
-b : Building summary-dbs
                tables: building tablename [-a:all tables list]
                term df:   (색인어의 빈도 수)
                pruning size: (색인어 재구성 사이즈)
```

예) BLUE DB 모든 Table 에 대해서 색인어가 50,000 이상이 것에 대해서 50,000 으로 Summary DB 구축하기

$>kristal_firefox ~/K2002 BLUE ./volume -b -a 50000 50000

예) BLUE DB 에서 BLUE01 SummaryDB 구축된 테이블 삭제하기

ex) kristal_firefox ~/K2002 BLUE ./volume -d BLUE01

❒ LIKE 검색 기능

LIKE 검색 기능은 KRISTAL 3.1.x 이상 버전에서만 지원한다. 또한 LIKE 검색은 메모리 DB 로 설정한 Section 에 대해서만 검색을 지원하는 기능이다. LIKE 검색은 불리언 모델에서만 기능을 지원한다. 문자열은 256byte 이은 절단되여 사용하므로 256byte 이상의 긴 문자열에 대해서는 검색 처리를 하지 않는 점을 고려해야한다.

기본적인 LIKE 문법은 아래와 같다.

Ex) TITLE: %보검% ➔ (*보검*)와 같은 의미를 가짐

Ex) TITLE: 한국% ➔ (한국*)와 같은 의미를 가짐

Ex) TITLE: %시스템 ➔ (*시스템)과 같은 의미를 가짐

Ex) TITLE: %한국%경제% ➔ (*한국*, *경제*)의 의미를 가짐

LIKE 검색을 할 때 다른 질의어와 혼용하여 사용이 가능하다.

Ex) CONTENT: 정보 & TITLE %보검% ➔ (정보 & *보검*)의 의미를 가짐

제 7 장 KRISTAL 데이터베이스 서버 운영

이 장에서는 서버 기동과 종료, 모니터링 기능 등의 KRISTAL-III 데이터베이스 서버 운영 방법에 대해서 설명한다.

7.1 서버 기동

KRISTAL-III 시스템은 하나의 데몬으로 구성되어 있으며, 서버 기동시 XML 형식의 환경 설정파일을 사용한다.

7.1.1 환경 설정파일 구성

❒ 환경설정 파일 예제

환경 설정파일 형식은 아래와 같으며 대소문자를 구분하지 않는다. 다음은 서버 기동을 위한 환경설정파일의 예이다.

```
<?xml version="1.0"?>

<ServerConfig>
        <Server ip_address="127.0.0.1"
                        port="50000"
                        no_child="3"
                        kristal_root="/home/k2002/K2002" />
        <Database directory="/home/k2002/TEST/BLUE/volume"
                        name="BLUE-DB"
                        description=”BLUE Web Site Directory Service” />
        <Retrieve max_sorting_size=”0” />
        <ResultSet      reuse=”false”
```

```
                    max_result_size="0"
                    set_size=”5000000” />
          <MemoryDB   memorydb_size="200">
                <MemorySection name="AN" target_tables="BLUE01 BLUE02 BLUE03"/>
          </MemoryDB>
<AccessList   ip=”127.0.0.1” />
<AccessList   ip=”203.254.176.*” />
 </ServerConfig>
```

이 데이터베이스는 127.0.0.1(local host)의 포트 50000 번을 통해 접근할 수 있으며 KRISTAL 패키지의 설치 디렉토리(kristal_root)는 /home/k2002/K2002 디렉토리이다. 또한 검색을 담당할 3 개의 자식 프로세스가 작동하고 있다(no_child=3).

데이터베이스의 물리적인 디렉토리는 /home/k2002/TEST/BLUE/volume 디렉토리이며 데이터베이스 이름은 BLUE-DB 이고 DB 에 대한 간략한 설명는 “BLUE Web Site Directory Service”이다.

검색결과에 대한 정렬 제한은 없다(max_sorting_size=).

KRISTAL 서버를 다시 기동하게 되면 이전에 검색한 결과 집합은 재사용하지 않는다(ResultSet reuse=”false”). 결과 집합에 저장할 검색 결과의 건수는 제한이 없으며(max_result_size=”0”), 데이터베이스에 대한 고유한 검색질의 500 만개까지 질의 집합을 저장한다(set_size=”5000000”)

메모리 DB 는 200MB 를 사용하며 BLUE01, BLUE02, BLUE03 테이블의 AN 섹션을 메모리로 적재한다.

BLUE-DB 데이터베이스에 대한 접근권한(Acces Right)은 127.0.0.1(Localhost)과 203.254.176.으로 시작하는 모든 IP 만이 가진다. 이외의 다른 IP 는 접근권한이 없으며 검색 또한 불가능하다. 127.0.0.1 은 지정하지 않더라도 자동으로 AccessList 에 지정된다

각 항목에 대한 설명은 다음을 참조하기 바란다.

대항목	세부 항목	설 명
Server	서버 자체에 대한 설정 부분이다	
	Ip_address	서버의 Ip 주소
	Port	서버가 사용할 port 번호 (기존에 다른 프로그램이 사용하지 않는 port 번호이어야 한다)
	no_of_child	검색 프로세스의 수 (동시에 처리할 수 있는 사용자의 수를 뜻한다)
	kristal_root	KRISTAL-III 가 설치된 홈 디렉토리 위치 (대소문자 구분)
Database	서버가 사용할 Database 에 대한 설정 부분이다	
	Directory	데이터베이스의 볼륨 위치 (대소문자 구분)
	Name	데이터베이스 영문 이름 (대소문자 구분안함)
	Description	데이터베이스 영문 설명
Retrieve	검색과 관련한 제한 설정 부분이다. (검색 속도와 관련이 있다.)	
	Max_sorting_size	소팅시 지정한 개수 이하인 경우에만 소팅한다. (0 일 경우 시스템 Default: 10000)
ResultSet	검색 결과를 저장하는 Result Set 에 대한 설정 부분	
	Max_result_size	검색 결과가 많을 경우 메모리와 속도 문제를 야기할 수 있으므로 저장될 검색 결과를 지정한 수만큼 제한한다. (제한할 수를 적어주면 되며, 0 일 경우는 무제한이다.) default = 0
	reuse	false 를 주면 Result Set 은 모든 기존 결과를 초기화하고 새로 작성된다. 하지만, True 일 경우에는 기존에 사용하던 검색 결과를 그대로 유지하여 사용한다. default = no
	Set_size	검색이 한번 수행될 때마다 result set 에는 검색결과가 한 slot 을 차지하고 저장된다. 이 slot 은 계속 증가하다가 지정한 set_size 를 넘어서면 모든 result set 을 초기화하고 0 부터 다시 시작된다. default = 100000
MemoryDB	메모리에 색인 데이터 일부를 올려서 빠른 검색을 지원하기 위한 설정	

	MemorySection	메모리에 올릴 대상 섹션을 name 으로 지정하고 대상 스키마을 target_schemas 에 지정한다. MemorySection 항목은 여러 개가 나열될 수 있으며 한 섹션에 지정되는 테이블 또한 여러 개가 나열될 수 있다. *Example)* *<MemoryDB>* *<MemorySection name="AN" target_schemas="blue_table"/>* *…* *</MemoryDB>*
AccessList	IP	데이터 접근 권한을 허가하는 IP 를 적어준다. aaa.bbb.ccc.ddd 형식의 주소에서 bbb 이하인 번호는 * 로 다수의 IP 를 지정할 수 있으며, *로 지정한 이하 번호는 무시된다. AccessList 는 여러 개가 올 수 있다. 여기에 지정되지 않는 IP 는 접근을 할 수 없다. 예) aaa.bbb.*.ddd = aaa.bbb.*.*

서버가 유휴 프로세스를 찾는 동안 클라이언트의 속도가 빨라 다음 질의를 던질 경우 서버 Busy 상태가 되므로 동시 사용자 수(no_child)는 최소 2 개 이상으로 설정할 것을 권장한다.

7.1.2 서버 기동

KRISTAL-III 서버(데몬)을 기동한다.

프로그램 이름

kristald

사용법

kristald [–DLvh] <config file>

설명

입력으로 환경설정파일인 <config file>은 반드시 필요하며, -DL 은 선택이다.

-D : 사용하면 데몬 모드(Daemon Mode)로 동작하므로, 터미널 상에 메시지를 표시하지 않는다. 일반적인 서비스에서 사용하므로 작업모드 또는 서비스모드라고도 한다. –D 옵션을 주지 않으면 콘솔 모드(Console Mode)로 작동하여 터미널 상에 메시지를 출력한다.

-L : 사용하면 로깅 모드로 동작하며, 서버로 입력되는 모든 메시지를 로그 파일에 저장한다. 주로 디버깅을 위한 목적으로 사용한다.

-v : 서버의 버전을 표시한다.

-h : 옵션 사용방법을 표시한다.

처음 KRISTAL 데몬을 띄울 때는 –D 옵션을 주지 말고 띄워 에러가 발생하지 않는지 확인 후에 다시 데몬 모드로 띄우는 것이 바람직하다. 다음은 KRISTAL 서버의 정상적인 작동 검증을 위한 예비 서버 기동 및 디버깅 서버 기동에 대해서 설명한다. 서버 환경설정파일의 이름은 blue.config.xml 이라고 하자. 먼저 KRISTAL 데몬을 다음과 같이 콘솔 모드로 작동시킨다.

콘솔 모드 기동 (목적: 서버 정상적인 작동 유무 점검)

```
kristald  blue.config.xml
```

위와 같이 KRISTAL 서버를 기동하면 콘솔에 서버의 메시지가 모두 출력된다. 이 모드를 콘솔 모드라고 한다. KRISTAL 관리도구나 클라이언트 라이브러리로 개발된 인터페이스를 통해 검색을 수행하여 서버가 정상적으로 작동하면 Crtl-c 를 눌러서 서버를 종료한다.

서버가 정상적으로 작동하는 것을 확인한 후에는 KRISTAL 서버를 –D 옵션을 사용하여 서비스 모드로 기동한다.

서비스 모드 기동 (목적: 정상적인 검색 서비스 수행)

```
kristald  -D  blue.config.xml
```

서비스 모드는 데몬 모드라고도 하며 –D 옵션으로 kristald 가 기동되어 백그라운드에서 KRISTAL 서버 작업을 수행한다. 이 모드가 가장 일반적인 KRISTAL 서버의 작업 모드이다.

KRISTAL 서버의 디버깅을 위한 목적이나 서버와 클라이언트의 통신 상황을 점검하기 위한 목적으로 통신 로깅 모드를 채택할 수 있다. 이 경우의 옵션은 –L 을 사용한다. 콘솔 모드에서는 사용자의 질의, 질의 처리과정 등이 모두 콘솔에 표시되므로 굳이 –L 옵션을 사용할 필요는 없다. 따라서 대부분의 경우 로깅 모드는 데몬 모드에서 사용된다. 즉 –D 옵션과의 조합(–DL)으로 사용된다.

로깅 모드 기동 (목적: kristald 디버깅 또는 서버-클라이언트 통신 분석)

```
kristald  -DL  blue.config.xml
```

로깅 모드를 사용하면 KRISTAL 서버와 클라이언트 간의 통신 상황을 모두 기록하기 때문에 서버에 부하를 줄 수 있다. 따라서 일반적인 작업 모드에서는 로깅 모드의 사용을 권장하지 않는다.

서버가 기동되면, 지정한 검색 프로세스의 수 +2 개의 프로세스가 생성된다. 이는 ”*ps –aef | grep kristald*” 로 확인할 수 있다. (system 의 종류에 따라 다른 옵션을 주어야 하는 경우도 있다. 자세한 내용은 각 system 의 매뉴얼을 참조하기 바란다.) 프로세스 번호 중 제일 작은 수의 번호가 메인 프로세스이며 이 메인 프로세스는 자식 프로세스의 관리와 로드 밸런싱 역할을 한다. 마지막 한 개를 제외한 나머지 프로세스들은 메인 프로세스의 자식 프로세스로 실제 검색을 담당한다. 마지막 프로세스는 서버의 검색 프로세스들이 모두 작업을 수행중일 경우, 즉 서버가 Busy 상태일 경우, 이 상태를 클라이언트에게 알리는 Busy 응답 전용 프로세스이다.

서버가 기동될 때 , 데이터베이스 볼륨위치에 메인 프로세스의 프로세스 번호를저장하고 있는 pid 파일이 생성된다. 이 pid 파일은 “kristald 포트번호.pid”의 이름을 가지고 있고 기동 모드에 상관없이 모두 pid 파일이 생상되므로 현재 데이터베이스 디렉토리를 사용하고 있는 모든 kristal 서버의 포트번호를 확인 할 수 있다.

로깅 모드일 경우 로그 파일은 “kristal 포트번호.pid” 의 이름을 가지고 데이터베이스 볼륨위치에 생성된다. 서버 종료 후에도 이 로그파일은 계속 남게되며, 서버 기동시 같은 이름을 가진 로그 파일이 존재할 경우 기존 내용은 사라지고 새로 저장된다. 서버가 새로 기동되지 않은 상태에서 계속하여 동작할 경우, 로그 파일은 크기는 계속 증가하며 이 파일에 대한 관리는 사용자 책임이다. 로그에 기록되는 내용은 클라이언트로부터 서버로 전달된 KRISTAL 프로토콜 전송 XML 메시지와 전송된 시간이다.

7.2 서버 운영

콘솔 모드로 서버를 기동하였을 경우는 화면에 메시지를 출력하기 때문에 서버의 상태를 계속 확인할 수 있다..

```
==========================================================================
Listen Port          : 20032          DB Name          : BIB1
Max Children         : 2              Min Children     : 2
Max FD               : 6              Listen FD        : 3
--------------------------------------------------------------------------
Total Work Count     : 109            Parent's Pid     : 19305
Current Children     : 2
Available Children   : 2
--------------------------------------------------------------------------
Busy Checker : available    | Pid : 19308   | FD : 6    | count : 0
  Child [0] : available     | Pid : 19306   | FD : 5    | count : 102
  Child [1] : available     | Pid : 19307   | FD : 5    | count : 7
==========================================================================
```

Listen Port 와 DB Name 은 이름 그대로이며, Max Children 과 Min Children 은 검색 프로세스의 최대 개수와 최소 개수로 사용자가 환경 파일에서 no_of_child 에 설정해준 값이다. Kristal 의 현재 버전에는 이 값이 같게 되어 있으나 차후에 검색 프로세스의 수를 동적으로 변화시킬 경우를 대비해 이런 형식으로 표시한다. 앞서 기술한 바와 같이 no_child 는 최소한 2 이상을 사용할 것을 권장한다.

Total Work Count 는 서버가 작업을 수행한 전체 서비스의 수이며 Parent's Pid 는 Load balancer 의 프로세스 Id 이다.

Current Children 은 검색 프로세스의 현재 개수다. KRISTAL 의 현재 버전에서 이 수는 항상 일정하며 no_of_child 와 같다.

Available Children 은 작업을 수행하지 않는 유휴 상태의 검색 프로세스 수를 말하는 것으로 KRISTAL 이 운영되는 동안 항상 값이 변한다.

이하는 각 검색 프로세스와 Busy Checker 프로세스의 상태를 보여준다. 프로세스가 사용중일 경우에는 using 이라고 표시되며 유휴상태인 경우에는 available 이라고 표시된

다. Pid 는 프로세스의 id 이며 fd 는 프로세스와 로드밸런스간의 파이프 통신을 위한 file descriptor 이며 마지막 count 는 해당 프로세스가 수행한 작업의 개수이다.

콘솔 모드 기동시에 위와 같은 메시지를 볼 수 있으므로 서버의 현재 상태를 계속 관찰 할 수 있는 장점이 있으나 메시지 출력시간으로 인해 데몬 모드로 띄울 때보다 약간의 속도 저하가 있을 수는 있다. 또한 콘솔을 계속 유지해야 하는 부담이 있으므로 서비스 용도로는 적당하지 않다.

데몬 모드로 기동시에는 속도는 더 빠르나 서버의 상태를 감시하지 못하는 단점이 있다. 따라서 검색속도가 크게 저하되거나 Server Busy 현상이 빈번하게 발생하면 KRISTAL 서버를 콘솔 모드로 띄우고 서버의 상황을 점검할 수 있다. 사용자 수의 증가로 인한 Server Busy 현상에 대해서는 no_child 를 증가시켜서 일부 해소할 수 있다.

검색프로세스의 수(동시 사용자수) 보다 많은 요청이 동시에 서버에 들어올 경우 서버는 다음과 같은 일을 수행한다.

1. 검색 프로세스의 수만큼의 요청을 각 프로세스가 작업을 수행한다.

2. 각 프로세스가 수행하는 요청 이후의 요청은 Busy Checker 가 서버의 상태가 Server Busy 임을 알리는 메시지를 리턴해준다.

3. Busy Checker 가 처리할 수 없을 정도의 더 많은 요청은 Server 의 Listen Queue 에 대기한다.

4. Listen Queue 가 꽉 채우고 초과되는 요청은 Client Library 에서 connect 시간을 측정하여 지정한 시간(현재 3 초)동안 Listen Queue 에 요청이 들어가지 못할 경우 Server Busy 상태임을 알린다. (Busy Checker 가 응답하는 것과 같은 Busy 상태이나 리턴되는 Error code 가 다르다.)

따라서 운영중에 Busy Checker 가 계속 수행될 경우에는 환경 설정 파일에서 검색 프로세스의 수를 늘릴 것을 권장하며(no_child 를 증가), Listen Queue 까지 넘치는 상황이 발생할 경우에는 별도의 물리적인 서버를 증설할 것을 권장한다.

7.3 서버 종료

KRISTAL-III 서버를 종료하기 위해서는 두 가지 방법이 있다. 콘솔 모드로 기동시켰을 경우에는 Ctrl-C 를 눌러 작업을 종료시키거나 다른 터미널로 서버에 접속하여 kristald_stop 이라는 종료 프로그램을 실행시키면 되며, 데몬 모드일 경우에는 kristald_stop 종료 프로그램을 사용하여야 한다.

콘솔모드에서의 종료시에는 종료시까지 '서버가 수행한 작업의 총수', '각 검색 프로세스가 수행한 작업수', 'Busy 상태여서 검색을 수행하지 못하고 리턴된 요청의 수' 등이 표시된다.

서버 종료 프로그램

프로그램 이름

kristald_stop

사용법

kristald_stop <config file>

설명

config file 은 기동시 사용한 환경설정 파일과 같다. KRISTAL 데몬 종료 도구인 kristald_stop 은 config file 내의 데이터베이스 볼륨 디렉토리 위치를 찾아 pid 파일을 오픈하고, pid 를 추출하여 kristal 서버가 종료될 수 있는 특별한 시그널 (SIGUSR1)을 전파한다. 이 시그널을 받은 kristal 서버는 모든 작업중인 데이터베이스를 안전하게 닫고 pid 파일을 삭제한 후 종료된다.

* 주의 : kristald 서버를 강제 종료 (kill signal 의 사용 등) 시킬 경우에는 pid 파일이 삭제되지 않으며 데이터베이스를 손상시킬 수 있다. 특히 데이터베이스에 대한 삽입/삭제/수정이 일어나는 경우에는 강제 종료를 가급적 피해야 한다.

7.4 데몬 상태 감시 도구

KRISTAL-III 는 데몬 상태를 감시하기 위한 도구를 제공한다. 이 도구는 KRISTAL-III 데몬에 상태 감시 서비스를 주기적으로 요청함으로써 상태를 감시한다.

프로그램 이름

kristal_mon

사용법

kristal_mon <Config List File> <Interval> <logfilename>

설명

입력으로 Config List File 과 Interval 을 주는데, Config List File 은 KRISTAL 데몬을 띄울 때 사용한 Config File 들의 위치를 적어놓은 파일이다. Logfilename 은 log 를 저장할 파일의 이름이다.

Interval 은 상태감시 간격이며 단위는 초이다.

결과

출력화면으로 나오는 것은 Interval 과 현재 시각, DB 이름과 DB 를 서비스하고 있는 Kristal 의 IP 와 Port 번호 , DB 의 설명 그리고 현재 상태이다. 현재 상태는 정상적으로 동작중이면 OK, 그렇지 않으면 Failure 가 출력된다.

예제

다음과 같이 상태 검사를 하고 싶은 데몬들의 config file 이름들을 list 로 적어준다.

Config List File : kmontest.conf

/home/k2002/K2002/bin/test-mhlee.xml

/home/gaia/kconfing/blue-test.xml

작성된 파일과 검색 주기, 로그 파일이름을 아규먼트로 하여 실행한다.

Kmon kmontest.conf 30 log

위 아규먼트로 실행하였을 때의 출력 예이다.

```
< KRISTAL-III Monitor 1.0 >

Start Time : Fri Dec 19 16:22:22 2003

Interval : 30 sec                              Fri Dec 19 16:22:22 2003

=============================================================

 DB          IP          PORT          STATUS          DESCRIPTION

=============================================================

BLUE-DB      127.0.0.1   50003          Failure          Test_DB

-------------------------------------------------------------------------------------------------------
```

log 파일에는 모니터링을 시작한 시간이 저장되며, 감시대상 서버가 다운된 시간과 DB 정보가 표시된다.

7.5 클라이언트 라이브러리

KRISTAL 데이터 관리 및 검색 프로그램은 데몬 방식의 서버로 구동 되는데, 사용자의 서버 접근을 통한 데이터 관리 및 검색을 용이하게 하도록 하기 위해 클라이언트 라이브러리를 제공한다. 클라이언트 라이브러리는 사용자의 서비스 요청에 필요한 정보를 입력 받은 후, 사용자를 대신해 프로토콜을 이용하여 서버와 통신을 수행함으로써 사용자의 요구에 적절한 결과를 제공한다.

KRISTAL 클라이언트 라이브러리는 C++ 프로그램과 Java 프로그램의 두 가지 버전이 존재한다. 라이브러리에서 사용되는 요청 함수의 사용법은 프로그램 언어의 특성에 의해서 약간씩 다른 부분이 존재하기는 하나 거의 동일하며 사용자는 자바 버전과 C++ 버전을 혼동없이 쉽게 이용할 수 있다.

C++ 클라이언트 라이브러리 사용법

사용법

사용자 프로그램에 KRISTAL_HOME/include/ClientLib.h 헤더 파일을 인클루드해서 컴파일하고, KRISTAL_HOME/lib/libclient.a 라이브러리 파일을 링크하여 최종 실행 프로그램을 생성한다.

설명

오라클에서 ProC 를 사용하는 것과 같은 원리로 사용자 프로그램에서 KRISTAL C++ 클라이언트 라이브러리를 사용하면 된다.

Java 클라이언트 라이브러리 사용법

사용법

KRISTAL_HOME/lib/xerces.jar 파일과 KRISTAL_HOME/lib/kristal.jar 파일을 CLASSPATH 에 등록하고 사용자 프로그램에서 클라이언트 라이브러리를 임포트한다. (ex. import kristal.client.*) 그리고 나서 사용자 프로그램을 컴파일하여 java 명령어를 이용하여 자바 가상 머쉰에서 프로그램을 실행하도록 한다.

설명

오라클에서 JDBC 를 사용하는 것과 같은 원리로 사용자 프로그램에서 KRISTAL Java 클라이언트 라이브러리를 사용하면 된다.

❒ 클라이언트 요청 서비스

KRISTAL-III 에서는 데이터베이스 리스트, 섹션 리스트, 검색 결과 리스트 등의 정보를 요청하고 가져오기 위한 자료 구조를 개별적으로 정의하지 않고 하나의 통합 Parameter 클래스로 정의하였으며, 섹션, 문서, 테이블 등 세부적인 정보들은 다시 몇 개의 클래스로 정의하였다. 좀더 세부적인 설명은 “Programmer's Manual”을 참조하기 바란다.

[표 7-1] 클라이언트 요청 서비스

서비스명	설명
GET_DB_INFO	DB 에 들어있는 Schema 들의 정보를 얻어온다.
GET_META_INFO_QUERY	사용자 질의, 서버에서 확장한 확장질의, 질의시 사용된 불용어, 순수 질의 단어, 검색된 테이블 이름 리스트, 공백처리 연산자, 검색 방법 등의 정보를 얻어온다.
GET_SET_INFO	셋에 저장된 정보들을 얻어온다.
RETRIEVE	검색을 요청한다. 이때 검색 대상이 되는 테이블 이름, 스페이스 연산 지정자와 검색 모델, 질의 확장, 불용어 제거 옵션을 정의하여 주어야 한다.

RETRIEVE_SIMILAR_DOCUMENTS	유사문서 검색을 수행한다. 이때 비교 대상 문서, 검색 대상 섹션이름, 유사도 값, 검색 대상 결과 셋을 지정한다.
RETRIEVE_IN_RESULT	결과 내 검색을 수행한다. 입력으로는 결과 집합인 결과 셋의 ID 와 질의확장 방법, 불용어 제거 옵션, 질의가 들어간다.
GET_DOCUMENTS_FROM_RESULT	검색된 문서들을 얻어온다. 입력 파라미터로는 결과 셋 번호, 가져올 문서들의 검색 결과 리스트상에서의 시작 위치, 가져올 문서의 개수와 문서 내용 중 가져오고 싶은 섹션들의 이름을 지정한다.
GET_DOCUMENTS_WITH_IDS	테이블 ID 와 문서 ID 를 지정하여 문서의 내용을 가져온다.
GET_DOCUMENTS_WITH_PRIMARY_KEY	문서 내용을 얻어온다. 앞의 GET_DOCUMENTS_XXX 들과 차이점은 primary_key 를 지정하여 문서를 얻어올 수 있다는 것이다.
GET_DOCUMENT_ID_WITH_DOCUMENT	지정한 문서의 문서 아이디를 가져온다. 문서의 지정은 문서의 전체 내용을 지정하던가, 혹은 문서의 Primary Key 를 지정하면 된다.
BROWSE_ALL_DOCUMENTS	지정한 Table 의 모든 문서를 Set 에 넣고 ID 를 리턴한다.
CALCULATE	지정한 섹션의 최대, 최소, 평균, 총합, 유일 키의 개수를 구한다. Query 에 MAX, MIN, AVG, SUM, CNT 등을 넣고 테이블 이름과 섹션 이름을 지정한다.
SORT_BY_SECTION	지정한 결과 셋 내의 값들을 지정한 섹션을 키로 하여 정열한다. Sorting_key_type 은 키가 되는 섹션들의 값들을 문자로 보고 정열을 하는지, 숫자로 보고 정열할지 여부를 결정한다.
PROCESS_DATABASE_SCHEMA	온라인 상에서 데이터베이스 스키마를 처리한다. 스키마 형식은 오프라인에서와 마찬가지로 XML 형식으로 구성하여야 한다. 다만 현재 버전에서는 데이터베이스 스키마 기능 중 데이터베이스 생성과 삭제 기능은 지원하지 않는다.

APPEND_DOCUMENT	한 개의 문서를 삽입한다. Documents 에서 삽입할 문서의 내용은 sections, 삽입을 원하는 테이블은 table_id 에 넣어준다.
DELETE_DOCUMENT	지정한 한 개의 문서를 삭제한다.
UPDATE_DOCUMENT	지정한 한 개의 문서를 변경한다. Documents 의 sections 에 변경할 문서의 내용을 넣어주며 여러 개의 문서를 넣어도 맨 앞의 한 개의 문서만 변경되며 나머지는 무시된다.
CHECK_STATUS	서버의 상태를 check 한다.
SAVE_USER_LOG	사용자가 임의로 작성한 로그를 서버측에 저장하는 프로토콜이다. 로그내용은 데이터베이스 디렉토리에 u_kristald.log 에 저장된다.
GET_XML_NODES_FROM_RESULT	검색된 결과 셋으로부터 XML Node 들을 가져온다. 입력 파라미터 중 Flag 는 XML Node 의 시스템 섹션들 중에 기본적으로 가져오지 않는 추가적인 정보를 얻고 싶을 때 세팅한다.
GET_XML_NODES_WITH_IDS	테이블 ID 와 노드 ID 를 지정하여 Node 의 내용을 가져온다.
GET_XML_NODES_INFO	주어진 XML 노드를 기준으로 왼쪽, 오른쪽 ,부모, 자식, 혹은 루트 노드를 가져온다.
GET_XML_TREE	주어진 노드를 기준으로 하위 노드들의 XML Tree 를 넘겨준다. 이때 node 의 순서는 DFS 를 따른다.
RETRIEVE_DOCUMETNS_WITH_FOREIGN_KEY	지정한 문서의 지정한 섹션과 매핑되는 다른 테이블의 문서들을 검색한다. 이 서비스는 문서 Navigation 구현을 위한 기본 서비스이다.
APPEND_XML_NODE	한 개의 XML 노드 혹은 계층 관계로 구성되어 있는 노드 그룹을 지정된 노드의 왼쪽, 오른쪽, 혹은 자식으로 삽입한다.
UPDATE_XML_NODE	한 개의 XML 노드를 수정한다. 방법은 UPDATE_DOCUMENT 와 같다.
DELETE_XML_NODE	한 개의 혹은 여러 개의 XML 노드를 삭제한다. 사용자가 지정한 노드가 리프 노드일 경우에는 하나의 노

	드만 삭제하지만, 리프 노드가 아닐 경우에는 자손 노드들을 포함하여 삭제한다. 삭제된 노드의 개수는 total_df 를 통하여 반환된다
MOVE_XML_NODE	한 개의 XML 노드를 기준 노드의 왼쪽, 오른쪽 혹은 자식으로 이동한다.
MULTIPLE_RETRIEVE	RETRIEVE 와 동일하나 여러 테이블을 그룹핑하여 검색 결과를 가져올 수 있다. 검색 대상이 되는 모든 테이블들의 스키마는 같아야 하며 RETRIEVE 와 달리 Integrate 옵션을 사용할 수 없다. 그룹 지정은 그룹핑될 테이블들의 수를 group_counts 에 넣어준다. 순서는 테이블 리스트에 넣은 테이블들의 순서를 따른다. Thesaurus_levels 를 주면 시소러스 사전에서 질의어를 확장하여 검색한다.

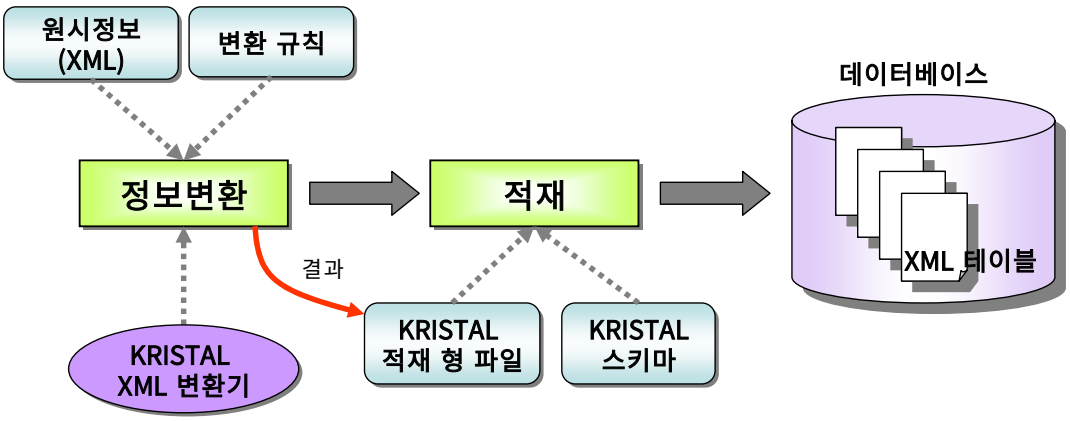
요청에 관련된 세부적인 입력 파라메터나 출력 파라메터에 대한 내용은 “Programmer's Manual”에 자세하게 나와있다. 개발자들은 “Programmer's Manual”을 좀더 상세하게 살펴볼 필요가 있다.

제 8 장 XML 처리

KRISTAL-III 는 PLAIN TEXT 뿐만 아니라, XML 문서에 대한 관리 및 검색도 지원한다. 본 장에서는 XML 문서의 적재를 위한 과정과 이를 위한 제반 도구 및 적재 예제를 위주로 하여 설명한다.

8.1 XML 문서 적재

KRISTAL-III 에서의 XML 문서 검색은 일반 문서 검색을 기반으로 이를 확장하여 처리하는 방식을 채택하고 있다. KRISTAL-III 에 XML 문서를 적재하기 위한 과정은 [그림 8-1]과 같다.

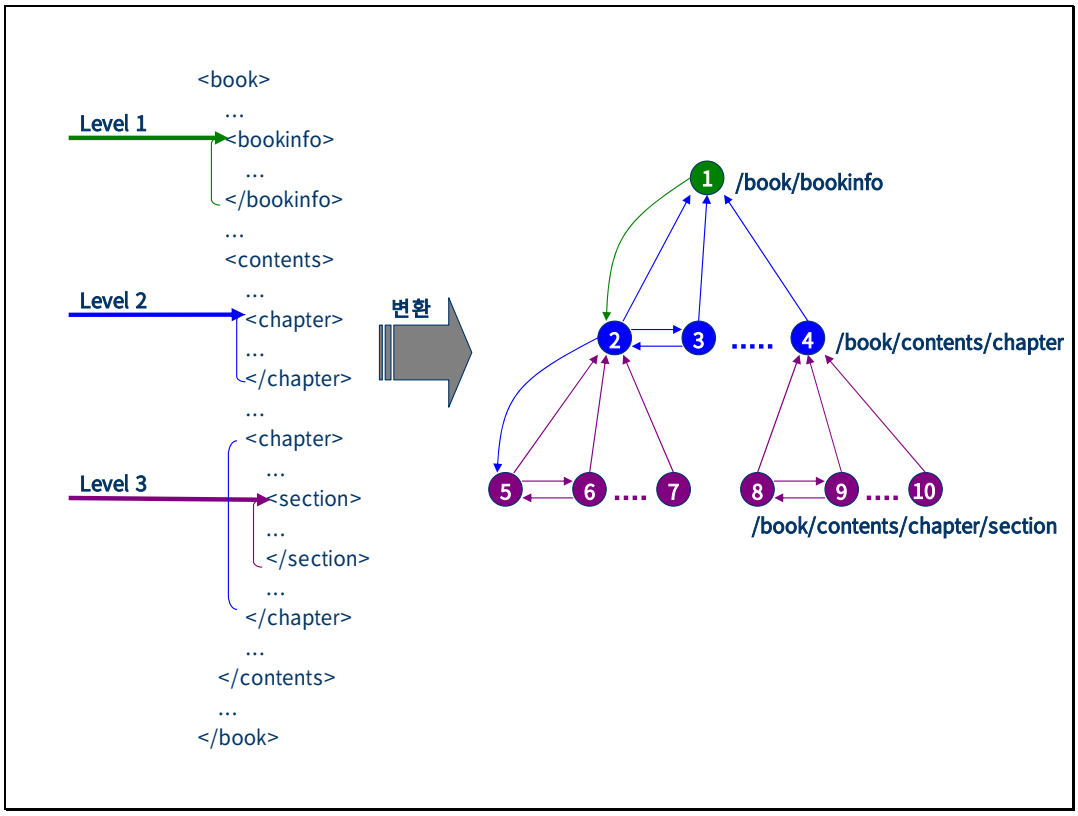


[그림 8-1] XML 문서 적재 과정

[그림 8-1]에서 보여지는 것과 같이 KRISTAL-III 에서 XML 문서를 서비스하기 위해서는 원본 XML 문서와 XML 문서를 변환하기 위한 변환 규칙이 필요하다. 서비스할 XML 문서에 대한 변환 규칙이 정의가 되면 XML 문서 변환 모듈을 이용하여 변환 작업을 수행하고, 스키마를 이용하여 결과로 생성된 적재형 데이터를 저장 시스템에 적재한다.

8.1.1 원시 문서의 구조 정보 수용

XML 문서를 Document Fragment 로 나누는 문서 분할은 복잡한 계층의 XML 문서를 의미 있는 엘리먼트를 중심으로 계층을 단순화시켜서 검색 및 계층적 결과 표현에 사용하도록 하는 작업으로, XML 문서의 부모-자식 관계, 형제 관계 등의 계층 정보를 유지하면서 문서를 단편화한다. 이때 계층 관계 및 병합 정보, 그리고 검색에 관련된 각종 정보를 정의하고 있는 변환 규칙을 이용하여 XML 문서를 구조적 특징을 포함한 단편화된 Document Fragment 로 변형시키는데, 이 각각을 단편노드라고 부른다. 이러한 단편노드는 원본 XML 문서의 구조 정보를 유지하면서 다루기 편한 트리 형태로 재구성된다.



[그림 8-2] XML 문서의 트리 표현

[그림 8-2]를 통해 알 수 있듯이 XML 문서를 단편노드로 분할하면 원본 XML 문서가 가지고 있던 복잡한 엘리먼트 계층 구조를 의미 있는 엘리먼트를 중심으로 이루어진 몇 레벨의 간략한 계층의 트리 구조로 재구성할 수 있다. 또한, 재구성된 트리 역시 원래의 XML 문서가 가지고 있던 노드 간의 부모-자식 관계 및 형제 관계를 그대로 수용하고 있기 때문에 원시 문서 구조 형태로의 복원도 지원할 수 있다.

관리자 입장에서는 복잡한 계층 구조의 XML 문서를 서비스의 목적에 따라서 중요하다고 판단되는 단편노드의 단위로 구분하고 단편노드 안에 존재하는 세부 엘리먼트 및 속성에 대해서 별칭(alias)을 사용하는 검색 필드를 미리 구성해 놓음으로써 보다 쉽게 관리 및 검색 서비스를 제공할 수 있다. 사용자 입장에서는 복잡한 XML 관련 문법을 알지 못하더라도, 키워드 방식의 인터페이스를 사용해 쉽게 XML 문서 검색 서비스를 이용할 수 있다.

8.1.2 XML 문서 검색 지원

XML 검색은 일반적으로 XML 문서의 계층적인 구조에 기반을 두고 있다. 이러한 계층적인 구조를 검색하기 위해서 XPath 나 XQuery 와 같은 XML 질의 표현식을 사용하는데, 이러한 질의는 사용자가 XML 문서상의 계층 구조를 정확하게 알고 있어야만 사용이 가능하다. 또한 XML 질의를 관계형 데이터베이스 시스템에서 처리하기 위해서는 개체와 개체 사이의 계층적인 구조를 검색하기 위한 빠른 조인 알고리즘이 필요하다. 그러나 KRISTAL-III 에서는 관리자가 전처리 과정으로 XML 문서의 계층 관계를 하나의 검색 필드로 정의해 놓을 수 있기 때문에 사용자에게 키워드 방식의 검색 인터페이스를 제공할 수 있을 뿐만 아니라, 구조 검색을 위한 조인을 수행할 필요도 없게 된다. 따라서 위에서 제시된 두 가지 단점을 효과적으로 극복할 수 있다.

예를 들어, 관리자가 EBOOK XML 문서의 /book/contents/chapter 를 하나의 레벨로 설정하고 chapter 엘리먼트의 하위에 존재하는 chapter-title 엘리먼트를 CHAPTER_TITLE 이란 IndexSection 으로 정의해 놓는다면, 사용자는 단순히 CHAPTER_TITLE 이란 섹션을 대상으로 키워드 검색을 수행해서 chapter 단위의 검색을 수행할 수 있으며 검색 결과를 생성하기 위한 book, contents, chapter 간의 조인도 필요가 없게 된다. 또한 XML 문서를 평이한 텍스트로 변환시켜서 색인을 수행한 것이기 때문에 검색 결과에 대한 랭

킹도 더불어 지원할 수 있게 된다. 이 경우에 있어서 평이한 텍스트 형태로 변환시켰다고 하더라도 최종 검색 결과 화면은 사용자가 정의한 각 레벨 정보에 의해서 계층화된 트리 구조로 보여질 수 있다.

8.2 XML 문서 변환기

KRISTAL-III 에서 XML 데이터를 서비스하기 위해서는 XML 문서를 KRISTAL-III 에 적재할 수 있는 포맷으로 변환시켜야 한다. 본 문서에서는 이러한 작업을 수행하는 KRISTAL-III XML 변환기(이하 변환기)에 대해서 설명하고자 한다.

변환기는 KRISTAL-III 에서 XML 문서를 처리할 수 있도록 하기 위해서 제공된다. 이 변환기는 XML 문서의 부모-자식 관계, 형제 관계 등의 계층 정보를 유지하면서 문서를 단편화 시킨다. 이때 사용자가 입력한 변환 규칙을 이용하여 XML 문서를 KRISTAL-III 에 적재할 수 있는 포맷으로 변형시킨다.

8.2.1 변환 규칙(Rule)을 위한 DTD

XML 문서의 KRISTAL-III 적재를 위해서 먼저 변환 규칙을 정의해야 한다. 변환 규칙은 XML 문서에 대한 검색 및 관리 서비스를 위해서 계층 관계 및 병합(Merge) 정보, 그리고 색인 섹션에 관련된 각종 정보를 담고 있는 텍스트 파일로서 XML 문서로 되어 있다. XML 문서를 단편화 시키는데 필요한 변환 규칙에 대한 DTD 는 다음과 같다.

```
<?xml version="1.0" encoding="EUC-KR"?>
<!ELEMENT Rule (LevelInfo+)>
<!ATTLIST Rule
    nodeRelation      (YES | NO) #IMPLIED
    nodeInclusion     (YES | NO) #IMPLIED>
<!ELEMENT LevelInfo (MergeSectionList?, IndexSectionList?)>
<!ATTLIST LevelInfo
    no                CDATA     #REQUIRED
    path              CDATA     #REQUIRED
    constraint   CDATA     #IMPLIED
    nodeDeletion      (YES | NO) #IMPLIED>
<!ELEMENT MergeSectionList (MergeSection+)>
<!ATTLIST MergeSectionList
    delimiter     CDATA     #IMPLIED>
```

```
<!ELEMENT MergeSection EMPTY>
<!ATTLIST MergeSection
    path            CDATA     #IMPLIED
    attr            CDATA     #IMPLIED
    type            (SELF-TEXT | SINGLE-TEXT | MULTI-TEXT | ALL | ALL-SE) #IMPLIED
    delimiter       CDATA     #IMPLIED
    length          CDATA     #IMPLIED
    textDelimiter   CDATA     #IMPLIED
    trim                (YES | NO) #IMPLIED
    newlineDeletion (YES | NO) #IMPLIED>>
<!ELEMENT IndexSectionList (IndexSection+)>
<!ELEMENT IndexSection EMPTY>
<!ATTLIST IndexSection
    name            CDATA     #REQUIRED
    path            CDATA     #IMPLIED
    attr            CDATA     #IMPLIED
    type            (SELF-TEXT | SINGLE-TEXT | MULTI-TEXT | ALL | ALL-SE) #IMPLIED
    delimiter       CDATA     #IMPLIED
    textDelimiter   CDATA     #IMPLIED
    trim                (YES | NO) #IMPLIED
    newlineDeletion (YES | NO) #IMPLIED>
```

변환 규칙 작성에 필요한 각 부분들에 대한 내용은 하위 절에서 예를 들어가면서 자세히 설명한다.

8.2.2 변환 규칙 정의

규칙 정보는 레벨, 병합, 색인에 관련된 변환 규칙에 대해 정의한다.

엘리먼트 이름 : **Rule**

관련 속성 : nodeRelation, nodeInclusion

속 성	설 명
nodeRelation	노드 간의 계층 관계(SYSTEM 섹션) 저장 여부 지정
nodeInclusion	노드의 엘리먼트(DOCUMENT 섹션) 저장 여부 지정

"nodeRelation"이 지정이 되어 있지 않을 경우, "nodeRelaton"는 "YES"를 기본 값으로 갖게 된다.

"nodeInclusion"이 지정이 되어 있지 않을 경우, "nodeInclusion"는 "YES"를 기본 값으로 갖게 된다.

사용 예:

⇨ <Rule>
XML 문서를 단편화 시킴에 있어서 노드 간의 계층 관계와 노드를 구성하는 엘리먼트를 함께 추출해서 저장한다.

⇨ <Rule nodeRelation="no" nodeInclusion="no">
XML 문서를 단편화 시킴에 있어서 노드 간의 계층 관계와 노드를 구성하는 엘리먼트를 저장하지 않는다.

8.2.2.1 레벨 정보 정의

레벨 정보는 XML 문서의 계층 관계와 별도로 사용자가 레벨을 지정하여 계층 관계를 새롭게 생성하는데 사용된다.

엘리먼트 이름 : **LevelInfo**

관련 속성 : no, path, constraint

속 성	설 명
no	새롭게 지정할 엘리먼트의 레벨
path	새롭게 지정할 엘리먼트 이름
constraint	레벨의 제약 조건을 지정

nodeDeletion	레벨로 지정된 노드의 삭제 여부 지정

“no”와 “path”는 반드시 지정을 해야 한다.

“constraint”가 지정 되어 있지 않을 경우, 값이 없는 것으로 간주한다.

“nodeDeletion”이 지정이 되어 있지 않을 경우, “nodeDeletion”는 “YES”를 기본 값으로 갖게 된다.

사용 예:

⇨ <LevelInfo no="1" path="/dataset/record">
XML 문서에서 “/dataset/record”의 엘리먼트를 레벨 “1”로 설정하여 검색 결과 트리에서 첫 번째 계층에 오도록 한다.

⇨ <LevelInfo no="3" path="/book/contents/chapter/section" nodeDeletion="NO">
문서 변환 시에 “section” 엘리먼트의 노드를 제거하지 않고 “chapter” 엘리먼트의 하위에 그대로 유지한다.

⇨ <LevelInfo no="1" path="/dataset/record" constraint="heading[@level='1']">
XML 문서에서 “/dataset/record”의 엘리먼트를 레벨 “1”로 설정하여 검색 결과 트리에서 첫 번째 계층에 오도록 한다. 단 이 경우에는 constraint 속성이 존재하기 때문에 “/dataset/record” 엘리먼트의 하위에 “heading” 엘리먼트가 존재하고 이 엘리먼트의 “level” 속성 값이 ‘1’인 경우에 한해서만 해당 “/dataset/record”의 엘리먼트를 레벨 “1”로 설정하도록 한다.

8.2.2.2 병합 정보 정의

병합 정보는 레벨 정보로 설정된 레코드(노드)의 식별 정보 섹션으로 병합 시에 서로 동일한 레코드인지를 식별하기 위한 정보로 사용된다.

엘리먼트 이름 : **MergeSectionList**

관련 속성 : delimiter

속 성	설 명
delimiter	여러 개의 MergeSection 값을 식별하기 위한 구분자 지정

"delimiter"가 지정이 되어 있지 않을 경우, "delimiter"는 공백을 기본 값으로 갖게 된다.

엘리먼트 이름 : **MergeSection**

관련 속성 : path, attr, type, delimiter, length

속 성	설 명
path	병합 섹션의 엘리먼트 지정
attr	속성 값을 병합 섹션으로 지정
type	병합 섹션의 정보 추출 타입 지정
delimiter	중복된 엘리먼트 값을 식별하기 위한 구분자 지정
length	병합 섹션의 최대 길이 지정
textDelimiter	type 속성으로 텍스트 추출 시 텍스트 구분자 지정
trim	추출될 텍스트의 trim 여부 지정
newlineDeletion	추출될 텍스트의 개행문자 제거 여부 지정

"path"가 지정이 되어 있지 않을 경우, "path"는 "."(self)를 기본 값으로 갖게 된다. 따라서 LevelInfo 에서 지정된 "path"와 동일한 엘리먼트를 지칭하게 된다.

"attr"이 지정이 되어 있지 않을 경우, 값이 없는 것으로 간주한다.

"type"이 지정이 되어 있지 않을 경우, "type"은 "MULTI-TEXT"를 기본 값으로

갖게 된다. 따라서 지정된 엘리먼트 하위에 존재하는 모든 텍스트를 추출하게 된다.

“delimiter”가 지정이 되어 있지 않을 경우, “delimiter”는 공백을 기본 값으로 갖게 된다.

“length”가 지정이 되어 있지 않을 경우 “length”는 0 을 기본 값으로 갖게 된다. 따라서 병합 섹션의 길이에 대한 제한 없이 지정된 모든 텍스트를 사용한다. (추출 텍스트의 의미 있는 사이즈는 1 이상이기 때문에 특별히 사용자가 0 으로 지정한 경우에 한해서는 사이즈에 제약을 두지 않겠다는 의미로 받아들인다.)

“textDelimiter”가 지정이 되어 있지 않을 경우, “textDelimiter”는 공백을 기본 값으로 갖게 된다.

“trim”이 지정이 되어 있지 않을 경우, “trim”은 “YES”를 기본 값으로 갖게 된다.

“newlineDeletion”이 지정이 되어 있지 않을 경우, “newlineDeletion”은 “YES”를 기본 값으로 갖게 된다.

속성 값 정의에 있어서 한 가지 특이한 점은 “attr” 속성이 지정된 경우 “type” 속성은 무시된다는 것이다. “attr” 속성이 지정된 경우에는 추출 방식에 상관없이 단지 해당 속성 값만을 추출하면 되기 때문이다.

사용 예 :

⇨ <MergeSection path="heading" attr="" type="MULTI-TEXT" length="100"/>
<LevelInfo>에서 path 속성 값으로 지정된 엘리먼트의 하위에 “heading” 엘리먼트가 존재할 경우 이 엘리먼트의 텍스트를 “MULTI-TEXT” 방식으로 추출하고, 추출된 텍스트의 사이즈를 최대 100 으로 지정한다.

⇨ <MergeSection path="heading[@lang='ko']" attr="" type="SINGLE-TEXT" length="100”/>
<LevelInfo>에서 path 속성 값으로 지정된 엘리먼트의 하위에 “heading” 엘리

먼트가 존재하고 이 엘리먼트의 속성 중의 하나인 “lang”이 ‘ko’ 값을 가질 경우 이 엘리먼트의 텍스트를 “SINGLE-TEXT” 방식으로 추출하고, 추출된 텍스트의 사이즈를 최대 100 으로 지정한다.

⇨ <MergeSection path="heading" attr="abstract" type="MULTI-TEXT" length="100"/>
<LevelInfo>에서 path 속성 값으로 지정된 엘리먼트의 하위에 “heading” 엘리먼트가 존재하고 이 엘리먼트의 속성으로 “abstract”가 존재할 경우 이 속성 값을 추출하고, 추출된 텍스트의 사이즈를 최대 100 으로 지정한다.

⇨ <MergeSection path="." attr="abstract" type="MULTI-TEXT" length="100"/>
<LevelInfo>에서 path 속성 값으로 지정된 엘리먼트에 속성으로 “abstract”가 존재할 경우 이 속성 값을 추출하고, 추출된 텍스트의 사이즈를 최대 100 으로 지정한다.

8.2.2.3 색인 섹션 정의

색인 섹션은 레벨 정보로 설정된 레코드(노드)의 사용자 정의 색인 섹션으로 레코드 안의 세부 엘리먼트에 대해서 색인과 검색을 지원하기 위해서 사용된다.

엘리먼트 이름 : **IndexSection**

관련 속성 : name, path, attr, type, delimiter

속 성	설 명
name	색인 섹션의 이름 지정
path	색인 섹션의 엘리먼트 지정
attr	속성 값을 색인 섹션으로 지정
type	색인 섹션의 정보 추출 타입 지정
delimiter	중복된 엘리먼트 값을 식별하기 위한 구분자 지정
textDelimiter	type 속성으로 텍스트 추출 시 텍스트 구분자 지정
trim	추출될 텍스트의 trim 여부 지정

newlineDeletion	추출될 텍스트의 개행문자 제거 여부 지정

"name"은 반드시 지정을 해야 한다. "name"을 제외한 IndexSection 엘리먼트의 속성 값 사용 방법은 MergeSection 엘리먼트의 속성 값 사용 방법과 동일하다. 따라서 "path", "attr", "type", "delimiter", "textDelimiter", "trim", "newlineDeletion"이 지정이 되어 있지 않을 경우, 기본 값은 MergeSection 의 경우와 같다.

사용 예 :

⇨ <IndexSection name="title" path="heading" attr="" type="MULTI-TEXT"/>
<LevelInfo>에서 path 속성 값으로 지정된 엘리먼트의 하위에 "heading" 엘리먼트가 존재할 경우 이 엘리먼트의 텍스트를 "MULTI-TEXT" 방식으로 추출하고, 추출된 텍스트를 "title" 섹션으로 지정한다.

⇨ <IndexSection name="author" path="bookinfo/author" type="SINGLE-TEXT" delimiter="|"/>
<LevelInfo>에서 path 속성 값으로 지정된 엘리먼트의 하위에 "bookinfo" 엘리먼트가 존재하고 그 하위에 다시 "author" 엘리먼트가 존재할 경우 이 엘리먼트의 텍스트를 "SINGLE-TEXT" 방식으로 추출하고, 추출된 텍스트를 "author" 섹션으로 지정한다. 단 이 경우에 path 속성 값으로 지정된 엘리먼트가 여러 개가 존재하면, 각 엘리먼트의 값을 모두 추출해서 delimiter 속성에서 지정된 구분자를 이용하여 합쳐준다.

⇨ <IndexSection name="ko_title" path="heading[@lang='ko']" attr="" type="SINGLE-TEXT"/>
<LevelInfo>에서 path 속성 값으로 지정된 엘리먼트의 하위에 "heading" 엘리먼트가 존재하고 이 엘리먼트의 속성 중의 하나인 "lang"이 'ko' 값을 가질 경우 이 엘리먼트의 텍스트를 "SINGLE-TEXT" 방식으로 추출하고, 추출된 텍스트를 "ko_title" 섹션으로 지정한다.

⇨ <IndexSection name="title" path="heading" attr="title" type="MULTI-TEXT"/>
<LevelInfo>에서 path 속성 값으로 지정된 엘리먼트의 하위에 "heading" 엘리먼트가 존재하고 이 엘리먼트의 속성으로 "title"가 존재할 경우 이 속성 값을

추출하고, 추출된 텍스트를 “title” 섹션으로 지정한다.

⇨ <IndexSection name="title" path="." attr="title" type="MULTI-TEXT"/>
<LevelInfo>에서 path 속성 값으로 지정된 엘리먼트에 속성으로 “title”가 존재할 경우 이 속성 값을 추출하고, 추출된 텍스트를 “title” 섹션으로 지정한다.

⇨ <IndexSection name="title" path="../title" attr="" type="MULTI-TEXT"/>
<LevelInfo>에서 path 속성 값으로 지정된 엘리먼트의 부모 엘리먼트에 “title” 엘리먼트가 자식 엘리먼트로 존재할 경우 이 엘리먼트의 텍스트를 “MULTI-TEXT” 방식으로 추출하고, 추출된 텍스트를 “title” 섹션으로 지정한다.

⇨ <IndexSection name="title" path="main-title" attr="" type="MULTI-TEXT"/>
<IndexSection name="title" path="sub-title" attr="" type="MULTI-TEXT"/>

<LevelInfo>에서 path 속성 값으로 지정된 엘리먼트의 하위에 “main-title” 엘리먼트 혹은 “sub-title” 엘리먼트가 존재할 경우 이 엘리먼트들의 텍스트를 “MULTI-TEXT” 방식으로 추출하고, 양쪽에서 추출된 텍스트를 하나로 합쳐서 “title” 섹션으로 지정한다.

⇨ <IndexSection name="comment" path="*/comment" attr="" type="MULTI-TEXT"/>
<LevelInfo>에서 path 속성 값으로 지정된 엘리먼트의 하위에 “comment” 엘리먼트가 존재할 경우 계층에 상관없이 이 엘리먼트 내에 존재하는 모든 텍스트를 “MULTI-TEXT” 방식으로 추출하고, 추출된 텍스트를 “comment” 섹션으로 지정한다. 단 이 경우에 * 문자의 적용 범위는 <LevelInfo>에서 지정된 컨텍스트 범위 안에서만 유효하며, 반드시 */로 시작해야 한다. 즉, 엘리먼트 패스의 중간에 *가 들어가서는 안된다.

⇨ <IndexSection name="category" path="/book/bookinfo/category" attr="" type="SINGLE-TEXT"/>
XML 문서 전체의 구조 내에서 path 속성 값으로 지정된 엘리먼트가 존재할 경우 이 엘리먼트의 텍스트를 “SINGLE-TEXT” 방식으로 추출하고, 추출된 텍스트를 “category” 섹션으로 지정한다. 패스 지정 부분에서 “/”로 시작하는 엘리먼트는 <LevelInfo>에서 지정된 컨텍스트 밖의 데이터를 참조할 필요가 있을

경우에만 사용한다.

⇨ <IndexSection name="category" path="역사" attr="" type="SELF-TEXT"/>
XML 문서 전체의 구조 내에서 path 속성 값으로 지정된 엘리먼트의 텍스트를 추출하는 것이 아니라, path 부분에 들어있는 텍스트 자체를 “category” 섹션으로 지정한다.

8.2.2.4 추출 방식 정의

문장 추출 방식은 XML 문서에서 텍스트를 어떠한 형태로 뽑아 올 것인가를 지정하는 것으로서, 요약 정보 생성 혹은 색인을 위한 텍스트 추출 시에 사용된다. “type” 속성에 사용할 수 있는 추출 방식으로는 다음과 같이 3 가지 방법이 있다.

- **MULTI-TEXT** : 엘리먼트 하위에 존재하는 모든 텍스트를 추출한다.
- **SINGLE-TEXT** : 지정된 엘리먼트 바로 하위에 존재하는 텍스트만을 추출한다.
- **SELF-TEXT :** XML 문서에 존재하는 텍스트를 추출하지 않고 규칙 기술자가 지정한 임의의 값을 사용한다.
- **ALL** : 지정된 엘리먼트를 포함하여 하위에 존재하는 모든 엘리먼트와 텍스트를 추출한다. 즉 있는 그대로의 모든 정보를 추출한다.
- **ALL-SE** : 지정된 엘리먼트를 포함하지 않고 하위에 존재하는 모든 엘리먼트와 텍스트를 추출한다.

사용 예 :

아래의 구조에서 path=”NodeA”로 지정되었다고 가정할 때, type 이 “MULTI-TEXT”인 경우에는 statementA statementB statementC statementA’가 추출되고, “SINGLE-TEXT”인 경우에는 statementA statementA’가 추출되고, “SELF-TEXT”인 경우에는 “NodeA”라는 문자열이 그대로 지정되고, “ALL”인 경우에는 엘리먼트 이름을 포함한 모든 텍스트가 추출된다. 즉 아래의 네모박스 그 자체가 된다. 그리고 “ALL-SE”인 경우에는 “ALL”과 같으나 가장 상위의 엘리먼트를 제외한

다는 차이점이 있다.

```
<NodeA>
        statementA
        <NodeB>
                statementB
                <NodeC>
                        statementC
                </NodeC>
        </NodeB>
        statementA'
</NodeA>
```

8.2.3 변환기 사용

변환기는 사용자가 지정한 XML 문서와 변환 규칙 문서를 입력으로 받아서 KRISTAL-III 에서 적재 가능한 텍스트 문서를 생성해주는 도구이다. 현재 변환기는 자바 프로그램으로 구현되어 있어서, 변환기의 사용전에는 JAVA 환경 설정 과정을 미리 수행하여야 한다.

8.2.3.1 JAVA 환경 설정

JAVA CLASSPATH 환경 변수에 아래 클래스 파일을 등록시킨다. KRISTAL-HOME 은 KRISTAL-III 가 설치된 최상위 디렉토리를 가리키는 환경 변수이다.

클래스 파일	클래스 설명
KRISTAL-HOME/src/libkconverter/libs/xerces.jar	XML 문서 파싱을 위한 패키지
KRISTAL-HOME/src/libkconverter/libs/kristal.jar	KRISTAL-III 서비스를 위한 패키지
KRISTAL-HOME/src/libkconverter/libs/kconverter.jar	XML 문서 변환을 위한 패키지
KRISTAL-HOME/src/libkconverter/examples	변환기 프로그램이 들어있는 디렉토리

8.2.3.2 변환기 사용법

원시 XML 문서와 변환 규칙 XML 문서를 입력으로 받아서 KRISTAL-III 적재 형 문서를 생성하는 방법과, 원시 XML 문서가 들어있는 디렉토리와 변환 규칙 XML 문서를 입력으로 받아서 배치 형식으로 처리하는 방법이 있다. 그리고 문서 인코딩 방식(-e 옵션)과 에러 로그 파일을 지정(-l 옵션)할 수 있는 옵션을 제공한다. 또한 파일의 생성 날짜를 사용자 임의로 지정(-d 옵션)할 수 있는 옵션도 제공한다. 파라메터 없이 java KConverter 를 수행하면 사용 방법을 보여준다.

사용 예:

⇨ java KConverter -e UTF-8 source.xml rule.xml target.txt
-e UTF-8 : target.txt 문서의 인코딩 방법을 지정. 인코딩이 지정된 경우 원본

문서의 인코딩을 무시한다. (생략된 경우에는 원본 문서의 인코딩을 따름)

source.xml : 원시 XML 문서

rule.xml : 변환 규칙 XML 문서

target.txt : KRISTAL-III 적재 형 문서

⇨ java KConverter –l logFile xml_dir rule.xml txt_dir

-l logFile : 변환 과정 중에 에러가 발생한 문서와 메시지를 logFile 에 기록한다.

xml_dir : 원시 XML 문서가 들어있는 디렉토리

rule.xml : 변환 규칙 XML 문서

txt_dir : 배치 형식으로 처리된 KRISTAL-III 적재 형 문서가 들어갈 디렉토리 (xml_dir 에 또 다른 디렉토리가 존재하는 경우 txt_dir 도 xml_dir 의 디렉토리 구조를 recursive 하게 따라감)

⇨ java KConverter -d 20050907111807 source.xml rule.xml target.txt

-d 20050907111807 : source.xml 문서의 생성 날짜를 무시하고 사용자가 입력한 값을 SYS.DATE 에 기록한다. 입력 방법은 -d 옵션을 주고 YYYYMMDDhhmmss 의 14 자리 포맷의 시간을 입력한다. –d file_time 은 문서의 생성 시각을 입력한다. –d current_time 은 현재의 시각을 입력한다. 옵션을 생략하면 SYS.DATE 에 널이 들어간다.

온라인으로 단편화된 XML 문서의 노드 그룹을 삽입(Insert)시키거나 머지(Merge)시키

기 위해서는 SendNodeGroup 프로그램을 이용하여야 한다. 파라메터 없이 java SendNodeGroup 을 수행하면 사용 방법을 보여준다.

사용 예:

⇨ java SendNodeGroup ip port tableid pivotid direction inputFile ruleFile topLevel

- ip - KRISTAL 서버의 IP 주소
- port - KRISTAL 서버의 PORT 번호
- tableid - pivotid 의 테이블 번호
- pivotid - pivotid 의 노드 번호
  pivotid 가 0 이면 노드 그룹 머지를 수행하고, 0 이 아니면 노드 그룹 삽입을 수행한다.
- direction – 노드 그룹을 삽입할 위치
  XML_ROOT_NODE 는 0, XML_LEFT_NODE 는 2, XML_RIGHT_NODE 는 3, XML_CHILD_NODE 는 4 를 입력한다.
- inputFile - 변환할 XML 문서
- ruleFile - 변환 규칙 파일
- topLevel - 변환된 노드 그룹의 TOP 노드 레벨
  topLevel 은 노드 그룹에서 사용자가 삽입하기를 원하는 부분 노드 그룹을 지정하는데 사용된다. topLevel 의 노드는 반드시 하나만 존재하여야 한다.

변환기 사용 시에 원시 XML 문서의 사이즈가 커서 자바 메모리 에러를 발생시킬 경우 자바 힙(heap) 사이즈의 최대 값을 적당한 값으로 조절해 주면 된다.

사용 예:

⇨ java -Xmx256m KConverter …

-Xmx256m : 자바 힙 사이즈의 최대 값을 256m 로 설정한다.

8.2.4 복원기 사용

복원기는 XML 데이터베이스 안에 존재하는 테이블을 대상으로 XML 문서를 복원하는 역할을 수행한다. 변환기에 의해서 단편화되어서 저장된 XML 문서는 각각의 분리된 노드로 존재하기 때문에 하나의 전체 XML 문서를 생성하기 위해서는 복원기를 이용하여 단편 노드를 결합시켜야 한다. 복원기는 원본 XML 문서 형태로의 복원도 가능하지만, 저장 이후에 편집된 작업들을 함께 반영하기 때문에 주로 새로운 문서 생성기로 사용된다. 현재 복원기는 C++ 프로그램으로 구현되어 있고, KRISTAL 설치 디렉토리 밑의 bin 디렉토리에 존재한다.

8.2.4.1 복원기 사용법

복원기는 데이터베이스 안의 테이블에 존재하는 루트 단편 노드의 수만큼 XML 문서를 복원한다.

사용 예:

kristal_dump_xml db_dir db_name table_name file_prefix element_name attribute_nam

db_dir : 데이터베이스 볼륨 디렉토리

db_name : 데이터베이스 이름

table_name : 테이블 이름

file_prefix : 복원되어서 저장될 XML 문서의 파일 접두어 (ex. 접두어가 “ebook”이면 “ebook1.xml, ebook2.xml, …, ebookn.xml”로 XML 문서 파일이 생성된다. 여기서 n 은 테이블 안에 존재하는 실제 XML 문서의 개수이다.)

element_name : 수정될 레벨 속성을 포함하고 있는 엘리먼트 이름

attribute_nam : 수정될 레벨 속성 이름

8.3 XML 문서 적재 예제

8.3.1 원시 XML 문서 작성

KRISTAL-III 에서 서비스하기 위한 원시 XML 문서를 작성한다.

다음과 같은 예가 있을 수 있다.

```
<?xml version='1.0' encoding='EUC-KR'?>
<book>
    <bookinfo>
        <cover>
            <title>더 멀리 더 가까이 통신</title>
            <subtitle>봉화에서 인터넷까지</subtitle>
            <authgrp>
                <author role="지음">토마스 슈트라이스구트</author>
                <author role="옮김">이민아</author>
            </authgrp>
        </cover>
    </bookinfo>
    <contents>
        <chapter>
            <chapter-title>더 멀리, 더 가깝게</chapter-title>
            <p>이 책에서 다루게 될 발명품 중에서 단 한 개도 쓰지 않고 하루를 보낸다고 상상해 보자. 빠르고 손쉬운 통신 수단이 없던 이전 시대의 사람들이 바로 그렇게 살았다. 친구하고 전화로 노닥거릴 수도, 읽을 책 한 권도 없다. 밤이 되어 봐야 볼 텔레비전도 없고 방안에서 호젓하게 라디오를 들을 수도 없다.</p>
            <p>통신 기술이 우리에게 주는 것은 물론 이런 오락거리만이 아니다. 통신은 때로 생존 수단이 되기도 한다. 119 전화 한 통으로 몇 분 안에 경찰이나 앰뷸런스를 부를 수 있다. 텔레비전이나 라디오 방송은 날씨를 비롯하여 위기 상황을 피할 수 있는 많은 정보를 준다.</p>
            <section>
                <section-title>인쇄기, 지식의 대중화</section-title>
                <p>사람의 통신 생활에서 최초의 혁명은 문자의 발명이었다. 이 사건은 수천 년 전의 고대 왕국으로 거슬러올라간다. 문자가 생겨남으로써 통치자는 선언문을 발표하고 상인들은 거
```

```
래 내역을 정리하며 이야기꾼들은 신화와 설화를 기록할 수 있게 되었다.</p>
                <p>처음에는 돌의 표면에 글씨를 새겼다. 그 다음으로 양피지 두루마리가 등장했는
데, 양가죽을 팽팽하게 늘인 뒤 문질러서 글씨를 쓸 수 있도록 표면을 얇게 만들어 사용했다.</p>
            </section>
        </chapter>
    </contents>
</book>
```

8.3.2 변환 규칙 정의

원시 XML 문서를 변환하기 위한 변환 규칙을 정의한다.

다음과 같은 예가 있을 수 있다.

```
<?xml version='1.0' encoding='EUC-KR'?>
<!DOCTYPE Rule SYSTEM "rule.dtd">
<Rule>
    <LevelInfo no="1" path="/book/bookinfo">
        <MergeSection path="cover/title" type="MULTI-TEXT" length="100"/>
        <IndexSectionList>
            <IndexSection name="TITLE" path="cover/title" type="MULTI-TEXT"/>
            <IndexSection name="SUB-TITLE" path="cover/subtitle" type="MULTI-TEXT"/>
            <IndexSection name="AUTHOR" path="cover/authgrp/author[@role='지음']" type="MULTI-
TEXT"/>
            <IndexSection name="TRANSLATOR" path="cover/authgrp/author[@role='옮김']"
type="MULTI-TEXT"/>
        </IndexSectionList>
    </LevelInfo>
    <LevelInfo no="2" path="/book/contents/chapter">
        <MergeSection path="chapter-title" type="MULTI-TEXT" length="100"/>
        <IndexSectionList>
            <IndexSection name="CHAPTER-TITLE" path="chapter-title" type="MULTI-TEXT"/>
        </IndexSectionList>
    </LevelInfo>
```

```
    <LevelInfo no="3" path="/book/contents/chapter/section">
        <MergeSection path="section-title" type="MULTI-TEXT" length="100"/>
        <IndexSectionList>
            <IndexSection name="SECTION-TITLE" path="section-title" type="MULTI-TEXT"/>
        </IndexSectionList>
    </LevelInfo>
</Rule>
```

8.3.3 KRISTAL 적재 형 문서 변환

원시 XML 문서에 변환 규칙을 적용하여 KRISTAL-III 적재 형 문서로 변환한다.

다음과 같은 예가 있을 수 있다.

```
@XML_RECORD
#SYS.RECID=1
#SYS.MERGE= 더 멀리 더 가까이 통신
#SYS.TYPE=L
#SYS.LEVEL=1
#SYS.PARENT=0
#SYS.PREVIOUS=0
#SYS.NEXT=0
#SYS.FIRSTCHILD=2
#SYS.ORDER=1
#AUTHOR= 토마스 슈트라이스구트
#TRANSLATOR= 이민아
#SUB-TITLE= 봉화에서 인터넷까지
#TITLE= 더 멀리 더 가까이 통신
#DOCUMENT=<book><bookinfo>
        <cover>
            <title>더 멀리 더 가까이 통신</title>
            <subtitle>봉화에서 인터넷까지</subtitle>
            <authgrp>
```

```
                <author role="지음">토마스 슈트라이스구트</author>
                <author role="옮김">이민아</author>
            </authgrp>
        </cover>
    </bookinfo></book>
  @XML_RECORD
  #SYS.RECID=2
  #SYS.MERGE= 더 멀리, 더 가깝게
  #SYS.TYPE=L
  #SYS.LEVEL=2
  #SYS.PARENT=1
  #SYS.PREVIOUS=0
  #SYS.NEXT=0
  #SYS.FIRSTCHILD=3
  #SYS.ORDER=1
  #CHAPTER-TITLE= 더 멀리, 더 가깝게
  #DOCUMENT=<book><contents><chapter>
            <chapter-title>더 멀리, 더 가깝게</chapter-title>
            <p>이 책에서 다루게 될 발명품 중에서 단 한 개도 쓰지 않고 하루를 보낸다고 상상해
보자. 빠르고 손쉬운 통신 수단이 없던 이전 시대의 사람들이 바로 그렇게 살았다. 친구하고
전화로 노닥거릴 수도, 읽을 책 한 권도 없다. 밤이 되어 봐야 볼 텔레비전도 없고 방안에서
호젓하게 라디오를 들을 수도 없다.</p>
            <p>통신 기술이 우리에게 주는 것은 물론 이런 오락거리만이 아니다. 통신은 때로 생존
수단이 되기도 한다. 119 전화 한 통으로 몇 분 안에 경찰이나 앰뷸런스를 부를 수 있다.
텔레비전이나 라디오 방송은 날씨를 비롯하여 위기 상황을 피할 수 있는 많은 정보를 준다.</p>
        </chapter></contents></book>
  @XML_RECORD
  #SYS.RECID=3
  #SYS.MERGE= 인쇄기, 지식의 대중화
  #SYS.TYPE=T
  #SYS.LEVEL=3
  #SYS.PARENT=2
  #SYS.PREVIOUS=0
  #SYS.NEXT=0
  #SYS.FIRSTCHILD=0
```

```
#SYS.ORDER=1
#SECTION-TITLE= 인쇄기, 지식의 대중화
#DOCUMENT=<book><contents><chapter><section>
        <section-title>인쇄기, 지식의 대중화</section-title>
        <p>사람의 통신 생활에서 최초의 혁명은 문자의 발명이었다. 이 사건은 수천 년 전의 고대 왕국으로 거슬러올라간다. 문자가 생겨남으로써 통치자는 선언문을 발표하고 상인들은 거래 내역을 정리하며 이야기꾼들은 신화와 설화를 기록할 수 있게 되었다.</p>
        <p>처음에는 돌의 표면에 글씨를 새겼다. 그 다음으로 양피지 두루마리가 등장했는데, 양가죽을 팽팽하게 늘인 뒤 문질러서 글씨를 쓸 수 있도록 표면을 얇게 만들어 사용했다.</p>
    </section></chapter></contents></book>
```

다음은 KRISTAL-III 적재 형 문서의 포멧 태그로서, 다음 절의 KRISTAL-III 데이터베이스 스키마에서 정의 각 섹션에 대응하게 된다. 이중 “#SYS.”로 시작하는 부분은 특별히 내부적으로 선언되는 시스템 섹션과 관련되어 있다.

```
@XML_RECORD          // 레코드 구분자
#SYS.RECID=          // 레코드 아이디
#SYS.MERGE=          // 레코드 병합 식별자
#SYS.TYPE=           // 레코드 타입
#SYS.LEVEL=          // 사용자 정의 레벨 깊이
#SYS.PARENT=         // 부모 레코드 아이디
#SYS.PREVIOUS=       // 이전 형제 레코드 아이디
#SYS.NEXT=           // 다음 형제 레코드 아이디
#SYS.FIRSTCHILD=     // 처음 자식 레코드 아이디
#SYS.ORDER=          // 형제 레코드 상의 순서
#DOCUMENT=           // 레벨로 지정된 XML 문서의 단편
#USER_DEFINED=       // 사용자 정의 섹션
```

8.3.4 KRISTAL 스키마

최종적으로, KRISTAL-III 적재 형 문서는 다음의 *1) KRISTAL-III 데이터베이스 스키마* 및 *2) 적재스키마*에 의해 적재 된다.

```
<DatabaseSchema>
    <CreateDatabase database-name="EBOOK_DB" volume-dir="/home/TEST/eBOOK/volume" />
    <CreateTableSchema name="ebook_schema" doc-type="xml">
        <Stopword file="/home/TEST/stopword/swords-eng"/>
        <Stopword file="/home/TEST/stopword/swords-han"/>

        <!-- KRISTAL SYSTEM SECTION -->
        <!-- 내부적으로 자동 생성되는 부분입니다.
        <BASICSECTION NAME="SYS.RECID" DATA-TYPE="KUINT" DEFAULT-VALUE="0" />
        <BASICSECTION NAME="SYS.MERGE" DATA-TYPE="KCHAR[512]" />
        <BASICSECTION NAME="SYS.TYPE" DATA-TYPE="KCHAR[1]" />
        <BASICSECTION NAME="SYS.LEVEL" DATA-TYPE="KINT" />
        <BASICSECTION NAME="SYS.PARENT" DATA-TYPE="KUINT" DEFAULT-VALUE="0" />
        <BASICSECTION NAME="SYS.PREVIOUS" DATA-TYPE="KUINT" DEFAULT-VALUE="0" />
        <BASICSECTION NAME="SYS.NEXT" DATA-TYPE="KUINT" DEFAULT-VALUE="0" />
        <BASICSECTION NAME="SYS.FIRSTCHILD" DATA-TYPE="KUINT" DEFAULT-VALUE="0" />
        <BASICSECTION NAME="SYS.ORDER" DATA-TYPE="KINT" />
        -->
        <!-- KRISTAL DOCUMENT SECTION -->
        <BASICSECTION NAME="DOCUMENT" DATA-TYPE="KSTRING"
                      REMOVE-STOPWORD="YES"
                      INDEX-TYPE="INDEX_BY_MA" />

        <UnionSection name="BASIC" include-sections="DOCUMENT" />
    </CreateTableSchema>
    <CreateTable table-name="ebook_table" with-schema="ebook_schema"/>
</DatabaseSchema>
```

[표 8-1] 데이터베이스 스키마 예제

XML 문서를 처리하기 위해 데이터베이스 스키마 테이블 정의에서는, 이전에서는 보지 못했던 시스템 섹션들(KRISTAL SYSTEM SECTIONs)과 문서 섹션(KRISTAL DOCUMENT SECTION)에 대한 내용이 추가 되어 있음을 볼 수 있다.

KRISTAL-III 는 XML 문서를 처리하기 위해 내부적으로 9 개의 시스템 섹션들을 정의해서 사용하는데, 그 용도는 XML 의 트리 관계의 유지, XML 문서 병합(MERGE)등과 같은 내용을 포함한다. 표 8-1 의 “SYS.”로 시작하는 섹션들이 시스템 섹션인데, 이 부분은 데이터베이스 스키마 내에 사용자가 명시하는 것이 아니라 내부적으로 자동 생성[25]되는 것임을 주의해야 한다. 시스템 섹션들도 기본섹션(Basic Section)형태로 정의 된 것이기에 일반 섹션들과 같은 방법으로 참조해서 사용할 수 있으나, 시스템 섹션에 대한 수정 등의 기능은 허용되지 않는다.

문서 섹션(KRISTAL DOCUMENT SECTION)은 단편화 된 XML 의 각 부분을 저장하기 위한 부분으로서, 문서의 복원 등의 용도로 사용한다. 문서 섹션에 색인을 생성하여 XML 의 단편화 된 문서 부분에 대한 검색을 수행할 수 도 있으나, KRISTAL-III XML 변환기에서 “IndexSection” 규칙을 사용하여 필요한 문서 부분에 대한 텍스트 만을 추출해서 사용하는 것이 보다 효율적인 방법이다.

테이블을 XML 타입으로 생성하기 위해서는 테이블 스키마 정의시 아래와 같은 형태를 취해야 한다. 문서 타입(doc-type)을 “xml”로 설정함으로써 XML 문서를 처리할 수 있는 테이블을 생성하도록 지시할 수 있다.

```
<CreateTableSchema name="ebook_schema" doc-type="xml">
```

[25] 시스템 섹션들은 색인이 허용되지 않는 섹션이기 때문에, 사용자 오류를 제거하고 스키마의 명확성을 높이기 위해 내부적으로 생성하는 것임.

시스템 섹션의 각각은 8.3.3 절에서 설명한 KRISTAL-III 적재 형 문서의 시스템 포멧 태그에 일대일 대응된다. 즉, KRISTAL-III XML 변환기의 결과로 생성된 문서중 시스템 포멧 태그로 정의 된 부분은 문서 적재 과정에서 시스템 섹션에 올바르게 연결되어야 한다.

```
<LoaderSchema database-name="EBOOK_DB"
                    volume-dir="/home/TEST/eBOOK/volume" kristal-root="/home/K2002">
    <DocStructure name="book_structure" border-string="@XML_RECORD">
        <!-- 내부적으로 자동 생성되는 부분입니다.
        <Tag name="#SYS.RECID=" mapping-section="SYS.RECID"/>
        <Tag name="#SYS.MERGE=" mapping-section="SYS.MERGE"/>
        <Tag name="#SYS.TYPE=" mapping-section="SYS.TYPE"/>
        <Tag name="#SYS.LEVEL=" mapping-section="SYS.LEVEL"/>
        <Tag name="#SYS.PARENT=" mapping-section="SYS.PARENT"/>
        <Tag name="#SYS.PREVIOUS=" mapping-section="SYS.PREVIOUS"/>
        <Tag name="#SYS.NEXT=" mapping-section="SYS.NEXT"/>
        <Tag name="#SYS.FIRSTCHILD=" mapping-section="SYS.FIRSTCHILD"/>
        <Tag name="#SYS.ORDER=" mapping-section="SYS.ORDER"/>
        -->
        <Tag name="#DOCUMENT=" mapping-section="DOCUMENT"/>
    </DocStructure>
    <LoaderMap table="ebook_table" doc-structure="book_structure"
                        file="/home/TEST/eBOOK/data/test.dat" encoding="EUC-KR" />
</LoaderSchema>
```

[표 8-2] 적재 스키마 예제

표 8-2 에서 주석 처리된 부분(내부적으로 자동 생성되는 부분)이 바로 시스템 섹션과 시스템 포멧 태그를 연결해 주는 부분으로서, 사용자의 이해를 돕기 위한 목적으로 기술한 것일 뿐 사용자가 이를 정의해서는 안됨을 주의해야 한다.

제 9 장 KRISTAL 데이터베이스 설계

오라클과 같은 DBMS 와 마찬가지로 KRISTAL 에서의 데이터베이스 설계는 전체적인 성능에 큰 영향을 미친다. 본 장에서는 다음과 같은 순서로 KRISTAL 데이터베이스 설계에 대해서 살펴본다.

- 문서 분석 및 섹션 색인방식 선택
- 테이블 분할
- 검색모델 선택 및 인터페이스 설계
- 서비스 시스템 설계

9.1 문서 분석 및 색인방식 선택

KRISTAL-III 에서 사용하는 문서의 분석은 간단하다. 일반적으로 KRISTAL 에 적재할 문서의 형태는 미리 고정되어 있으므로 섹션별로 적절한 데이터 타입을 지정해주면 된다. 이 경우 고정된 길이의 문자열에 대해서는 KCHAR 형을 사용하고 숫자의 경우에는 KINT, KUINT, KFLOAT 등의 숫자형태를 사용할 것을 권장한다. 이외에 문헌의 요약문이나 본문 등과 같은 가변길이의 섹션은 KSTRING 을 선택하여야 한다.

고정길이 섹션과 가변길이 섹션에 아래에서 보다 자세하게 살펴보자.

9.1.1 고정길이 섹션

먼저 고정길이 섹션 타입은 고정된 길이를 가지는 섹션에 적용할 수 있다. 여기에 사용되는 섹션 타입으로는 KCHAR, KINT, KFLOAT, KUNIT 등이 있다. 대개의 경우 사용가능한 색인방식은 INDEX_AS_IS 또는 INDEX_AS_NUMERIC 의 두 방식이다. 데이터 형태별로 예를 들어서 섹션 타입 및 색인방식에 대해 고려해보자.

```
#CONTROL_CODE=KISTI050008
```

시나리오: 위 예제는 데이터 제작 기관에서 고유하게 부여하는 데이터베이스의 제어 번호를 보여주고 있다. 이 코드는 각 기관에 따라 달라질 수 있으나 전체적인 문서를 분석한 결과 기관에 따라 길의의 차이는 있으나 최대길이가 16 바이트인 것으로 나타났다. 또한 모두 영문 알파벳과 숫자의 혼합형이었다. 또한 데이터베이스의 주요키로 등록하여 데이터의 중복 등록을 방지해야 한다. 그러나 검색에서는 사용하지 않으며 관리목적으로만 사용한다.

섹션 설정: <PrimaryKey sections=”CRTLCODE” /> <BasicSection name=CRTLCODE data-type=”KCHAR[16]” index-type=”DO_NOT_INDEX” />

설명: CRTLCODE 는 주요키의 역할을 해야 하므로 PrimaryKey 섹션으로 등록해야 한다. 이 섹션은 섹션값이 가변 길이기는 하나 최대 길이가 16 바이트이므로 섹션길이의 예측이 가능하므로 KSTRING 대신 KCHAR[16]으로 선언한다[26]. 또한 이 섹션에 대한 검색은 없고 관리의 목적으로만 사용하므로 색인을 할 필요는 없다.

적용방법: 주요키(Primary Key)를 이용한 문서 접근, 주요키의 중복 체크

```
#LOCUS_NAME=A16SRRNA
```

시나리오: 위 예제는 유전자 데이터베이스에서 지정하는 유전좌의 위치로 각 유전자에 대해서 고유하게 부여되는 부호로 영문과 숫자의 조합으로 이루어져 있다. 최대 길이는 13 바이트인 것으로 분석되었다. 다른 기관에서 제작하는 데이터이기 때문에 주요키를 통한 데이터의 중복을 우리가 검사할 필요는 없다. 그러나 이 섹션에 대한 검색

[26] KCHAR 형은 KSTRING 형에 비해서 문서 관리가 유리한 장점이 있다.

은 가능해야 한다.

섹션 설정: <BasicSection name="LOCUS" data-type="KCHAR[13]" index-type="INDEX_AS_IS" />

설명: 주요키로 사용할 필요가 없으므로 PrimaryKey 로 등록하지 않는다. 길이가 최대 13 바이트였으므로 데이터 타입은 KCHAR[13]으로 선언하였다. 이 섹션값은 검색 대상이 되며, 영문 알파벳과 숫자의 조합으로 구성되는 단일 키워드로 구성되기 때문에 INDEX_AS_IS 로 색인방식을 채택하였다.

적용방법: 주요키(Primary Key)를 이용한 문서 접근 또는 LOCUS 섹션에 대한 불리안(BOOLEAN) 검색

```
#INPUT_DATE=2005-03-08
```

시나리오: 위 예제는 문서의 작성날짜를 의미한다. 모든 문서는 동일한 형태로 "년도-월-일"의 10 자리로 구성되어 있다. 날짜는 검색에서 날짜를 제한하기 위한 항목으로 사용하여야 한다.

섹션 설정 1: <BasicSection name="DATE" data-type="KCHAR[10]" index-type="INDEX_AS_IS" />

설명: 앞서의 설명과 유사하다.

섹션 설정 2: <BasicSection name="DATE" data-type="KCHAR[10]" index-type="DO_NOT_IDEX" />

설명: 섹션 설정 1 은 앞에서 나온 LOCUS 예제의 설명과 유사하다. 그러나 섹션 설정 2 는 검색을 해야 함에도 불구하고 색인을 하지 않았다. 이 경우에는 DATE 섹션의 값을 메모리 DB 로 올리는 경우에 해당할 수 있다. 앞서 설명한 바와 같이 메모리 DB

는 DB 수정을 지원하지 않으므로 게시판과 같은 서비스에서의 날짜는 반드시 첫번째와 같이 색인을 하기를 권장하며, 메모리 DB 를 사용하는 경우에도 검색에 필요할 경우에는 색인을 미리 해두기를 권장한다.

적용방법: 메모리 DB, 불리안 방식의 검색을 통한 검색결과 제한

```
#BIRDTH_DATE=19990122
```

시나리오: 위 예제에서는 생일을 보여주고 있다. 모든 문서는 동일한 형태로 “년도월일”의 8 자리로 구성되어 있다. 날짜는 검색에서 나이를 제한하는 항목으로 사용하여야 한다.

섹션 설정: <BasicSection name=”BIRTH_DATE” data-type=”KINT” index-type=”INDEX_AS_NUMERIC” />

설명: 이 섹션은 특수문자가 들어있지 않으므로 숫자(정수)로 색인할 수 있다. 따라서 여기서는 KINT 데이터 타입을 지정하고 숫자 색인방식인 INDEX_AS_NUMERIC 을 적용하였다.

9.1.2 가변길이 섹션

가변길이 섹션 타입은 섹션값의 길이가 일정하지 않은 섹션에 적용할 수 있으며, 데이터 타입은 KSTRING 으로 지정한다. 정보검색에서 사용하는 대부분의 섹션이 가변길이 섹션으로 적용된다. 가변길이 섹션에 대해서는 INDEX_BY_MA, INDEX_BY_TOKEN, INDEX_BY_CHAR, INDEX_BY_MIX_CHAR 등의 색인방식을 적용할 수 있다. 생물서열에 대해서는 INDEX_DNA, INDEX_PROTEIN 등의 색인방식을 적용할 수 있다. 예제를 위주로 섹션에 대한 정의에 대해서 살펴보자.

```
#KOR_TITLE=금속 비누의 성질과 역할
```

시나리오: 위 예제는 과학기술 문헌 데이터베이스에서 한글 제목의 예제를 보여주고 있다. 한글 제목은 기본적으로 검색의 대상이 되며 검색어는 조사를 제외한 명사를 위주로 검색되어야 한다.

섹션 설정: <BasicSection name=”KTIT” data-type=”KSTRING” index-type=”INDEX_BY_MA” remove-stopword=”YES” />

설명: 검색의 편의를 위하여 섹션이름은 KTIT 로 간략하게 하였다. 제목은 가변길이이므로 KSTRING 을 설정한다(다만 제목과 같이 짧은 섹션인 경우에는 분석을 통해 최대길이를 산출하여 KCHAR 형으로 잡아도 무방하다). 한글 제목은 자연어인 한국어로 기술되므로 한글 형태소 분석을 통해 명사를 추출하여야 한다. 따라서 형태소 분석을 이용하는 색인방식인 INDEX_BY_MA 를 사용하였다. 그리고 과학기술문헌을 대상으로 하는 데이터베이스이므로 일반적인 부사나 형용사 등은 색인에서 제거하도록 remove-stopword 속성을 참으로 설정하였다.

색인어 예: “금속”, “비누”, “비누의”, “성질”, “성질과”, “역할”

추천검색모델: 웹 검색과 마찬가지로 사용자가 원하는 문서를 찾고자 하는 경향이 강할 것이므로 벡터검색모델을 추천한다. 그러나 DB 관리 등과 같이 재현율(Recall)이 중요한 경우라면 불리안(BOOLEAN) 검색모델을 사용하여야 한다.

```
#ENG_TITLE=Preliminary studies of thick films
```

시나리오: 위 예제는 과학기술 문헌 데이터베이스에서 영문 제목의 예제를 보여주고

있다. 영문 제목은 기본적으로 검색의 대상이 되며 검색어는 명사를 위주로 검색되어야 한다.

섹션 설정: <BasicSection name="ETIT" data-type="KSTRING" index-type="INDEX_BY_TOKEN" remove-stopword="YES" />

설명: 검색의 편의를 위하여 섹션이름은 ETIT 로 간략하게 하였다. 제목은 가변길이이므로 KSTRING 을 설정한다(다만 제목과 같이 짧은 섹션인 경우에는 분석을 통해 최대길이를 산출하여 KCHAR 형으로 잡아도 무방하다). 영어 문서는 일반적으로 단어 단위의 재분석이 불필요하다. 따라서 영문 제목은 어절(토큰) 단위로 색인을 하면 충분하므로 INDEX_BY_TOKEN 형식으로 색인을 지정한다. 그리고 검색에 불필요하다고 예상되는 단어들은 색인에서 제거하도록 remove-stopword 속성을 참으로 설정한다.

색인어 예: "preliminary", "studies", "films"[27]

추천검색모델: 위의 예제와 동일하며 재현율이 중요한 고려대상이 아니라면 벡터검색모델을 추천함.

#TEXT= . . .
前述하였듯이 漢字는 새롭게 外國語로서 習得하여야 하는 영어와는 달리 우리의 日常 言語에 자연스럽게 內包되어 있으므로 우리가 조금만 노력한다면 충분히 활용이 가능한 無形의 資源이라 할 수 있는 바, 世界化 시대에 漢字敎育이 한층 더 强化되어야 하는 이유는 바로 여기에 있다고 본다.

[27] 불용어는 주로 to, on, in, of 등과 같은 전치사와 very, nice 등과 같은 부사나 형용사를대상으로 한다. 이 예에서는 불용어 목록에 "of"와 "thick"이 포함되어 있는 것으로 가정하였다. 이 예제서 보여지는 것처럼 thick films 이 중요한 의미를 지님에도 불구하고 잘못된 불용어의 선택으로 인해 thick 이 검색되지 않아서 서비스의 정확도가 떨어질 수 있음을 주의하기 바란다.

```
...
```

시나리오: 위 예제는 국한문이 혼용된 본문의 예제를 보여주고 있다. 본문에 대해서는 검색을 해야 하며 한글로 검색하더라도 한자 부분이 검색되어야 한다.

섹션 설정: <BasicSection name="TEXT" data-type="KSTRING" index-type="INDEX_BY_MA" remove-stopword="YES" hanja-to-hangul="YES" />

설명: 본문은 기본 섹션으로 TEXT 라는 이름으로 지저하였다. 본문의 길이는 문서마다 다양할 것이므로 데이터 타입은 KSTRING 으로 선언하였다. 또한 본문에는 검색에 불필요한 단어들이 다수 포함되어 있을 것으로 추정하여 불용어를 처리한다. 한자로 기술된 부분은 한글로도 검색이 되어야 하므로 hanja-to-hangul 속성을 YES 로 지정하였다. 이 경우 색인시에는 모든 한자가 한글로 변환된다. 다음은 TEXT 섹션에 대해서 추출되는 색인어의 예이다.

색인어 예: "전술", "하였듯이"[28], "한자", "외국어", "습득", "영어", …

추천검색모델: 벡터공간모델

```
#BODY=..., 亂初李舜臣·元均等, 經營創立時, 湊合各色軍兵, 且請道內各司奴子, 使之輪回入防, 仍成規例, 行之至今. 遠地之人, 憚於往來, 自備番布, 代給各浦居人, 居人受布而爲生, 防軍給價而歸家, 兩相便利, 而其中或有邊將之侵暴, 士卒之刁蹬, 貽害多端, 轉成積弊, 水陸換定, 以爲永久之計, 則僻邑者無遠赴之苦, 納布者無被徵之患, 乘船者無風濤之恤, 而近山傍海之
```

[28] KRISTAL-III 의 INDEX_BY_MA 색인 타입은 어절단위 즉 INDEX_BY_TOKEN 방식의 색인을 함께 수행함에 유의하기 바란다.

```
民，良賤相雜，專屬舟師，則良民甚病，前後此論之不行，似由於此，如是變通，尙有難便 ...
```

시나리오: 위의 예제는 고서(古書)의 본문를 보여주고 있다. 이 고서는 한자로만 씌어 있으나, 한글로도 검색이 가능하여야 한다. 예를 들어 “이순신”으로 검색했을 경우 본 예제의 문서가 검색되어야 한다.

섹션 설정: <BasicSection name=”BODY” data-type=”KSTRING” index-type=”INDEX_BY_MIX_CHAR” hanja-to-hangul=”YES” use-index-bigram=”YES” />

설명: KRISTAL-III 는 한자에 대해서는 낱자로 색인할 수 있는 특수한 색인 방식두 가지를 제공한다. 하나는 여기서 선택한 INDEX_BY_MIX_CHAR 이며 다른 하나는 다음 예제에서 설명할 INDEX_BY_MIX_MA 이다. BODY 섹션은 한자로만 구성되어 있으므로. 한자의 낱자 단위로 색인한다. 또한 hanja-to-hangul 속성을 켜서 한자의 경우 한글로도 색인이 되도록 하였다. INDEX_BY_MIX_CHAR 방식은 각 한자를 낱자로 색인하므로 검색속도가 저하되는 단점이 있다. 이를 보완하기 위하여 use-index-bigram 옵션으로 연이어 출현하는 색인을 동시에 색인하는 방식을 취한다. 이렇게 저장된 BODY 섹션은 한자로도 한글로도 검색이 가능하다..

색인어 예: “亂”, “란”, “난” “初”, “초”, “李”, “이”, “리”, “舜”, “순”, …, “李^舜”, “舜^臣”

추천검색모델: 불리안모델. 이러한 문서의 경우, 색인방식의 특이성 때문에 벡터공간 모델을 적용하기는 어렵다. 그러나 재현율이 중요한 고려대상이 아니고 상위 문서에 원하는 문서를 제공하길 원하는 경우에는 벡터검색모델을 사용할 수도 있다.

```
#POEM=黃鳥歌(황조가)
翩翩黃鳥 雌雄相依 念我之獨 誰其與歸
(꾀꼬리 오락가락, 암수 서로 노니는데, 외로워라 이 내 몸은, 뉘와곰 돌아가랴)
```

시나리오: 위의 예제는 고시(古詩)의 예제를 보여주고 있다. 황조가는 고구려 2 대 유리왕이 지은 시로 한글로 변역된 시가 첨부되어 있는 상황이다. 고시는 한자로도 검색이 가능하여야 하고 한글로도 검색하여야 한다.

섹션 설정: <BasicSection name="POEM" data-type="KSTRING" index-type="INDEX_BY_MIX_MA" hanja-to-hangul="YES" use-index-bigram="YES" />

설명: POEM 섹션은 한시와 한글 번역시가 공존하기 때문에 한자는 한자대로 한글은 한글대로 색인하는 INDEX_BY_MIX_MA 색인 방식을 지정한다. 이 경우, 한자는 낱자 단위의 INDEX_BY_MIX_CHAR 방식의 색인이 되며 한글은 IDNEX_BY_MA 방식으로 색인된다. 또한 hanja-to-hangul 속성을 켜서 한자의 경우 한글로도 색인이 되도록 하였다. 한자의 경우 낱자로 색인을 하게 되는데 이러한 경우 각 색인어에 저장되는 정보가 너무 많기 때문에 검색속도가 저하되는 단점이 있다. 이를 보완하기 위하여 use-index-bigram 옵션으로 연이어 출현하는 색인을 동시에 색인하는 방식을 취한다. 이렇게 저장된 POEM 섹션은 한자로도 한글로도 검색이 가능하다..

색인어 예: "黃", "황", "鳥", "조", …, "황조가", …, "翩^翩", "翩^黃", "黃^鳥" …

추천검색모델: 불리안 검색모델.

```
#AHTHOR=홍_길동;최_치원;박_문수;을지_문덕;
```

시나리오: AUTHOR 섹션은 저자를 나타내는 섹션으로 ";" 구분자로 저자간을 구분하고 있다. 또한 성과 이름을 구분하기 위해 "_"를 사용하고 있다. 이 섹션은 이름전체에 대하여 검색을 수행하여야 한다. 즉 "홍길동"으로 검색할 경우 이 문서가 검색되어야 한다.

섹션 설정: <BasicSection name="AUTHOR" data-type="KSTRING" index-

type=”INDEX_BY_TOKEN” delimit-chars=”;” delete-chars=”_” />

설명: 저자 섹션은 고도로 통제된 단어들을 포함하고 있다. 반면 저자의 수는 예측 가능하지 않으므로 AUTHOR 섹션을 가변길이 섹션 타입인 KSTRING 으로 정의하였다. 각 색인어는 구분자를 통해 분리되어야 하므로 delimit-chars 에 “;”에 선언하였고 성을 구분하기 위한 특수문자인 “_”는 제거되어야 하므로 delete-chars 로 선언하였다.

색인어 예: “홍길동”, “최치원”, “박문수”, “을지문덕”

추천검색모델: 불리안검색모델

9.2 테이블 분할

KRISTAL-III 는 다중 스키마를 지원할 뿐만 아니라, 동일한 스키마를 가지는 다수의 테이블을 지원한다 (동일한 스키마를 가지는 테이블의 묶음을 논리적으로는 테이블 클러스터라 부른다). 이러한 구조는 다양한 정보서비스 환경에서 유연한 서비스 구조를 선택하여 설계할 수 있는 KRISTAL-III 의 구조적 유연성의 바탕이 된다. 본 절에서는 다양한 환경에서 어떻게 테이블을 구성할 것인가에 대한 지침을 제공하고자 한다.

먼저 정보서비스 환경을 살펴보면, 데이터의 관리관점에서는 관리와 검색의 두가지 개념으로 구분할 수 있다. 대부분의 정보서비스는 검색을 위주로 하는 서비스이지만 데이터베이스 수정이 빈번하게 발생하는 서비스에서는 관리 위주의 서비스가 구성되기도 한다. 또한 대용량 데이터베이스의 경우에는 검색서비스와 관리서비스를 이원화하는 구조를 채택할 수도 있다. 검색과 관리는 서로 시스템적으로 속도 면에서 서로 상충하는 데이터 관리방식이다. 즉 검색을 빠르게 하기 위한 테이블 구성을 하게 되면 관리 속도가 저하되며 관리속도를 빠르게 하면 검색속도가 저하되는 단점이 발생할 수 있다. 따라서 검색 서비스와 관리서비스에서는 이러한 점을 고려하여 적절한 테이블 구성을 설계하여야 한다[29].

[29] 문서 관리와 검색 성능이 서로 상충하는 것은 대용량 데이터베이스의 경우이다.

먼저 KRISTAL 에서는 단일 기계에서 서비스하는 데이터의 건수를 1500 만건 이하로 할 것을 권장한다[30]. 1500 만건의 의미는 INDEX_BY_MA 또는 INDEX_BY_TOKEN 등과 같은 일반적인 색인방식을 사용하고 각 문서의 크기가 초록을 포함하는 서지 사항 정도의 데이터베이스에 해당한다. INDEX_BY_CHAR 나 INDEX_BY_MIX_CHAR 등과 같은 낱자 색인이나 INDEX_DNA, INDEX_PROTEIN 등과 같은 N-Gram 방식의 색인에 대해서는 단일 기계에서의 서비스 권장량은 더 감소할 수 있다. 이러한 데이터 분산에 의한 검색서비스의 성능향상은 검색속도가 중요한 대용량 데이터베이스에서는 심각하게 고려해보길 권장하는 사항이다.

9.2.1 검색 데이터베이스의 테이블 분할

정보검색시스템을 구축하여 만들어지는 대부분의 정보서비스는 검색서비스로 볼 수 있다. 이러한 경우 일반 사용자는 수동적으로 정보를 받아들이는 입장이며 제공되는 정보의 수정에 참여하는 경우는 거의 없다고 볼 수 있다. 이러한 경우에는 빠른 검색에 주안점을 두어서 서비스 시스템을 설계하게 된다.

KRISTAL-III 를 이용하여 문서의 수정 및 삭제가 발생하지 않는 검색서비스를 구성하는 경우에는 테이블의 구성을 간단하게 할 것을 권고한다. 즉, 동일한 스키마에 속하는 테이블의 개수를 최소화하는 것이 좋다(계속 설명하는 바와 같이 중소형 데이터베이스에서는 테이블의 개수가 검색에 미치는 영향이 크지는 않다). 예를 들어 2000 만건의 서지정보 데이터베이스를 약 100 만건씩 20 개의 테이블로 분할하여 적재한 경우와 2000 만건 모두를 1 개의 테이블로 구성하는 경우 검색속도의 차이가 크다. 그 원인은

수만~수십만에 이르는 일반적인 데이터베이스에서는 이러한 문제가 거의 발생하지 않는다. 수백만 건 이상을 다루는 데이터베이스에서는 관리와 검색의 속도가 서로 상충하는 문제가 발생한다.

[30] 1500 만건은 단지 권고안이다. 예산이나 관리의 효율성 때문에 부득이 3000 만건 이상의 데이터베이스를 단일 기계에서 서비스하고 있는 실제 서비스도 있다.

동일한 정보를 가져오기 위하여 1 개 테이블 구조에서는 디스크를 1 번 접근하면 되지만 20 개 테이블로 구성될 경우에는 20 번의 디스크 접근이 필요하기 때문이다. 이러한 경우 검색속도는 수배의 차이가 나게 된다.

9.2.2 관리 데이터베이스의 테이블 분할

앞서 설명한 바와 같이 검색서비스만을 위한 데이터베이스는 비교적 간단한 권고안으로 설명할 수 있다. 그러나 관리위주의 데이터베이스는 이와는 다른 특수한 상황에 직면하게 된다. 수시로 데이터베이스의 업데이트가 일어나는 경우에는 단일 테이블로 구성할 경우 단일 테이블의 크기가 너무 커서 데이터베이스 업데이트에 많은 시간이 소요된다. 업데이트가 수시로 발생하는 경우에는 테이블을 많이 분할할수록 유리하다.

아래에서는 수시로 데이터베이스에 새로운 항목이 추가되거나 수정, 삭제가 빈번한 데이터베이스를 대상으로 설명한다. 이러한 경우는 아래와 같이 데이터베이스 전체에 대해서 수정이 발생하는 신문사의 보도사진 데이터베이스와 게시판 서비스 등이 이에 해당한다고 볼 수 있다. 보도사진 데이터베이스에 대한 분석의 예를 살펴보자.

신문사의 보도사진 데이터베이스

- 현재 지난 10 년동안의 총 100 만 장의 사진에 대한 정보를 수록하고 있다.
- 각 사진에 대한 등록정보는 촬영자, 촬영일시, 촬영장소, 제목, 기사 본문 및 몇가지의 신문사에 특이한 플래그(flag) 정보들을 담고 있다.
  - ✓ 촬영자는 INDEX_BY_TOKEN 으로 색인
  - ✓ 촬영일시는 INDEX_AS_IS 로 색인
  - ✓ 촬영장소는 INDEX_BY_TOKEN 으로 색인
  - ✓ 제목은 INDEX_BY_MA 로 색인
  - ✓ 기사 본문은 INDEX_BY_MA 로 색인
  - ✓ 각종 플래그는 검색에 활용되는 경우에는 INDEX_AS_IS 로 그렇지 않은 경우에는 색인을 하지 않음.
- 매일 300 여장의 사진이 기자에 의해 직접 등록된다.

- 매일, 마감이 임박한 오후 3~4 시에 동시 사용자 20~30 명의 동시 수정이 몰리는 경향이 있다.
- 사진 기자는 먼저 사진과 제목 정도를 입력하고 이후에 급한 마감이 끝나는 대로 업데이트 작업을 수행하는 경향이 있다.
- 신문사 웹사이트 사용자나 신문 구독자의 요청에 따라 잘못된 정보의 수정이 수시로 발생한다.

이 데이터베이스는 사진을 위주로 하고 이 사진과 관계된 기사들이 입력되기 때문에 100 만건 정도이기 때문에 모든 데이터를 단일 테이블로 적재하였을 경우에는 수정/신규 사진 등록 등에서 많은 시간이 소비된다. 실례로 서버 PC 급의 기종에서 단일 사진의 수정/삭제/등록에 약 2 분이 소요되는 것이 관찰되었다. 이 데이터베이스의 구성을 단일 테이블에서 다중 테이블로 변경할 경우 수정 시간은 대폭 감한다. 실제로 이 DB 를 단일 테이블에서 20 여개의 테이블로 분할하였으며 이 경우 검색시간은 전체적인 검색시간은 약간 느려졌지만 – 사용자는 감지할 수 없을 정도였음 – 수정/삭제/등록 시간은 10 초 이내로 감소하였다. 또한 마지막 테이블은 신규 등록 사진을 위한 목적으로 사용함으로써 신규 등록 사진에 대한 정보의 수정 시간은 이보다 훨씬 감소시킬 수 있다. 그리고 일정 수 이상의 사진이 등록되면 새로운 테이블을 또다시 추가함으로써 신규 사진의 등록 서비스를 향상시킬 수 있다.

게시판과 같이 신규 게시물이 위주로 서비스되는 데이터베이스의 경우에도 유사한 구조로 테이블을 분할하여 서비스를 구성할 것을 권장한다[31].

[31] KRISTAL-III version 2.0 이후 버전에서는 이와 같은 단점을 개선하여 다수의 테이블로 분할하지 않고도 빠른 속도로 수정/등록/삭제할 수 있는 구조를 채택하기 위한 개선작업을 수행하고 있다.

9.3 검색모델 선택 및 인터페이스 설계

검색 모델의 선택은 사용자의 만족도에 큰 영향을 미친다. 다양한 환경에서 사용자의 검색 결과에 대한 만족도를 향상시키기 위해 KRISTAL-III 에서는 세가지 검색모델을 기본적으로 제공하고 있다. 또한 각 검색모델에서는 서비스제공자가 선택할 수 있는 옵션들을 여러가지 제공함으로써 해당 데이터베이스에 적합한 검색 기능을 구현할 수 있도록 하였다.

9.3.1 KRISTAL 의 검색모델

KRISTAL-III 가 제공하는 검색모델은 AND, OR, NOT, NEAR, WITHIN 등의 불리안 연산자를 지원하는 불리안 검색 모델(BOOLEAN MODEL)과 문서의 우선순위를 지원하는 벡터공간모델(VECTOR SPACE MODEL), 그리고 불리안 검색 모델에서 문서의 우선순위를 지원하는 벡터불리안 모델(VECTOR-BOOLEAN MODEL)의 세가지가 있다. 아래에서는 각각의 검색모델에 대해서 예제를 통하여 간략하게 살펴본다.

9.3.1.1 불리안모델 (Boolean Model)

불리안 검색모델은 전통적으로 도서관의 서지정보검색시스템에서 활용되어온 검색모델이다. 이 모델에서는 AND, OR, NOT, WITHIN, NEAR 등과 같은 불리안 연산자를 이용하여 불리안 검색식에 적합한 문서를 모두 사용자에게 제공하는 방식을 취한다.

불리안질의 예제 1) (TITLE: 정보 | Information) & (AUTHOR: Salton)

위의 예제는 도서의 제목에 “정보” 또는 “Information”이 포함된 것으로 저자가 “Salton”인 서지정보를 검색한다. 이처럼 불리안 검색식은 사용자가 미리 이 도서에 대한 정보를 알고 있다면 검색 결과에 대해서도 어느 정도 정확하게 예측할 수 있는 장점이 있다. 또한 Salton 이 “Information Retrieval”이라는 책을 저술했다는 것을 알고 있다면 위의 불리안 질의 예제 1 을 다음과 같이 보다 정확하게 줄 수도 있다.

불리안질의 예제 2) (TITLE: Information /w1 Retrieval) & (AUTHOR: Salton)

예제 2 에서는 제목에 “Information”과 “Retrieval”이라는 두 단어가 연이어 출현하고 저자가 Salton 인 결과만을 반환할 것이다.

위의 예제에서 보는 바와 같이 불리안 검색 모델은 사용자가 원하는 문서를 모두 검색해야 하는 재현율(Recall) 중심의 검색모델로서, 사용자의 의도를 검색에 정확하게 반영하여 결과를 제공할 수 있는 장점이 있다. 반면, 불리안 연산외에도 시스템에 따라서 복잡한 검색식을 사용하여야 하기 때문에 전통적인 도서관시스템에서는 시스템의 사용자가 전문검색가로 제한되기도 하였다. 또한 불리안 검색은 수식에 입각한 검색을 수행해야 하므로 다소 복잡한 연산이 요구되어 검색시간이 느리다는 단점이 있다. 현재 KRISTAL-III 시스템의 불리안 검색 모델은 다음에 설명할 벡터모델과는 속도면에서 수배 이상 느린 것으로 파악된다.

9.3.1.2 벡터모델 (Vector Model)

불리안 모델이 전통적인 도서관시스템의 검색모델이라면, 벡터공간모델은 웹이 발달하면서 널리 사용되기 시작한 모델이다. 웹의 확장으로 웹검색엔진은 당시까지는 다뤄보지 못했던 초 대용량의 데이터를 대상으로 검색하게 되었고, 불리안 검색모델은 속도의 측면과 사용자의 검색결과에 대한 만족도에서 웹검색엔진에 채택되기에는 무리가 있었다. 이의 대안으로 각광받기 시작한 것이 벡터공간모델로 사용자가 원하는 문서를 빠른 시간내에 검색해준다.

벡터공간모델은 사용자의 질의에 가장 적합한 문서를 상위에 제공하는 정확도(Precision) 중심의 검색모델이다. 즉 수많은 문서 중에서 사용자가 원하는 것으로 판단되는 문서를 최상위에 제공함으로써 사용자가 빠른 시간내에 원하는 정보를 획득하도록 도와주는 검색모델이라고 할 수 있다.

전통적인 도서관 시스템에서는 데이터의 양도 적었을 뿐만 아니라 데이터의 형태 또한 서지 정보에 불과했으므로 불리안 검색모델이 적합한 검색 모델이었다. 더욱이 제목, 저자, 출판 정보 등과 같은 서지정보에 포함된 정보의 양으로는 사용자의 질의와

서지 정보간의 적합성 정도를 판단하기는 어려웠다. 그러나 웹의 급격한 성장으로 다양한 형태의 원문정보가 인터넷 상에 존재하게 됨으로써 불리안 검색 모델에 의한 검색은 사용자의 욕구를 충족시키기 어렵게 되었다. 너무나 많은 양의 문서가 존재하고 이 문서들은 사용자가 사전에 인지하지 못한 상태의 문서들이기 때문에 불리안 검색으로 원하는 정보를 제공하기 위해서는 매우 복잡한 불리안 연산을 수행해야 한다. 웹의 확대는 검색시스템의 사용자를 전문검색가에서 일반 대중으로 확대시켰기 때문에, 대다수의 일반 대중에게 복잡한 불리안 검색식을 요구하는 것은 무리였다. 반면, 벡터공간모델은 복잡한 검색식을 요구하지 않으며 매우 빠른 시간에 검색을 수행할 수 있다. 다음 예제를 보자.

벡터질의 예 1) TEXT: 정보검색 Salton

위의 벡터질의 예 1 은 AND, OR, NOT, WITHIN 등과 같은 불리안 연산자를 요구하지 않는다. 단어의 나열이면 충분하다. 벡터공간모델로 이러한 검색을 수행하면 그 결과로는 Salton 과 관련된 정보검색 연구들에 대한 문서들을 최상위에 제공하게 된다. 대용량 데이터베이스를 대상으로 하는 검색에서는 원하는 정보를 빠른 시간내에 상위에서 찾을 수 있으므로 벡터공간모델이 적합한 검색 모델이 된다.

9.3.1.3 벡터불리안모델 (Vector-Boolean Model)

벡터불리안 모델은 불리안 연산자를 사용하면서도 사용자의 질의와 해당 문서간의 적합도를 판정하여 우선순위를 결정해주는 검색모델이다. 불리안검색모델과 벡터검색모델의 복합형 검색모델로서, 재현율(Recall)과 정확도(Precision)를 모두 고려하는 검색모델이라고 할 수 있다. 그러나 불리안 모델과 벡터 모델을 동시에 지원해야 하기 때문에 검색속도가 매우 느리다는 단점이 있다. 불리안 모델에 비해서도 경우에 따라서는 두 배이상 검색속도가 느린 단점이 있다. 특수한 경우가 아니면 벡터불리안 검색모델의 사용을 권장하지 않는다.

9.3.2 검색모델의 선택

웹검색엔진에서는 검색 질의의 길이가 평균적으로 2 단어 이내라는 통계가 있다. 이러한 통계는 검색모델의 선택에 있어서 매우 중요한 의미를 가진다. 즉 매우 짧은 질의만으로 사용자가 원하는 정보를 제공해야 한다는 것이다. 오늘날 대부분의 인터넷 사용자는 웹검색엔진을 기본적으로 사용하고 웹검색엔진 방식의 검색에 익숙해져 있기 때문에, 검색을 지원하는 정보서비스에 접속하는 대부분의 사용자는 웹검색엔진 방식의 서비스 형태를 기본적으로 바랄 것으로 예상할 수 있다. 앞서 언급한 바와 같이 웹검색엔진의 검색은 정확도(Precision)에 중점을 두고 있으며, KRISTAL 에서는 벡터모델을 통해 이러한 검색방식을 지원한다.

검색모델의 선택은 데이터베이스에 따라, 서비스의 목적에 따라, 색인방식에 따라서 다양하게 달라질 수 있다. 일반적으로 INDEX_BY_MA 색인방식을 사용하는 경우에는 벡터공간모델을 권장한다. 반면 INDEX_BY_CHAR, INDEX_BY_MIX_CHAR 등과 같은 낱자단위 색인에서는 불리안검색모델을 권장한다. 영어 문서를 INDEX_BY_TOKEN 으로 색인할 경우에는 벡터모델을, 그외에 이름, 기관명 등과 같은 특수한 섹션을 INDEX_BY_TOKEN 방식으로 색인하였을 경우에는 불리안모델을 권장한다. 데이터의 양이 작거나 재현율이 매우 중요한 서비스에서는 불리안 모델을 권장한다.

9.3.3 웹 인터페이스

도서관의 서지정보 검색시스템 등과 같은 초기의 정보서비스에 비해 현대의 정보서비스는 사용자 인터페이스 면에서 급격한 변혁을 겪어왔다. 초기 정보서비스의 경우 각종 명령어와 검색식을 사용자가 숙지하고 검색해야 했으므로 소수만이 정보에 접근가능했다. 그러나 웹검색엔진과 웹 인터페이스는 대다수의 일반 대중에게 직접 검색을 통해 원하는 정보에 접근가능하도록 하는 획기적인 변혁을 가져왔다. 몇 개의 단어만으로 원하는 정보를 획득할 수 있다는 것은 가히 혁명적인 정보 환경의 변화라 할 수 있을 것이다.

KRISTAL 을 사용하여 구축하는 정보서비스도 이와 유사한 인터페이스를 가지도록 권장한다. 일반 대중의 경우 문서의 구조나 데이터베이스의 구성 등에 대해서 미리 인

지하고 검색하는 경우가 드물기 때문에 웹검색엔진과 유사한 검색환경을 만들어주는 것이 좋을 것이다. 이를 위해 KRISTAL 에서는 문서의 구조에 상관없이, VirtaulSection 이나 UnionSection 을 이용하여 기본적으로 검색 대상이 되어야 하는 섹션을 지정함으로써 사용자가 시스템에 대한 깊은 이해없이도 쉽게 검색을 수행할 수 있도록 할 수 있다. 또한 웹의 장점은 다수의 인터페이스를 제공할 수 있다는 것이다. 고급 사용자를 위해서는 상세하게 검색식을 구성할 수 있는 상세검색 인터페이스를 별도로 제공하는 것도 사용자의 만족도를 높일 수 있는 한 방편이다.

부록. A 용어집

가변길이 크기가 가변적임.

결과셋 검색결과를 저장하고 저장된 곳의 위치를 대표하는 대표값.

고정길이 문자열 크기가 고정된 문자열.

다중테이블, 멀티테이블 같은 구조를 가진 문서들의 집합을 테이블이라고 하며, 다수의 테이블을 다중테이블 또는 멀티테이블이라 한다.

단위명사 명사가 단독으로 존재하는 단어

데몬(Daemon) 시스템의 운영에 관련된 작업을 백그라운드 상태에서 실행하는 프로그램

데이터베이스 하나의 데몬이 서비스하는 데이터들의 집합.

데이터베이스 스키마 데이터베이스를 관리하기 위해 kristal_dbadmin 이 사용하는 스키마를 일컫음

레코드 문서가 저장되는 물리적인 공간을 의미함.

복합명사 명사 여러 개가 결합된 형태의 단어

볼륨 데이터들이 저장되는 물리적인 공간을 의미.

불리언 검색 불(Bool) 연산자를 사용하여 설정된 조건(기준)에 맞는 항목을 찾는 데이터베이스 검색 방법

불용어(Stopword) 색인을 수행할 때 색인어로서 무의미한 단어

사전파일	파일 형태로 존재하는 사전
색인	특정 문서에서 그 문서를 표현하는 주요 단어(명사 등)를 추출하는 과정
색인기	특정 문서에서 그 문서를 표현하는 주요 단어(명사 등)를 추출하는 시스템
색인어	색인기를 통하여 추출된 색인 결과 단어
서버	서비스를 제공하는 시스템
섹션	문서의 특정부분으로 검색대상이 되는 최소단위.
스키마	자료의 구조 및 내용 그리고 이러한 자료들에 대한 논리적, 물리적 특성에 대한 정보들을 표현하는 데이터베이스의 논리적 구조
스테밍, 스태밍	영어 단어에 대해서 어근을 분리해 내는 과정
시그널	시스템의 상태에 따라 발생되는 신호
아규먼트	입출력 인자를 의미함
원시데이터	KRISTAL-20002 데이터베이스에 적재하기 전의 문서.
유니코드	세계 각 국의 언어를 통일된 방법으로 표현할 수 있게 제안된 국제적인 코드 규약의 이름
유닉스	미국 벨 연구소가 개발한 중형 전산기용 운영 체계
재현율	대상 문서 집합에서 적합한 전체 문서 중 검색 시스템이 찾아낸 적합 문서의 비율
적재 스키마	데이터베이스 내에 데이터를 적재하기 위해서

	kristal_dbadmin 이 사용하는 스키마를 일컫음
정보검색관리 시스템	정보 검색시스템에 DBMS 의 데이터관리 기능을 추가한 시스템
정형섹션	하나의 문서는 다수의 섹션으로 구분되고 섹션들 중에서 고정길이를 가진 섹션을 의미함.
정확도	검색 시스템이 제공한 결과 문서 중 적합한 문서의 비율
주요키(Primary Key)	테이블 내에서 각 문서를 대표할 수 있는 유일한 키.
카탈로그	데이터베이스의 특징 정보들이 저장되어 있는 테이블
클라이언트	서버에 요구하고 정보나 자원을 이용하는 하드웨어나 소프트웨어
클러스터	같은 스키마(문서구조)를 가진 테이블들의 집합.
태그	데이터의 형태나 종류를 식별하기 위해 항목이나 레코드에 붙여지는 꼬리표
터미널	컴퓨터의 관점에서 최종 단말기 위치에 연결되어 동작되는 장치
파라미터	
패키지	특정분야의 업무 처리에 사용되는 각종 응용 소프트웨어를 하나로 묶어 상품화 한것
프로세스	컴퓨터에서 프로그램을 실행하여 작업을 처리하는 단위
필드	데이터 레코드를 구성하는 각 요소
API	애플리케이션이 시스템 소프트웨어에 어떤 처리를 위해

	서 호출할 수 있는 서브루틴 또는 함수의 집합
BLOB (Binary Large Object)	사운드, 이미지, 프로시저, 콤플렉스 구조 등을 나타내기 위한 2 진 정보를 포함한 객체
DBMS	DataBase Management System.
KRISTAL-III	2002-2003 기간 중 KISTI 에서 개발한 정보검색관리 시스템의 이름
UTF-8	유니코드 표현방법의 한 형태, 아스키 코드 처리를 쉽게 하기 위해 아스키 코드는 1 바이트, 나머지 코드는 2 바이트 혹은 3 바이트로 구성하여 인코딩됨. UCS-2 나 UCS-4 로 변환할 수 있음
XML	데이터의 구조적 정보를 표현할 수 있는 마크 업 언어

색 인

얼

얼

L

M

N

P

R

S

T

U

V

얼

X