User Manual

오디세우스/OOSQL

Version 5.0

Manual Release 2

2016 년 8 월

Advanced Information Technology Research Center (AITrc)

KAIST

목 차

1. 오디세우스/OOSQL 디렉토리 구조

디렉토리/파일	설명
OOSQL	Linux 운영체제용 오브젝트 파일과 헤더 파일들
document	문서
source	소스 코드
example	예제 파일

소스 코드는 C 언어 및 C++ 언어로 작성되었으며 약 355,000 라인의 규모를 가진다.

2. 오디세우스/OOSQL 컴파일 방법

Linux 운영체제에서 소스 코드를 컴파일하기 위한 환경은 다음과 같다.

- 32bit
  - 플랫폼: Linux 2.6
  - 사용 컴파일러: gcc 4.1.2 Compiler
- 64bit
  - 플랫폼: Linux 2.6
  - 사용 컴파일러: gcc 4.4.7 Compiler

오디세우스/OOSQL 소스 코드 디렉토리에서 다음 과정을 거쳐서 컴파일 한다.

1) 오디세우스/COSMOS 바이너리와 헤더 파일을 적당한 디렉토리에 설치한다. 오디세우스/COSMOS 는 32bit, 64bit 대용량화, 64bit 비대용량화 버전 [1]이 존재한다. 각 버전에 대한 설명은 오디세우스/COSMOS User Manual 을 참고한다.
(오디세우스/OOSQL 의 컴파일을 위해서는 Linux 운영체제 용 오디세우스/COSMOS 의 바이너리와 헤더 파일이 필요하다.)

2) 압축된 오디세우스/OOSQL 소스 코드 파일을 적당한 디렉토리에 압축 해제한다. 압축 해제하였을 때, ./LOM, ./GEOM, ./OOSQL 등 3 개의 서브 디렉토리와 setup 파일, 그리고 Make.sh 파일이 생성되는지를 확인한다.

[1] 2016 년 8 월 현재, 오디세우스/COSMOS 64bit 비대용량화는 32bit-64bit type mismatch 로 인한 버그가 일부 존재함

3) 오디세우스/OOSQL 소스코드를 컴파일하기 위한 각종 설정을 저장하고 있는 setup 파일을 수정한다. setup 파일에서 설정하는 환경변수의 종류와 의미는 아래와 같다. LOM, GEOM[2], OOSQL 은 소스 코드 파일을 각 서브 디렉토리에서 컴파일하고 바이너리 파일을 생성하므로 소스코드의 위치와 바이너리의 위치가 같다. 따라서 O_ROOT 와 O_KAOSS 환경변수의 설정에 유의하고 다른 환경변수는 default 값을 사용하는 것을 추천한다. O_OOSQL_EXPORT 는 오디세우스/OOSQL 의 컴파일 후 바이너리 파일을 복사하는 경로를 지정한다. SWIG 와 BISON, 그리고 PYTHON 은 응용프로그램의 위치를 각각 지정한다.

(오디세우스/OOSQL 의 컴파일을 위해서는 Linux 운영체제 용 SWIG 1.3.29 와 Bison 1.28, 그리고 Python 2.4 의 바이너리와 헤더 파일이 필요하다.)

환경변수	Default 값	의미
O_ROOT	직접 설정함	소스 코드의 루트 디렉토리
O_SERVER	$O_ROOT	오디세우스/OOSQL 소스 코드의 루트 디렉토리
O_KAOSS	$O_ROOT/COSMOS	오디세우스/COSMOS 바이너리의 위치
O_LOM_COMMON	$O_ROOT/LOM	LOM 소스코드의 위치
O_LOM_SERVER	$O_ROOT/LOM	LOM 바이너리의 위치
O_GEOM	$O_ROOT/GEOM	GEOM 소스코드/바이너리의 위치
O_OOSQL_COMMON	$O_ROOT/OOSQL	OOSQL 소스코드의 위치
O_OOSQL_SERVER	$O_ROOT/OOSQL	OOSQL 바이너리의 위치
O_COMMON	$O_ROOT/OOSQL	OOSQL 유틸리티의 위치
O_DLLBROKER	$O_LOM_SERVER/RPCdll/dllbroker	LOM 통신 모듈(dllbroker)의 위치
O_DLLSERVER	$O_LOM_SERVER/RPCdll/dllserver	LOM 통신 모듈(dllserver)의 위치
O_OOSQL_EXPORT	직접 설정함	OOSQL 컴파일 완료 시 바이너리 및 헤더가 복사되는 위치
SWIG	직접 설정함	SWIG 바이너리의 위치

[2] GIS module 인 GEOM 은 2016 년 8 월 현재 multi threading 이 안되어 있으므로 multi-thread execution 에서는 사용하지 말아야 한다.

BISON	직접 설정함	Bison 바이너리의 위치
PYTHON_HEADER	직접 설정함	Python 헤더의 위치

4) LOM 서브 디렉토리와 OOSQL 서브 디렉토리의 Makefile 을 열어서 오디세우스/COSMOS 의 버전(32bit, 64bit 대용량화, 64bit 비대용량화)에 따라 SERVERFLAG 를 설정한다. 오디세우스/COSMOS 32bit 버전 또는 오디세우스/OOSQL 64bit 비대용량화 버전을 사용하는 경우에는 DSUPPORT_LARGE_DATABASE2 플래그를 해제한다. 오디세우스/COSMOS 64bit 대용량화 버전을 사용하는 경우에는 DSUPPORT_LARGE_DATABASE2 플래그를 설정한다.

5) Make.sh 스크립트 파일을 실행하여 오디세우스/OOSQL 을 컴파일한다. Make.sh 파일은 LOM, GEOM, OOSQL 을 순차적으로 컴파일하는 스크립트 파일이다. 컴파일 한 후에 OOSQL 서브 디렉토리에 오디세우스/OOSQL 오브젝트 파일(liboosql.so)이 생성되었는지 확인한다. 오브젝트 파일이 생성되어 있지 않다면 컴파일이 정상적으로 이루어지지 않은 것이므로 컴파일 오류를 확인하여 문제를 해결한 후에 다시 컴파일 해야 한다.

본 문서에서는 오디세우스/OOSQ 의 컴파일 및 사용 방법에 관한 내용을 중점적으로 설명한다. 오디세우스/OOSQL 의 설계 및 구현, 모듈 별 함수 설명에 대한 내용은 다른 문서들을 참고하도록 한다. 아래의 리스트는 다른 문서들과 각 문서의 내용을 간략히 기술한 것이다.

- User Manual: 컴파일 및 사용 방법
- Functional Specification: 모듈 별 함수 설명 (2016 년 8 월 현재 release 되지 않음)
- Reference Manual: API function 들의 사용 방법
- External Document: 오디세우스/OOSQL 의 개념 (2016 년 8 월 현재 release 되지 않음)

3. 데이터베이스 스키마 작성

데이터베이스 스키마 작성은 검색 시스템 성능을 결정하는 중요한 부분으로 전문가와 상의하도록 한다.

목표

데이터베이스 시스템에서 사용하는 데이터를 데이터베이스에 표현하기 위해 스키마를 작성한다.

내용

설계된 사용자 인터페이스에 알맞은 데이터베이스 스키마를 SQL 을 사용하여 작성한다.

필수 사항

1) 오디세우스를 이용한 데이터베이스 스키마를 설계하는 경우, 숫자, 문자열, 텍스트 필드를 적절히 사용하도록 구성하여야 한다. 이 중 텍스트 필드는 반드시 인덱스를 만들지만, 숫자와 문자열 필드에 대해서는 선택적으로 인덱스를 구성할 수 있으므로 적절한 인덱스를 구성해야 한다.

2) 숫자와 문자열 필드에 대한 인덱스는 선택률이 낮은 것에만 만들어야 한다. 선택률이 낮다는 것은 주민등록번호, 등록번호, 이름 같이 동일한 값을 가진 데이터들이 많이 존재하지 않은 것을 의미하며, 선택률이 높은 경우로는 도서에서 발행국가, 언어, 출판년도 등이 있을 수 있다. 선택률이 높은 필드에 인덱스를 생성하게 되면 검색시 이 인덱스를 참조하게 되어 검색이 늦게 된다.
(**선택률** : 질의의 결과로 나온 문서의 수 / 전체 데이터베이스의 문서 수)

3) 선택률이 높은 필드에 대해서 인덱스를 만들고자 한다면 텍스트 인덱스를 사용해야 한다. 예를 들어 도서정보 검색 시스템을 구현하고자 할 때 발행연도의 경우 그 선택률은 높지만 검색에 사용해야 할 경우가 있다. 이 때 일반 인덱스를 이용하면 높은 선택률 때문에 검색 속도가 상당히 느리게 된다. 하지만 이때 텍스트 인덱스를 이용하면 빠른 검색 속도를 얻을 수 있다. 숫자 데이터를 텍스트 타입을 이용해서 데이터베이스를 구축한 경우라도 SELECT 문에서 MATCH 내에 BETWEEN 을 이용해서 범위 검색도 가능하다. 예를 들어 발행연도가 2000 년부터 2010 년 사이의 도서를 검색하고 싶은 경우 아래와 같은 질의를 입력하면 된다.
SELECT * FROM BOOK WHERE MATCH (YEAR, BETWEEN (“2000”, “2010”)) > 0

4) 데이터베이스 스키마를 작성할 때 주어진 데이터들의 컬럼을 어떤 타입으로 선언해야 하고 어떤 인덱스를 구성할지를 결정하는 것은 전체 검색 시스템의 성능을 좌우할 만큼 중요한 문제이다. 이러한 문제는 기존의 데이터베이스 관리 시스템 혹은 IR 검색 시스템에서 사용하던 스키마를 오디세우스로 옮길 때 컬럼 타입과 인덱스 타입을 정하는 것에서도 마찬가지이다.
도서검색시스템의 스키마를 작성 할 때 주민등록번호, 발행연도, 학문 분야와 같은 컬럼들을 텍스트 타입으로 선언해서 텍스트 인덱스를 구성해야 하는지 아니면 VARCHAR, 혹은 CHAR 로 선언해서 B-Tree 인덱스를 구성해야 하는지를 판단할지 결정하는 문제를 그 예로 들 수 있다.

이러한 선택이 필요할 때 다음과 같은 rule 을 이용해서 데이터베이스 스키마를 작성하는 원칙으로 삼을 수 있다.

Rule 1) 주어진 Equality(=) 연산에 대한 선택률이 낮은 컬럼에 대해서는 원래 타입대로 사용해서 B-Tree 인덱스를 구성한다. (예: 주민등록번호 등과 같이 주어진 연산에 대해서 선택률이 낮은 컬럼은 원래 데이터베이스에서 사용하던 것과 동일한 타입을 사용해서 B-Tree 인덱스를 구성한다.)

Rule 2) 주어진 Equality(=) 연산에 대한 선택률이 중간인 컬럼에 대해서는 텍스트 타입을 사용해서 인덱스를 구성 한다. (예: 발행연도 등과 같이 주어진 연산에 대해서 중간 정도의 선택률을 가진 컬럼은 TEXT 타입을 사용해서 인덱스를 구성한다.)

Rule 3) 주어진 Equality(=) 연산에 대한 선택률이 아주 높은 컬럼에 대해서는 데이터베이스를 분리 한다. (예: 학문분야(물리, 화학 등등), 도서형태(저널, 단행본 등등) 같은 컬럼은 각각의 타입에 대해서 테이블을 분리해서 데이터를 저장한다.)

Rule 4) Rule 1 과 Rule 3 에 해당하더라도 부분 일치 검색이 필요한 컬럼의 경우에는 TEXT 타입을 사용해서 인덱스를 구성 한다. 선택률에 관계 없이 부분 일치 검색(입력된 키워드를 포함하는 문서를 찾는 검색)이 요구될 때는 TEXT 타입으로 인덱스를 구성한다. (예: 다수의 저자의 이름을 저장하고 있는 컬럼에서 한 사람의 이름을 이용해서 원하는 문서를 찾고자 할 때는 TEXT 타입을 사용해서 데이터베이스를 구축한다.)

5) 검색 시스템에서 사용하는 테이블은 가능하면 하나의 테이블로 구성하는 것이 좋다. 검색에 두 개 이상의 테이블이 연관이 있게 되면 검색에서 조인이 생기게 된다. 일반적으로 조인 처리에는 많은 시간이 소모되므로 가급적이면 검색에는 하나의 테이블로 모든 검색이 이루어지도록 스키마를 정의한다.

6) 모든 경우에 검색 시스템에 사용하는 테이블을 하나의 테이블로 만들 수 있는 것은 아니다. 1:1 매핑에서는 하나의 테이블로 만드는 것이 큰 문제가 없지만 1:n, m:n 매핑에서는 조인을 사용하지 않고 검색 시스템을 만드는 것이 문제가 있다. 예를 들어 도서검색에서 하나의 도서는 총서명, 보조서명 등 여러 개의 서명을 가질 수 있다. 이것을 하나의 테이블로 만들게 되면 테이블의 필드가 늘어나야 한다. 즉 1:n 의 매핑인 경우, n 개의 필드를 추가해야 되며, 검색에 있어서도 n 개의 OR 연산

이 필요하다. 이러한 경우에는 테이블을 2 개로 나누어야 하며 검색에 있어서도 조인이 필요하게 된다. 이러한 형태로 스키마를 구성한다고 항상 성능이 떨어지는 것은 아니다. 이렇게 스키마를 구현한다 하더라도 검색으로 얻어지는 조인 결과의 개수가 크지 않는다는 조건을 만족하면 오히려 n 번의 OR 연산보다 조인으로 처리하는 것이 효과적이다.

참고 사항

1) 스키마가 만들어진 후에 각 테이블을 저장할 볼륨을 부여해야 한다. 이때 하나의 볼륨에 여러 개의 테이블을 넣을지 각기 분리된 볼륨에 하나의 테이블을 넣을지 결정해야 한다. 이 두 방법 각각의 장단점을 살펴보면 다음과 같다.

여러 개의 테이블을 하나의 볼륨 안에 저장하는 방법

- 장점: 각 테이블이 하나의 볼륨을 공유하기 때문에 데이터가 증가하더라도 디스크 공간을 서로 효율적으로 사용할 수 있다.
- 단점: 이미 구성된 데이터베이스에 새로운 데이터가 입력될 경우, 새로운 데이터가 저장되는 위치가 그것이 속하는 테이블의 데이터들이 저장된 위치와 떨어져 있을 수 있어서 clustering 이 유지되지 않을 수 있다. 또한, 어떤 한 테이블에만 update 가 일어나더라도 볼륨 전체에 lock 이 걸리게 되므로 다른 테이블에 필요 없이 lock 이 걸리게 되는 문제점이 있다.

하나의 테이블을 각각 하나의 볼륨 안에 저장하는 방법

- 장점: 테이블들이 각각의 볼륨 안에 존재하므로, 새로운 데이터가 입력되었을 때도 모든 테이블을 하나의 볼륨 안에 저장할 때 보다 clustering 이 더 잘 유지되어 검색이 빨라진다.
- 단점: 모든 테이블을 하나의 볼륨 안에 저장하는 방법에 비해 디스크 공간을 효율적으로 사용하지 못한다. 또한, 서로 다른 볼륨에 저장되어 있는 테이블 간에는 조인이 되지 않는 단점이 있다.

따라서 각 테이블을 저장할 볼륨은 디스크 공간 활용, clustering, locking, 테이블 간의 연관성 등을 고려하여 결정해야 한다.

4. 오디세우스/OOSQL 설치

오디세우스에 대한 오작동 문의의 대부분이 잘못된 설치로 인한 것이다. 따라서 아래의 사항을 준수하여 시행착오를 줄여야 한다. 설치과정은 Linux 에 능통한 기술자의 도움을 받아 수행하는 것이 효과적이다.

목표

오디세우스 데이터베이스 관리 시스템의 엔진인 오디세우스/OOSQL 을 설치하는 과정이다.

내용

적절한 디렉토리에 오디세우스 데이터베이스 관리 시스템의 엔진인 오디세우스 라이브러리와 관련 유틸리티를 설치한다. 설치를 위해서는 본 매뉴얼의 제 2 장에서 소개된 오디세우스/OOSQL 컴파일 방법을 참고하거나, 또는 제공되는 오디세우스/OOSQL 바이너리 파일을 사용한다. 설치되는 내용은 OOSQL 라이브러리, 프로그래밍을 위한 Header 파일, 관련 유틸리티, 텍스트에서 키워드를 추출하는 키워드 추출기, 예제 프로그램이 설치된다.

필수 사항

1) 오디세우스를 설치하게 되면 라이브러리, include 파일, 키워드 추출기, 유틸리티, sample 프로그램 등이 설치된다. 설치한 후에는 디렉토리 또는 파일을 필요한 사용자가 액세스할 수 있도록 권한을 조정해야 한다.

2) 오디세우스 설치과정에서 setup 이라는 파일이 생성되는데 여기에는 오디세우스가 수행하는데 필요한 환경 변수들이 있다. 이들 환경 변수를 설치한 시스템에 맞게 수정해야 한다. 오디세우스의 정상적인 동작을 위해서는 이러한 환경 변수들이 정확히 설정되어 있어야 한다. 사용되는 환경 변수는

   ODYS_TEMP_PATH: 오디세우스가 수행 시 사용하는 임시 파일을 위한 디렉토리
   ODYS_OODB: 오디세우스 데이터베이스의 위치를 지정하는 디렉토리
   IR_SYSTEM_PATH: 키워드 추출기가 있는 디렉토리
   COSMOS_LOG_VOLUME: 잘못된 연산으로 데이터베이스의 파괴를 막기 위한 로그볼륨의 위치
   COSMOS_COHERENCY_VOLUME: 다중 서버 환경을 위한 coherency 볼륨의 위치

   등이 있다. 따라서 오디세우스를 설치한 후에는 이러한 환경 변수를 정확히 수정을 해 주어야 한다.

   그 외 오디세우스의 라이브러리는 수행 시 찾을 수 있게 LD_LIBRARY_PATH 에 오디세우스의 라이브러리가 있는 디렉토리를 추가하여야 한다.

3) 오디세우스 설치 시 제공되는 setup 의 환경변수를 수정하는 것만으로는 시스템 수행에 필요한 환경이 모두 정해지는 것은 아니다. Linux 환경에서는 shell 상에서 “source setup” 을 수행해서 setup 파일의 내용을 시스템에 반영해야 한다. 참고로 setup 파일은 운영체제의 tcsh 쉘을 기준으로 작성되었다. 따라서 자신이 현재 tcsh 을 사용하고 있는 경우 무방하지만 그렇지 않은 경우 tcsh 로 현재 shell 을 변경한다. 이 작업을 login 할 때마다 매번 하는 것을 방지하기 위해서는 로그인 시에 자동으로 이루어지도록 .login 이나 .cshrc 등의 파일에 첨부 하는 것도 좋은 방법이다.

   이렇게 로그인 시에 자동으로 수행되게 하지 않았을 경우는 유틸리티 프로그램이나 script 를 돌리기 전에 “source setup”을 수행시켜 주거나 script 첫 부분에 환경변수를 설정하여야만 이 명령을 넣어주어야 시스템이 올바르게 동작할 수 있다. 또한, 프로그램에 따라 다른 환경변수를 사용하기 위해서는 각 프로그램을 script 로 만들고 script 위에 필요한 환경변수를 정하게 하고 수행해야 한다.

   오디세우스의 데이터베이스에 관련된 작업은 모두 script 로 만들어 수행하는 것이 타이핑 에러를 막을 수 있고, 정확한 수행이 이루어지는데 도움이 되므로 반드시 script 로 만들어 수행하여야 한다.

4) 오디세우스를 설치한 이후에는 오디세우스가 수행되면서 사용할 로그 볼륨을 format 해야 한다. OOSQL_FormatLogVolume 이라는 명령어를 사용한다. 명령어의 매개변수로 로그 볼륨을 저장할 device 명에는 적절한 파일 경로와 로그 볼륨의 크기를 입력한다. 로그 볼륨의 크기는 한 트랜잭션에서 최대로 update 하는 데이터의 크기보다 조금 크게 정해야 한다.

   로그 볼륨을 생성하였으면 COSMOS_LOG_VOLUME 이라는 환경 변수에 만들어진 로그 볼륨의 device 이름을 등록하여 다른 오디세우스 프로그램들이 수행될 때에 로그 볼륨을 참조할 수 있게 한다. 로그 볼륨이 만들어 있지 않거나 환경변수에 로그 볼륨 위치를 명시하지 않으면 프로그램 수행 중 에러가 발생한다.

5) 다중 서버 환경에서 자료의 갱신, 삽입, 삭제가 발생하는 경우에는 반드시 coherency 볼륨을 사용해야 한다. OOSQL_FormatCoherencyVolume 은 다중 서버 환경을 위한 coherency 볼륨을 초기화하는 유틸리티이다. 다중 서버 환경에서 한 프로세스 내의 버퍼 내용이 변경되면 이러한 변경 내용이 coherency 볼륨에 기록되고 다른 프로세스는 이를 참조하여 버퍼의 내용을 일치시킨다. OOSQL 응용프로그램이 생성된 coherency 볼륨을 사용하기 위해서는 $COSMOS_COHERENCY_VOLUME 이라는 환경변수를 coherency 볼륨이 있는 위치를 가리키도록 설정해야 한다.

6) 환경 변수를 설정한 후에 이것이 올바르게 동작하는지를 알기 위해서 확인을 해주어야 한다. 먼저 shell 상에서 “setenv”를 실행시켜서 설정한 환경변수 들이 올바

르게 정의되어 있는지 확인한다. 이것이 확인이 되면 다음에는 라이브러리들이 올바르게 연결 되어있는지를 확인해야 한다. LD_LIBRARY_PATH 환경 변수에 정의된 디렉토리가 잘못되거나 순서가 잘못될 경우 전혀 엉뚱한 라이브러리가 프로그램에 연결 될 수 있다. 따라서 "ldd[3] <실행 파일 이름>"을 수행해서 필요한 라이브러리가 모두 연결 되어 있는지 그리고 올바르게 연결되어 있는지 확인을 해야 한다. 특히 이 과정은 script 를 사용하여 프로그램을 수행할 때 빈번히 일어나는 실수로 프로그램 수행 시 정확한 환경변수가 선언되어 있는지 확인할 필요가 있다.

참고 사항

1) 오디세우스의 수행에 필요한 라이브러리를 하나의 디렉토리에 설치하지 않고 여러 디렉토리에 설치하는 경우가 있다. liboosql.so 를 /usr/lib 에도 설치하고 ~/OOSQL/lib 에도 설치하는 경우를 그 예로 들 수 있다. 이렇게 중복해서 설치할 경우는 라이브러리를 업데이트 할 때 중복된 모든 라이브러리를 변경해야 하는 번거로움이 따른다. 따라서 가능하면 오디세우스 관련 라이브러리와 실행 파일을 한군데에만 설치하도록 한다. 부득이하게 중복 설치 해야 할 경우는 ldd 와 which[4]명령을 이용해서 올바른 라이브러리와 실행 파일이 연결되어 있다는 것을 확인해야 한다.

5. 데이터베이스 볼륨 구축

이 과정은 검색 시스템에서 제공할 데이터를 데이터베이스에 입력하는 과정으로 데이터베이스 생성, 스키마 생성, 데이터 로딩, 인덱스 생성, 데이터베이스 테스트로 나뉘어진다. 비교적 기계적으로 이루어지는 작업이나, 프로그램 수행시간이 긴 작업이다. 따라서 한번의 착오로 많은 시간을 소비할 수 있으므로 아래의 사항에 유의하여 착오를 최소화하여야 한다.

데이터베이스 볼륨 구축에 사용되는 유틸리티들의 정확한 사용법은 '오디세우스/OOSQL Reference Manual'에 나와 있다. 이를 숙독하고 작업을 진행하여야 한다.

5.1. 데이터베이스 생성

목표

데이터베이스를 저장하기 위한 저장 공간을 할당한다.

[3] Linux 환경에서 제공하는 유틸리티로 실행 파일이 사용하는 dynamic library 의 연결 상태를 출력한다.

[4] Linux 환경에서 제공하는 유틸리티로 실행 파일이 어디에 있는 지를 알려준다.

내용

오디세우스에서 제공하는 유틸리티를 사용하여 데이터베이스를 위한 저장 공간을 할당한다.

필수 사항

1) 데이터베이스의 생성에는 오디세우스에서 제공하는 OOSQL_CreateDB 유틸리티를 사용한다. 이 유틸리티의 매개변수(parameter)로 device 이름과 할당할 페이지 개수를 주도록 되어 있다. 이때 존재하지 않는 device 를 지정하거나 있더라도 권한(permission)이 없는 device 를 지정하게 되면 error 가 발생한다. 따라서 적절한 device 를 지정하도록 한다.

2) 데이터베이스를 저장하는 디바이스의 종류에 따라 데이터베이스의 성능이 달라진다. 디바이스는 raw device 를 사용하는 것이 좋다. raw device 를 사용하지 않을 경우에는, OS 에서 수행하는 불필요한 버퍼링으로 인해 처리 시간과 메모리가 낭비되는 문제가 발생한다. 특히 버퍼링을 위해 사용되는 메모리의 크기는 디바이스의 크기(수백 MB ~ 수 GB) 만큼 커질 수도 있기 때문에 시스템에 과부하를 줄 수도 있다.

3) 할당할 데이터베이스의 크기에 있어 할당한 페이지 개수가 너무 적으면 데이터가 입력되면서 할당한 페이지 개수를 넘어가게 되어 error 가 발생한다. 따라서 입력할 데이터의 양과 앞으로 늘어날 데이터를 고려하여 충분한 크기로 잡아야 한다. 할당한 페이지 개수는 크기는 입력할 데이터와 앞으로 입력이 예상되는 크기의 10 배 정도 (offset 을 저장하지 않으면 약 6 배)로 잡으면 데이터 입력 및 인덱스를 위한 충분한 공간을 확보하므로 큰 문제가 발생하지 않는다.

4) Raw device 를 이용해서 볼륨을 구축하고자 할 때 실제 raw device 의 용량보다 더 많은 용량의 공간을 할당하려고 하면 “invalid file format”등의 에러가 발생할 수 있다. 이 때는 OOSQL_CreateDB 의 매개변수로 설정한 number_of_page 값을 확인해 봐야 한다. 일반적으로 페이지의 크기는 4KByte 이므로 볼륨의 크기는 4KByte * number_of_page 로 계산 되어진다. number_of_page 값을 너무 크게 잡아서 raw device 의 크기를 넘어가면 위와 같은 에러가 발생 할 수 있다. Raw device 의 크기는 disk partition 시에 결정된다. Raw device 의 최대 크기는 32bit 버전에서 2GB, 64 bit 버전에서 8 EB 이다.

5) 만일 데이터베이스를 잘못 만들었을 경우에는 OOSQL_DestroyDB 를 하고 다시 OOSQL_CreateDB 를 수행한다.

참고 사항

1) 만들어진 데이터베이스에 있는 모든 데이터와 테이블을 삭제하려면 OOSQL_InitDB 를 수행한다. 따라서 이미 만들어진 데이터베이스의 내용을 지우고 다시 데이터를 넣으려면 처음에 데이터베이스를 만들 때에 사용한 OOSQL_CreateDB 대신에 OOSQL_InitDB 를 수행하고 그 다음 단계들을 수행하면 된다.

2) 볼륨을 구성할 때 raw device 는 하나 당 2GB 짜리를 최대 4000 개까지 붙일 수 있다. 따라서 확장성을 위해서 raw device 를 이용해서 볼륨을 구성할 때 처음부터 2GB 짜리를 사용해야 데이터가 늘어나도 최대 8TB 가지 커버할 수 있다. 그렇다고 확장성을 위해서 처음부터 필요한 모든 raw device 를 붙일 필요는 없다. 저장 공간이 모자라는 경우, OOSQL 의 유틸리티인 OOSQL_AddDevice 를 이용해서 필요할 때마다 볼륨에 raw device 를 추가 시킬 수 있다. 자세한 사항은 OOSQL Reference Manual 을 참조한다.

5.2. 스키마 생성

목표

데이터베이스에 설계된 스키마를 생성한다.

내용

설계된 스키마를 SQL 로 바꾸어 수행하여 데이터베이스에 스키마를 생성시킨다. 그리고, 텍스트 검색을 위해 키워드 추출기를 등록한다.

필수 사항

1) 스키마 생성에서는 검색 시스템에서 사용될 테이블을 정의하는 과정으로 스키마를 데이터베이스에 생성하는 방법은 여러 가지가 있을 수 있다. i) isql 을 사용하여 interactive 하게 작성하는 방법이 있을 수 있고, ii) 파일에 스키마를 적어 놓고, isql 혹은 오디세우스/Web-PHP 툴을 이용하여 생성하는 방법도 있다. 또한 iii) OOSQL API 를 사용하여 C 프로그램을 만들고 수행하는 방법도 있을 수 있다. 구현에서는 가능한 ii)번 방식을 사용하여 구현하는 것이 좋다. 스키마가 완전히 고정되었다면 iii)번 방법도 가능하다. i)번 방법은 테스트를 위해 임시로 스키마를 만들 때를 제외하고는 사용하지 않도록 한다.

2) 스키마를 정의하는데 있어 주의할 데이터 형태로는 char(10), varchar(10)이 있다. char(10)는 2 글자만 입력해도 뒤 부분은 모두 null 로 채워 항상 필드의 크기는 10 으로 유지된다. 반면에 varchar(10)은 2 글자만 입력하면 2 글자를 입력하고 뒤에 1 byte 만 null 로 채워져 크기가 3 이 된다.

3) 테이블의 텍스트 필드에 사용할 키워드 추출기에 대한 정보를 등록한다. 오디세우스에서는 텍스트 데이터를 입력하기 위해 모든 텍스트 타입에서 사용할 수 있는 기본(default) 키워드 추출기를 등록할 수 있다. 그러나 경우에 따라 다른 형태의 키워드 추출 방식이 필요할 수 있으므로 특정 테이블에 특정 애트리뷰트에 대해 각기 다른 키워드 추출기를 사용할 수 있다.

   특정 애트리뷰트에 대해 키워드 추출기를 등록하기 위해서는 먼저 키워드 추출기를 만들고, 데이터베이스에 특정 키워드 추출기를 등록하기 위한 인터페이스를 작성해야 한다.

   키워드 추출기를 만드는 예제는 OOSQL/Example/null_keyword_extractor 프로그램을 참조하면 된다. 이렇게 만들어진 키워드 추출기를 OOSQL 유틸리티를 사용하여 데이터베이스에 연결 시킨다. 유틸리티로는 InstallDefaultKeywordExtractor, InstallKeywordExtractor, SetKeywordExtractor 가 제공된다.

4) 오디세우스의 키워드 추출기는 형태소 분석 없이 단어를 분리하는 키워드 추출기를 제공한다. 키워드 추출기는 OOSQL/Example/null_keyword_extractor 에 소스 형태로 제공되고, 바이너리는 OOSQL/bin/NullKeywordExtractor.so 로 제공된다.

5) 키워드 추출기에서는 모든 영문을 소문자로 바꾸어서 키워드를 뽑아 내도록 되어 있다. 그래야 질의 처리기에서 대소문자 구별 없이 원하는 키워드를 가진 문서를 검색할 수 있다. 따라서 실제 검색 시스템을 구축할 때는 입력된 키워드가 인덱스에 대문자인지 소문자인지에 대한 고려는 필요 없다.

   하지만 필요에 따라 키워드 추출기를 새로 작성해야 할 때는 대소문자에 대한 고려가 필요하다. 즉, 사용자 정의 키워드 추출기를 작성할 때, 뽑힌 키워드를 모두 소문자로 변환하는 과정을 추가하지 않으면 그 키워드 추출기를 이용해서 구축된 텍스트 인덱스는 올바로 동작하지 않는다. 예를 들어서 데이터에는 그 키워드가 존재하는데 텍스트 인덱스는 대문자 형태로 된 키워드가 존재하는데 소문자의 키워드로 검색하여 원하는 문서를 찾지 못하는 경우가 발생 할 수 있다.

5.3. 데이터 로딩

목표

검색 시스템에서 제공할 데이터를 데이터베이스에 로딩한다.

내용

설계된 스키마가 저장되어있는 데이터베이스에 검색에 사용될 데이터를 로딩한다. 데이터베이스에는 반드시 스키마가 생성되어 있어야 한다.

필수 사항

1) 데이터를 로딩하는 방법은 OOSQL 에 포함되어 있는 OOSQL_LoadDB 를 사용한다.

2) OOSQL_LoadDB 형식은 맨 위에 입력하려는 테이블과 각 애트리뷰트를 명시하고 각 필드에 대해 실제 데이터를 기술한다. 주의할 것은 char 또는 varchar 타입과 text 타입의 데이터를 구분하기 위해 ‘과 “이 사용된다. 따라서 실제 데이터 안에 이러한 문자가 있을 때는 \’, \”로 대치해야 한다. 예를 들어 ‘한국과 ‘세계’ 동향’이라고 하면 ‘한국과 ‘가 하나의 데이터로 취급되어 에러가 발생한다. 따라서 이것을 입력하기 위해서는 ‘한국과 \‘세계\’ 동향’으로 입력해야 한다. 또한 carriage return 을 입력하기 위해서는 ‘\n’으로 표시한다.

3) 위에서 설명한 것과 비슷한 문제로 데이터를 처리하다가 문자열에 \ \ ‘가 나타날 수 있다. 이때 \는 다음의 문자를 특수 문자로 생각하여 \\ 로 처리하게 되어 단지 ‘만 입력한 것과 같은 결과가 된다. 따라서 데이터 중간에 ‘가 있는 것으로 처리되어 에러가 발생한다. 문자열 \\“ 또는 \\‘ 는 \\\“, \\\’로 바꾸어 주어야 한다.

4) 로딩할 데이터를 담은 텍스트 파일의 첫 부분에는 그 데이터들이 로딩될 class 에 대한 정의가 들어가야 한다. 이것은 다음과 같은 형식으로 가진다.

%class <class 이름> (<컬럼 이름 1> <컬럼 이름 2> ….)

만약 첫 부분에 이러한 class 정의가 들어가 있지 않으면 데이터 로딩 시 에러가 발생한다. 이때 주의해야 할 점은 데이터를 로딩하기 이전에 이 class 가 데이터베이스 안에 만들어져 있어야 하고, 그 컬럼의 이름에 정의된 것과 동일해야 한다는 것이다. 또 하나 주의해야 할 점은 이렇게 class 에 대해 정의하는 문장은 한 라인에 쓰여져야만 한다는 것이다. 즉 중간에 줄 바꿈 없이 전체 정의를 써야 한다.

5) 로딩 전에 반드시 OOSQL_CheckDataSyntax 유틸리티를 사용하여 OOSQL_LoadDB 유틸리티의 입력 파일을 검사한다. OOSQL_CheckDataSyntax 유틸리티는 OOSQL_LoadDB 유틸리티의 입력 파일이 문법에 맞게 되어 있는지를 검사하고 에러가 있는 경우 에러의 원인과 해결 방법을 표시해준다.

6) 로딩하고자 하는 데이터의 용량이 클 경우, 파일에 텍스트 형식으로 데이터들을 저장한 후 한꺼번에 로딩하는 방법을 사용하는 것이 효율적이다. 로딩을 위한 텍스트 파일의 첫 부분에 class 정의가 들어가고 이어서 로딩할 데이터들을 나열하게 된다. 데이터들은 튜플의 구분 없이 각 컬럼에 들어갈 내용을 단순히 나열해서 작

성을 한다. 따라서 만약 중간에 하나의 컬럼이라도 비게 되면 데이터 로딩이 잘못 될 수 있다. 예를 들어 A,B,C 라는 필드를 가진 class 에 데이터를 로딩하고자 할 경우 두 번째 튜플의 B 필드에 입력할 데이터가 없을 때 문제가 생긴다.

문제가 생기는 예는 다음과 같다.

"a1"
"b1"
"c1"
"a2"
"c2" <- B 컬럼의 데이터가 없어서 텍스트 파일에 작성하지 않음
"a3"
"b3"
"c3"

와 같이 필드를 저장하게 되면 실제로 저장될 때는

A	B	C
a1	b1	c1
a2	c2 (원래는 data 가 없음)	a3
b3	c3	

로 저장 되서 잘못된 결과를 나타내게 된다.

따라서 데이터 로딩을 위한 텍스트 파일을 만들 때는 데이터가 없는 필드에도 그것을 표시할 정보를 작성해 주어야 한다. 예를 들어 TEXT 타입 컬럼의 경우 입력할 데이터가 없으면 " " 식으로 공백을 넣어주고, 숫자 컬럼의 경우는 0 같은 숫자를 넣어주어 모든 필드가 파일상에 표시되도록 해야만 올바르게 데이터를 로딩할 수 있다.

5.4. 인덱스 생성

목표

빠른 검색과 키워드 검색을 제공하기 위해 인덱스를 생성한다.

내용

문자열 또는 숫자에 대해 인덱스를 만들기 위해서는 SQL 의 create index 를 사용하여 만들 수 있다. 텍스트 데이터에 대해서는 로딩 시 즉시 텍스트 인덱스를 생성할 수도 있고, 로딩 후 일괄적으로 할 수도 있다.

필수 사항

1) 숫자, 문자 필드와 텍스트 필드에 대해 모두 인덱스를 만들 수 있는데 숫자, 문자 필드는 선택적으로 인덱스 생성이 가능하고 텍스트 필드는 필수적으로 텍스트 인덱스를 만들어야 한다.

2) 숫자와 문자 필드는 SQL 문의 create index 를 사용하여 만든다. SQL 문을 수행하는 방법은 앞에서 스키마를 입력하는 방법과 마찬가지고 isql 을 이용하여 수행하는 방법, 오디세우스/Web-PHP 툴을 이용하는 방법, OOSQL API 를 이용한 프로그램을 작성하여 사용하는 방법이 있다. 이 중 하나의 방법을 사용하여 'create [unique | cluster] index *index_name* on *table_name* (*attribute*);'의 형식으로 SQL 문을 입력하고 수행한다.

   만일 잘못된 인덱스를 만들었을 때는 'drop index *index_name;*'의 SQL 문을 위의 방법 중에 하나로 수행시켜 인덱스를 삭제한다.

3) 숫자와 문자 필드의 인덱스는 인덱스를 만들어 효과가 있는 것에만 만든다. 즉 주민등록번호, 이름 같이 같은 값을 갖는 튜플이 적은 필드에만 인덱스를 만든다. 이에 대한 자세한 설명은 제 5.2 절의 스키마 생성 부분을 참조하기 바란다.

4) 인덱스 생성 과정에서는 데이터베이스에 많은 데이터가 기록이 된다. 이 과정에 대해 로그에 기록을 남겨 시스템 오류로 인해 데이터베이스가 파괴되는 것을 방지한다. 그런데 여러 개의 인덱스를 한번에 만들게 되면 인덱스 생성 과정에서 생기는 로그가 너무 커져 경우에 따라 로그 볼륨의 크기를 넘어가게 된다. 따라서 가능하면 인덱스 생성은 하나의 인덱스를 만들고 commit 을 하고 다시 인덱스를 만들고 commit 하게 하여 로그의 양이 필요 이상으로 커지는 것을 방지해야 한다.

5) 텍스트 필드의 인덱스를 만드는 과정은 먼저 키워드를 추출하고 키워드 추출 결과로 텍스트 인덱스를 만든다. 자세한 과정은 오디세우스 교육과정 자료와 교육 비디오에서 자세히 설명되었고 '오디세우스/OOSQL Reference Manual'을 참고하면 된다. 작업을 시작하기 전에 교육비디오와 매뉴얼을 철저하고 읽고 각 과정에 대해 정확히 이해하고 작업에 들어가야 한다.

6) 데이터베이스 스키마를 작성할 때 어떤 필드에 대해서 인덱스를 구축 할 것인가 뿐만 아니라 인덱스를 형성할 때 사용할 키의 성질까지도 고려해야 한다. 예를 들어 사람 이름 같은 경우는 부분 일치 검색 보다는 exact matching 의 경우가 많다. 따라서 인덱스를 구성하는 키가 기본형으로 변화(stemize) 되어서는 안 된다. 하지만 책 제목의 경우 제목 전체를 검색하는 경우 보다는 제목의 단어 일부를 이용한

부분 일치 검색의 경우가 많다. 이런 경우는 제목에 들어간 단어들을 기본형으로 변형시킨 것을 키로 해서 인덱스를 구성해야 한다. 또 연도를 텍스트 타입으로 구성해서 인덱스를 만들고자 하는 경우에는 숫자를 키로 해서 인덱스를 구성해야 한다. 따라서 텍스트 인덱스를 구성하고자 하는 모든 필드에 대해서 인덱스를 구성할 키의 성격에 대해서 명확하게 정의를 내리고 이것에 따라서 실제 데이터베이스를 구축할 때 알맞은 형식의 인덱스, 키워드 추출기를 사용하도록 해야 한다. 이에 대한 설명은 제 5.2 절을 참조하기 바란다.

7) 텍스트 데이터를 'deferred mode'(데이터 입력 후 나중에 인덱스를 만드는 방법)로 입력하고 인덱스를 구축하는 방법에는 두 가지가 있다. 하나는 OOSQL_MakeTextIndex 를 이용한 방법하여 한번에 만드는 방법이 있고 다른 하나는 OOSQL_ExtractKeyword, OOSQL_MapPosting, OOSQL_SortPosting, OOSQL_BuildTextIndex, OOSQL_UpdateTextDescriptor 를 차례로 사용하여 각 단계별로 순차적으로 수행하는 방법이다. 이 두 방법이 하는 일은 동일하다. 즉 첫 번째 방법은 두 번째 방법에서 사용하는 각 단계를 script 형태로 하나로 묶은 것이다. 중간 단계부터 수행해야 하는 경우에는 각각의 유틸리티를 별도로 수행한다. 예를 들어 세 번째 단계인 OOSQL_SortPosting 에서 소팅을 위한 Linux 의 임시 파일을 위한 저장 공간이 부족한 경우, 에러가 발생하는데 만일 후자 방법을 선택하면 파일을 저장할 공간을 확보하고 이 프로그램부터 다시 수행한다.

8) 텍스트 인덱스 구축과정에서 키워드 추출 결과 등의 임시 파일이 생성되는데, 이를 저장할 수 있을 만큼의 파일 시스템 공간(포스팅 파일 크기의 총합 * 2 배)이 확보되어 있어야 한다.

9) 벌크로딩 유틸리티에서는 임시 볼륨을 설정할 수 있도록 되어 있는데 임시 볼륨의 사용량은 키워드 추출 및 키워드 정렬이 종료된 후 생성되는 SortedPosting 파일을 분석하여 추정할 수 있다. 임시 볼륨의 사용량은 다음과 같은 공식에 의해 계산할 수 있다.

임시 볼륨 사용량 = ((SortedPosting 파일에서 중복이 제거된 키워드들의 길이의 합 * 2) + (SortedPosting 파일의 라인 수 * 16)) * 2

참고 사항

1) 텍스트 인덱스 생성에서 가장 시간이 많이 걸리는 부분 중의 하나는 키워드 추출 과정이다. 경우에 따라 데이터베이스의 내용 변경 없이 데이터베이스의 볼륨 크기를 조정하거나, 데이터베이스의 볼륨을 저장하는 device 를 바꾸는 일이 생길 수 있다. 이때 키워드 추출과정을 다시 수행하면 필요 없이 데이터베이스 구축시간이

많이 걸리게 된다. 따라서 키워드 추출된 결과를 저장하고 있는 SortedPosting 이라는 확장자를 가진 파일을 백업하고 이것을 사용하여 키워드 추출과정 없이 데이터베이스를 새로 구축할 수 있다. 이에 관련한 사용방법은 교육비디오와 '오디세우스/OOSQL Reference Manual'에 자세히 나와 있으므로 이것을 참조하기 바란다.

5.5. 데이터베이스 볼륨 테스트

목표

만들어진 데이터베이스 볼륨이 정상적으로 만들어져 있는지를 확인한다.

내용

만들어진 데이터베이스 볼륨을 이용하여 질의 문을 수행하여 만들어진 데이터베이스 볼륨이 정상적으로 동작하는지 확인한다.

필수 사항

1) 오디세우스에 포함되어 있는 isql 을 사용하여 SQL 언어로 볼륨이 정확하게 생성되었는지 확인한다. 기본적인 데이터베이스 볼륨의 테스트에는 isql 만으로도 충분하다. 필요에 따라 볼륨의 세밀한 것까지 확인이 필요한 경우에는 OOSQL API 를 사용하여 프로그래밍을 해야 한다. isql 을 구동하는 방법은 아래와 같다.

```
isql <db name>
```

구동에 성공하였을 경우 오디세우스 데이터베이스가 사용 가능한 상태가 되며, 사용자의 SQL 질의를 대기한다. 테이블 및 튜플의 생성/갱신/삭제를 수행할 수 있으며, 특정 테이블에 대한 질의를 수행할 수 있다. SQL 질의는 항상 세미콜론(;)으로 종료되어야 한다. 프로그램의 수행을 종료하고자 하는 경우 세미콜론 없이 quit 를 입력하면 현재 갱신된 내용이 데이터베이스에 commit 되고 isql 프로그램을 종료한다. 테스트를 위해 사용할 수 있는 간단한 SQL 문장은 오디세우스 디렉토리의 example 서브 디렉토리에 있다. 테스트 SQL 문장을 ISQL 로 실행하는 방법은 아래와 같다.

```
isql test  <  /오디세우스 경로/example/test.sql
```

위 방법으로 실행한 결과가 테스트 결과 샘플 파일 test.result 와 같으면 오디세우스 데이타베이스가 정상적으로 작동하고 있음을 간단히 확인할 수 있다.

test.sql 의 내용

```
CREATE TABLE test1 ( a integer, b varchar(30) );
INSERT INTO test1 VALUES (10, 'abc');
INSERT INTO test1 VALUES (20, 'def');
INSERT INTO test1 VALUES (30, 'ghi');
SELECT * FROM test1;
UPDATE test1 SET b='aaa' WHERE a=30;
SELECT b FROM test1 WHERE a>15;
quit
```

test.result 의 내용

```
First query results:
----------+----------+
         a|         b|
----------+----------+
        10|       abc|
        20|       def|
        30|       ghi|
----------+----------+

Second query results:
----------+
         b|
----------+
       def|
       aaa|
----------+
```

2) 한번에 설계상의 착오 없이 데이터베이스를 구성한다는 것은 대단히 어려운 일이다. 따라서 데이터베이스 볼륨 테스트에서 설계상의 오류가 발견되는 것이 일반적이다. 이러한 경우, 전체 데이터에 대해서 처음부터 볼륨을 재구성해야 한다. 따라서 대용량의 데이터를 이용한 볼륨 구축의 경우, 전체 볼륨을 구축, 테스트, 문제가 있을 때 수정, 재구축의 과정을 거친다면 볼륨 구축 주기가 너무 길게 된다.

   이러한 문제를 해결하기 위해서는 볼륨 구축을 위해서 일부 데이터(예를 들어서 100 만건)를 이용해서 소용량의 테스트 볼륨을 구축하는 방법을 사용한다. 테스트 볼륨을 이용해서 제대로 구축되었는지를 검사하고 모든 것이 완벽하다고 판단되었

을 때에만 전체 데이터에 대해 볼륨을 구축해야 한다. 이렇게 해야만 시행 착오에 의한 시간 낭비를 크게 줄일 수 있다.

3) 볼륨을 테스트 하기에 앞서 로그 볼륨을 반드시 설정해야 한다. 대용량의 볼륨을 구축하기 위해서는 일반적으로 많은 시간이 걸린다. 따라서 테스트 중에 볼륨이 파손되면 상당한 시간적 손해를 입게 된다. 볼륨의 파손은 질의의 오작동, 테스트 하는 사람의 실수, 테스트 수행 중에 발생하는 시스템의 문제 등으로 인해서 데이터베이스가 올바르지 못한 상태로 되는 것을 말한다. 이런 대부분의 문제들은 로그 볼륨을 설정해 줌으로서 방지할 수 있다. 따라서 테스트에 앞서서 로그 볼륨을 반드시 설정해 주어서 이러한 오작동에 의한 볼륨 파손을 막아야만 한다.

4) 데이터베이스 볼륨에 대한 테스트에는 다음과 같은 항목을 검사한다.
   - 각 컬럼별로 데이터가 모두 들어가 있는가 검사
   - 임의 검색한 튜플의 데이터와 입력에 사용한 데이터가 일치하는지 확인
   - 인덱스를 구성한 모든 컬럼에 대해 검색이 빠르게 되는지 확인
   - 질의 결과로 얻어진 결과 건수가 질의에 검색되어야 할 실제 데이터 건수와 일치하는지 확인
   - 텍스트 검색에서 추출된 키워드로 검색이 가능한지 확인
   - 여러 필드를 조합하여 검색하는 통합질의가 정확히 수행되는지 확인
   - 임의의 새로운 데이터의 insert 가 수행되는지, 또한 데이터베이스에 반영되었는지 확인
   - 입력된 임의의 문서에 대한 delete 가 수행되는지, 또한 수행된 결과가 데이터베이스에 반영되었는지 확인
   - 입력된 임의의 문서를 update 가 수행되는지, 또한 수행된 결과가 데이터베이스에 반영되었는지 확인
   - 기타 검색 시스템에 필요한 기능이 정확히 수행되는지 확인

   위의 검사에서 조금이라도 미심쩍은 부분이 있으면 반드시 해결하고 전체 볼륨 구축에 들어가야 한다. 그렇게 하지 않으면 데이터베이스 볼륨을 다시 만들어야 하는 일이 발생되어 대규모 데이터베이스의 경우에는 오랜 시간을 허비할 수 있다.

5) 테스트 과정에서 시스템 동작 시 사용되는 가능한 모든 질의를 수행해 보고 이에 대한 결과와 응답 시간을 측정해 놓은 것이 좋다. 즉 검색 시스템에서 사용자의 입력으로 만들어질 질의를 만들어 수행해 보고 정확한 질의 결과가 얻어지는지 확인한다. 또한 특히 검색에 있어는 검색 시간을 측정하여 놓은 것이 중요하다. 추후에 속도 향상 작업이 있을 수 있는데 테스트 과정에서 검색 시간을 측정해 놓으면

설계상의 오류를 미리 찾아 수정할 수 있을 뿐만 아니라, 속도 향상 결과를 평가하기 위한 자료로 사용될 수 있으므로 질의의 종류에 따라 체계적으로 정리해 놓아야 한다.

참고 사항

1) 로그 볼륨을 설정하고 데이터베이스에 대한 연산을 수행한다 하더라도 디스크 에러 등의 하드웨어에 문제가 생겼을 경우는 볼륨이 손상되게 된다. 이런 경우는 볼륨구축을 위한 시간상의 손해는 물론이고 귀중한 데이터마저 복구 불능의 상태가 될 수 있다. 따라서 이러한 것을 막기 위해서 정기적인 볼륨 백업은 필수적이다. 볼륨 테스트 과정부터 이런 백업을 수행하도록 해야 한다.