SQL BNF 문법 기반의 자동 질의 생성기를 이용한 DBMS 테스트

김정겸○*, 황민호*, 권숙연**, 임종혁***, 배유진****, 하만재*****
*한국과학기술정보연구원, **공주대학교 ,***울산대학교, ****삼성SISO, *****농심데이타시스템
mcjg82@kisti.re.kr , minho@kisti.re.kr , sookyoun@kongju.ac.kr , iidieii@nate.com
yujin.bae@samsung.com , hmj@nongshim.co.kr

Automated Query based on SQL BNF Grammar for Testing DBMS

Kim Jeong-kyeom○*, Hwang Min-ho*, Kwon Sook-youn**, Lim jong-hyeok***, Bae yu-jin****, Ha man-jae*****
*Korea Institute of Science and Technology Information(KISTI), **KONGJU NATIONAL UNIVERSITY,
***University of ULSAN, ****Samsung India Software Operations(SISO), *****Nongshim Data System(NDS)

요 약

현대의 데이터베이스 서버는 거대하고 복잡한 소프트 시스템의 구조이다. 복잡한 SQL(Structured query language) 언어는 점점 늘어가고 ANSI 표준을 바탕으로 새로운 형태로 발달하고 있다. 데이터베이스 서버를 테스트하는 작업은 꾸준히 진행되어 왔으며 앞으로도 계속 도전하고 있는 과제중 하나이다. 그 과제에 적합한 새로운 테스트 기법의 개발을 위해서는 보편적으로 막대한 인력과 비용이 요구된다. 본 논문에서는 수동적인 테스트에서의 막대한 인력과 비용의 문제로부터의 해결책을 제공하기 위해서 자동화된 SQL 쿼리 테스트 프레임워크를 제시한다. 본 프레임워크는 SQL의 기본이 되는 SQL BNF(Backus-Naur Format) 문법을 기본으로 하여 문법적, 의미적으로 정확한 “지능적인” SQL 쿼리를 랜덤하게 자동적으로 생성 한다. 생성된 “지능적인“ 쿼리는 논리적 모델에서 얻어지고, 통계적인 정보를 통해 사용자에게 유용한 체크리스트를 제공한다. 각각의 데이터베이스 개발업체는 그들의 데이터베이스와 새롭게 개발되는 데이터베이스를 통합적으로 테스트 환경을 제공함에 따라 테스트 과정에서의 인력과 비용의 문제를 해결하고, 데이터베이스의 장단점을 파악하는 기준을 제공하여 품질 향상에 도움이 될 것이다.

1. 서 론

데이터베이스 서버를 테스트하는 작업은 분명히 복잡한 형태이며 하위 시스템의 수반함이 고려된 작업이며 새롭게 발매되는 모든 버전의 특징 역시 고려되어진 형태이다. 즉, 새로운 데이터베이스 서버가 시장에 발매되기 위한 테스트 작업에 수많은 노력이 필요하다는 것이다. 그렇기 때문에 본 논문에서 제시하는 자동화는 매우 필수적이며 품질 보증까지도 고려되었다는 것이다. 예를 들어, 소비자 측면에서 서버의 오류는 보통 생산성의 하락 혹은 데이터베이스 업체의 수익에도 손해가 될 수 있는 치명적인 상황을 의미하기 때문에 품질 보장이 중요한 것이다.

Oracle, MySQL, MS-SQL등과 같은 DBMS(Database management system)에서는 그들의 특정 시스템을 위한 시스템 테스트 방식과 특징을 보완하는 기능을 제공한다. 본 프레임워크에서는 위와 같은 데이터베이스 서버에 대비하여 증명된 모든 중요성 특성에 대한 통합된 체크 리스트를 제공하고, 어플리케이션 개발자 입장에서 다양한 접근을 통해서 데이터베이스 솔루션의 전면적으로 평가하는 과정에 도움을 주는 역할을 수행할 것이다.

자동화된 방식으로 SQL 쿼리를 생성하는 작업은 매우 유용하다. 본 프레임워크는 FOP(Feature Oriented Programming) 기법을 사용하였다. 본 기법은 순차적 개발 방법론의 콘셉트를 바탕으로 작은 프로그램의 세부사항을 하나하나 추가하여 스스로 복합적인 구조의 프로그램에 관계를 형성하는 기법이다. FOP 기법에서, 증가되는 사항은 feature이다[1]. 이 콘셉트를 본 프레임워크에 적용하여 하위 절에 종속된 다양한 옵션을 적용한 "SELECT" 기반의 쿼리를 생성하는 것이다. 하위 절은 “where", “group by”, “having”, “window"등의 형태를 말한다.

2. SQL Grammar Test Framework 설계

2.1 개발 환경

본 시스템은 Oracle, MySQL, DB2 등 3개의 데이터베이스 서버와 본 프레임워크가 실행되는 어플리케이션 서버로 구성되어 있다. 서버의 유동성을 위해서 리눅스 기반의 Ubuntu를 사용하였으며, FOP 이용한 객체 지향적인 개념을 위해 개발 언어는 Java를 기본으로 하였으며, 프레임워크 상 DBMS와의 연결의 유동성을 위해 JDBC와 Connection Pool[5]의 개념을 사용하였고, 각각의 데이터베이스 서버의 테스트 결과와 모니터링 진행 사항을 보

여주는 사용자 인터페이스에는 Swing을 통해서 구현하였다. [표 1]에서 개발 환경을 나타내었다.

표 1. 개발 환경

서버 형태	시스템 사양	OS	개발언어
Oracle 10g	1GHz/512MB	Ubuntu	JAVA
MySQL	1GHz/512MB	Ubuntu	JAVA
DB2 Server	1GHz/512MB	Ubuntu	JAVA
Application	1GHz/512MB	Ubuntu	JAVA, SWING

2.2 FOP 프로그래밍

본 프레임워크에서는 프로그래밍 디자인 방법론으로 FOP 기법을 사용하였다. FOP는 순차적 개발 방법론의 콘셉트를 바탕으로 작은 프로그램의 세부사항을 하나하나 추가하여 스스로 유기적으로 복합적인 역트리 구조를 만들어 프로그램에 관계를 형성하는 기법이다. 여기에서 추가되는 사항은 feature(특성, 특징)이다. 이 콘셉트를 본 프레임워크에 적용하면 쿼리 전체의 형태에서 [표 2]에 나오는 하위 절의 형태로 feature가 나누어진다. 자동으로 생성되는 쿼리 형태는 단순한 쿼리부터 Sub-Query가 포함하는 쿼리의 형태까지 다양한 구성이 가능하며, 각각의 Clause에 따른 문법적(Syntactic), 의미적(Semantic) "지능적인" 쿼리를 만드는 알고리즘은 feature 내부에서 구현되어 있다. 또한, clause 내부에서 칼럼 명, 테이블 명 등 실제 데이터베이스 도메인이 요구되는 키워드에 대해서 다음 설명될 Database 논리 모델 정보를 자동적으로 맵핑하는 과정의 알고리즘과 조건절에서 사용되는 연산자, 칼럼 형태에 따른 함수의 사용 역시 feature 형태에 포함되어 있다.

표 2. SELECT Query 구성

No	Clause	Description
1	Simple query	SELECT columns FROM tables
2	WHERE	WHERE columns op condition
3	GROUP BY	GROUP BY columns
4	HAVING	HAVING columns
5	Aggregate Function	function (columns)
6	ORDER BY	ORDER BY columns sort type
7	JOIN expression	table join type table ON condition
8	Sub-Query (1 level)	FROM (sub-query) alias
		WHERE column op (sub-query)
		Having column op (sub-query)

2.3 DB Logical Model 설계

현재 Oracle, MySQL, DB2등의 DBMS Vendor별로 SQL BNF 문법을 기본으로 한 쿼리를 생성하는 알고리즘을 사용하고 있지만, 그들의 DBMS 특성을 살리기 위해서 Grammar를 재 정의하여 구조를 다양화하고, 데이터 타입으로 통합, 확장하여 사용하고 있다. 본 프레임워크에서는 통합된 데이터베이스 테스트 환경을 만들기 위해서 다양한 DBMS를 공통적으로 만족하는 데이터 타입과 일정한 제약 조건[표 3]을 이용하여 테스트 DB를 생성하는 Logical Model Information으로 이용하였다. 칼럼 내부에 사용되는 데이터 타입으로는 숫자 형으로는 INT(정수형), FLOAT(실수형) 타입의 데이터를, 문자형으로는 CHAR(한 글자의 문자형), VARCHAR(두 글자 이상의 문자형) 타입의 데이터를 사용하였으며, 일반적인 테이블 간의 관계를 통한 제약 조건을 이용하기 위해서 키 역할을 하는 기본 키와 참조키와 칼럼의 제약 조건인 NULL과 NOT NULL을 사용하였다.

표 3. Common Logical Model Information

Data Type	Constraint
INT(정수형)	PRIMARY KEY(기본키)
FLOAT(실수형)	REFERENCE(참조키)
CHAR(한 글자의 문자형)	NULL
VARCHAR(두 글자 이상의 문자형)	NOT NULL

2.4 시스템 제약 조건

본 논문에서는 쿼리의 랜덤 생성과 Test Database의 변경으로 인한 테스트 결과의 오류를 줄이기 위해서 SQL BNF Grammar의 범위를 "SELECT"로 제한하였고, 일관된 논리 데이터 모델을 이용하여 테스트 데이터베이스를 생성하였다. 의미적, 문법적인 오류를 제거하기 위해서 Feature 단위별 로직을 통해서 지능적인 쿼리를 생성하였기에 제한된 범위로 한정되어 있어 확장성 측면에서의 유연성이 제한되어 있다. 테스트 과정에서의 시간적 지연을 방지하기 위해서 행수의 제한을 통해서 두 가지 방법의 비교 방식을 사용하였고, timeout 값을 이용하여 지연되는 쿼리에 대한 처리 과정이 사용되었다.

◆ Runtime 제한 - 쿼리의 실행 오류를 방지하기 위해, 일정한 실행 시간을 초과한다면 timeout 값이 화면상에 표시되며 쿼리 실행이 중지된다.

◆ Row 수의 제한 - 비교 기법은 비교를 위해 DBMS에 저장된 Result Set과 마찬가지로 클라이언트의 메모리 제약에 의존된 DBMS에서 리턴 되는 Result Set을 위해 실행된다.

◆ Scope의 제한 - 본 프레임워크는 BNF 문법 파일에서 오직 SELECT 쿼리만 생성하는 Production Rule(77행)으로 범위를 제한한다.

◆ 일관된 논리 데이터 모델 - 일관된 논리 데이터 모델이 다양한 DBMS에 동등하게 사용되었다. (테이블 명, 칼럼 명, 칼럼의 데이터 타입, 기본 키, 칼럼 제약, 테이

블 레벨 속성을 동등한 상태로 테스트를 진행한다는 의미이다.)

◆ 본 프레임워크는 실제 복잡한 형태의 쿼리를 생성하기 위한 로직으로 확장성의 측면에서 좋지 못하다. 만약 사용자가 문법에 하위 항목을 추가하여 더 복잡한 쿼리를 생성하고 싶다면, 프레임워크가 새로운 특성을 기본으로 한 클래스를 자동적으로 만들어내야 한다.

3. SQL Grammar Test Framework 구축

SQL Grammar Test Framework는 SQL 기본 문법이 되는 SQL BNF 문법 파일과 논리 정보를 이용하여 생성된 테이블 리스트 정보를 이용하여 쿼리를 생성한다. Logical 정보를 이용하여 DBMS에 테스트를 위한 TEST DB를 생성하며, 각각의 DBMS에 연결된 Connection을 이용하여 쿼리를 실시간으로 실행하여 결과를 받아온다. 받아온 결과에 대해서 테스트 된 DBMS간의 비교를 통해서 BNF 문법에 대한 만족도와 DBMS의 성능 평가를 할 수 있는 시스템 구조이다. [그림 1]은 시스템의 구조도이며, [그림 2]는 전체 아키텍처를 보여준다.

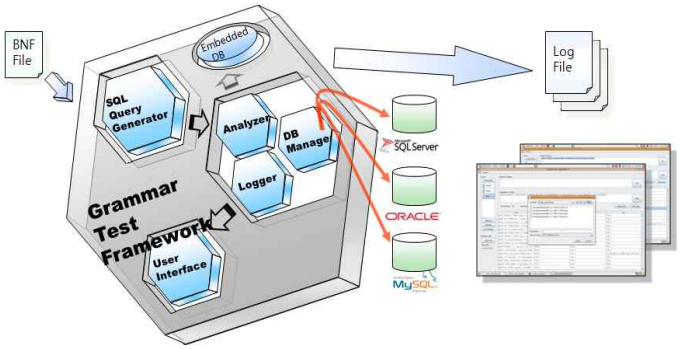

그림 1. System 구조도

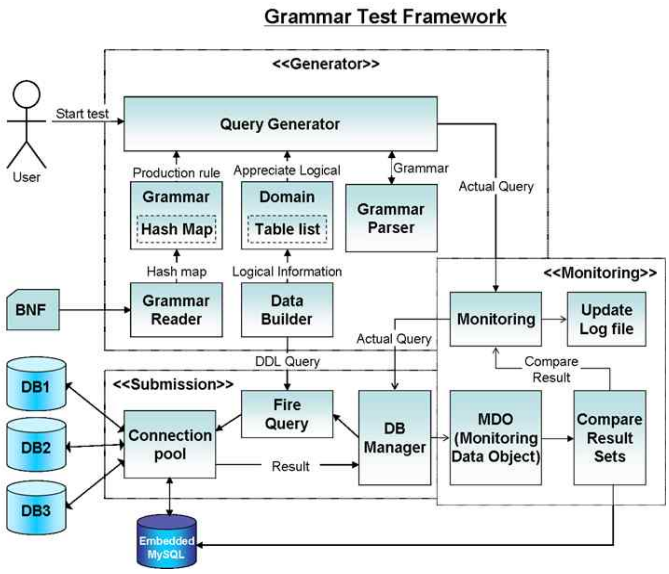

그림 2. System Architecture

3.1 Data Builder

Data Builder 모듈은 다양한 DBMS에 공통적으로 적용되는 DDL(Data Define Language)에 2.3에 언급된 논리 모델 정보를 이용하여 DBMS에 테스트를 위한 데이터베이스를 생성하고, 쿼리를 생성하기 위해서 스키마 정보를 Domain 모듈에서 사용되는 목록에 저장한다. 또한, DBMS에 만들어진 각각의 테이블에 존재하는 데이터를 자동적으로 랜덤하게 불러온다. 언제든지 테스트를 위해서 프레임워크에 추가되는 DBMS을 위해서 테스트 데이터베이스를 삭제하고 재생성하는 역할을 수행한다.

3.2 SQL BNF Grammar

본 프레임워크의 기본이 되는 입력은 SQL BNF 문법 파일이다[3]. SQL BNF 문법은 모든 SQL 문장의 생성 알고리즘을 가지고 있는 규칙사전의 형태를 지니고 있으며, 여러 개의 Production Rule 문장으로 구성되어 있다. [표 3]에 나온 예시처럼 Production Rule는 하나의 문장의 구조의 문법으로써,[::=] Symbol을 기준으로 좌측 어구[left phrase]와 우측 어구[right phrase]로 나누어진다. 좌측 어구는 우측 어구로 대체할 수 있으며, [<>] Symbol 내부에 키워드가 존재하는 것을 Non-terminal, Symbol이 없는 키워드를 Terminal이라고 부른다[2].

Terminal은 실제 쿼리에 사용되는 키워드로 “SELECT", "ALL", "DISTINCT" 그리고 함수이름(COUNT) 등이 있으며, Non-terminal은 동일한 Production Rule의 좌측 어구로 재사용 또는 다른 Production Rule의 좌측 어구로 사용되어 Terminal과 Non-terminal로 나누어진다. [그림 3]에서는 본 시스템의 사용되는 SQL BNF 문법의 일부분을 나타낸 화면이다.

표 4. SQL BNF 구성요소

Production Rule	<set quantifier> ::= DISTINCT | ALL
Non-terminal	<select list>, <derived column>
Terminal	SELECT, ALL, MAX, MIN, COUNT

```
<query specification> ::= SELECT [<set quantifier>] <select list> <table expression>
<set quantifier>      ::= DISTINCT | ALL
<select list>         ::= <asterisk> | <select sublist>
<select sublist>      ::= <derived column>
<derived column>      ::= <value expression>
<value expression>    ::= <column reference> | <set function specification>
<column reference>    ::= <column name>
<set function specification> ::= <general set function>
<general set function> ::= <set function type> <left paren> <function column>
                           <right paren>
<set function type>   ::= AVG | MAX | MIN | SUM | COUNT
<table expression>    ::=
                           <from clause>
                           [ <where clause> ]
                           [ <group by clause> ]
                           [ <having clause> ]
                           [ <window clause> ]
..... and so on
```

그림 3. BNF Grammar for ISO/IEC 9075-2:2003

3.3 Query Generator

Query Generator 모듈은 문법적, 의미적으로 정확한 "지능적인" 쿼리를 생성한다. ANSI SQL 2003 규칙을 따르는 SQL BNF Grammar를 이용하여 문법적인 정확성을 얻고, 프레임워크에 사용되는 FOP 프로그래밍 패턴과 논리 모델 정보의 적절한 조합으로 의미적으로 문제가 없는 쿼리를 생성한다. 시스템에서는 BNF Grammar를 기반으로 feature를 식별하고, 개별적인 Class를 통해 각각의 Feature를 처리하였고, 상황에 따라서 클래스들의 Instance가 적절히 배치되어 쿼리를 구성하였다.

[그림 4]에 나타난 화면과 같이 미리 정의된 논리 데이터 Schema를 사용하여 Data Builder 모듈을 생성하고, Domain 모듈에 논리 모델 정보를 전달하여 테이블 리스트에 저장한다. SQL BNF 문법은 Grammar Reader 모듈에 탑재되고 Hash-Map 형태로 Grammar 모듈에 저장되며, 모든 Production Rule은 Grammar Parser를 통해 Hash Map에 저장되기 위해 파싱된다. 마지막으로 실제 쿼리를 얻기 위해 Hash Map에 존재하는 Production Rule을 파싱하는 과정을 거치며, 논리 모델 정보가 요구되는 Terminal을 만났을 경우 Table 리스트에서 해당되는 정보를 Dynamic하게 사용하고 Non-Terminal 정보를 만나게 되면 해당되는 Production Rule 호출하기 위해서 다시 Hash-Map를 파싱하는 과정을 통해서 쿼리를 생성하게 된다.

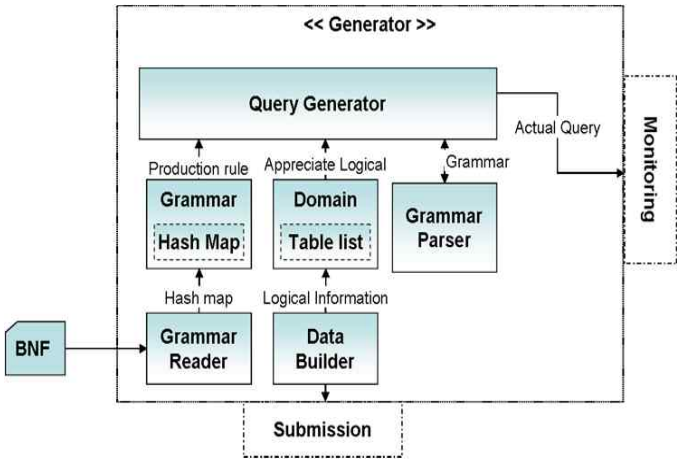

그림 4. Generator Architecture

3.4 Submission

Submission 모듈은 각각의 DBMS에 대한 연결을 생성하여 Connection Pool에 저장한다. Generator 모듈로부터 받은 Logical Model Information을 이용하여 DDL 쿼리를 이용하여 연결된 DBMS에 실행함으로써 테스트 데이터베이스를 생성하게 한다. Monitoring 모듈로부터 받는 실제 쿼리를 받아 Connection Pool을 통해 DBMS에 실행한 후 그 Result Set을 다시 Monitoring 모듈에 돌려준다. Connection Pool는 테스트 할 DBMS와 3.5에 설명될 Embedded MySQL[4]에 대한 Connection을 Connection Object의 형태로 관리하며, 연결이 실패하는 경우 재 연결에 대한 시간의 소비를 절약하고 시스템의 효율성을 강화하는 목적으로도 사용된다. [그림 5]에서는 Submission Module의 구조를 표현하였고, Generator와 Monitoring에 연계되는 과정을 도식화하여 보여주었다.

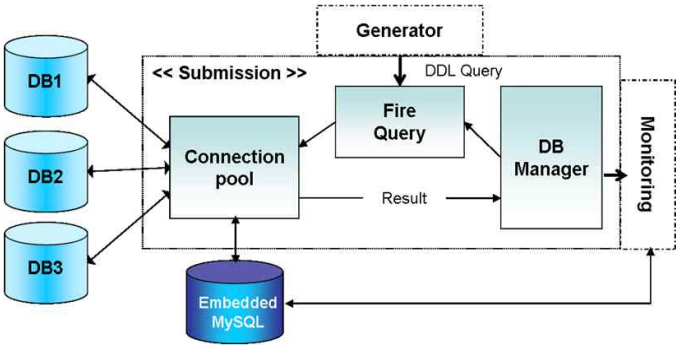

그림 5. Submission Architecture

3.5 Monitoring

[그림 6]에 설명되어 있는 구조와 같이 Monitoring 모듈은 쿼리를 실행한 결과를 비교하여 로그 파일에 작성한다. Result Set의 비교는 크기에 따라서 두 가지 방법으로 비교하는 과정을 사용하였다. Result Set Size가 프레임워크의 시스템이 작동하는 클라이언트의 성능 저하를 가져오지 않는다면, 클라이언트 내에서 Result Set List를 직접 비교하는 방법이 효율적이다. 하지만, Result Set의 크기가 메모리 제한을 초과하더라도 Result Set이 "ORDER BY"와 같이 명확히 정렬되어 있다면 클라이언트에서 블록 단위로 비교할 수 있다.

Result Set 크기가 시스템에 큰 영향을 줄 경우에는 특정한 DBMS(현 시스템에서는 설치의 유동성과 시스템 부하를 줄이기 위해서 Embedded MySQL를 사용하였음)를 이용하여 Result Set 리스트를 저장하고 DBMS에서 제공하는 함수로 Checksum Method를 이용하여 비교하는 방법이 있다. Checksum은 CRC(Cyclic Redundancy Checksum) 값의 구체적인 파일이다. 이 방법을 이용하여 파일 내부의 Consistency를 비교하는 용도로 사용할 수 있다. 대부분의 DBMS는 Checksum 값을 얻는 함수를 제공하고 있다. [그림 7]에서는 Checksum을 이용하여 비교하는 과정을 도식화 하였다. 비교된 결과는 로그 파일에 저장되며, 로그 파일에는 쿼리와 성공/실패 여부 및 비교 결과 등의 정보로 구성되어 있다.

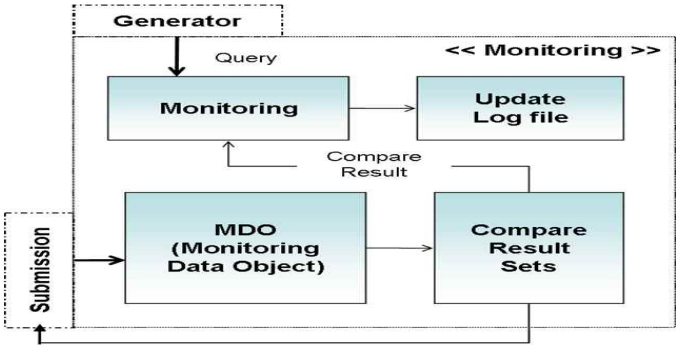

그림 6. Monitoring Architecture

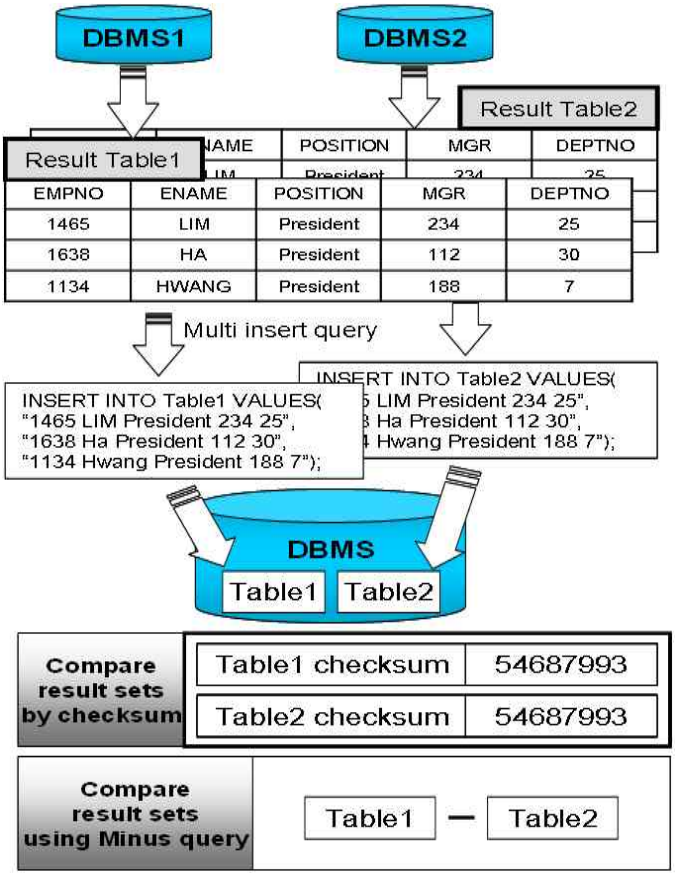

그림 7. Checksum Comparison

4. SQL Grammar Test Framework 성능 테스트

본 프레임워크의 시스템 성능 테스트를 통해서 얻을 수 있는 핵심은 속도와 신뢰성이다. 여기서 말하는 속도는 테스트를 시작하고, 얼마나 많은 쿼리가 빠른 시간에 실행되고, 비교된 결과 값이 로그에 저장되는 속도를 말한다. 신뢰성이란 SQL BNF Grammar를 이용하여 생성된 쿼리가 문법적, 의미적인 오류가 없는 "지능적인" 형태로 테스트에 사용되는지 여부를 말한다.

쿼리의 실행 시간, Result Set의 형태로 인한 비교 과정에 따라서 전체 수행 속도의 영향을 받지만, 분당 200~300개 쿼리를 수행하고, 결과를 받아온다. 신뢰적인 측면에서 문법적, 의미적인 쿼리를 구별하지 않은 상태에서 랜덤으로 시스템을 작동하였을 경우 40% 이하의 쿼리에 대한 Result Set이 반환되거나 특정 DBMS에서는 SQL error 메시지를 출력하였으나, FOP 내에서 문법적 오류, 의미적 오류를 발생하는 구조에 대한 처리 알고리즘을 통해서 95% 이상의 정상적인 쿼리 실행 및 Result Set의 비교과정이 수행되었다. [그림 8]에서는 본 시스템을 이용하여 실제 테스트가 수행되는 과정을 보여주었고, 그림 하단의 로그 파일의 내용을 통해서 수행되는 쿼리의 수, 정상적으로 수행된 쿼리의 수, 실패된 쿼리의 수, 성공 및 실패율이 작성되어 있다.

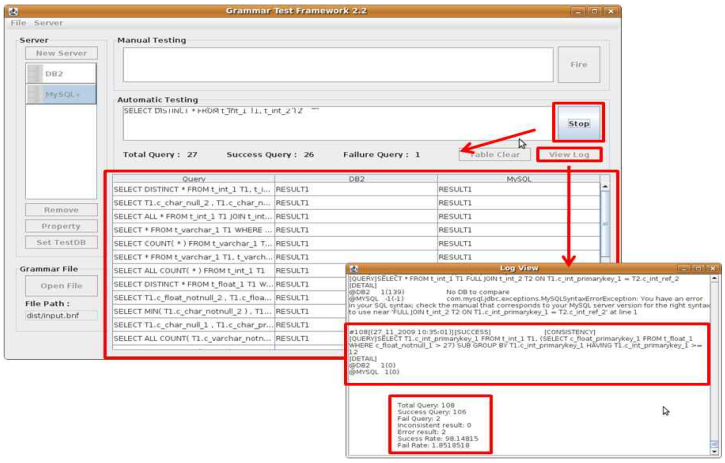

그림 8. SQL Grammar Test Framework 화면

5. 결론

본 논문에서는 SQL 문법이 되는 SQL BNF 문법을 이용하고, 공통된 논리 모델 정보를 사용함으로 통합된 DBMS 테스트 프레임워크를 제안하였다.

본 프레임워크는 다양한 DBMS의 테스트 작업에서의 수고를 감소시킬 뿐만 아니라, 각각의 DBMS 품질과 성능을 비교하는 작업과 DBMS가 ANSI-SQL 2003 문법에 정확한지를 검증하는 작업을 수행한다. 또한, 새로운 DBMS을 개발하는 경우 및 DBMS의 신뢰성 테스트를 위한 도구로 활용 가능하다.

현재 DML(Data Manipulation Language), DCL(Data Control Language), DDL등의 모든 SQL 문장에 대해서 지원되지 않았으며, 오직 한명의 사용자에게 서비스를 제공하고 있다. 앞으로의 연구 방향은 더 많은 문법 파일을 따름으로써 확장된 영역에서 SQL 문장을 지원하고, 다양한 플랫폼에서 여러 개의 DBMS를 다수의 사용자가 테스트할 수 있는 범위로 확장시킬 것이다. 또한, 초소형 DBMS 테스트의 영역으로도 범위를 넓혀갈 것이다.

참고문헌

[1] Sagar Sunkle "Feature-Oriented Decomposition of SQL:2003",Master's thesis, Department of Computer Science, University of Magdeburg, oct 24,2007.

[2] Sagar Sunkle, Martin Kulmanna and Nobert Siegmund "Generating Highly Customizable SQL Parsers",SETMDM '08, Mar 29, 2008.

[3] http://savage.net.au/SQL/sql-2003-2.bnf.html

[4] http://www.mysql.com/products/embedded/

[5] http://java.sun.com/developer/onlineTraining/Programming/JDCBook/conpool.html

[6] http://en.wikipedia.org/wiki/Feature_Oriented_Programming

[7] Salvador Trujillo, Don Batory, Oscar Diaz "Feature Oriented Model Driven Development: A Case Study for Portlets", International Conference on Software Engineering(ICSE) 2007.

[8] Don Batory "Feature Models, Grammars, and Prosositional Formulas", Software Product Line Conference 2005.