KIPS Tr. Software and Data Eng.
Vol.5, No.9 pp.425~432 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2016.5.9.425

Design and Implementation of SQL Inspector for Database Audit Using ANTLR

Chen Liu[†] · Taewoo Kim[††] · Baowei Zheng[†††] · Jeongmo Yeo[††††]

ABSTRACT

As the importance of information audit is getting bigger, the public corporations invest many expenses at information system audit to build a high quality system. For this purpose, there are much research to proceed an audit effectively. In database audit works, it could audit utilizing a variety of monitoring tools. However, when auditing SQLs which might be affected to database performance, there are several limits related to SQL audit functionality. For this reason, most existing monitoring tools process based on meta information, it is difficult to proceed SQL audit works if there is no meta data or inaccuracy. Also, it can't detect problems by analysis of SQL's syntax structure. In this paper, we design and implement the SQL Inspector using ANTLR which is applied by syntax analysis technique. The overall conclusion is that the implemented SQL Inspector can work effectively much more than eye-checked way. Finally, The SQL inspector which we proposed can apply much more audit rules by compared with other monitoring tools. We expect the higher stability of information system to apply SQL Inspector from development phase to the operation phase.

Keywords : SQL Audit, Audit Tool, Database Performance, SQL Audit Tool, Database Audit

ANTLR를 사용한 데이터베이스 감리용 SQL 검사기의 설계 및 구현

Chen Liu[†] · 김 태 우[††] · Baowei Zheng[†††] · 여 정 모[††††]

요 약

정보시스템 감리의 중요성이 커지면서 공공기관이나 기업에서 높은 품질의 시스템을 구축하기 위해 많은 비용을 투자하고 있으며 효율적인 감리 작업을 수행할 수 있는 도구에 관한 연구도 많이 진행되고 있다. 정보시스템의 핵심인 데이터베이스 관련 감리 작업에서 다양한 모니터링 도구를 활용해 많은 검사항목에 대해 감리할 수 있지만, 데이터베이스 성능에 많은 영향을 미칠 수 있는 SQL 감리에는 기능적으로 부족한 면이 존재한다. 대다수의 모니터링 도구들은 메타 정보 기반으로 검사하기 때문에 메타 정보가 없거나 정확하지 않으면 SQL 감리 작업을 수행하기가 어렵고 SQL 문장의 구체적인 문제점을 도출할 수도 없다. 따라서 본 연구에서는 ANTLR를 활용한 SQL 검사기를 설계하고 구현한다. 구현된 SQL 검사기를 통해 기존의 수작업으로 SQL을 검사하는 것보다 효율적으로 수행할 수 있다. 그리고 기능적인 측면에서 다른 모니터링 도구에 비해 더 많은 검사 규칙을 SQL 검사 작업에 적용할 수 있다. 본 연구에서 제시한 SQL 검사기는 개발 단계부터 운영단계까지 감리 작업을 수행하여 정보시스템의 안정성을 높일 수 있다고 기대한다.

키워드 : SQL 감리, 감리 도구, 데이터베이스 성능, SQL 감리 도구, 데이터베이스 감리

1. 서 론

정보시스템 구축사업을 성공하기 위해 1987년 한국전산원에 의해 정보시스템 감리가 도입되었다. 정보시스템 감리는 정보시스템에 대한 위험요인들을 조기 발견 및 개선하는 방안을 통해 위험요소를 최대한 줄이는 것을 말한다[1]. 데이터베이스 감리는 정보시스템 감리의 부분집합이며, 데이터베이스 관련 정보를 수집하고 검사를 하여 위험요인들을 발견 및 개선한다.

정보시스템 감리는 정보시스템 관련 정보를 수집한 후 감리인이 판단하여 위험요소를 검사하는 방식으로 수행된다. 하지만 전문가가 수작업으로 점검하는 방식은 감리인 별로

※ 이 논문은 부경대학교 자율창의학술연구비(2016년)에 의하여 연구되었음.
† 준 회 원 : 부경대학교 컴퓨터공학과 박사과정
†† 비 회 원 : 부경대학교 컴퓨터공학과 석사과정
††† 비 회 원 : 부경대학교 정보공학과 공학박사
†††† 정 회 원 : 부경대학교 컴퓨터공학과 교수
Manuscript Received : March 25, 2016
First Revision : April 22, 2016
Accepted : May 13, 2016
* Corresponding Author : Jeongmo Yeo(yeo@pknu.ac.kr)

점검 항목에 대한 이해도에 따라 다르게 수행하여 일관되지 못한 결과를 도출할 수 있다. 또한, 검사 작업의 양이 많을 경우 많은 시간이 소요된다[1]. 정보시스템 감리의 효율성을 향상시키기 위해 다양한 연구가 이루어졌으며, 그중에 정보시스템 감리 품질을 확보하고 감리 증적을 객관적으로 확보하기 위한 애플리케이션과 데이터베이스 성능개선 관련 감리 도구들이 포함되어 있다[2]. 본 연구는 다양한 정보시스템 감리 분야 중 데이터베이스 성능에 큰 영향을 미칠 수 있는 요소인 SQL에 대한 감리 분야에 초점을 둔다. 규모가 큰 시스템일수록 많은 SQL 문장을 사용하게 된다. 시스템 오픈하기 전에 감리 수단으로 성능 이슈를 검사하여 개선하지 않으면 운영 단계에서 추가적인 비용을 투자하여 성능 문제를 유발된 SQL 문장을 도출해서 튜닝해야 한다. 기존 도구들은 정해진 시간에 수집한 메타정보를 기반으로 검사를 수행하기 때문에 운영 중인 시스템에서 발생한 돌발적인 위험요소들을 모니터 하여 문제점을 도출한다. 하지만 메타정보가 없을 경우에는 검사하지 못하므로 사전 예방 목적으로 사용하지 못한다. 또한, 운영 단계에서 문제를 도출하더라도 SQL 성능 측면에서 문제를 유발하는 이유까지 제시하지 못한다.

이러한 문제를 해결하기 위해 본 논문에서는 SQL 문장에 대해 구문 분석 기법을 적용해 ANTLR[3]를 활용한 검사도구를 구현하고자 한다.

2. 관련 연구

2.1 감리 점검 항목

한국정보화진흥원이 정보시스템 감리 수행 가이드[4]에서 정보시스템 감리에 대한 전반적인 감리 절차, 감리 수행 방법 및 준수사항, 감리 점검 체계에 따른 표준 점검 항목을 제시하였다. 가이드에서는 정보화 사업 유형을 정보기술 아키텍처 구축, 정보화 전략 계획 수립, 시스템 개발(구조적/정보 공학적 모델, 객체지향/컴포넌트 기반 모델), 데이터베이스 구축, 시스템 운영, 유지보수, 사업관리 등으로 분류하였다.

또한 감리 작업 수행 중에 개별 감리원의 경험, 기술력, 노하우에 따라서 동일한 점검 항목에 대해 다른 관점으로 해석하여 점검활동을 수행함으로써 일관성이 부족한 경우가 발생할 수 있으므로 [5] 및 [6]에서 감리원이 동일한 점검 항목에 대하여 동일한 관점, 기준에 따라서 점검활동을 수행할 수 있도록 정보화 사업 유형별로 감리 관점/점검 기준을 제시하였다. [5]에서 제공하는 표준 점검 항목은 절차, 산출물 및 성과 세 가지 감리 관점에서 점검 기준 기반으로 정의하였다.

2.2 SQL 감리 지원 도구

SQL 감리 작업을 효율적으로 수행하고 정확한 검사 결과를 도출하기 위해서 많은 도구 및 방법을 활용하고 있다.

AWR(Automatic Workload Repository)[7]는 오라클에서 제공한 일정 시간을 설정한 후에 자동으로 해당 시점이나 시간대에 데이터베이스 내 여러 가지 정보를 수집하여 저장하는 저장소이다. AWR은 수집해 온 통계정보를 분석하여 성능 문제를 발견 및 Self-tuning에 목적을 두고 있다. AWR의 수집한 정보는 특정 시간에 데이터베이스의 모든 객체, 자원 사용량, 세션 활용 정보, 자원 많이 소비하는 SQL 및 관련된 정보 등을 분석함으로써 성능 문제를 트러블슈팅 한다. 이와 같은 기반의 Toad, Orange, SQL Gate for Oracle 등의 도구들은 강력한 실시간 감시 기능을 제공하지만 SQL로 인한 성능 문제가 발생하여야 발견할 수 있다는 문제가 있다. 또한 특정 SQL을 지정하여 검사하는 기능을 제공하지 않는다.

AWR의 단점을 보완하기 위해 오라클에서 STA(SQL Tuning Advisor)[7]를 제공한다. STA는 AWR에서 수집한 정도나 공유 메모리 영역에 캐싱된 정보를 기반으로 SQL 문장에 대해서 세부적으로 분석한다. STA은 SQL를 옵티마이저를 통해서 다양한 실행계획을 통해 반복적으로 실행해 보아 비용을 비교하여 개선 방안을 제공해 준다. 그러나 아주 세부적으로 분석하기 때문에 하나의 SQL 문장을 분석하는 데에 너무 많은 시간을 소모되며, 배치 방식으로 처리하려면 절차적은 프로그래밍 언어인 PL/SQL 등을 통해서 구현하여야 한다. 게다가 통계정보를 기반으로 분석하기 때문에 통계정보가 정확하게 수집된 운영 단계에서만 적용이 가능하다.

[8]연구에서 제시한 성능 분석기는 실행계획 정보를 수집하여 실행계획에서 큰 비용이 발생 가능성 있는 오퍼레이터를 기준으로 SQL 문장을 추출하는 방식으로 검사한다. 제시한 성능 분석기는 배치 방식으로 처리가 가능하고 사용자가 필요에 따라서 검사 규칙을 정의할 수 있지만, SQL 실행계획 정보 기반으로 수행하기 때문에 SQL의 실행계획이나 통계정보가 존재하지 않은 경우에 검사할 수 없다. 게다가 SQL 문장의 구조에 대해 검사할 수 없기 때문에 많은 검사 규칙을 구현하기 어렵다.

Adhoc 쿼리[9]방식은 AWR에서 수집한 정보나 데이터 사전 등 정보를 사용자가 직접 접근하여 감리 수행 중에 직접 SQL를 사용한다. Adhoc 방식은 가장 자유자재로 검사하는 규칙을 구현할 수 있으며, SQL 문장 구조에 대해서 검사하려고 할 때 정규표현식을 사용할 수 있다. 하지만 정규표현식을 통해서 SQL 문장 구조에 대해서 검사할 때 정규표현식의 한계점으로 복잡한 검사 규칙을 구현하기 어렵다.

2.3 ANTLR 파서 생성기

ANTLR는 오픈 소스인 파서 생성기 중의 하나이며 사용하기가 편리하고 예외처리 기능 등 많은 장점이 있어서 주목받고 있다. 구문 분석에는 렉서(Lexer) 및 파서(Parser) 두 개의 컴포넌트를 사용한다. Lexer는 입력받은 일련의 문자열 스트림을 차례로 읽으면서 미리 정의한 토큰(Token)을 적용하여 토큰나이징(Tokenizing) 작업을 수행한다. 토큰나이징한 결과를 파서(Parser)에게 전달해 주면 파서가 정의된

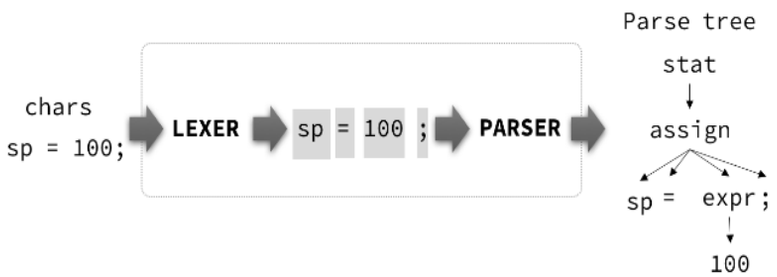

Fig. 1. Process of Syntax Parsing by ANTLR [9]

문법 규칙을 적용하여 입력한 문자열에 대해 조직화한다. 최종적으로 입력한 문자열을 파스 트리(Parse Tree) 데이터 구조로 만들어 준다. 이와 같은 과정을 통해서 구문 분석을 하기 위해 개발자가 직접 파서를 구현하거나 파서 생성기를 통해 파서를 생성하는 두 가지 경우가 있다. 파서 생성기로 구현하는 경우에 개발자가 문법 규칙만 정의하면 파서 생성기가 정의한 문법을 기반으로 컴포넌트를 자동 생성해 주기 때문에 직접 파서를 구현하는 것보다 편리하다. 개발자가 정해진 표현식을 이용하여 구문 분석하고자 하는 텍스트의 문법 규칙을 정의하면 ANTLR는 렉서 및 파서 등 컴포넌트를 생성해 준다[8, 10-14].

개발자가 구문 분석 과정을 거쳐 생성된 파스 트리를 사용하여 원하는 처리를 수행할 수 있다. ANTLR에서는 파스 트리에 대해 추가적인 처리하기 위해 리스너(Listener) 및 비지터(Visitor) 두 가지 Tree-Walking 검색 메커니즘뿐만 아니라 XPATH 및 패턴 기반 처리 방법도 제공한다.

3. ANTLR를 사용한 데이터베이스 감리용 SQL 검사기의 설계 및 구현

본 장에서는 데이터베이스의 좋은 성능을 보장하기 위해 데이터베이스에서 사용하고 있는 SQL 문장 중 성능 문제 유발 가능성이 있거나 개발 표준을 위배한 SQL 문장을 검사하는 것을 목적으로 한다. 기존의 설계된 도구들은 검사 대상 데이터베이스의 통계 정보에 의존하기 때문에 개발 환경에서 적용하지 못하는 한계점이 있다. 이를 해결하기 위해 ANTLR를 이용하여 SQL 구문 분석을 통해 통계정보가 없는 개발 단계에서도 문제가 되는 SQL을 검사하는 데이터베이스 감리용 SQL 검사기를 설계 및 구현하고자 한다.

점검 항목은 감리 영역 범위 안에서 수행하는 검사 작업의 목적을 제시한다. 점검 항목별로 상세화 된 내용을 검토 내용/검토 항목이라고 한다. 정보시스템 감리에서 검토 내용/검토 항목을 제공하였지만 그대로 적용하기에 포괄적이므로 상세화가 필요하다. 본 연구에서 검사를 위해 검토 내용/검토 항목을 상세화 하여 정의한 것을 검사 규칙(Inspect Rule)이라고 한다. 검사 규칙은 검사 단계 처리 중 구문 분석 기법 적용 여부에 따라 쿼리 프로세스(Query Process) 및 파싱 프로세스(Parsing Process)로 나눌 수 있다. SQL 감리 수행 과정 중 검사 작업에 대해 수집된 메타정보를 조회만으로 원하는 검사 결과를 도출할 수 있는 경우는 쿼리 프로세스라 하며, SQL 문장에 대해서 구문 분석 기법을 통해 파스 트리를 생성한 후에 검사를 수행하는 경우는 파싱 프로세스라고 한다.

3.1 SQL 검사기의 설계

1) 감리 점검 항목 분석 및 검사 규칙 도출

검사에 필요한 점검 항목들은 감리 작업의 계획 단계에서 도출 및 확정한다. 확정된 점검 항목별로 점검 방법을 분류하고, 점검 시 검사할 점검 자료를 정한다. 동시에 점검 항목에 대한 구현 방법도 분류한다.

검사를 진행하기 위해 점검 항목 도출 작업이 선행되어야 하며, 점검 항목을 도출하기 위한 다양한 방법들이 존재한다. 첫째, 한국정보화진흥원에서 공표한 정보시스템 감리 지침에서 점검 항목을 도출할 수 있다. 둘째, 고객사나 발주기관에서 현재 사용하고 있는 SQL 개발 표준지침서에서 도출할 수 있다. 셋째, 데이터베이스에 관련된 전문서적에서 도출할 수 있다. 넷째, 고객사 담당자와 인터뷰를 통해서 도출할 수 있다.

점검 항목이 확정되면 점검 항목에 대한 검사 방법을 정한다. 점검 항목에 대한 검사 방법은 전문가 수작업 방식, 프로그램을 활용하는 방식, 전문가 수작업 및 프로그램 활용 방식으로 나뉜다. 전문가 수작업 방식은 전문가의 주관적인 판단 기준에 의해 검사하는 방식이다. 프로그램을 활용하는 방식은 프로그램으로 점검항목을 검사하는 방식이지만 점검 항목 중 전문가의 경험에 의해 판단되는 부분이 존재할 수 있기 때문에 모든 점검 항목을 구현하기 어렵다. 전문가 수작업 및 프로그램 활용 방식은 앞의 두 방식을 결합한 방식이다. 전문가 수작업 및 프로그램 활용 방식은 두 방법의 단점을 보완하였지만 전문가가 처리할 범위가 넓으면 작업의 효율성을 기대하기 어렵다. 그러므로 본 연구에서는 구문 분석 기법을 적용하여 프로그램이 더 많은 점검 항목을 처리할 수 있도록 처리 범위를 넓혀 심사 작업의 효율성 및 정확성 문제를 보완하고자 한다.

점검 항목 분석하여 도출한 결과의 일부는 Table 1과 같고 표 중 구현 방법 항목에서 P는 파싱 프로세스를 의미하고 Q는 쿼리 프로세스를 의미한다.

Table 1. A part of Inspect Rule List

No.	검사규칙	검사 방법	검사 대상	구현 방법
1	중첩 스칼라 서브 쿼리 경우가 존재여부 검사	프로그램	SQL	P
2	용량이 100G 이상인 테이블에 대해 FULL TABLE SCAN 방식으로 스캔하는 경우를 검사	프로그램	Table, ExecPlan	Q
3	바인드 변수 미사용한 경우 검사	프로그램	SQL	P
4	묵시적 형변환 발생여부 검사	프로그램	SQL, ExecPlan	Q
5	조건절 좌측 컬럼 가공 여부 검사	프로그램	SQL	P
6	힌트 사용여부 검사	프로그램, 감리인	SQL	P
7	5개 이상의 테이블 조인여부 검사	프로그램	SQL	P
8	SELECT * 형태인 SQL 존재 여부 검사	프로그램	SQL	P
…	…	…	…	…

2) SQL 검사기 처리 프로세스

SQL 감리 작업은 메타정보 수집 및 수집된 메타 정보에 대해서 검사하는 단계로 나뉜다. 본 연구에서 수집 단계를 다루고자 하는 부분이 아니므로 모든 메타 정보가 미리 수집된 상태로 가정하고 이를 기반으로 검사하는 단계를 다룬다. 수집 단계에서 수집한 메타 정보와 이를 저장하기 위한 레파지토리는 선행연구[15]에서 제시하였고 메타 정보의 종류는 데이터 사전 정보, 동적 성능 뷰 정보 및 검사 대상 SQL 문장으로 나눌 수 있다.

검사 작업을 수행하기 위해서는 검사대상 및 구현된 검사 규칙들을 선택하여 등록한 후에 작업 수행 서비스(JobExec Service)에게 실행을 요청한다. 작업 수행 서비스는 레파지토리(Repository)에서 등록된 검사대상(Inspect Target) 및 구현된 검사규칙 리스트(Inspect Rule List)를 가져온 후에 해당 검사기(Inspector)를 호출하여 검사 작업을 수행한다. 검사기는 실행 중에 검사규칙마다 필요한 메타 정보를 레파지토리에서 가져와서 처리하고, 처리 후 검사한 결과 및 수행 내역을 다시 레파지토리에 저장한다. 모든 점검 항목에 대해서 검사가 완료되면 사용자는 레파지토리에서 검사한 결과 및 검사 작업 수행 내역을 조회할 수 있다.

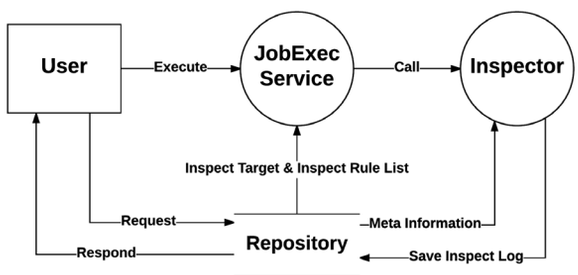

Fig. 2. Data Flow Diagram of SQL Auditor

작업 수행 서비스를 실행한 후 검사 규칙을 분류한다. 검사 규칙은 쿼리 프로세스와 파싱 프로세스로 분류하여 각각 리스트로 생성한 후 검사기를 호출하여 검사한다.

검사기에서 검사 규칙이 쿼리 프로세스일 경우 메타 정보 존재 여부를 먼저 확인한 후에 처리한다. 파싱 프로세스인 경우 검사기에서 메타 정보의 존재 여부뿐만 아니라 SQL 문장 구문 분석 작업 수행을 위한 검사대상 SQL 문장들의 존재 여부도 확인해야 한다. 검사할 SQL 문장이 존재하면 ANLTR로 생성된 파서를 호출하여 SQL 문장의 파스 트리를 생성한다. 파스 트리를 생성한 후 다양한 검사 로직을 적용하여 처리한다.

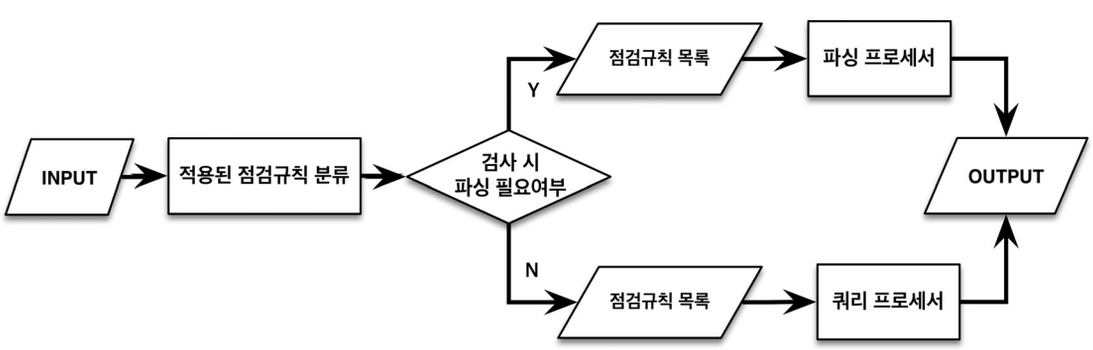

Fig. 3. Process of SQL Auditor

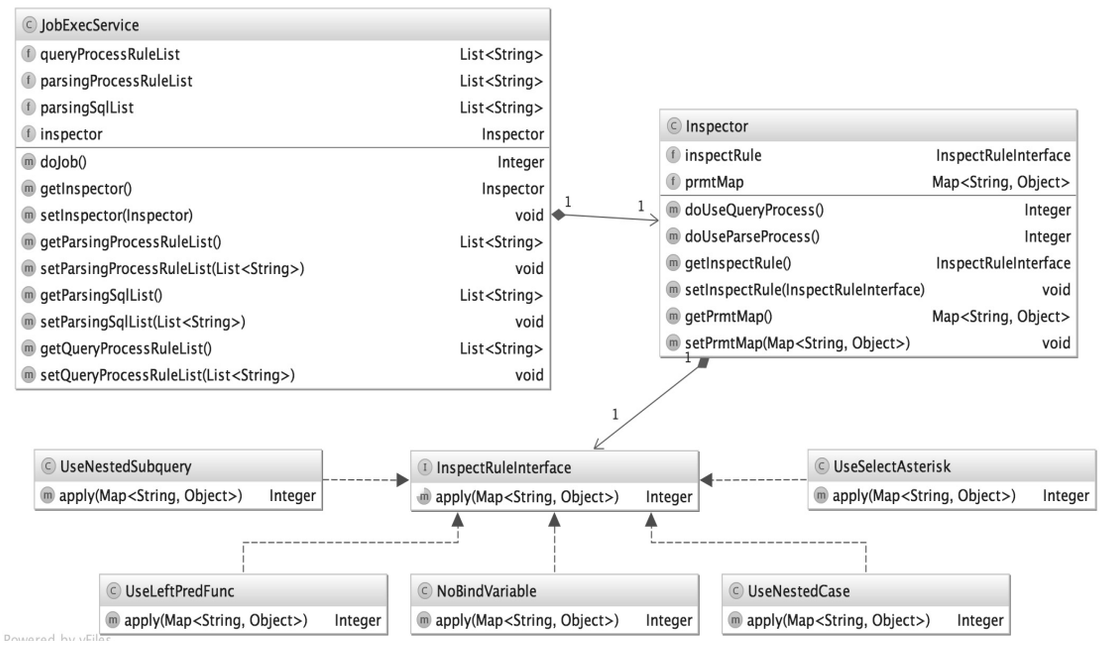

Fig. 4. Class Diagram of SQL Auditor

3) SQL 검사기 클래스 다이어그램

SQL 검사기의 도출된 처리 프로세스를 기반으로 소프트웨어 설계 절차에 맞추어 클래스 다이어그램을 설계한다.

감리 작업을 수행하는 감리인이 최종적으로 확정된 검사 규칙들 및 수집해 온 검사자료들을 저장소에서 가져와 모든 검사 작업 수행 처리 역할 담당인 JobExecService에게 입력으로 전달해 준다. 입력을 받은 JobExecService은 SQL에 대해서 파싱 작업이 필요한지에 따라 queryProcessRuleList 및 parsingProcessRuleList로 분류하며 파싱 대상인 SQL 문장들은 parsingSqlList에 저장한다. 이러한 수행 절차들이 끝나게 되면 doJob메서드를 호출하여 생성한 검사기를 통해서 작업을 수행한다. 검사기는 검사 자료를 입력받고 정해진 규칙을 호출하여 실행하는 역할을 담당하고 있다. 검사기 클래스는 받아온 검사 규칙 명칭에 의해서 해당하는 검사 규칙 클래스를 호출한다. 모든 검사 규칙 클래스들은 InspectRuleInterface를 구현하여 apply메서드를 오버라이드 한다. 이러한 디자인 패턴을 적용하여 검사 규칙의 처리 로직과 검사 규칙 수행자와 분리되어 좋은 확장성을 가져온다. 따라서 추후 새로운 규칙을 적용해야 할 경우에는 수시로 추가하여 바로 적용할 수 있다.

3.2 SQL 검사기의 구현

1) 구현 환경

본 연구에서 제시한 SQL 검사기를 Table 2와 같은 환경에서 구현하고 실험을 수행하였다.

Table 2. Development Environment

SERVER	CPU	Intel i5 4 Core
	Memory	8GB
	OS	Windows 10 64bit
DBMS		Oracle 11g r2
IDE		Intellij IDEA
ETC.		ANTLR
		Spring MVC
		…

2) 검사 규칙 구현

본 절에서는 설계 단계에서 설계된 내용에 따라 검사 도구를 구현하게 되며, SQL 문장을 구문 분석 적용 여부에 따라 쿼리 프로세스 및 파싱 프로세스로 나누어서 구현한다.

a) 쿼리 프로세스 처리 과정

검사 작업을 실행한 후에 작업 수행 서비스는 레파지토리에서 검사 작업 대상 리스트를 가져온다. 검사 작업 대상 리스트에 포함되어 있는 각 대상에 대해서 하나씩 검사한다. 작업 수행 서비스가 작업 대상에 대해서 적용할 검사 규칙 리스트도 레파지토리에서 가져온 후에 검사 규칙 리스트 중에 있는 각 검사 규칙의 명칭을 통해 해당 검사 규칙을 구현된 클래스를 호출하여 apply메서드를 통해서 적용하여 검사한다. 검사 규칙 리스트 중의 모든 검사 규칙은 이와 같은 과정에 따라 모두 처리 완료되면 하나의 검사 작업 대상에 대한 검사 작업이 완료되며, 검사 작업 대상에 있는 모든 작업 대상에 대해서 검사가 완료되면 작업 수행 서비스가 종료된다.

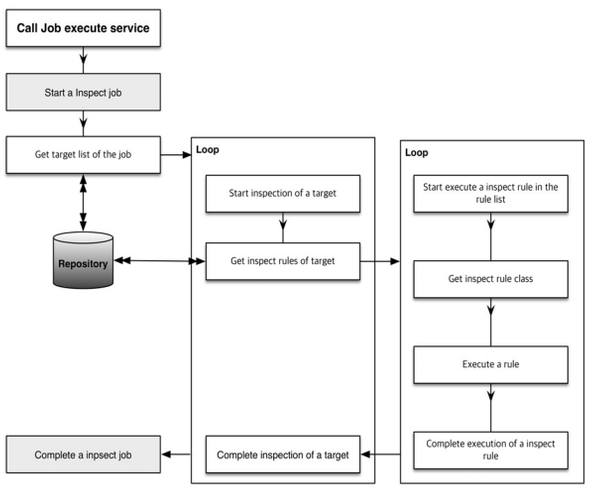

Fig. 5. Process ofqueryprocess inspectrules

b) 파싱 프로세스 처리 과정

파싱 프로세스 검사 방식은 작업 대상 리스트에 있는 각 작업 대상에 대해서 검사를 수행한다. 검사하기 전에 레파지토리에서 해당 검사 대상에서 수집해 온 모든 SQL 문장 및 적용할 파싱 프로세스 규칙 리스트를 가져온다. 쿼리 프로세스로 검사할 때 검사 규칙 별로 수행하지만 쿼리 프로세스로 검사할 때 SQL 문장마다 ANTLR로 구문 분석기(ANTLR Engine)를 통해서 구문 분석하여 파스 트리를 생성한 후에 파스 트리에 대해서 파싱 프로세스 검사 규칙을 구현된 클래스를 호출해서 적용한다. 모든 검사 규칙을 적용한 후에 하나의 SQL 문장에 대한 검사 작업이 완료되고, 수집한 모든 SQL 문장에 대해서 이와 같은 구문 분석 및 검사하는 과정을 수행하여야 하나의 검사 작업 대생에 대한 검사 작업이 완료될 수 있다.

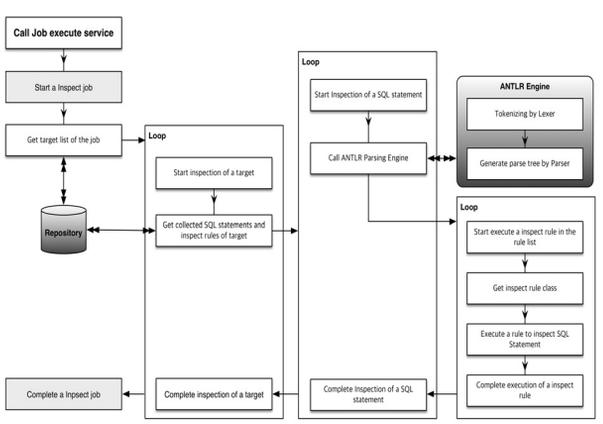

Fig. 6. Process of parsing process inspect rules

c) ANLTR로 통한 구문 분석기의 구현

쿼리 프로세스는 검사 규칙의 클래스 안에 메타 정보를 조회하여 규칙 위배 항목들을 찾아주는 SQL 문장들이 임베디드 되어 있다. 파싱 프로세스는 검사 규칙의 클래스 안에 ANTLR로 생성된 구문 분석기를 통해서 구문 분석한다. 이후 생성된 파스 트리를 전달받아 파스 트리의 필요한 부분을 XPATH나 패턴 매칭 등 방법을 통해 추출하여 검사한다.

ANTLR는 필요한 구문 분석기를 생성해 주기 위해 정해진 문법으로 구문 분석 대상의 문법 규칙을 표현해야 한다. 따라서 본 연구에서는 오라클 SQL 문법 규칙에 따라 ANLTR를 적용할 수 있는 문법 파일을 구현하고 이를 통해서 구문 분석기를 생성하였다.

Table 3. A part of ANTLR syntax

```
select_statement
   :
   subquery_factoring_clause?
   (
   (
   SELECT    ( hint )? ( DISTINCT | UNIQUE| UNION |
           ALL )? select_list
   (   INTO data_item (COMMA data_item)*)?
   FROM
   table_reference_list
   (   where_clause )?
   (   hierarchical_query_clause )?
   (   group_by_clause )?
   (   model_clause )?
   (   union_clause )*
   (   fu1=for_update_clause )?
   (   order_by_clause )?
   (   fu2=for_update_clause )?
   )
   | subquery
   )
   ;
```

문법 파일을 통해서 ANTLR로 구문 분석기를 생성한 후에 이를 통해서 생성된 파스 트리를 검사 규칙 클래스에 전달해 주면 해당 규칙을 검사할 수 있는 로직을 구현한다. Table 4는 검사 규칙을 구현하기 위해 XPATH로 파스 트리 해당 부분을 추출할 때 사용하는 문법의 예시를 보여 준다.

Table 4. A part of XPATH syntax

No.	검사규칙	XPath 추출 예시
1	중첩 스칼라 서브 쿼리 경우가 존재여부 검사	//select_statement//subquery/select_statement//subquery/select_statement
3	바인드 변수 미사용한 경우 검사	//where_clause//NUMBER//where_clause//QUOTED_STRING
5	조건절 좌측 컬럼 가공 여부 검사	//where_clause//condition_expr
6	힌트 사용여부 검사	//hint
8	SELECT * 형태인 SQL 존재 여부 검사	//displayed_column/'*'
…	…	…

3.3 SQL 검사기 적용 실험 및 기능 비교

1) 적용 시험

본 연구에서 구현한 SQL 검사기를 사용해서 수집된 850개 SQL 문장에 대해 파싱 프로세스를 적용하여 검사하였다.

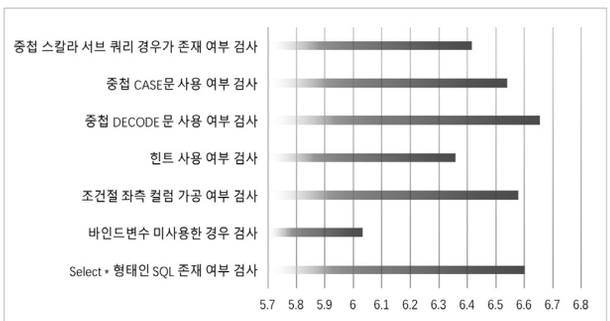

Fig. 7. Result of Experiment

검사 결과를 통해 쿼리 프로세스로 검사하기에 적용하기 힘든 검사규칙들에 대해서 파싱 프로세스를 적용하여 구현한 규칙에 위배되는 SQL문을 도출하였으며, 검사 규칙을 검사하는데 소요되는 시간이 평균 6.4초로 사람이 눈으로 검출하는데 소요되는 시간에 비해 많은 시간이 절약되었음을 확인할 수 있었다. 검사 규칙마다 검사에 드는 소요시간의 차이가 발생하는 것은 검사 규칙을 처리하는 각각의 복잡도로 인해 발생된다.

2) 기능 비교

본 절에서는 본 연구에서 제시한 SQL 검사기와 관련 연구에서 언급한 지원 도구들과의 SQL 감리 과정 중 필요한 기능에 대해 비교 분석하고자 한다.

SQL 감리는 수집 작업을 선행해야 하며 수집한 정보에 따라 검사하는 방법이 다를 수 있다. AWR 기반으로 구현된 도구들은 정해진 시점에 AWR 기능을 통해서 통계 정보를 수집하여 검사한다. AWR 기반으로 구현된 도구들은 시스템 구축한 후 운영 단계에서 정확한 정보 수집이 가능하기 때문에 개발 단계에서 적용하여 검사하기가 힘들다. 본 연구에서 제시한 도구 중 쿼리 프로세스 검사 방식도 통계 정보가 없는 경우에 수행할 수 없고 통계 정보가 정확하지 않으면 검사 결과의 오류를 초래할 수 있다. 이를 보완하기 위해 파싱 프로세스를 통해서 통계 정보가 없을 경우에도 SQL 문장을 구문 분석하여 검사하는 기능을 개발하였다.

또한 AWR에 수집된 정보를 기반으로 SQL 성능을 최적화하기 위해 STA 기능을 통해서 자체 튜닝을 실시한다. 이 기능도 AWR 기반으로 구현된 도구와 같은 단점을 가지고 있다. STA 기능은 특정 SQL에 대해 문제점을 분석하고 최적화시키기 위해 아주 많은 시간 필요하며 배치 방식으로 검사하기 위해 절차형 프로그래밍 언어와 결합해서 구현해야 되는 단점도 가지고 있다. 또한 STA에서는 사용자가 지정한 검사 규칙을 개발하여 추가할 수 없기 때문에 SQL 검사 규칙에 대한 확장성이 떨어진다. 하지만 본 연구에서 제시한 도구에서 구문 분석 기반으로 하기 때문에 통계정보를 의존하지 않아도 구현이 가능하고 새로운 검사 규칙을 추가하는 요구사항이 생기면 수시로 개발하여 적용할 수 있다. 그리고 개발할 때부터 배치 방식으로 SQL 문장을 배치 방식으로 수집하고 검사하는 데에 초점을 맞추었기 때문에 STA 기능을 보완하는 측면에서 봤을 때 확장성을 향상이 되고 효율적이다.

[7]에서 제시한 방법은 수집한 실행계획을 검사하여 성능 문제가 존재하는지를 검사하는 방식으로 구현되었다. 이는 본 연구에서 제시한 도구에서 쿼리 프로세스 검사 규칙에 포함되어 있다. 하지만 본 도구에서는 추가적으로 구문 분석 기법을 도입하는 것은 더 많은 검사를 수행할 수 있게 되었다.

Adhoc 쿼리 및 정규표현식을 결합하여 검사하는 방식은 유연하게 검사 규칙을 구현할 수 있지만 Adhoc 쿼리 및 정규표현식의 한계로 인해 복잡한 구문 분석 요구가 생기면 구현하기 어렵다는 단점이 있다.

본 연구에서는 ANTLR를 적용하였기 때문에 복잡한 구문 분석 규칙도 구현할 수 있을 뿐만 아니라 SQL 문법 규칙이 변경되면 ANTLR 문법 파일만 수정하여 바로 적용이 가능한 장점을 가지고 있다. 또한 Adhoc 쿼리 및 정규표현식을 사용하여 수동으로 구현한 경우 프로그램 코드에서 관련된 부분을 모두 수정하는 문제를 개선할 수 있다.

4. 결 론

본 연구에서는 ANTLR를 사용하여 데이터베이스 감리에 활용할 수 있는 SQL 검사기를 설계하고 구현하였다. 구현된 SQL 검사기로 테스트하며 일반적인 감리 시 활용하고 있는 모니터링 도구와 기능적인 특징을 비교하였다. 본 연구에서 제시한 도구의 목적이 일반적인 모니터링 도구들과 달라 같은 검사 규칙에 대해서 테스트하여 성능 비교는 불가능하지만 SQL 문장에 대해 더 많은 규칙을 적용하여 검사할 수 있다는 기능적인 보완이 이루어졌다. 또한 구현된 SQL 검사기를 통해 기존의 전문가가 눈으로 검사하는 방식보다 효율적이면서 객관적으로 정확하게 검사할 수 있어 데이터베이스 감리 중 SQL 검사 작업에 큰 도움을 줄 수 있을 것이다. 뿐만 아니라 개발 단계부터 SQL 문장을 검사하여 추후 데이터베이스 운영 시의 성능 및 안정성을 향상시킬 수 있을 것이라 기대한다.

References

[1] Jong-won Kim, “System Audit Improvement Through Identifying Database Query Audit Inspection Item,” Master dissertation, Incheon National University, Incheon, KOREA, 2013.

[2] National Information Society Agency, “A Survey and Application Plan for Audit Tools,” National Information Society Agency, 2001.

[3] T. J. PARR, “The Definitive ANTLR Reference: Building Domain-Specific languages,” The Pragmatic Bookshelf, 2013.
[4] National Computerization Agency, “The Guide for Information System Auditing,” National Computerization Agency, 2013.
[5] National Information Society Agency, “Information Systems Audit Cookbook V2.0,” National Information Society Agency, 2007.
[6] National Information Society Agency, “Information Systems Audit Guidelines V1.0,” National Information Society Agency, 2009.
[7] Oracle, Oracle Database Performance Tuning Guide 11g Release [Internet], http://docs.oracle.com/database/121/TGDBA/toc.htm.
[8] Gwangil Park, “Design and implementation of an SQL performance analyzer for DATABASE performance improvement,” Master dissertation, Chungang University, Seoul, KOREA, 2010.
[9] YourDictionary [Internet], http://www.yourdictionary.com/ad-hoc-query.
[10] Parsing [Internet], https://en.wikipedia.org/wiki/Parsing.
[11] T. J. Parr, R. W. Quong, “ANTLR: A Predicated-LL(k) Parser Generator,” *Software-practice and Experience*, Vol.25, No.7, pp.789-810, 1995.
[12] Haiyan Wang and Hebiao Yang, “ANTLR-based SQL Grammatical Analysis Strategy and its Implementation,” *Computer Application and Software*, Vol.30, No.11, pp.68-70, 2013.
[13] Danyang Cao and Donghui Bai, “Design and implementation for SQL parser based on ANTLR,” *2nd International Conference on Computer Engineering and Technology*, Vol.4, pp.276-279, 2010.
[14] Xia Liu, Li Tao, Yuhong Zhou, Kevin Ma, and Xiaoqiang Liu, “The Automatic Marking Method of SQL Script Based on Syntax Analysis and Levenshtein, Distance,” *Software Engineering and Applications*, Vol.3, pp.9-14, 2014.
[15] Chen Liu, Taewoo Kim, Baowei Zheng, and Jeongmo Yeo, “Design and Implementation of SQL Audit Tool for Database Performance,” *KIPS Transactions on Software and Data Engineering*, Vol.5, No.5, 2016.

Liu Chen
e-mail : liuchen@pukyong.ac.kr
2011년 부산외국어대학교 E-Business 학과(학사)
2013년 부경대학교 컴퓨터공학과(석사)
2013년~현 재 부경대학교 컴퓨터공학과 박사과정
관심분야 : 데이터 아키텍처, 데이터 모델링, 데이터베이스 성능 튜닝, 데이터베이스 감리

김 태 우
e-mail : mtkim7895@pukyong.ac.kr
2015년 부경대학교 컴퓨터공학과(학사)
2015년~현 재 부경대학교 컴퓨터공학과 석사과정
관심분야 : 데이터 모델링, 데이터베이스 성능 튜닝, 데이터베이스 감리

Baowei Zheng
e-mail : zhengbw@en-core.com.cn
2004년 중국 서안전자과학기술대학교 컴퓨터학과(학사)
2005년 위덕대학교 컴퓨터멀티미디어공학과 (공학석사)
2011년 부경대학교 정보공학과(공학박사)
2011년~현 재 (주)엔코아 차이나 법인장
관심분야 : 데이터 모델링, 데이터 품질, 데이터 거버넌스, 데이터 분석, 데이터베이스 성능 튜닝

여 정 모
e-mail : yeo@pknu.ac.kr
1980년 동아대학교 전자공학과(학사)
1982년 부산대학교 전자공학과(공학석사)
1993년 울산대학교 대학원 전자 및 전산기공학과(공학박사)
1986년~현 재 부경대학교 컴퓨터공학과 교수 / (주)엔코아 사외이사
관심분야 : 데이터 아키텍처, ITA/EA