**BenchGAD 인수인계 확인서[[1]](#footnote-0)**

인계자 : 구상언,정현태,정명원

인수자 : 김성현

인계날짜 : 2018/09/02

**목차**

1. **Client**

**1.1 작업환경 및 코드 저장소**

**1.2 인터페이스 설명**

1. **Server**

**2.1 서버 구상도 및 개요**

**2.2 코드 설명**

**2.3 프로그램 및 패키지 인스톨**

**2.4 방화벽 정책**

**2.5 서버 실행 방법**

**2.6 Future Work**

**2.7 참고자료**

1. **ExCon(Experiment Conductor)**

**3.1 개요**

**3.2 MapD**

**3.3 프로그램 내부**

**1. Client**

**1.1 작업환경 및 코드 저장소**

1.1.1 OS와 Tool

인터페이스를 개발하는데 있어 MacOS 운영체제를 사용하였고, Java의 Swing을 개발하는데 있어서 편의를 위해 NetBean IDE로 작업했다.

1.1.2 GitHub

https://github.com/BevoLEt/BenchGAD\_0 에 코드의 History와 관련 문서자료가 남아있다.

**1.2 인터페이스 설명**

1.2.1 유스케이스 다이어그램

우선적으로 구현되어야 하는 기능들을 나타내는 유스케이스 다이어그램이다. 유스케이스에서 제공 되어야할 기능으로는 구성 설정, 인풋 데이터 제공, 벤치마킹 시작, 로그인 및 회원가입, 시스템 모니터링, 결과 데이터 확인, 유저 정의 인터페이스 설정이 있다. 추가적으로 현재 유스케이스에서는 명시되어 있지 않은 기능으로 콘솔창, 병목 지점 탐지, R 및 파이썬 언어를 통한 통계 등이 있다.

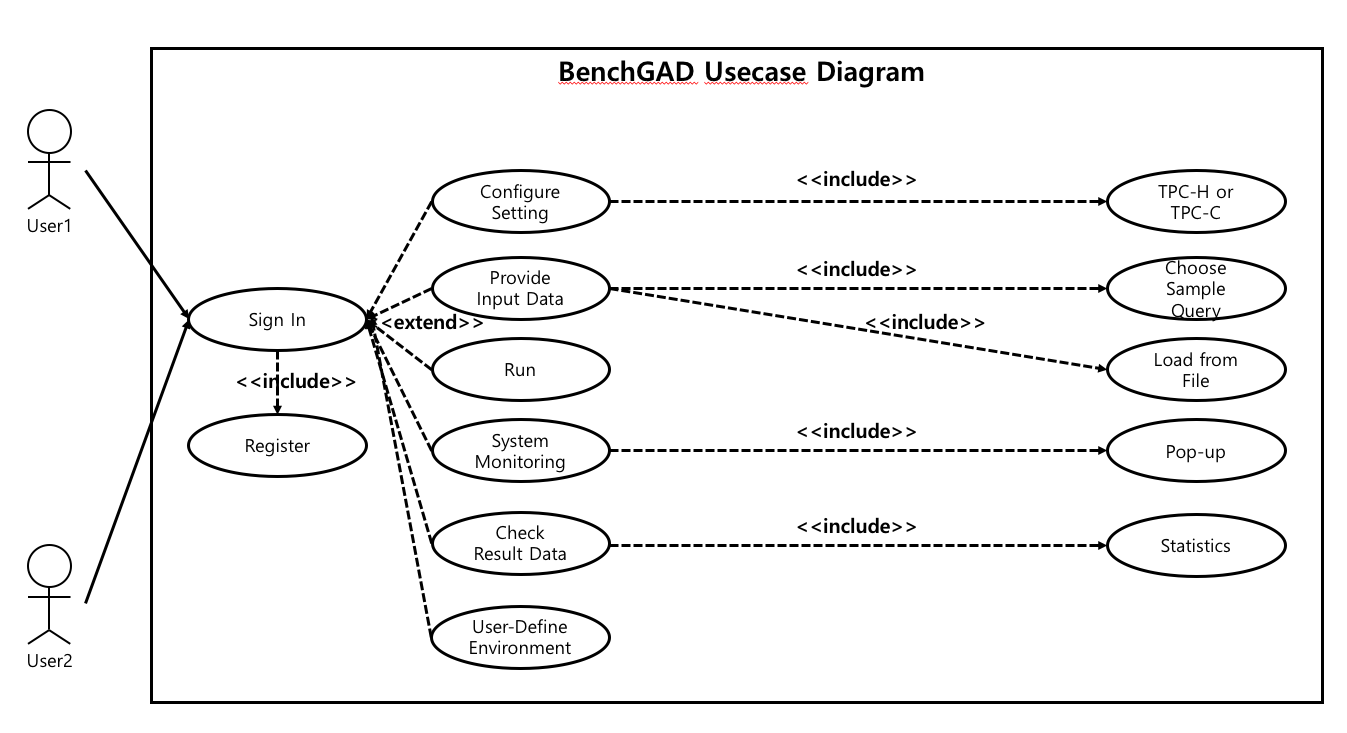


figure 1.1 유스케이스 다이어그램



figure 1.2 BenchGAD 인터페이스

1.2.2 구현되어 있는 기능

1) 벤치마크 구성 설정

좌측 버튼 패널의 톱니바퀴 버튼을 누를시 나오는 패널이다. 맨 위의 콤보박스를 통해 TPC-C, TPC-H, User-Define옵션을 고를 수 있다. TPC-H일때는 Transaction Refresh Rate항목은 작성할 수 없도록 한다. 옵션을 다 작성한뒤 Save버튼을 눌러야 벤치마킹엥 사용할 옵션이 저장된다. Save to my Computer버튼을 통해 현재의 옵션 정보를 내 컴퓨터에 .pn파일로 저장을 할 수 있고, Load Setting 버튼을 통해 이 파일을 불러와 프로젝트에 적용시킬 수 있다.

**추가적으로 구현해야할 목표는 각 쿼리마다 구성 설정을 할 수 있도록 해야하고, User-Define옵션이 제대로 구현될 수 있도록 해야한다.**

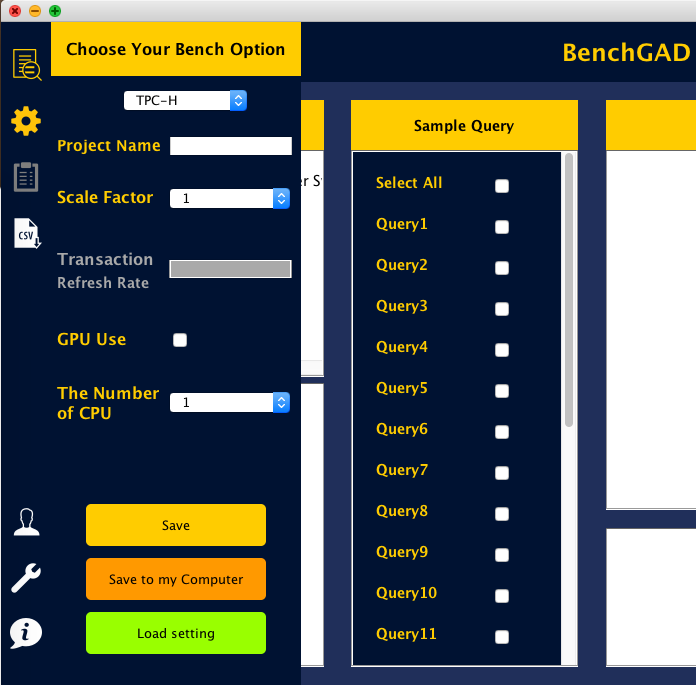


figure 1.3 구성 설정

2) 시작 화면 창

프로그램을 처음 실행하면 유저한테 보여지도록 설계된 창이다. Register 버튼을 통해 계정을 생성하고 Sign In 버튼을 통해 로그인 하여 BenchGAD 클라이언트를 이용할 수 있도록 한다.

**현재는 창만 구현되어 있기 때문에, 로그인 및 회원가입 기능을 만들어 연동시킬 필요가 있다.**

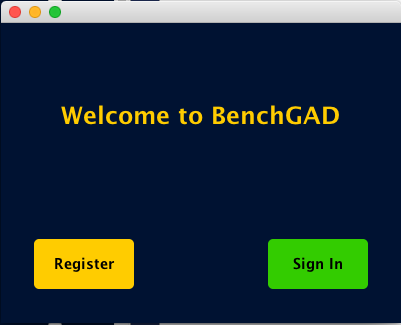


figure 1.4 시작화면 창

3) 벤치 마킹 패널

유저 관점에서의 벤치마킹 시나리오는 다음과 같다. DBMS를 선택하고, Sample Query를 선택하거나 유저가 직접 Query를 수정하거나 파일을 통해 읽어온다.

DBMS 선택 패널은 선택 가능한 위쪽의 JTree와 아래쪽의 선택 되어진 JList 부분으로 나뉘어 진다. JTree의 항목을 더블 클릭 할 경우 JList에 추가가 되고, JList의 항목을 더블 클릭 할 경우 JList에서 해당 항목이 삭제된다. 중복은 허용하지 않는다.

Sample Query 선택 패널은 TPC-H의 Sample Query 22개를 선택할 수 있다. 각 항목의 이름을 누르면 오른쪽 Query 작성 부분에서 내용을 확인할 수 있고, 필요에 따라 추가적으로 수정을 할 수 있다. 각 항목에 체크를 하거나 Select All 박스를 통해 전 항목을 추가 혹은 제외시킬 수 있다. **현재는 각 항목을 체크해도 체크한 모든 항목이 서버에 보내지는 것이 아니라 오른쪽 Query 작성 부분의 내용만 보내지니 이 부분의 수정이 필요하다.**

오른쪽 위에 있는 버튼을 통해 FileChooser를 열 수 있고, .txt파일을 읽어 들어와 Query작성 부분의 내용을 채울 수 있다.

**오른쪽 밑의 패널은 현재 아무 기능이 없지만, 현재 프로젝트에 적용된 설정들을 유저에게 보여주기 위해 남겨둔 패널이다. 추후에 수정이 필요하다.**

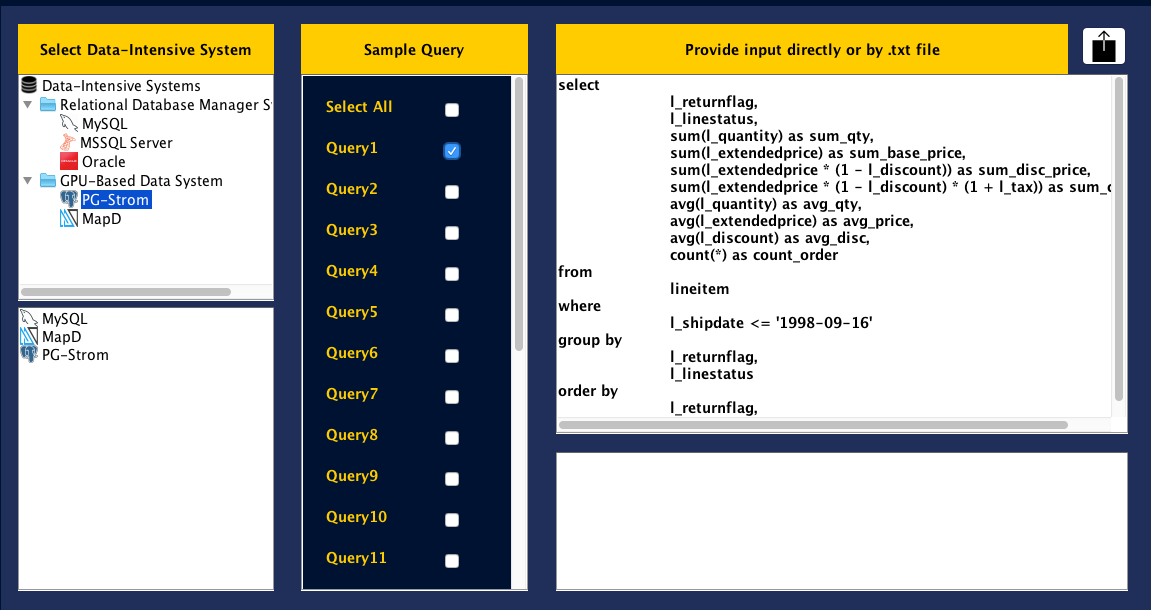


figure 1.5 벤치 마킹 패널

4) 하단 패널

메인 프레임의 하단에 위치한 패널이다. 왼쪽의 notification 버튼을 누를 경우 팝업창의 사용 여부를 선택할 수 있도록 설계하였다. **현재는 notification이 켜졌다는 팝업만 뜨도록 되어있고, 추후에는 작업이 끝났을 때, 서버 연결이 불안정할때 등을 알려주도록 해야한다.**

오른쪽의 play 버튼은 위에서 설정한 벤치마킹 옵션을 토대로 벤치마킹을 실행한다. 벤치마킹을 실행하여 결과가 나왔을 경우만 결과 패널로 넘어갈 수 있도록 설계를 해야한다.



figure 1.6 하단 패널

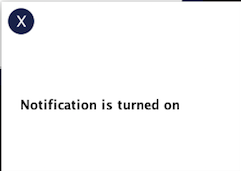


figure 1.7 팝업 창

5) 어바웃 프레임

사이드 패널에 있는 about버튼을 누를 경우 나오는 창이다. 현재는 BenchGAD 팀원의 이름과 역할이 나와있고, 유저가 사용에 있어서 피드백이 있을 경우 연락할 수 있는 이메일을 남겨 두었다.



figure 1.8 어바웃 창

1.2.3 구현되어야 하는 기능

1) 결과 패널

현재의 결과 패널에는 벤치마킹결과를 한줄로 나타내는 기능밖에 없다. 결과가 나왔을 경우에만 패널에 접근할 수 있게 해두었다.

**추후에는 벤치마킹의 결과를 테이블이나 차트와 같은 시각화된 자료로 나타낼 수 있어야 하고, R과 Python등을 이용해 유저가 직접 결과를 분석할 수 있도록 하는 기능도 추가되어야 한다.**

2) export 기능

벤치마킹의 결과를 csv와 pdf로 저장할 수 있는 기능이 추가되어야 한다.

3) 계정과 관련된 기능들

BenchGAD를 이용하기 위해서는 로그인을 하는 것이 전제이기 때문에, 로그인 및 회원가입 기능, 계정 관리 기능, 벤치마킹과 관련된 정보를 계정과 연동시키는 기능들이 필요하다.

4) 유저 디파인 설정

BenchGAD의 인터페이스와 여러가지 설정등을 유저가 직접 설계할 수 있는 기회를 제공해야 한다. 버튼의 위치, 벤치마킹 타입 등이 이에 해당한다.

주로 비쥬얼 스튜디오 코드와 서브라임 텍스트의 기능을 가져오는 쪽으로 설계하였다.

**2. Server**

**2.1 서버 구성도 및 개요**



BenchGAD 프로젝트에 사용된 서버는 Rest API 를 사용하는 웹서버이다. 따라서 클라리언트와 서버는 HTTP라는 규칙을 이용해서 서로 통신하게 된다. 서버의 주 기능은 API 제공과 Mongo DB 와의 연결이다. 우선 BenchGAD 의 서버 및 데이터베이스는 rhel 7.4 버전에서 구동된다. 서버 프로그램으론 Node js를 사용하여 구현하였으며 Data Base 는 No-sql DB 인 Mongo DB 를 사용한다. Node js 는 v8 버전, Mongo DB 는 v4.01, 두 프로그램 모두 community edition을 사용하고있다. 현재 dkelab 서버 컴퓨터에 올라가 있는 서버 프로그램 같은 경우는 별도의 설치 작업 없이 현재 실행이 가능하다.

**2.2 코드 설명**[[2]](#footnote-1)

**2.2.1 JAVA CLASS**

1. **Alldata.java** : 서버에 보낼 데이터를 AllData 에 정의하고 저장하면 된다. REST 클래스의 postinfo(Alldata topost); 에 인자로 사용하면 된다.
2. **REST.java** : BenchGAD 클라이언트와 Excon 와 같은 Java application과 Node js 서버의 통신을 위해 만든 클래스이다. getInfo(String name) 와 postinfo(Alldata topost) 메소드로 이루어져있다. getInfo의 경우 서버에서 리퀘스트를 보내 원하는 정보(현재는 name 으로 구분) 을 얻어오는 기능이며, postinfo의 경우 Alldata 에 저장된 정보를 서버로 전송하게 된다. 현재 서버가 REST api 를 사용하는 웹서버임으로 http 통신을 사용하고 있다. REST.java 클래스에는 jackson lib 를 사용하여 json converting 기능을 사용하고 있다. [[3]](#footnote-2)

**2.2.2 Node js**

[[4]](#footnote-3)

1. **app.js** : 서버 실행을 담당하는 코드이다. port num 과 ip 를 설정할 수 있다.
2. **package-lock.json & package.json** : lock 의 경우 자동적으로 패키지가 추가될경우 생성되는 코드이므로 수정을 지양하며 package.json 의 경우 현재 프로젝트에 사용된 패키지 dependencies 정보와 스크립트가 저장되어있다.
3. **node.modules** : 패키지 및 모듈이 저장된 폴더이다. 수정 지양



4) **index.js** : Rest API 와 mongodb 스키마 정의 및 저장,로드 기능 코드가 존재한다.[[5]](#footnote-4)

5) **user.controller.js** : CRUD 관련 함수를 모듈화 시켜 정의한 코드이다.

**2.3 프로그램 및 패키지 인스톨** [[6]](#footnote-5)

모든 프로그램은 rhel 7.4 에서의 설치를 가정하고 있다.

**2.3.1 Node js 설치**

1. 터미널 실행 후 **$curl --silent --location** [**https://rpm.nodesource.com/setup\_8.x**](https://rpm.nodesource.com/setup_8.x) **| sudo bash -**
2. **$sudo yum -y install nodejs**
3. **$node --version** 을 통해 설치작업이 완료되었는지 확인

**2.3.2 Mongo DB 설치**

1. **/etc/yum.repos.d/mongodb-org-4.0.repo** 를 만들고 난 뒤 아래와 같이 내용을 **수정**한다.

[mongodb-org-4.0]

name=MongoDB Repository

baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.0/x86\_64/

gpgcheck=1

enabled=1

gpgkey=<https://www.mongodb.org/static/pgp/server-4.0.asc>

2) 터미널 실행 후 **$sudo yum install -y mongodb-org**

3) **$mongo --version** 을 통해 설치작업이 완료되었는지 확인

4) **$systemctl enable mongod** 를 통해 mongod 등록

**2.3.3 Node js 패키지 설치** [[7]](#footnote-6)

1. **express 패키지 설치** : $sudo install express 사용하는 이유 : 프레임워크 사용을 통한 빠르고 효율적인 서버 개발을 위함
2. **body-parser 패키지 설치** : $sudo npm i body-parser --save 사용하는 이유 : express 의 요청 바디의 데이터에 접근하기 위해 사용
3. **mocha 패키지 설치** : $sudo npm i mocha --save-dev 사용하는 이유 : Node js 서버를 테스팅 하기 위한 패키지
4. **mongodb 패키지 설치** : $sudo npm install mongodb 사용하는 이유 : mongodb - Node js 상호작용 관련 모듈을 사용하기 위함 일종의 드라이브이다.
5. **mongoose 패키지 설치** : $sudo npm install mongoose 사용하는 이유 : Object mapper 로써 mongodb sql 을 모르더라도 javascript 문법만으로 mongodb를 사용하기 위함

**2.4 방화벽 정책**

**2.4.1 firewalld 설치**

1. 터미널 실행 후 **$yum install firewalld**
2. **$systemctl start firewalld**
3. **$systemctl enable firewalld**

**2.4.2 port 추가**

1. **$firewall-cmd --permanent --zone=public --add-port=PORT/tcp** (PORT 에는 포트넘버기입 ex 8000)[[8]](#footnote-7)
2. **$firewall-cmd --reload** 로 서비스를 재실행하면 반영된다.

**2.5 서버 실행 방법**[[9]](#footnote-8)

1. 터미널 실행 후 **$cd BenchGAD\_Server**
2. **$systemctl start mongod** 를 통해 MongoDB 백그라운드 실행
3. **$node app** 서버가 실행되고 콘솔에 서버가 실행되었다는 문구가 나타난다.
4. 새로운 터미널을 생성한 뒤 **$mongo** 를 타이핑 후 mongo shell 에 접속

**2.6 Future Work**

1. Server Node js **코드 리팩토링**
2. 데이터베이스 **스키마 수정**
3. rest api **설계 조정**
4. Client, ExCon, Server 를 제어할 수 있는 **관리 프로그램 제작**
5. 테스트 툴을 통한 **서버 테스트 작업**

**2.7 참고자료**

<https://www.mkyong.com/java/how-to-send-http-request-getpost-in-java/> -- java http 통신

<http://webframeworks.kr/tutorials/nodejs/api-server-by-nodejs-01/#tocAnchor-1-8> -- node js 를 이용한 rest api server 제작방법

<http://bcho.tistory.com/1096> -- moongose 사용방법

<https://nodejs.org/ko/download/package-manager/> -- 2.3.1 Node js

<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-red-hat/> -- 2.3.2 Mongo DB

<https://www.lesstif.com/pages/viewpage.action?pageId=22053128> -- 2.4 방화벽정책

**3. ExCon (Experiment Conductor)**

**3.1 개요**

ExCon은 BenchGAD의 모듈 프로그램중 하나이다. 서버에 저장되어 있는 클라이언트의 요청을 JSON 형식으로 정리되어 받아 요청에 맞게 여러 DBMS에 할당될 실험을 감독하고, 수행하는 역할을 한다. ExCon은 JDBC를 이용하여 실험 대상 DBMS에 연결하며, 실험 수행 시간 뿐만 아니라 각종 GPU 관련 metrics를 뽑아오는 것이 목적인 프로그램이다.

**3.2 MapD**

본 항목에서는 ExCon에 연결되는 GPU 가속 DBMS 중 하나인 MapD의 설치 방법과 JDBC 라이브러리를 설명한다. 또한, MapD에 대해 TPC-H를 빌드하는데 필요한 옵션 등을 설명한다.

**3.2.1 설치**

**CUDA가 설치되어 있다고 가정.**

1. [MapD 홈페이지](https://www.mapd.com/)에 접속하여, **Download**->**Community-**>**GPU+CPU**->**Linux/CentOS**->**x86\_64**->**yum** 선택
2. 아래로 스크롤 하여, **Preparation** 선택 후 Installation CUDA를 제외한 항목 실행[[10]](#footnote-9)
3. 이후 홈페이지의 지시를 따르면 됨.

### **3.2.2 설치 후 동작 및 점검**

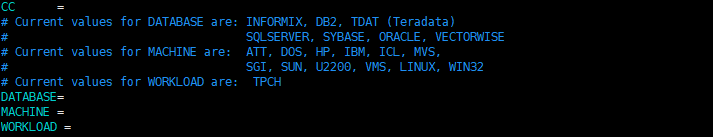
1. 홈페이지의 지시를 따른 이후, $MAPD\_PATH에 지정된 경로로 이동한다.
2. ./insert\_sample\_data 를 실행한 이후, 2번 데이터만 MapD에 삽입한다 [[11]](#footnote-10)
3. ./bin 디렉토리에 들어가 ./mapdql을 실행한 뒤, 기본 계정 정보로 로그인[[12]](#footnote-11)하여 \t 를 입력해 알맞게 삽입되었는지 확인한다.
4. 계정의 홈 디렉토리에서, **.bashrc** 파일에서 export $MAPD\_BIN=$MAPD\_PATH/bin 을 추가한다. 이는 mapdql 등 MapD와 관련된 여러 프로그램을 편리하게 사용하기 위함이다.
5. 방화벽 설정중, sudo firewall-cmd --zone=public --add-port=9091/tcp --permanent && sudo firewall-cmd --reload 을 실행하여 mapdql의 포트 또한 개방한다. 이는 추후 JDBC 연결을 위함이다.

### **3.2.3 JDBC**

**RHEL 7으로 설치된 MapD를 기준으로 한다.**

1. $MapD\_PATH/bin/ 디렉토리 내의 **mapd-1.0-SNAPSHOT-jar-with-dependencies.jar, mapdjdbc-1.0-SNAPSHOT-jar-with-dependencies.jar** 을 MapD의 JDBC를 이용해 작성하고자 하는 프로그램 내에 추가시킨다.
2. $MapD\_PATH/SampleCode/ 내의 SampleJDBC.java 파일 내에 JDBC를 작성하는 routine으로 코드가 있어 참고하며 작성한다.

### **3.2.4 TPC-H**

1. [TPC 문서 페이지](http://www.tpc.org/tpc_documents_current_versions/current_specifications.asp)에서 TPC-H 소스코드를 다운 받는다.
2. 다운받은 zip 파일을 unzip으로 압축해제 후 <tpc-h\_version>/dbgen/ 디렉토리 내로 들어간다.
3. makefile.suite 파일을 Makefile로 복제한다.
4. Makefile에서, 다음과 같은 부분을

CC =gcc

DATABASE=SQLSERVER

MACHINE=LINUX

WORKLOAD=TPCH

로 변경한다.

1. make 를 실행한다.[[13]](#footnote-12)

**(\*\*) qgen.c 코드 수정사항**

random seed 관련 출력 stdout -> stderr로 변경

SET\_ROWCOUNT 관련 출력 주석처리. (\*.sql 파일에서도, :n 제거)

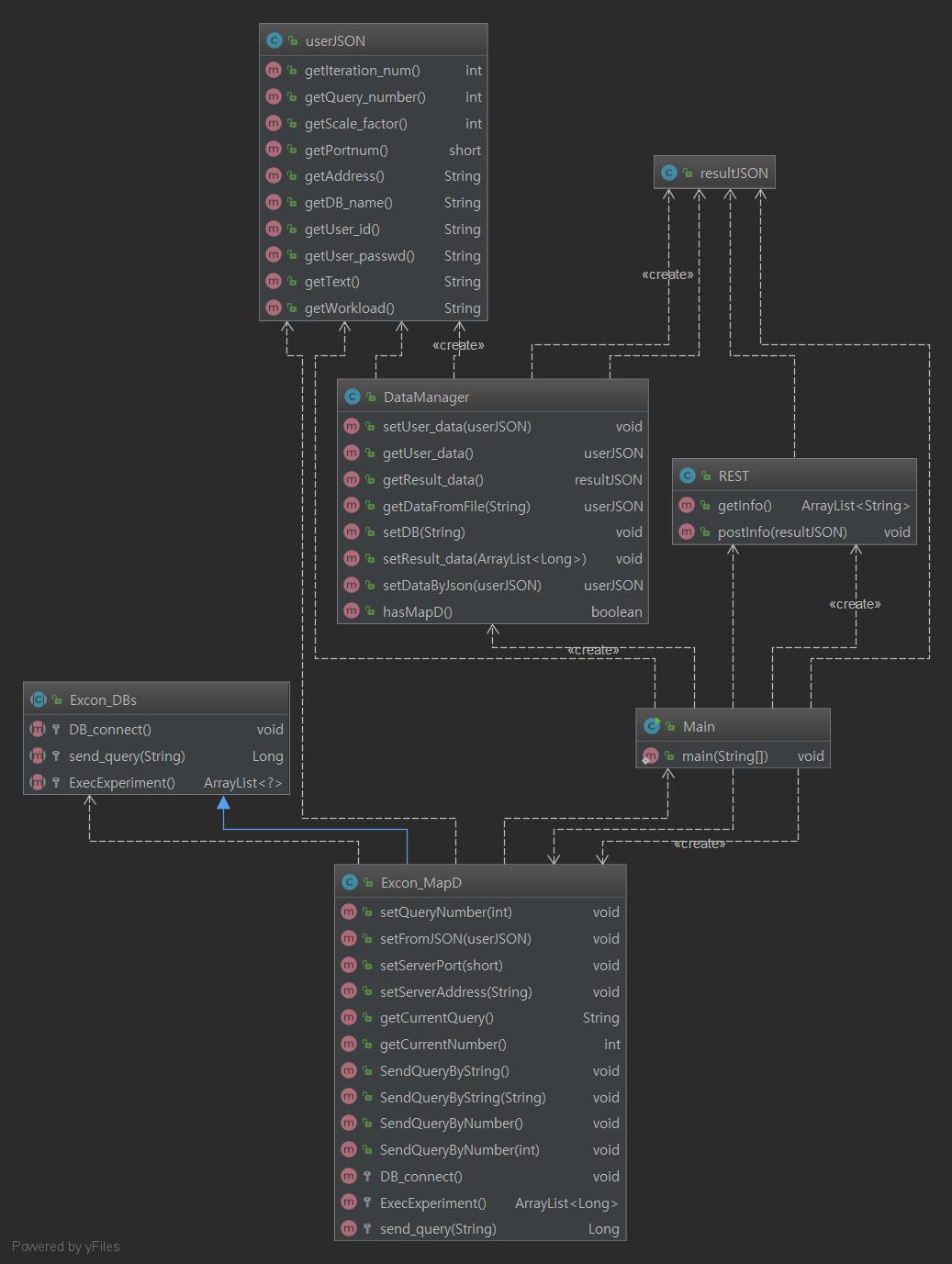
헤더를 출력하는 라인 주석 처리(mapd에서 sql의 주석을 인식하지 않기 때문에 제거하였음)

**3.3 프로그램 내부**

이 항목에서는 인계날짜까지 개발된 ExCon의 구조와 기능을 설명한다.

**3.3.2 클래스 다이어그램**

**그림은 다음 페이지에 게시되어 있다**. 클래스 다이어그램은 IntelliJ IDEA 내에서 그렸으며, Dependency가 설명되어 있다. 파란색 굵은 선은 상속을 의미하며, 점선으로 이어진 것은 해당 클래스를 사용하여 객체를 생성했거나, 클래스의 기능을 사용하고 있음을 보여준다.



**3.3.2 클래스별 설명**

각 클래스별로 { <패키지 이름> 클래스 이름 <상속> }와 같이 표시되었다.

**<src> Main**

1. REST 클래스를 이용하여 서버의 Mongo DB로 부터 userJSON 형태의 입력을 json 형태로 받아온다.
2. 받아들인 입력을 간단히 parsing 하여 DataManager를 통해 user\_data를 설정하고, Excon\_MapD에 데이터를 넘겨주어 Excon\_MapD의 ExecExperiment()를 이용해 실험을 수행한다.
3. 수행한 실험을 resultJSON 형태로 작성하여 REST를 이용해 Mongo DB에 게시한다.

**<src> Excon\_DBs**

추후에 추가할 DB들을 위한 abstract 클래스. JDBC를 연결할때 주요 정보들이 정의되어 있다.

**Field**:

int scale\_factor; → 사용할 workload의 scale\_factor

Connection conn = null; → JDBC Connection

Statement stmt = null; → JDBC Statement

ResultSet rs = null; → JDBC ResultSet

String TPC\_type; → 현재 테스트하는 TPC benchmark의 타입

String query; → query 문

String user; → DB user

String passwd; → DB password

String server\_address; → DB가 있는 서버의 주소

short port\_number; → JDBC가 통할 수 있는 DB의 포트 숫자

String JDBC\_DRIVER; → JDBC 드라이버의 이름

static String DB\_URL; → 완성된 DB의 URL

String DB\_name; → DB의 이름

int query\_number; → 현재 TPC benchmark의 쿼리 숫자

int grammar\_type; → 문법 유형

int iterations; → 실험 반복횟수

**Method**:

protected abstract void DB\_connect(); → 해당 DB에 연결

protected abstract Long send\_query(String \_query); → 해당 DB에 쿼리 전송

protected abstract ArrayList <?> ExecExperiment(); → 실험수행

**<src> Excon\_MapD < extends Excon\_DBs >**

**Field:**

userJSON Mapd\_config;

resultJSON Mapd\_result;

**Method:**

/\*-- 생성자 --\*/

Excon\_MapD(String \_user, String \_passwd)

Excon\_MapD(String \_user, String \_passwd, String address, short \_portnum)

Excon\_MapD(userJSON \_userdefined)

/\*-- set --\*/

public void setQueryNumber(int \_query) → query\_number 설정

public void setFromJSON(userJSON \_userdefined) → json 으로부터 Mapd\_config 설정

public void setServerPort(short \_portnum) → 서버 포트숫자 설정

public void setServerAddress(String Address) → 서버 주소 설정

/\*-- 쿼리 전송 --\*/

public void SendQueryByNumber()

public void SendQueryByNumber(int \_number)

protected ArrayList <Long> ExecExperiment() → 실험 수행

protected void DB\_connect() → DB 연결

protected Long send\_query(String \_query) → 쿼리 전송

**주의점:**

모듈화가 많이 이루어지지 않은 코드임. 특히, query를 보내는 부분은 조금 복잡해 보일 수 있음.

* + 1. **<src> REST**

**Method:**

public ArrayList <String> getInfo() → MongoDB로 부터 데이터를 받아오는 메소드

public void postInfo(resultJSON) → MongoDB로 데이터를 게시하는 메소드

**주의점:**

URL 주소가 hardcoded 되어있음.

* + 1. **<manage\_data> DataManager**

**Field:**

static userJSON user\_data;

static resultJSON result\_data;

ObjectMapper JsonManager;

**Method:**

public DataManager()

public DataManager(userJSON \_userdefined)

public void setUser\_data(userJSON \_userdefined)

public userJSON getUser\_data()

public resultJSON getResult\_data()

public userJSON getDataFromFile(String \_fname)

public void setDB(String \_name)

public void setResult\_data(ArrayList<Long> \_result)

public userJSON setDataByJson(userJSON \_userdefined)

**주의점:**

DataManager를 따로 떼어놓은 이유는, 추후 Thread를 이용할 때 Data collision을 피하기 위함이었다. 개발자의 선호와 필요에 의해 위 메소드들을 Excon\_DBs 내에 내장하거나, 삭제하는 것도 가능하다.

* + 1. **<manage\_data> userJSON**

**Filed:**

String project\_name;

String DB\_name;

String dblist[];

int dbnum;

int scale\_factor;

boolean gpu\_use = true;

int transaction\_refresh\_rate = 20;

String text;

String workload;

int iteration\_num = 10;

int query\_number;

String address = "155.230.34.143";

short portnum = 9091;

String user\_id = "mapd"; //db\_id

String user\_passwd = "HyperInteractive"; // db\_password

String \_id;

int cpu\_count;

int status;

int \_\_v;

**Method:**

각 필드에 대한 get 함수

* + 1. **<manage\_data> resultJSON**

**Field:**

String project\_name;

String exp\_result;

// avg, min, max 등 여러 통계 값을 처리해보면 됨

**3.3.3 현재 발견된 문제점**

* + 1. JDBC 연결을 위해 mapdql의 포트인 9091을 개방했다. 추후 보안에 문제가 생길 수도 있음.
    2. JDBC로 작성하는 경우, 기본으로 GPU 모드로만 작동한다. CPU 모드로 바꿀 수 있는 방법을 찾아보면 좋을 것임. $MAPD\_STORAGE 내에 mapd.conf 파일이 있으니 참고바람[a].
    3. CUPTI로 자세한 프로파일링을 할 수 있는 방법을 찾으면 좋을 것임. 리눅스상 CUDA설치시 같이 설치 되는 /usr/local/cuda/extras/CUPTI/sample/callback\_metirc/callback\_metric.cu 내용에 CUPTI의 metric API를 이용할 수 있는 방법을 볼 수 있음
    4. MapD 는 subquery를 지원하지 않는다[b]. subquery가 있는 SQL 코드를 풀어 쓸 수 있는 방법을 고안하면 좋을 것임.
    5. TPC-H의 일부 query는 MapD에서 에러로 인하여 동작하지 않는다[c].
    6. 지원한다고 되어 있는 일부 metric은 지원하지 않는 것[d][e]처럼 보인다. (CUDA ver 9.1)
    7. MapD JDBC에서 지원하지 않는 기본 JDBC의 메소드가 일부 있다.[[14]](#footnote-13)

**3.3.4 Future Works**

* + 1. **로그 분석자**(log analyzer) 개발이 필요하다. log4jdbc를 이용해 관련 쿼리에 대한 log를 생성하여 쿼리의 성공여부와 에러 종류를 출력하는 것이 필요할 것이다.
    2. **문법 교정자**(Grammar corrector)가 필요할 것이다. 각 DBMS 들은 쿼리를 프로그램에 맞게 문법을 수정하거나, 지원하지 않는 경우도 있다. 같은 기능을 수행할 수 있도록 하는 것이 관건이다.
    3. log4jdbc에서 도출해내는 시간 결과가, stmt.execute() 아래 위로 타이머를 삽입한 시간보다 짧게 나온다. 이에 대한 조사가 필요해보인다.
    4. CUPTI를 이용해 MapD에서 실행시 관련 metric을 조사해보는 것이 좋을 것 같다.CUPTI metric을 스냅샷으로 찍을 수 있다면, execute() 아래 위로 GPU 상태를 CUPTI를 이용해 스냅샷을 찍어 metric을 뽑아내는 것도 좋을 것 같음.
    5. mapdjdbc 상에서 **GPU cache 옵션을 끄는 방법**을 찾을 수 있으면 좋을 것이다. mapd.conf 파일에서 관련 옵션을 flag로 지정할 수 있는 부분은 보이지 않는다. java의 mapd 라이브러리 내에 clear\_gpu\_memory 메소드 군집이 있다. 실험을 시행할 때 이를 이용해 쿼리를 보내기 전 cache를 지우는 것을 시도해보는 것도 괜찮을 것 같다.
    6. 분석할 DB를 추가해야한다. ExCon\_DBs 클래스를 이용하는 것이 편리할 것이다.

**3.3.5 함께 볼 문서들**

* 1. [Configuration Flags and Runtime Settings in MapD](https://www.mapd.com/docs/latest/4_configuration.html)
  2. type not serializable error에 대한 MapD [개발자의 답변](https://community.mapd.com/t/type-not-serializable-error-meaning/1337)
  3. Query generation 정리
  4. cupti\_metirc
  5. [NVIDIA CUPTI 문서](https://docs.nvidia.com/cuda/cupti/index.html)

1. <https://github.com/BevoLEt/BenchGAD_0> 에 BenchGAD 관련 코드,문서가 다 저장되어있다. [↑](#footnote-ref-0)
2. 서버에서 사용될 javascript 언어로 된 코드 설명과 클라이언트, Excon에 사용될 java 언어로 된 코드 설명을 포함한다. [↑](#footnote-ref-1)
3. jackson-all-1.9.0.jar 와 jackson-databind-2.9.6.jar 를 사용하고 있다. [↑](#footnote-ref-2)
4. BenchGAD\_Server 디렉토리 현황이다. [↑](#footnote-ref-3)
5. 코드에 대한 설명은 index.js 의 코드에 코멘트로 남겼다. [↑](#footnote-ref-4)
6. 되도록이면 terminal 에서 명령어를 통한 설치를 권장한다. [↑](#footnote-ref-5)
7. node js 패키지 설치 시 npm 을 사용한다. npm 커맨드가 동작하지 않는다면 node js 설치가 제대로 안되었다는 뜻이니 재설치를 권장한다. [↑](#footnote-ref-6)
8. 현재 BenchGAD 서버는 포트 3000을 사용 하고 있다. [↑](#footnote-ref-7)
9. dkelab 연구실의 경우 workstation 에 BenchGAD vm 에서 실행하면 된다. (pw는 관리자에게 물어보도록..) [↑](#footnote-ref-8)
10. EPEL 을 필요로 하는 경우, [epel 배포 페이지](https://fedoraproject.org/wiki/EPEL)에서 epel-release-latest-7 을 선택하여 다운로드 후 설치한다. [↑](#footnote-ref-9)
11. 다른 번호의 데이터는 용량이 커 삽입에 시간이 걸린다. [↑](#footnote-ref-10)
12. ID = mapd PASSWD = HyperInteractive (대소문자 구분) [↑](#footnote-ref-11)
13. 생성되는 dbgen 및 qgen 파일은 <tpc-h\_version>/dbgen/ 디렉토리 안에서만 사용할 것을 권장한다. [↑](#footnote-ref-12)
14. mapdjdbc 라이브러리가 큰 규모인 편이라 모든 리스트를 작성할 수 없다. 메소드를 사용하려할 경우 error 메시지를 출력하니 참고할 것. [↑](#footnote-ref-13)