목차

[◎ 목적과 조건 및 환경 2](#_Toc153360050)

[개발 목적 2](#_Toc153360051)

[개발 환경 2](#_Toc153360052)

[조건 4](#_Toc153360053)

[◎ 기술 설명 6](#_Toc153360054)

[QT 6](#_Toc153360055)

[SQLite Database 6](#_Toc153360056)

[HW 7](#_Toc153360057)

[SW 7](#_Toc153360058)

[◎ 설계 차트 10](#_Toc153360059)

[Service와 DB 그리고 QT 시퀀스 다이어그램 10](#_Toc153360060)

[service의 플로우 차트 12](#_Toc153360061)

[QT의 플로우 차트 13](#_Toc153360062)

[SQLtie Database 구상 14](#_Toc153360063)

[◎ 테스트 15](#_Toc153360064)

[QT 테스트 15](#_Toc153360065)

[SQLite 테스트 26](#_Toc153360066)

[HW 정보 수집 테스트 28](#_Toc153360067)

[SW 정보 수집 테스트 36](#_Toc153360068)

[로그 파일 테스트 38](#_Toc153360069)

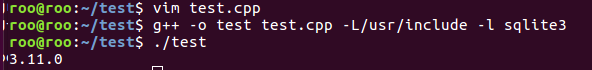
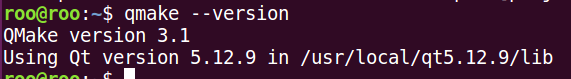
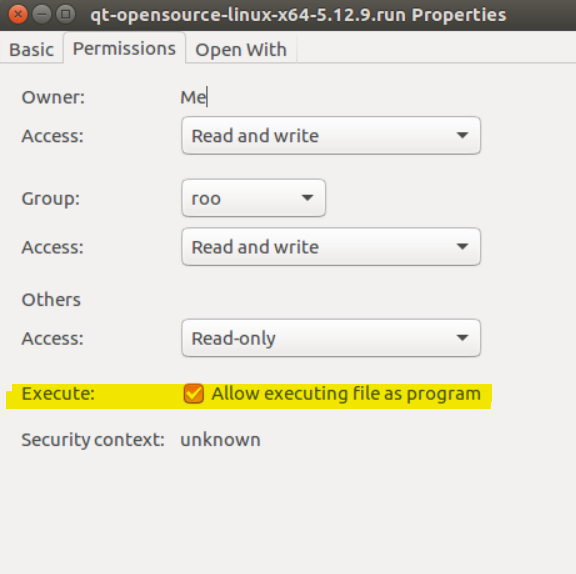
# ◎ 목적과 조건 및 환경

-L/usr/include -lsqlite3

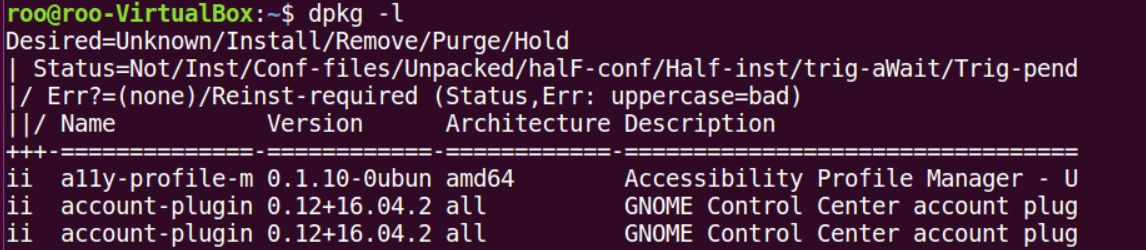
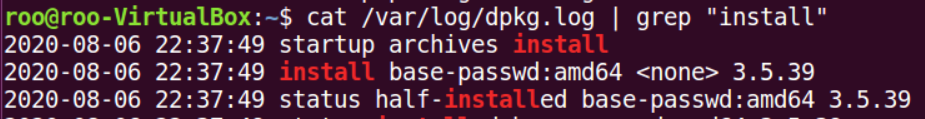
## 개발 목적

* PC에 존재하는 HW 정보와 설치된 SW 정보를 수집하여 DB에 저장하는 Service를 개발한다

## 개발 환경

* Ubuntu OS를 사용한다.
  + 운영 체제 : Ubuntu (64-bit, 16.04 버전)
  + 기본 메모리 : 4096MB
  + 프로세서 : 2
  + 저장소 : 200GB
* C++ 언어를 사용한다.
  + **$ sudo apt-get update** / 저장소 업데이트
  + **$ sudo apt-get install g++** / g++ 설치
  + **$ g++ --version** / 설치된 버전 확인
  + **$ sudo apt-get install vim** / vim 설치
* SQLite DB를 사용한다. (<https://yooloo.tistory.com/59>) (<https://basiclike.tistory.com/173>)
  + **$ sudo apt-get install libsqlite3-dev** / sqlite3 설치
  + **$ sqlite3 --version** / sqlite3 버전 확인
  + **$ sqlite3** / sqlite3 실행
  + **.quit or .exit** / sqlite3 종료
  + **$ g++ -o [생성 파일명] [파일명.cpp] -L/usr/include -l sqlite3** / 컴파일 방법  
      
    <C++을 이용한 SQLite의 버전 확인 테스트>
* QT 라이브러리를 사용한다. 버전은 5.12.9 버전을 사용한다.
  + **$ sudo apt-get update** / 저장소 업데이트
  + **$ sudo apt-get install build-essential libgl1-mesa-dev** / 필수 개발환경 패키지 설치
  + **$ wget https://download.qt.io/archive/qt/5.12/5.12.9/single/qt-everywhere-src-5.12.9.tar.xz** / qt-everywhere-src-5.12.9.tar.xz다운로드
  + **$ tar -xJf qt-everywhere-src-5.12.9.tar.xz** / qt-everywhere-src-5.12.9.tar.xz 압축 해제
  + **$ cd qt-everywhere-src-5.12.9** / qt-everywhere-src-5.12.9 폴더로 이동
  + **$ ./configure -opensource -confirm-license -prefix /usr/local/qt5.12.9** / 오픈 소스 라이선스로 QT를 빌드, 라이선스 동의를 확인하지 않고 자동으로 동의, 빌드된 QT를 /usr/local/qt5.12.9 디렉토리에 설치하도록 설정
  + **$ make** / make 파일 생성
  + **$ sudo make install** / make 파일 설치
  + **$ echo 'export PATH=/usr/local/qt5.12.9/bin:$PATH' >> ~/.bashrc  
    $ source ~/.bashrc** / 환경변수 설정
  + **$ qmake --version** / QT 버전 확인  
    
  + **$ sudo apt install qtcreator** / QT Creator 설치
  + **$ qtcreator** / QT Creator 실행
  + 혹은 다음 링크에서 qt-opensource-linux-x64-5.12.9.run를 다운로드 한 후 설치한다.(<https://download.qt.io/archive/qt/5.12/5.12.9/>)
    - 설치 전에, 다운로드 한 파일을 우클릭을 하여 Properties에 들어가 아래 버튼을 클릭한다.  
      

## 조건

* 설치된 SW 정보는 deb과 rpm으로 설치된 기준으로 판단한다.
  + rpm
    - Ubuntu 환경임으로 rpm을 설치하기 위해 먼저 alien을 설치한다.
      * **$ sudo apt-get install alien** / alien 설치
    - **$ sudo apt-get install rpm** / rpm 다운로드
    - **$ rpm -qa** / rpm으로 설치한 패키지들을 보여준다. 설치된 패키지명과 버전만 출력한다.  
      ex) php-5.3.28-2.w5
    - **$ yum list installed** / rpm(or yum)으로 설치된 패키지들을 보여준다. 패키지명, 버전, 설치 여부를 출력한다.  
      ex) php.x86\_64 5.3.25-2.w5 installed
  + deb
    - **$ dpkg -l** / deb 형식으로 설치된 패키지들을 보여준다. 패키지명, 버전, 아키텍쳐, 설명을 출력한다.  
      
    - **$ cat /var/log/dpkg.log | grep “install ”**
      * cat은 파일을 출력하거나 여러 파일을 연결해 주는 명령어이다. cat [경로/파일] 을 한다면 해당 경로에 있는 파일의 내용을 출력한다.  
        설치 날짜, 설치 시간, 동작, SW명, 아키텍쳐, 이전버전, 설치버전 의 형식으로 출력된다.  
        
      * | grep “” 는 “ “ 안의 문자열을 찾아낸 부분만을 출력하게 바꿔준다. “위의 명령어는 “install “의 문자열을 찾아 해당 문자열이 존재하는 것만 출력해준다. install 이 되어있는 것만 찾는 이유는 설치된 것만 찾아내기 위함이다.
* 주기적으로 HW와 SW 정보를 확인하여 DB에 저장한다.
  + HW 정보
    - OS 정보
    - NIC
    - 메모리
    - CPU
  + SW 정보
    - SW 명
    - SW 설치 버전
    - 설치된 날짜
* QT를 이용하여 DB에 저장되어 있는 SW 정보 출력 되는 GUI를 개발한다.
  + service에서 DB에 저장한 SW 정보를 QT가 읽어와 GUI로 보여준다.
  + 모든 정보를 가져온 후, 사용자가 선택하면 선택한 것만 볼 수 있도록 만든다.  
    ex) SW명 선택 시 SW명의 정보만 보이며, SW 설치 정보, 설치된 날짜는 보이지 않는다.
* 개발되는 service와 GUI에 대해서 동작 분석 및 검색이 가능 하도록 로그 파일을 남긴다.
  + 로그 형식은 다른 로그 텍스트 파일을 참고하여 만든다.  
    ex) 로그날짜 로그시간 로그내용

# ◎ 기술 설명

## QT

* QT는 기본 언어를 C++로 하는 GUI제작 크로스 플랫폼 프레임워크이다.
* C++를 주로 사용하지만, 파이썬, 루비, C, 펄, 파스칼과도 연동된다.
* SQL 데이터베이스 접근, XML 처리, 스레드 관리, 단일 크로스 플랫폼 파일 관리 API를 제공한다.
* UI 구성 목적으로 사용한 것으로 생각된다.
* SQLite에 저장된 SW 정보를 가져와 출력한다.
* Qt Designer라는 위지위그 방식 툴이 존재해 초보자가 사용하기 쉽다.
* QWidget을 상속한 모든 클래스에서 setStyleSheet를 이용해 css파일을 사용하여 편집할 수 있다.
* 윈도우 환경에서 지원을 잘 못해준다는 단점이 존재한다. 또한 UI가 현대화되어 있지 않다는 단점이 존재한다.
* 대체 라이브러리로는 GTK, wxWidgets 등이 존재한다.

## SQLite Database

* 관계형 데이터베이스 관리 시스템을 제공하는 소프트웨어 라이브러리다. 설정, 데이터베이스 관리 및 리소스 측면에서 가볍다.
* 서버리스, 독립실행형, Zero-configuration, 트랜잭션 처리 등의 기능을 가지고 있다.
* 동적 유형을 사용하는 특징을 가지고 있다. 데이터 유형에 관계 없이 모든 열에 모든 값을 저장할 수 있다.
* 단일 데이터베이스에 연결하여 여러 데이터베이스 파일에 동시에 액세스 할 수 있다. 이는 다른 데이터베이스의 테이블을 조인하거나 단일 명령으로 데이터베이스 간에 데이터를 복사할 수 있다.
* 동시에 여러 사용자가 데이터베이스에 접근하는 것을 제한한다는 단점이 존재한다.
* 저장 프로시저, 트리거 등과 같은 데이터베이스 기능이 제한적이거나 지원되지 않을 수 있다.
* 컬럼에 데이터 타입을 지정할 수는 있지만 필수는 아니다. 만약 지정해준다면 유형은 아래와 같이 나누어진다.  
  
  + TEXT 타입의 컬럼에 INTEGER 또는 REAL 데이터 타입의 값이 포함 된 경우, TEXT 타입으로 변환되어 저장된다.
  + NUMERIC 타입의 컬럼에 TEXT 타입의 값이 포함 된 경우, INTEGER 타입 또는 REAL 타입으로 변환을 해보고 성공하면 데이터 타입으로 저장되지만, 실패하면 TEXT 타입으로 저장된다.
  + INTEGER 타입의 컬럼에 정수로 나타낼 수 있는 REAL 타입의 값(예 : 34.0 등), 또는 같은 타입의 TEXT 타입의 값이 포함된 경우 INTEGER 형으로 변환되어 저장된다.
  + REAL 타입의 컬럼에 INTEGER 값이 포함 된 경우 REAL 타입으로 변환되어 저장한다.
  + NONE 타입의 컬럼의 경우는 변환이 처리가 없다.
* 컬럼에 데이터 타입을 지정하는 방법의 예시이다.
  + create table test (id int, name text); / integer 대신에 int로 작성이 가능하다.
  + create table test(id integer, name varchar); / text 대신에 varchar로 사용이 가능하다.

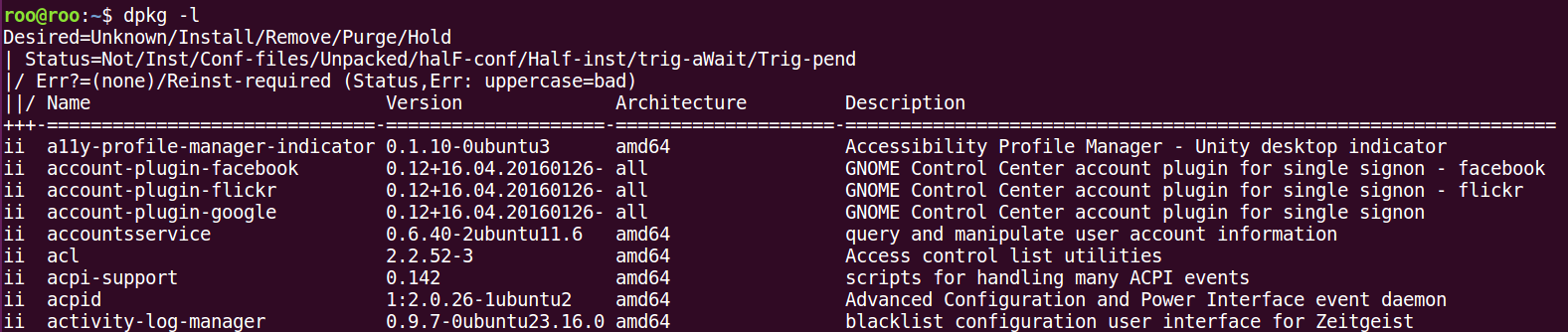
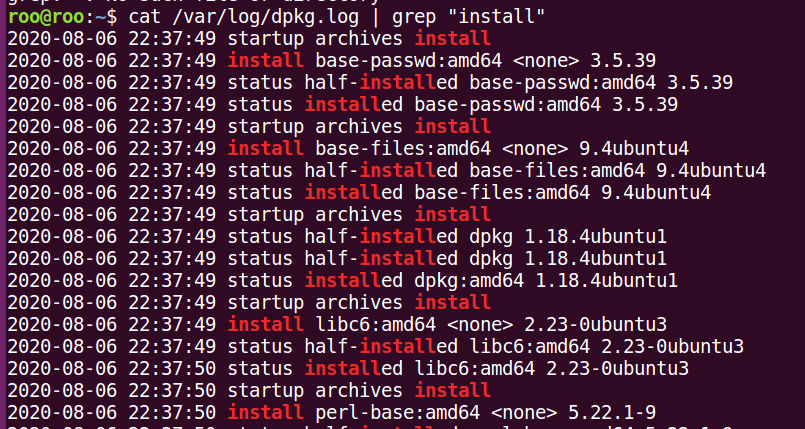
## HW

HW의 OS 정보, NIC, Memory, CPU 정보를 수집해야 한다.

* OS 정보 **$ cat /etc/os-release** / OS 정보를 확인할 수 있다.
* CPU 정보  
  **$ cat /proc/cpuinfo** / CPU 정보를 확인할 수 있다.
* NIC 정보  
  **$ cd /sys/class/net** / 각각의 네트워크 어뎁터 폴더에서 NIC 정보를 확인할 수 있다.
* Memory 정보  
  **$ cat /proc/meminfo** / Memory 정보를 확인할 수 있다.

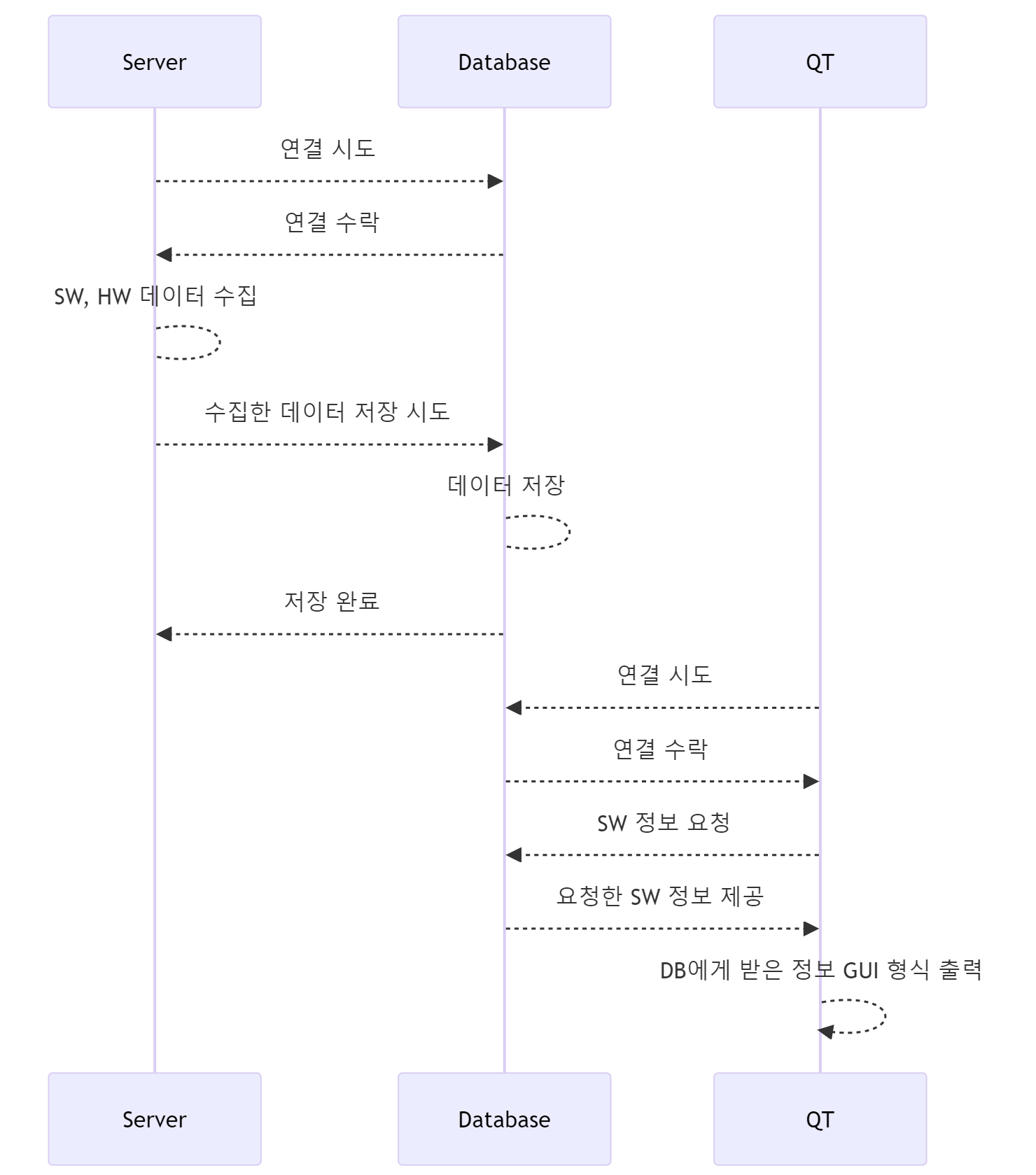
## SW

deb와 rpm으로 설치된 SW 정보(SW 명, SW 설치 버전, 설치된 날짜)를 수집해야 한다.

* deb  
  .deb란 데비안의 소프트웨어 패키지 포맷의 확장자이며 데비안 소프트웨어 포맷의 바이너리 패키지에서 자주 사용되는 파일 이름이다.  
  dpkg에 의해 관리 유지 되며, dpkg는 데비안, 우분투, 칼리, 민트 등에서 사용되는 패키지 관리 도구이다.  
  추가로 apt 역시 dpkg를 사용해서 만든 고급 관리 도구이다.
  + **$ dpkg -l** / 설치된 패키지 목록 확인
    - dpkg는 데비안 패키지 관리 시스템의 기초가 되는 소프트웨어이다.
    - deb 패키지(확장자가 .deb로 끝나는 파일)의 설치, 삭제를 위해 사용된다.
    - dpkg -l 는 설치된 패키지 목록을 보여준다.  
      SW명, 버전, 아키텍쳐, 설명으로 구분해 정보를 보여준다.  
      
  + **$ cat /var/log/dpkg.log | grep “install ”**
    - cat은 파일을 출력하거나 여러 파일을 연결해 주는 명령어이다. cat [경로/파일] 을 한다면 해당 경로에 있는 파일의 내용을 출력한다  
      즉, 위의 명령어에선 /var/log/경로에 있는 dpkg.log의 내용을 출력한다.
    - grep을 사용해 설치된 파일이란 의미인 “install “이 포함되어 있는 문장만 출력한다.  
      설치 날짜 / 설치 시간 / 동작 / SW명 / 아키텍쳐 / 이전버전 / 설치버전으로 구분해 정보를 보여준다.  
      
  + 위의 두 가지 명령어 중 SW명, SW 설치 버전 뿐만 아니라 설치된 날짜 또한 수집을 해야하므로 **$ cat /var/log/dpkg.log | grep “install ”** 명령어를 사용한다.
* rpm(RedHat Package Manager)  
  rpm이란 레드햇 계열(Redhat, CentOS, Fedora)의 리눅스 배포판에서 사용하는 프로그램(패키지) 설치 관리 도구이다. \*.rpm 식의 파일 이름을 갖고 패키지의 버전과 실행 환경에 따라 규칙성 있는 패키지 명을 가지게 된다.  
  레드햇 계열에서 사용이 되지만 데비안 계열(Debian, Ubuntu)에서 Alien을 이용해 \*.rpm 파일을 직접적으로 설치하거나 \*.deb 형식으로 변경하여 설치할 수 있다.
  + **$ sudo alien -i [RPM파일명]** / 직접적으로 바로 설치하는 명령어
  + **$ sudo alien -c [RPM파일명]** / rpm 파일을 deb 파일로 변환 명령어 **sudo dpkg -i [Deb로 변환된 파일명]** / deb로 변환시킨 파일 설치 명령어

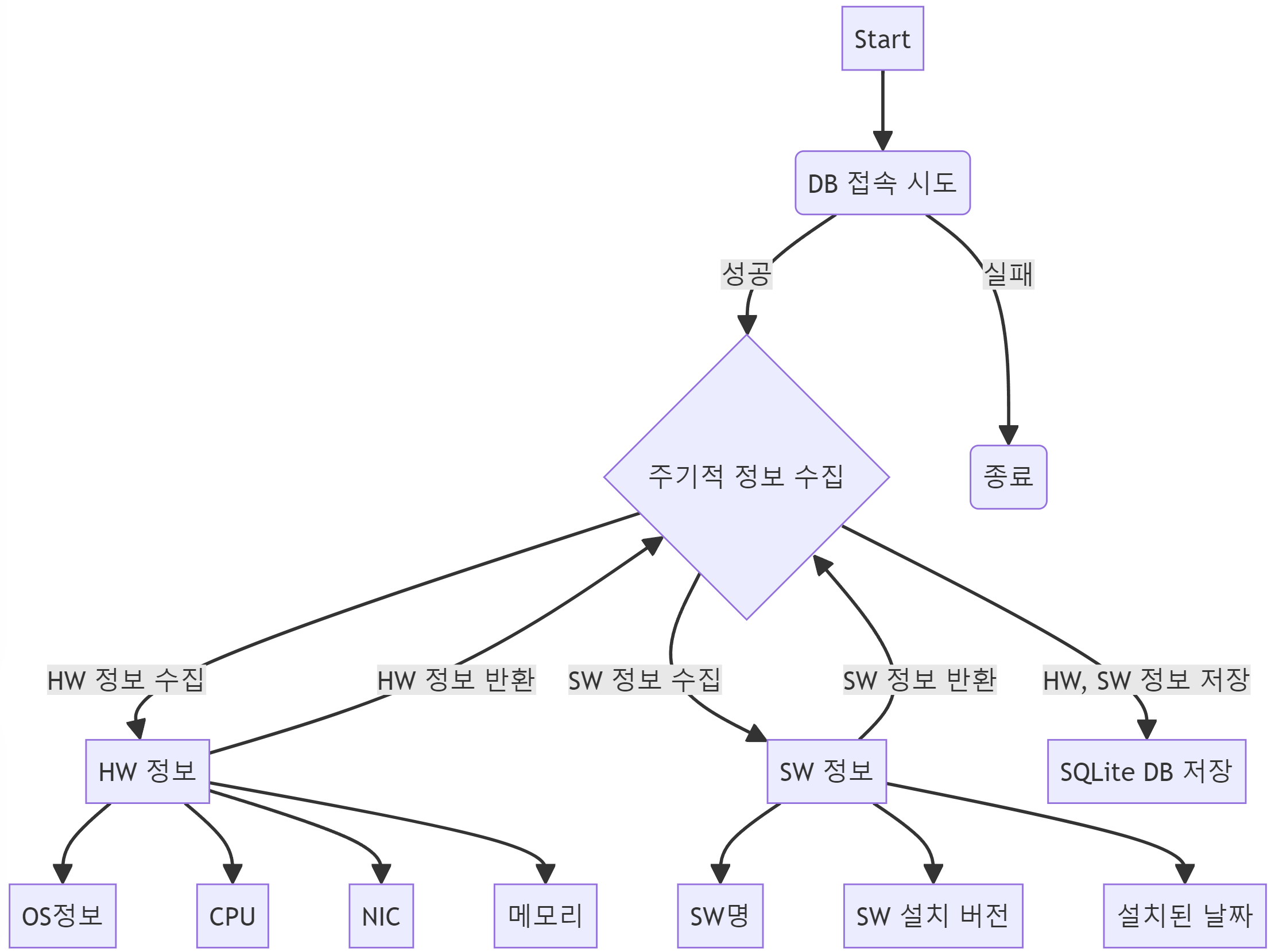
# ◎ 설계 차트

## Service와 DB 그리고 QT 시퀀스 다이어그램



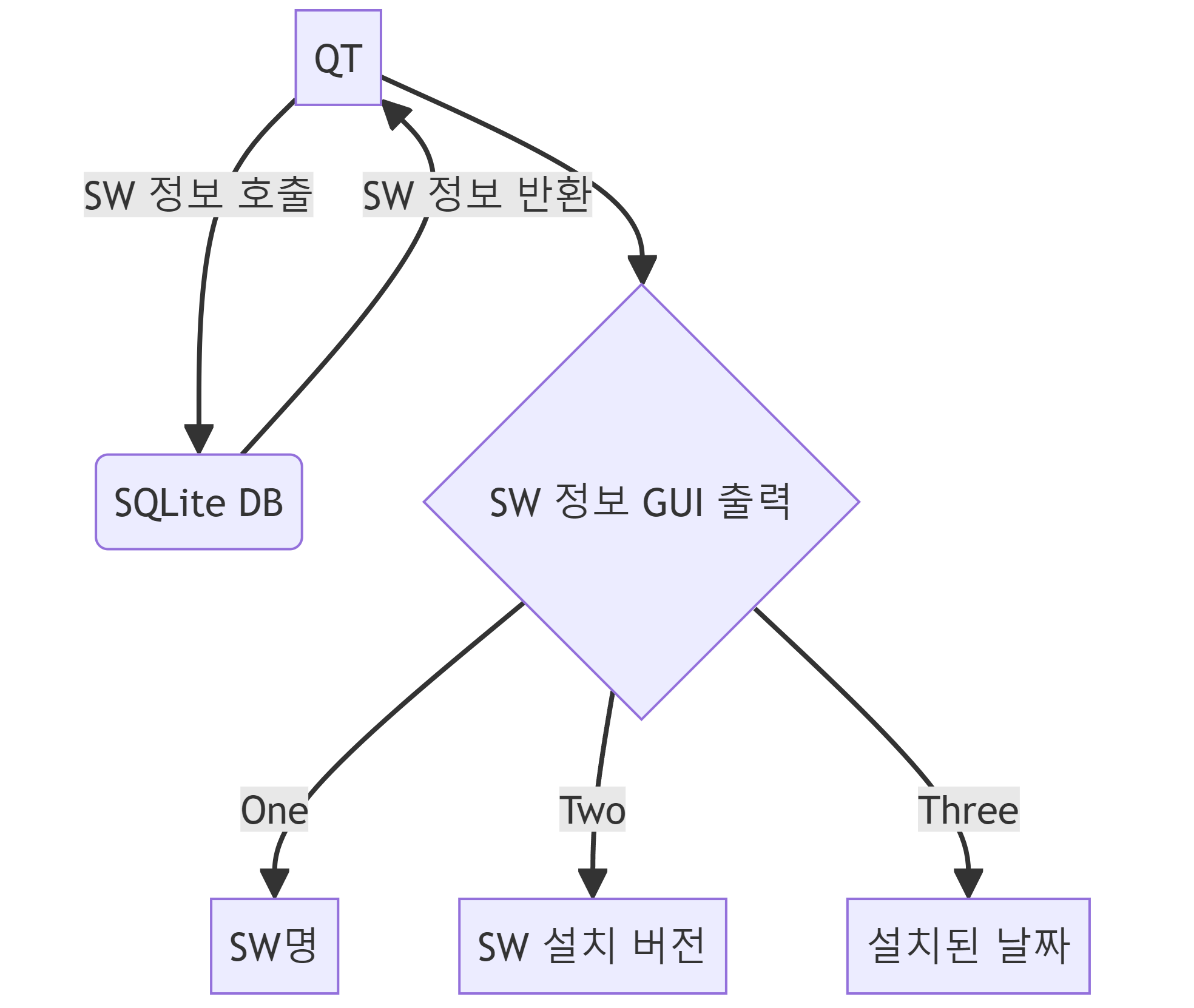
1. Server에서 DB에 연결을 시도하고 DB에 접근이 가능하다면 수락한다.
2. DB와 연결이 된다면 SW와 HW의 정보를 수집 후 DB에 넘긴다.
3. 넘겨받은 정보에 해당하는 DB, Table, Column이 존재한다면 정보를 저장한 후 저장에 성공했다고 알려준다.
4. QT에서 DB에 연결을 시도한다. DB는 접근에 이상이 없다면 연결이 된다.
5. QT에서 DB에게 Server가 저장한 SW 정보를 요청한다. DB는 QT에게 저장된 SW 정보를 넘겨준다.
6. QT에서 GUI 형식으로 SW 정보를 보여준다.

## service의 플로우 차트



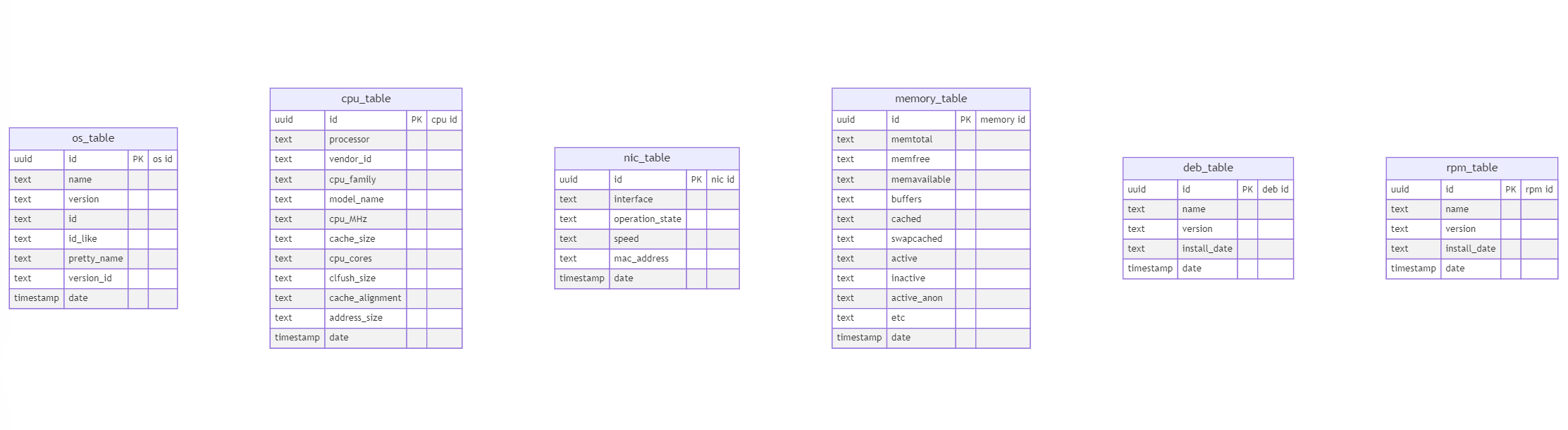
1. 코드 실행
2. SQLite DB 접속
3. 주기적으로 HW(OS정보, CPU, NIC, 메모리)정보와 SW(deb와 rpm의 SW명, SW 설치 버전, 설치된 날짜)정보 수집
4. 수집된 정보를 SQLite DB에 저장

## QT의 플로우 차트



1. QT에서 SQLite DB에 저장되어 있는 SW 정보 수집
2. 수집한 SW(deb와 rpm의 SW명, SW 설치 버전, 설치된 날짜)정보를 GUI 형식으로 출력

## SQLtie Database 구상

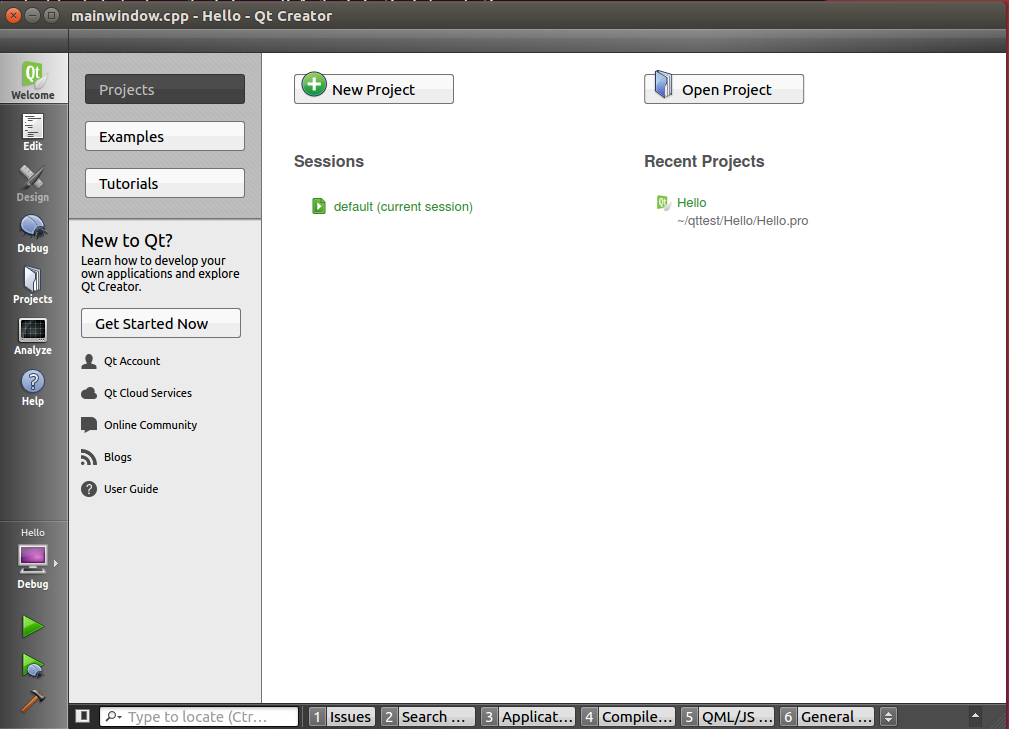
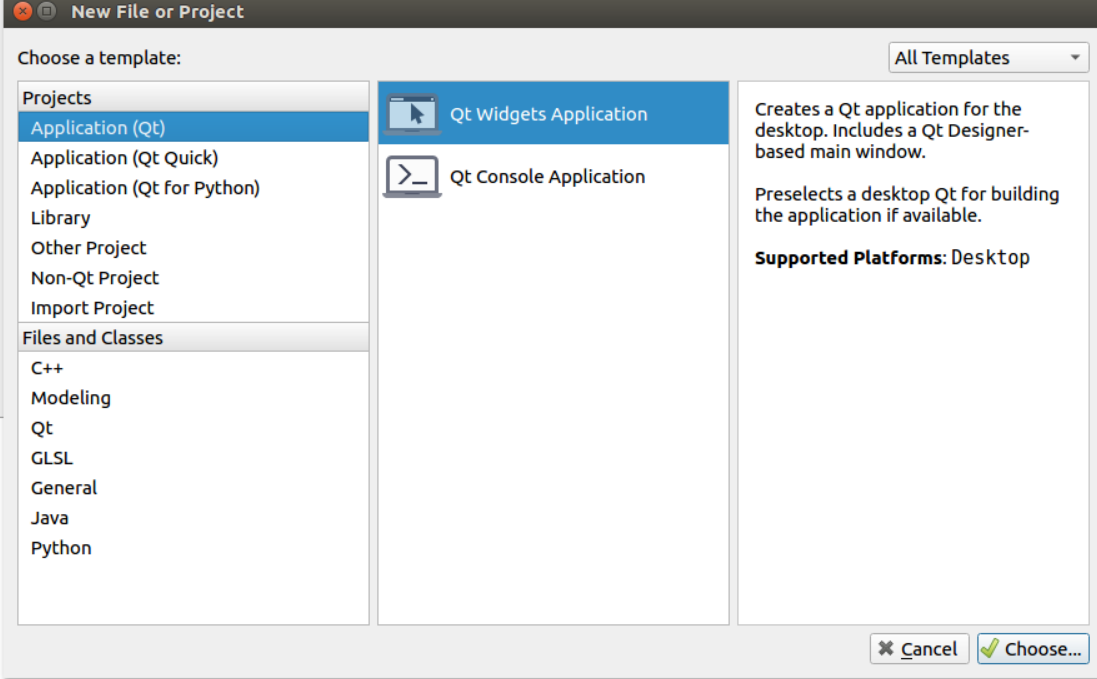
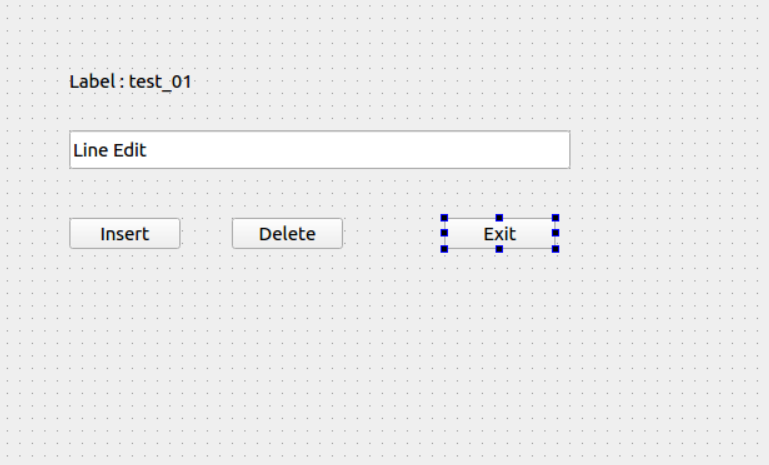
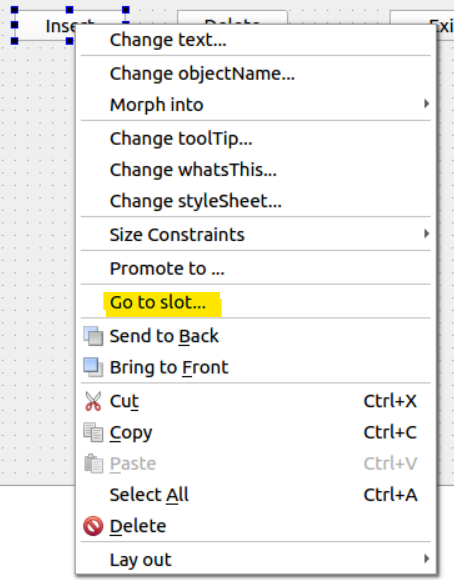
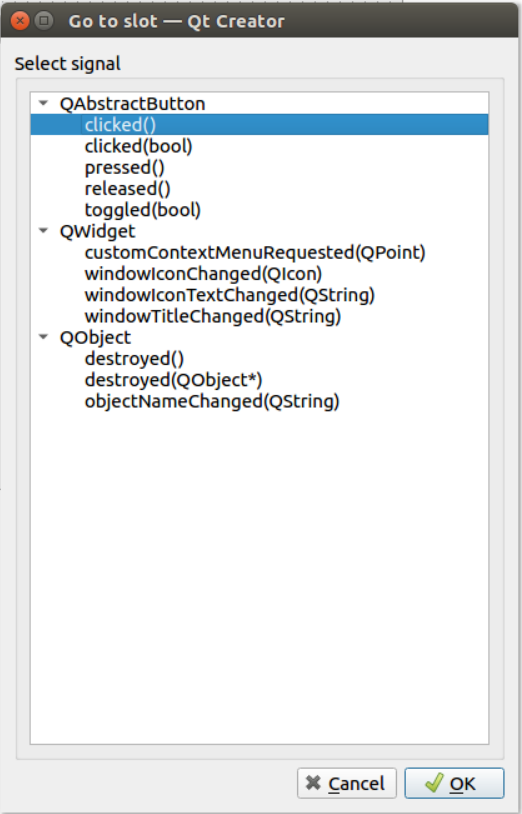
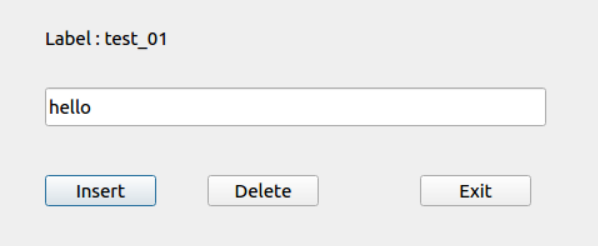
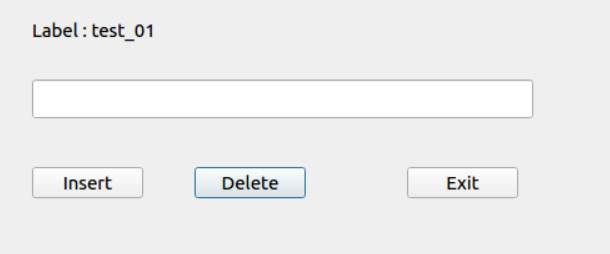
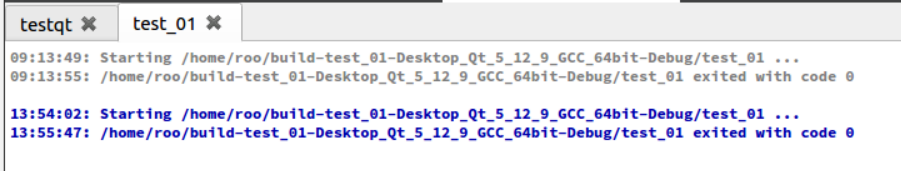


* os\_table
  + name, version, id, id\_like, pretty\_name, version\_id정보를 text 형식으로 저장한다.
  + 데이터가 삽입 된 시간을 timestamp를 사용해 date에 저장한다.
* cpu\_table
  + processor, vendor\_id, cpu\_family, model\_name, cpu\_MHz, cache\_size, cpu\_cores, clfush\_size, cache\_alignment, address\_size 정보를 text 형식으로 저장한다.
  + 데이터가 삽입 된 시간을 timestamp를 사용해 date에 저장한다.
* nic\_table
  + interface, operation\_state, speed, mac\_address 정보를 text 형식으로 저장한다.
  + 데이터가 삽입 된 시간을 timestamp를 사용해 date에 저장한다.
* memory\_table
  + memtotal, memfree, memavailable, buffers, cached, swapcached, active, inactive, active\_anon, 등의 정보를 text형식으로 저장한다.
  + 데이터가 삽입 된 시간을 timestamp를 사용해 date에 저장한다.
* deb\_table
  + sw\_name, sw\_version, sw\_install\_data의 정보를 text 형식으로 저장한다.
  + 데이터가 삽입 된 시간을 timestamp를 사용해 date에 저장한다.
* rpm\_table
  + sw\_name, sw\_version, sw\_install\_data의 정보를 text 형식으로 저장한다.
  + 데이터가 삽입 된 시간을 timestamp를 사용해 date에 저장한다.

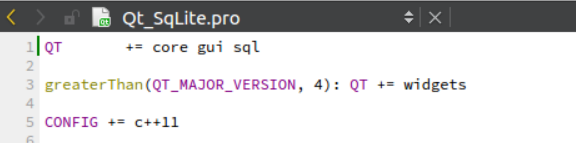
# ◎ 테스트

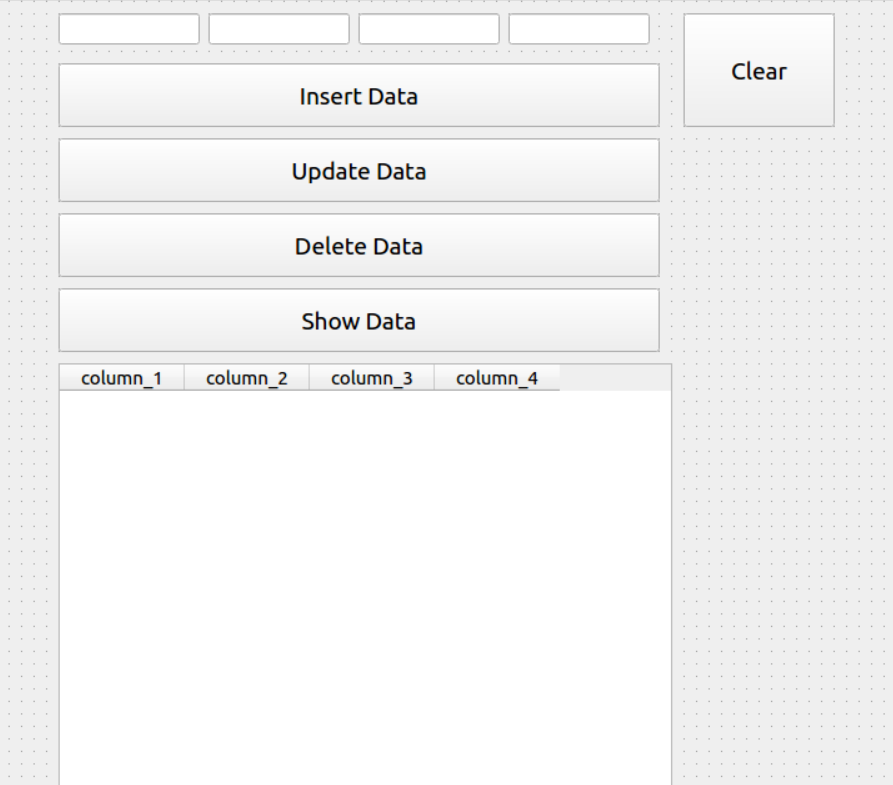
## QT 테스트

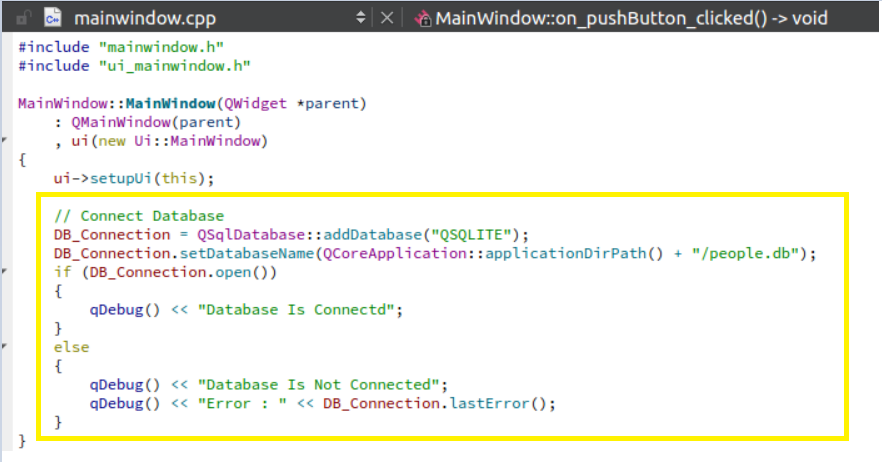
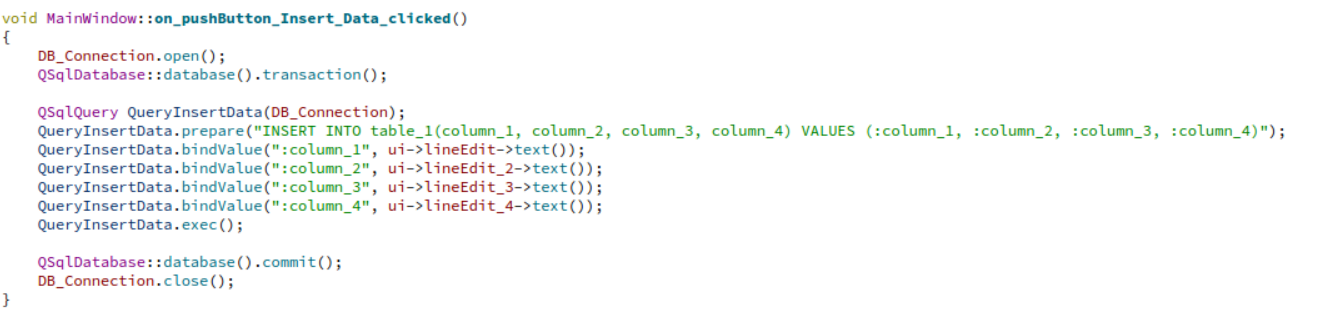
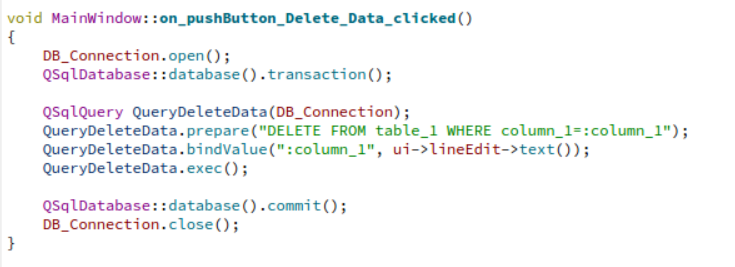
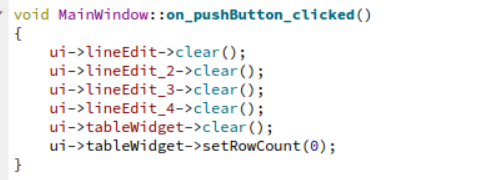
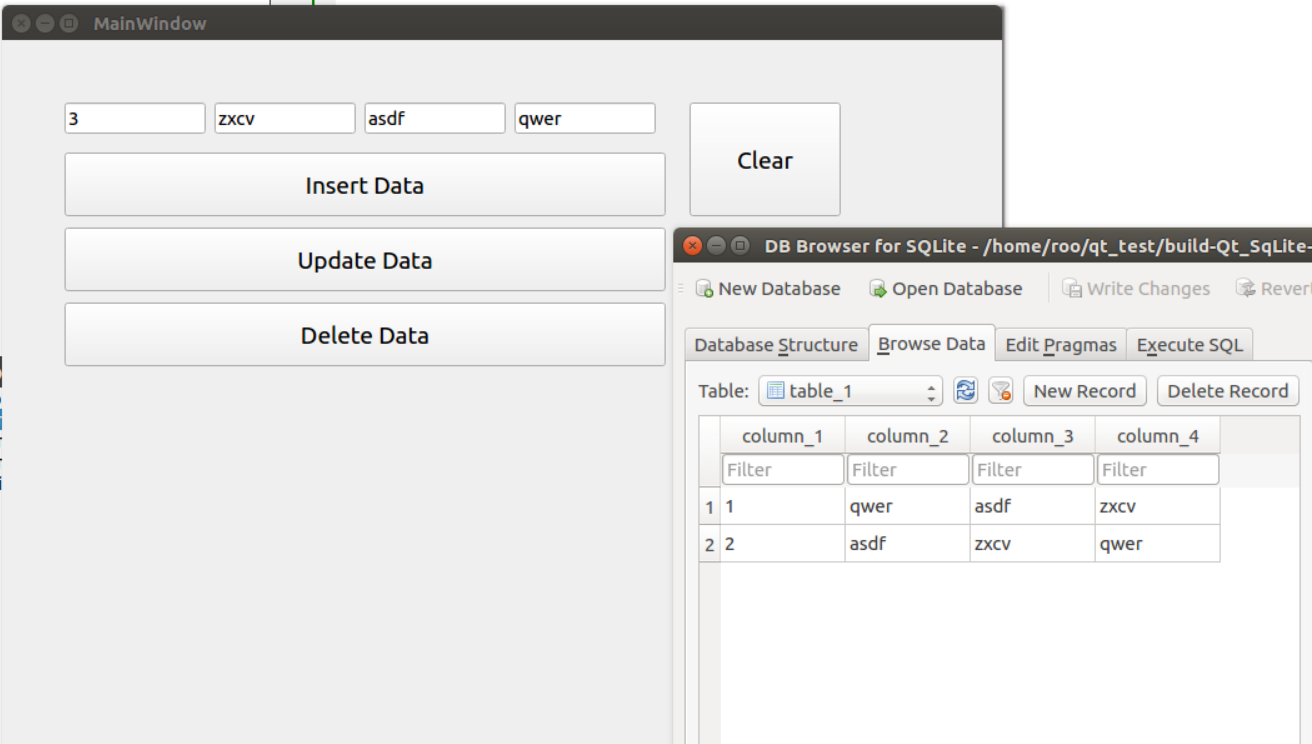
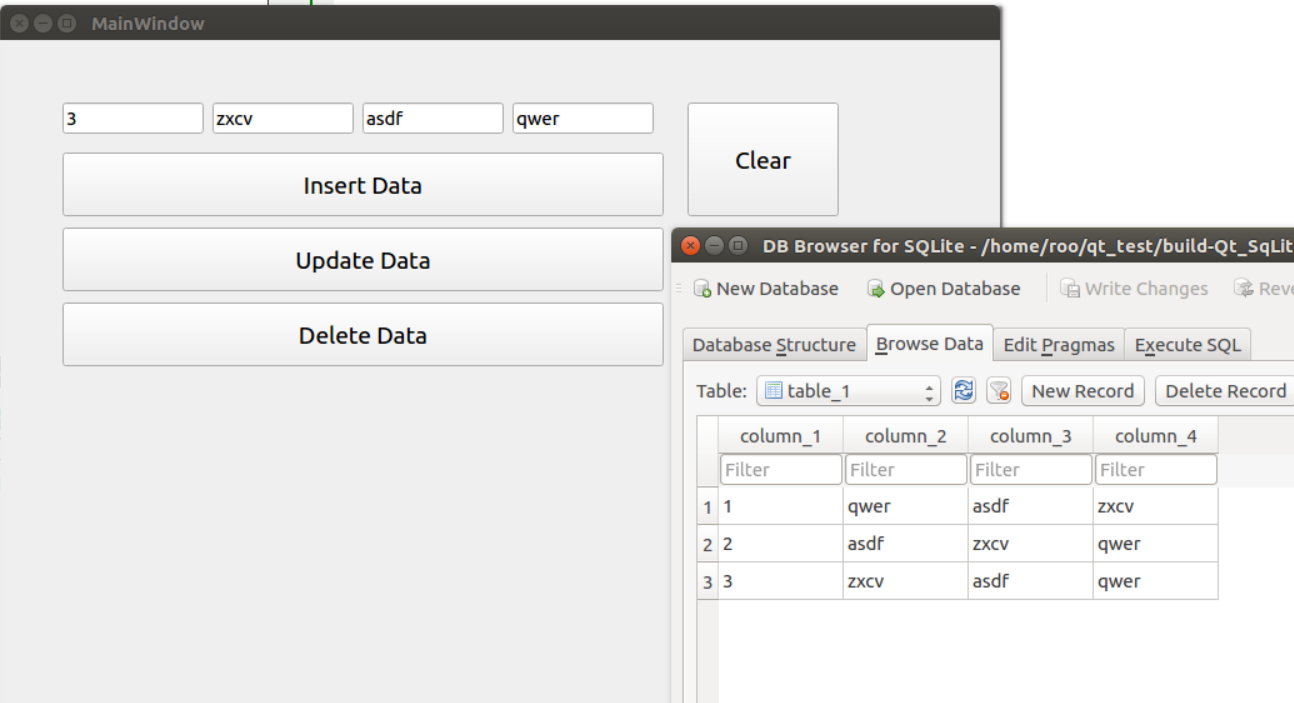
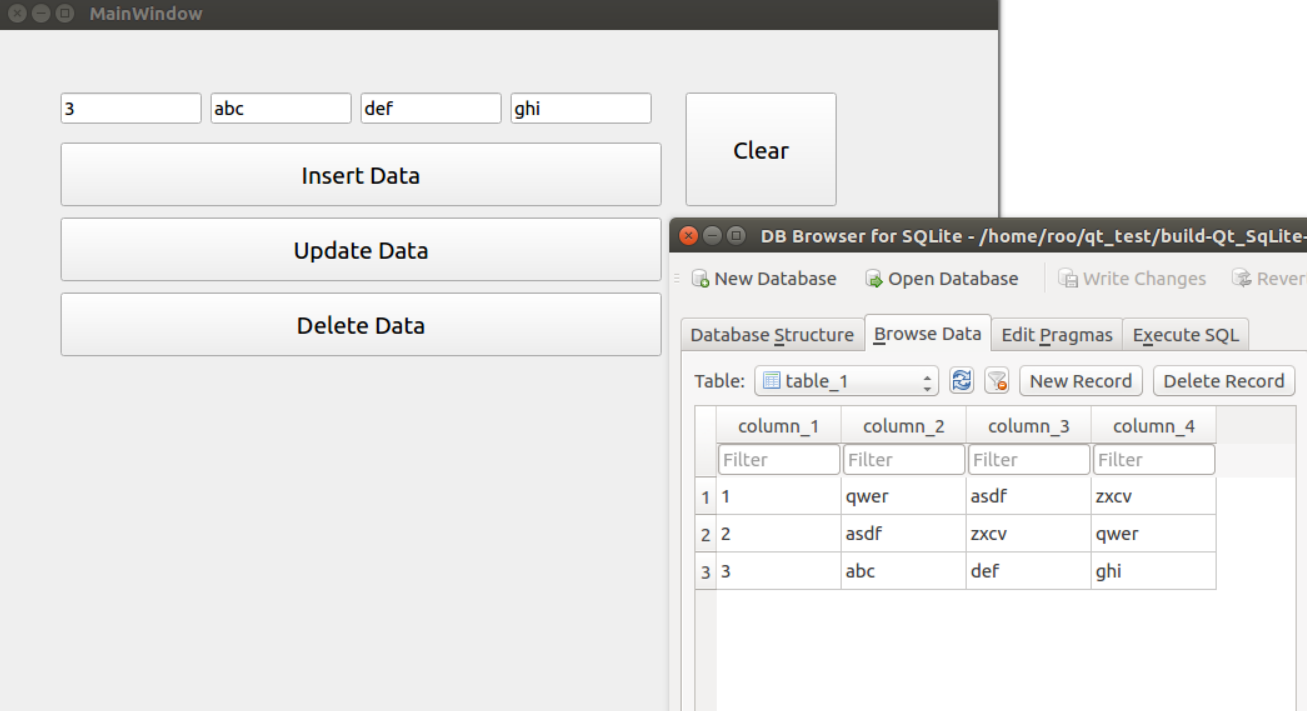
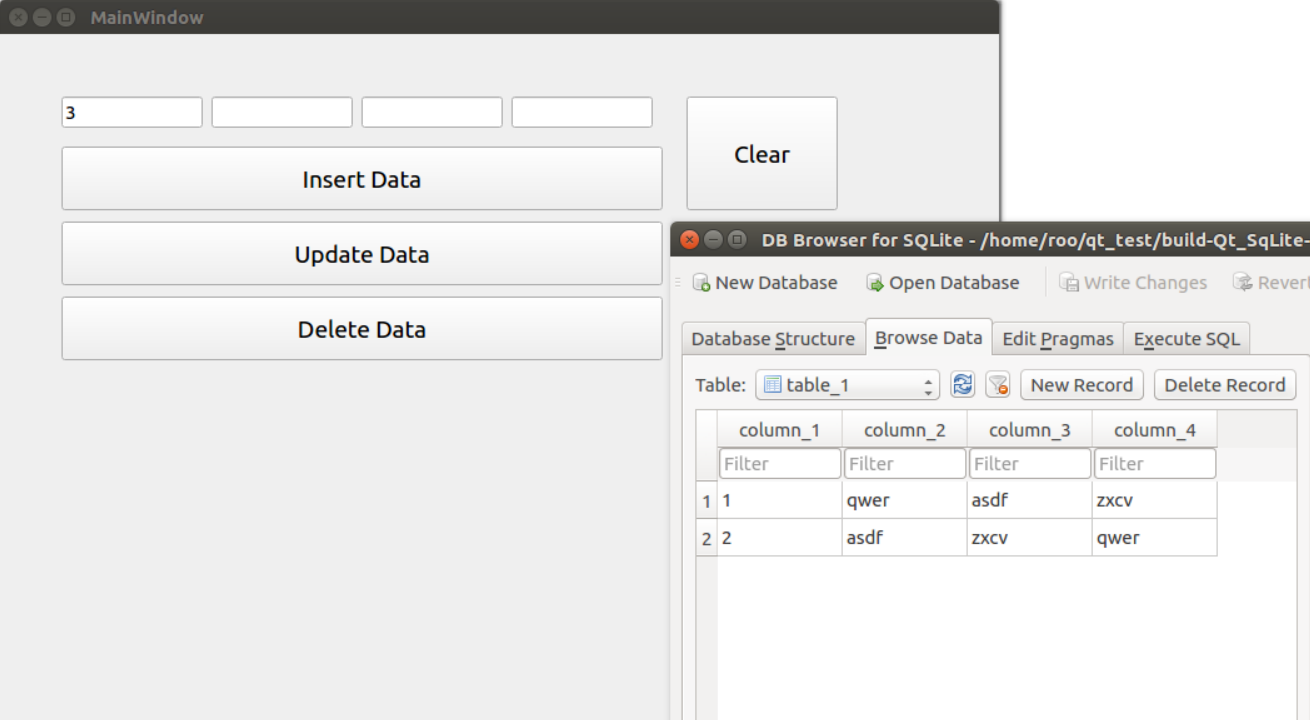
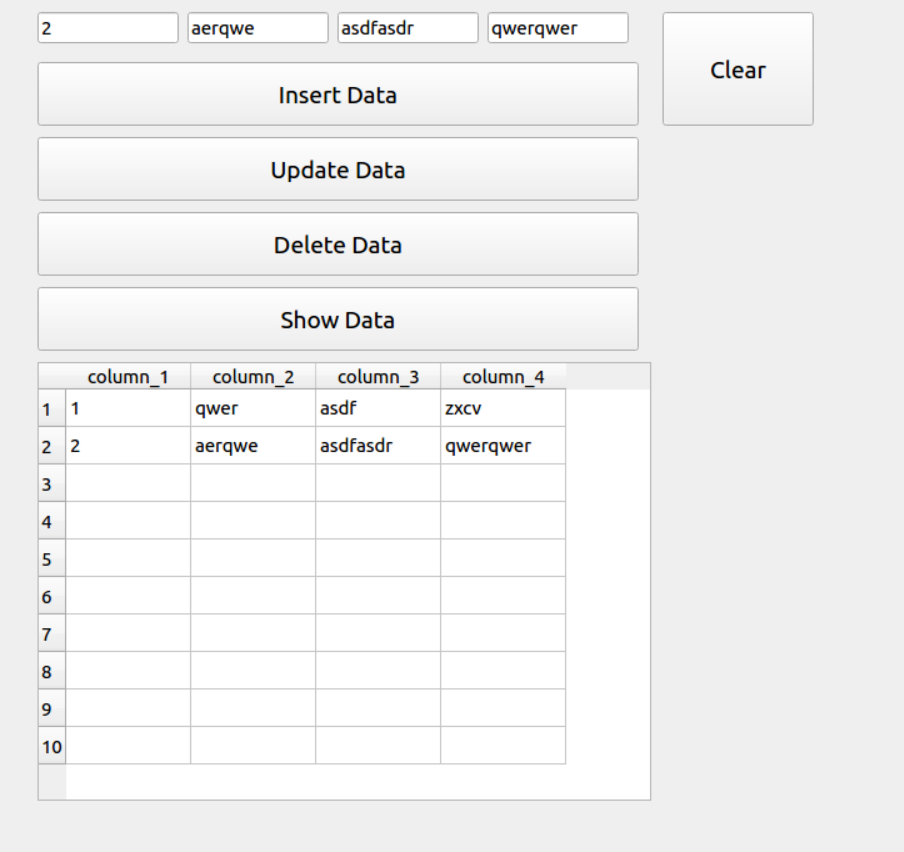
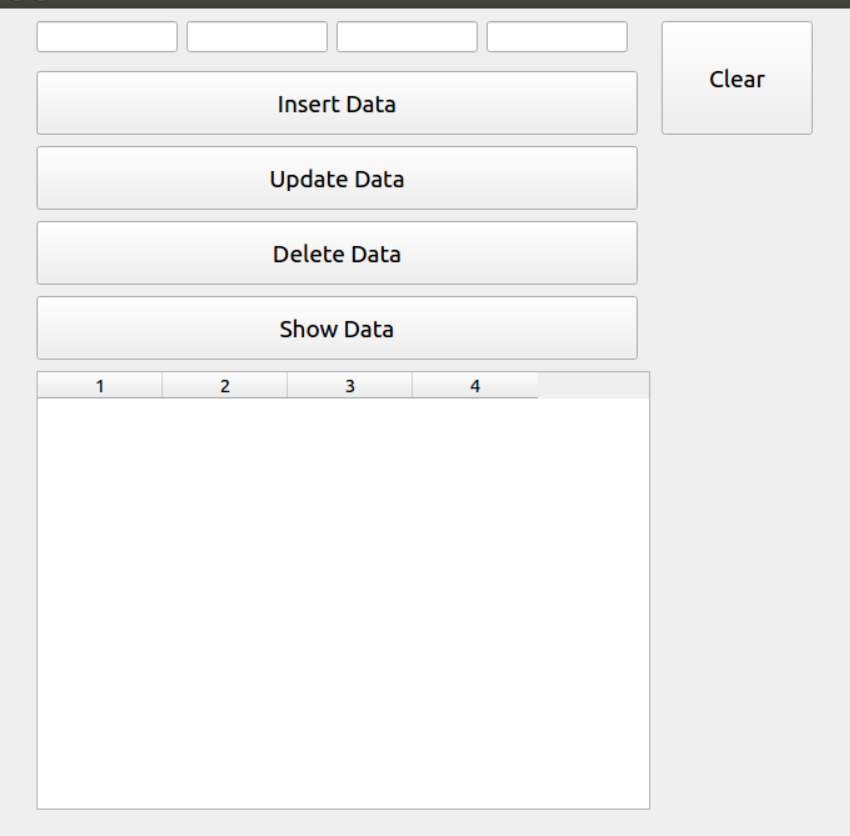
**[1번 테스트] GUI 실행해보기**

* QT Creater 실행 화면  
    
  프로젝트가 존재하지 않는다면 NewProject, 존재한다면 OpenProject를 선택한다.
  + NewProject를 선택 시 GUI 형식으로 만들기 위해 QT Widgets Application을 선택하여 실행한다.  
    
* GUI 형식으로 출력 될 UI 구성  
  
  + Label을 이용해 제목을 표현
  + Text를 이용해 원하는 내용 출력
  + Button을 이용해 눌렀을 때 원하는 작동이 실행되게 구성
* 구성한 UI 작동을 위한 코드 작성
  + UI에서 이벤트를 작동 시킬 위젯을 우클릭 후 Go to slot 을 누른다.  
    
  + 위젯에 따라 다르지만, button의 경우 아래와 같이 출력된다. 원하는 방식을 선택 후 OK 버튼을 누른다. (클릭만 하면 작동하게끔 만들면 됨으로 clicked()를 선택했다.)  
    
  + 누르게 된다면 아래와 같이 위젯의 이벤트를 위한 함수가 생성된다.  
    ex) void Dialog::on\_InsertButton\_clicked()  
    
    - ui : gui로 표현될 것
    - lineEdit : text ui의 이름
    - close() : 종료
* 코드 실행 결과
  + Insert 누를 시 hello 출력  
    
  + Delete 누를 시 text 비우기  
    
  + Exit 누를 시 종료  
    

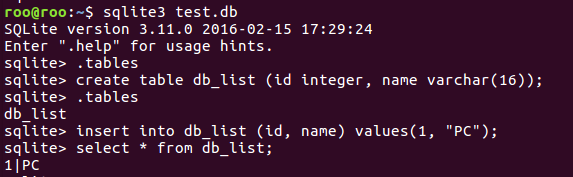
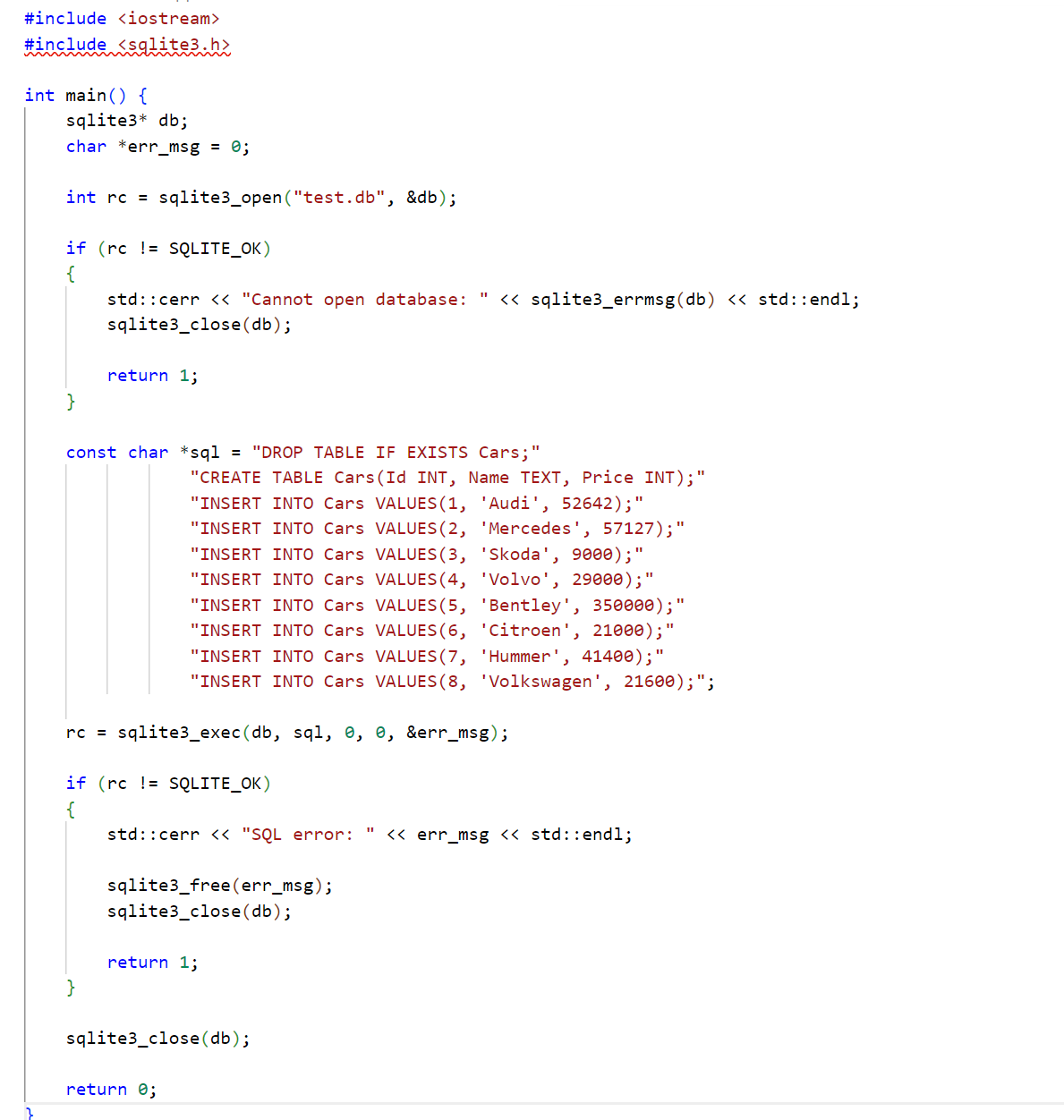
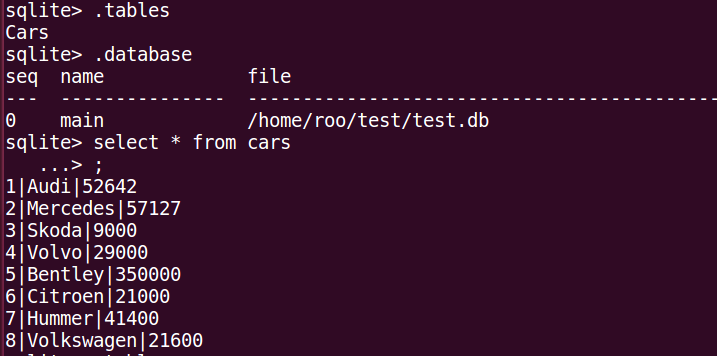
**[2번 테스트] DB에 접근하여 Insert, Update, Delete 등 구현해보기**

* New File 생성 부분 생략 (위와 동일)
* .pro 파일에 QT += sql를 추가한다.  
  
* UI 부분을 아래의 사진과 같이 구성한다.
  + lineEdit : 사용자가 원하는 값을 입력하는 텍스트
  + Insert Data : 값을 DB에 삽입하는 버튼
  + Update Data : DB에 저장되어 있는 값을 업데이트 하는 버튼
  + Delete Data : DB에 저장되어 있는 값을 지우는 버튼
  + Show Data : DB에 저장되어 있는 값을 보여주는 버튼
  + Table Widget : DB에서 가져온 정보를 보여주기 위한 위젯
  + Clear : 편의성을 위해 lineEdit 텍스트에 있는 값들을 모두 지우는 버튼

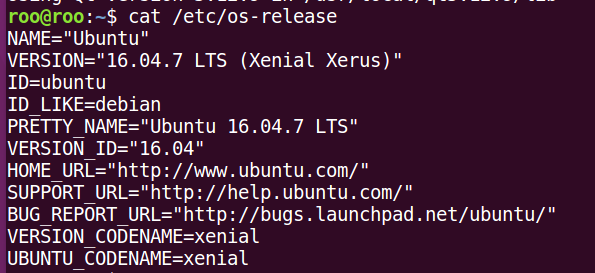


* .h 파일에 #include <QtSql> 헤더를 추가한 후 class 내의 private 부분에 QSqlDatabase DB\_Connection; 을 작성해준다. 이는 QT의 DB를 사용해 DB에 접근하기 위해 사용된다.  
  
* mainwindow.cpp 파일 부분이다. DB에 접근하기 위해 아래 코드와 같이 작성해준다. 현재 코드는 people.db 라는 파일에 접근하여 존재한다면 연결이 되고, 존재하지 않는다면 오류 메시지를 출력하게 만들어져 있다.  
  
* 아래는 각각 Insert, Update, Delete, Clear을 구성하고 있는 코드이다.
  + Insert Button  
      
    미리 구성되어 있는 테이블과 컬럼에 접근해 쿼리문을 이용하여 입력된 텍스트에 맞춰 Insert 하는 코드이다.
  + Update Button  
      
    저장되어 있는 column\_1번의 값에 접근해 column\_2~4번까지의 값을 변경하는 코드이다.
  + Delete Button  
      
    저장되어 있는 column\_1번 값에 해당하는 column 값들을 제거하는 코드이다.
  + Show Button  
      
    DB에 접근하여 table\_1 이라는 이름의 테이블을 선택하여 정보를 확인한다. 최대 10개의 정보(늘리거나 줄이기 가능)를 확인할 수 있다.  
    table 위젯에 10개의 Row를 생성하고 0번부터 불러온 데이터의 정보 개수만큼 반복하며 column\_1부터 column\_4까지 값을 채워넣는다.
  + Clear Button  
      
    입력되어 있는 text들과 불러온 내용을 전부 지우는 코드이다.
* 실행 결과
  + 아무것도 하지 않고 텍스트에 값만 입력한 모습  
    
  + Insert Data 버튼을 누른 이후  
    
  + Update 버튼을 이용해 3번 값을 변경하는 모습  
    
  + Delete 버튼을 이용해 3번 값을 지우는 모습  
    
  + Show 버튼을 이용해 값을 보여주는 모습  
    
  + clear 버튼을 이용해 모든 텍스트를 지운 모습  
    
* 현재는 DB와 연동하고 작동이 되는지 확인을 하기 위해 CRUD 부분을 전부 넣었지만 추후에는 Read 부분만을 남겨 사용자가 원하는 정보를 선택해 Show 버튼을 누른다면, DB에 저장되어 있는 정보들 중 원하는 정보만 출력되게 만들어낸다.

## SQLite 테스트

* SQLite 접속 후 직접 테이블을 생성하고, 내용을 저장  
  
  + sqlite3 test.db : sqlite3를 이용해 test.db 파일에 접속한다.
  + create table [테이블명] (column명) : 테이블과 저장할 컬럼을 생성한다.
  + insert into [테이블명] (column명) values(저장할 정보) : 정보를 테이블의 형식에 알맞게 삽입하여 저장한다.
* C++ 언어를 사용해 SQLite에 접속을 한 후 테이블을 생성하고 내용을 저장  
  
  + #include <sqlite3.h> : sqlite3를 사용하기 위한 헤더
  + sqlite3\_opne(“db파일명”, &sqlite3 변수명) : sqlite3를 이용해 파일에 접속
  + sqlite3\_exec(sqlite3 변수명, 쿼리문 작성되어 있는 변수명, 쿼리 결과로 출력되는 행 하나마다 호출되는 콜백(필요없으면 0), 콜백의 첫 번째 인자(필요없으면 0), 오류가 발생했을 경우 할당된 오류 메세지를 가리킬 포인터) : sqlite3\_prepare\_v2(), sqlite3\_step(), sqlite3\_finalize() 세 함수를 묶은 함수로 편리하게 쿼리를 실행시킬 수 있다.
  + sqlite3\_free(오류 메시지를 카르키는 포인터) : 할당된 오류 메시지는 free 해줘야한다.
  + sqlite3\_close(sqlite3 변수명) : db 종료
* 컴파일 방법  
  
* 컴파일 이후 실행 결과  
  
* 추후에는 service 코드에서 SQLite에 접근한 후 HW정보와 SW정보를 수집하여 해당하는 데이터를 DB에 저장하는 부분을 구현한다..

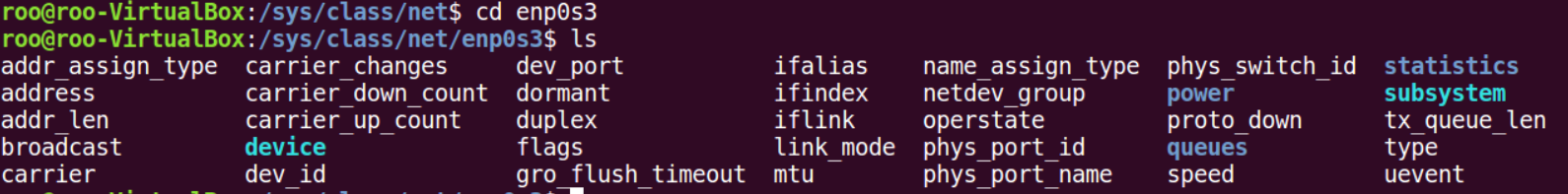
## HW 정보 수집 테스트

* **$ cat /etc/os-release** 를 사용하여 OS 정보를 확인하는 모습  
  

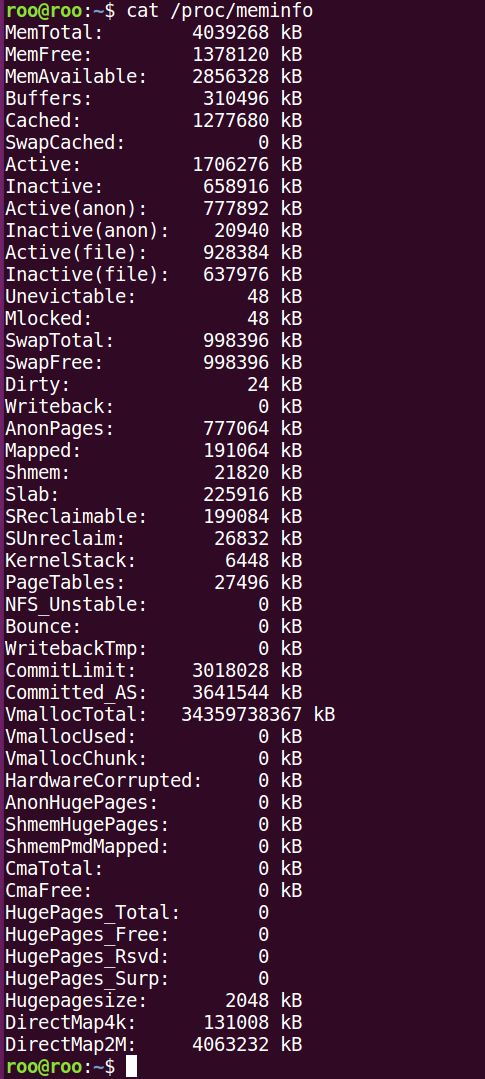
|  |  |
| --- | --- |
| NAME | 운영 체제의 이름 |
| VERSION | 운영 체제의 전체 버전 정보 및 코드명 |
| ID | 운영 체제의 식별자 |
| ID\_LIKE | 운영 체제가 유사한 운영 체제 그룹에 속하는 경우에 대한 정보 |
| PRETTY\_NAME | 사용 중인 운영 체제의 이름 |
| VERSION\_ID | 운영 체제의 버전 ID |
| HOME\_URL | 운영 체제와 관련된 링크 정보 |
| SUPPORT\_URL | 운영 체제와 관련된 링크 정보 |
| BUG\_REPORT\_URL | 운영 체제와 관련된 링크 정보 |
| VERSION\_CODENAME | 운영 체제의 코드명 |
| UBUNTU\_CODENAME | 우분투 운영 체제의 코드명 |

* OS 데이터는 모든 데이터를 수집해 DB에 저장한다.
* **cat /proc/cpuinfo** 를 사용하여 CPU 정보를 확인하는 모습  
  

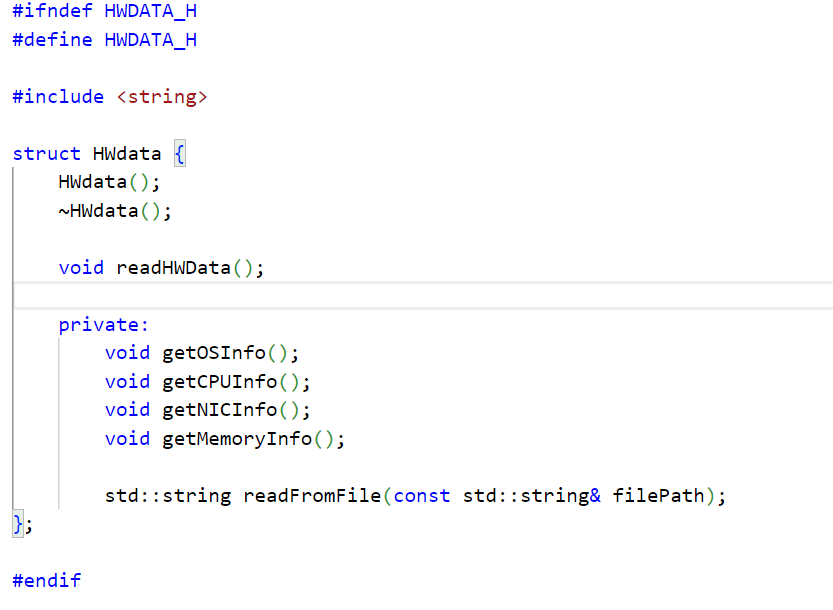
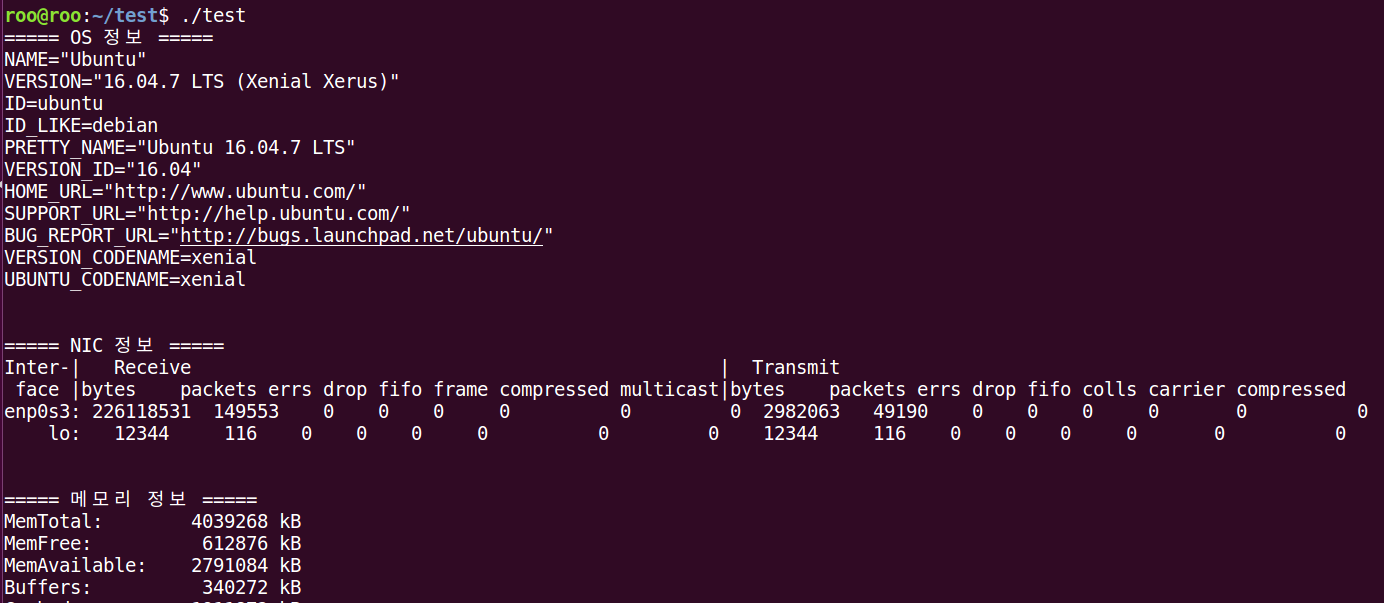
|  |  |
| --- | --- |
| processor | 프로세서의 번호(인덱스) |
| vendor\_id | CPU 제조 업체 |
| cpu family | CPU 패밀리 |
| model | CPU 모델 |
| model name | CPU 모델명 |
| stepping | CPU 스테핑 번호 |
| cpu MHz | CPU 클럭 속도 |
| cache size | CPU 캐시 크기 |
| physical id | 물리적인 CPU의 식별자 |
| siblings | 하나의 물리적인 코어에서 실행 가능한 스레드 수 |
| core id | 물리적인 CPU 코어의 식별자 |
| cpu cores | 물리적인 CPU가 가지고 있는 코어의 수 |
| apicid | CPU의 고유한 APCI 식별자 |
| initial apicid | CPU의 고유한 APCI 식별자 |
| fpu | 부동 소수점 계산 지원 여부 |
| fpu\_exception | 부동 소수점 예외 지원 여부 |
| cupid level | CPUID 명령어 사용 시 얻을 수 있는 최대 정보 수준 |
| wp | Write Protection 기능 활성화 여부 |
| flags | CPU가 지원하는 기능과 명령어 집합을 나열한 목록 |
| bugs | CPU에 관련된 버그나 이슈에 대한 정보 제공 |
| bogomips | CPU의 성능을 대략적으로 측정하는 데 사용되는 BogoMIPS라는 단위 |
| clflush size | CPU에서 지원하는 캐시 라인 크기 |
| cache\_alignment | CPU 캐시의 정렬 값 |
| address size | CPU가 지원하는 물리적 및 가상 주소 크기 |
| power management | CPU에서 지원하는 전력 관리 기능을 나열한 목록 |

* CPU 데이터의 경우 processor, vendor\_id, cpu\_family, model\_name, cpu\_MHz, cache\_size, cpu\_cores, clfush\_size, cache\_alignment, address\_size 정보를 수집하여 DB에 저장한다.
* **$ sys/class/net**에 존재하는 폴더의 내용을 읽어 NIC 정보를 수집한다.  
    
  

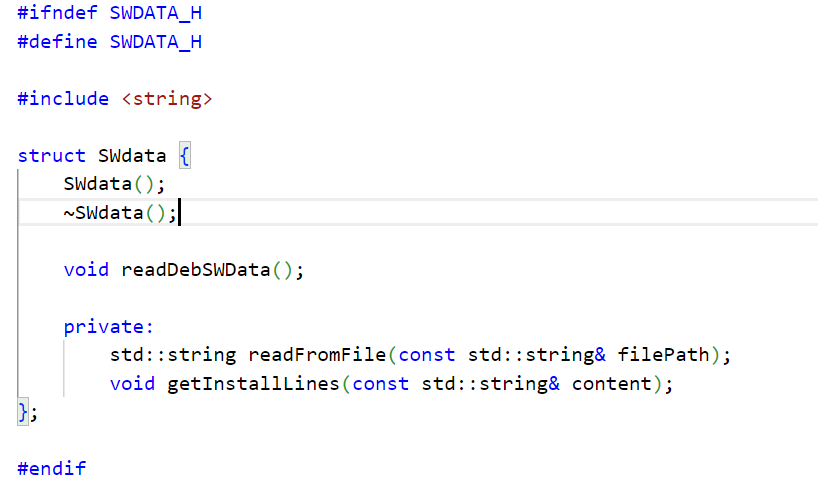
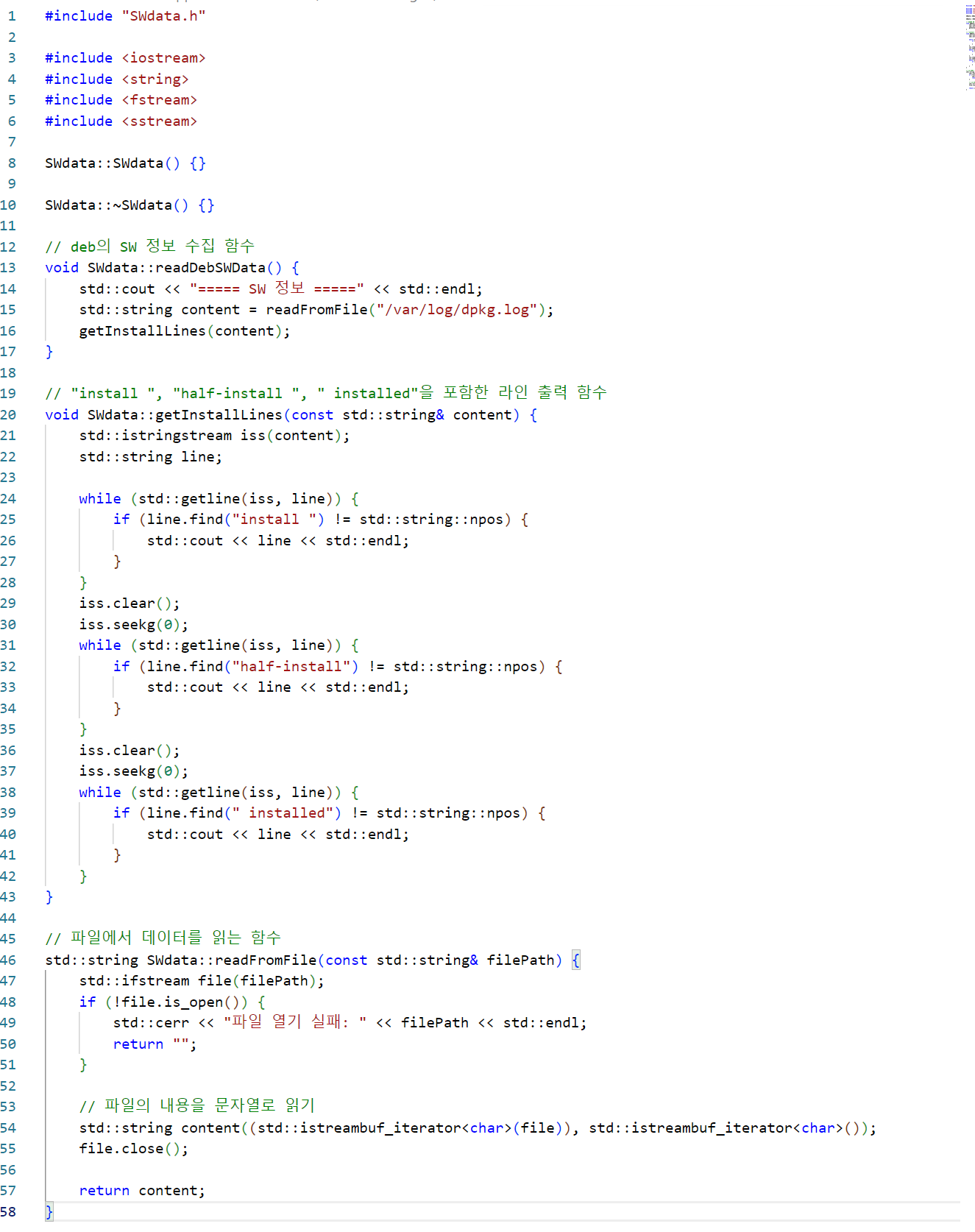
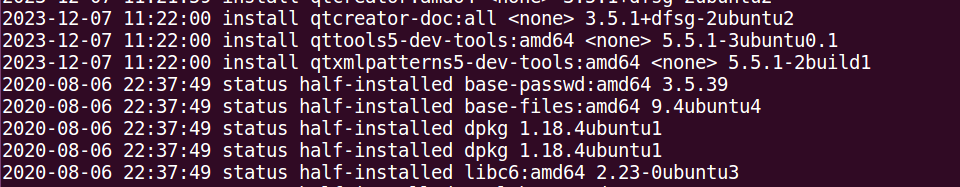
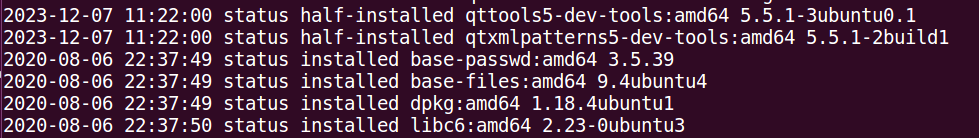
|  |  |
| --- | --- |
| addr\_assign\_type | 네트워크 인터페이스의 주소 할당 방법을 지정하는 파일. |
| dev\_id | 네트워크 디바이스의 ID를 나타내는 파일. |
| link\_mode | 네트워크 인터페이스의 링크 모드를 나타내는 파일. |
| proto\_down | 프로토콜이 비활성 상태인지 여부를 나타내는 파일. |
| address | 네트워크 인터페이스의 MAC 주소를 포함하는 파일. |
| dev\_port | 네트워크 디바이스의 포트를 나타내는 파일. |
| mtu | 최대 전송 단위(MTU)를 나타내는 파일. |
| queues | 네트워크 인터페이스의 큐 수를 나타내는 폴더 |
| addr\_len | 네트워크 인터페이스 주소의 길이를 나타내는 파일. |
| dormant | 네트워크 인터페이스가 비활성 상태인지 여부를 나타내는 파일. |
| name\_assign\_type | 네트워크 인터페이스의 이름 할당 방법을 나타내는 파일. |
| speed | 네트워크 인터페이스의 속도를 나타내는 파일. |
| broadcast | 브로드캐스트 주소를 나타내는 파일. |
| duplex | 네트워크 인터페이스의 전이중(duplex) 모드를 나타내는 파일. |
| netdev\_group | 네트워크 인터페이스가 속한 그룹을 나타내는 파일. |
| statistics | 네트워크 인터페이스의 통계 정보를 포함하는 폴더. |
| carrier | 네트워크 인터페이스의 캐리어 상태를 나타내는 파일. |
| flags | 네트워크 인터페이스의 플래그를 나타내는 파일. |
| operstate | 네트워크 인터페이스의 운영 상태를 나타내는 파일. |
| subsystem | 네트워크 인터페이스가 속한 서브시스템을 나타내는 폴더 |
| carrier\_changes | 캐리어 상태 변경 횟수를 나타내는 파일. |
| gro\_flush\_timeout | GRO(Generic Receive Offload) 플러시 타임아웃을 나타내는 파일. |
| phys\_port\_id | 물리 포트 ID를 나타내는 파일. |
| tx\_queue\_len | 전송 큐 길이를 나타내는 파일. |
| carrier\_down\_count | 캐리어 다운 횟수를 나타내는 파일. |
| Ifalias | 네트워크 인터페이스의 별칭을 나타내는 파일. |
| phys\_port\_name | 물리 포트 이름을 나타내는 파일. |
| type | 네트워크 인터페이스의 유형을 나타내는 파일. |
| carrier\_up\_count | 캐리어 업 횟수를 나타내는 파일. |
| Ifindex | 네트워크 인터페이스의 인덱스를 나타내는 파일. |
| phys\_switch\_id | 물리 스위치 ID를 나타내는 파일. |
| uevent | Udev 이벤트를 나타내는 파일. |
| device | 네트워크 인터페이스가 속한 디바이스를 나타내는 심볼릭 링크 폴더 |
| iflink | 네트워크 인터페이스의 ifindex에 대한 심볼릭 링크. |
| power | 네트워크 인터페이스의 전원 관리 정보를 제공하는 폴더. |

* nic의 경우 interface(각 폴더명), operation\_state, speed, mac\_address 정보를 수집하여 DB에 저장한다.
* **$ cat /proc/meminfo** 를 사용하여 Memory 정보를 확인하는 모습  
  

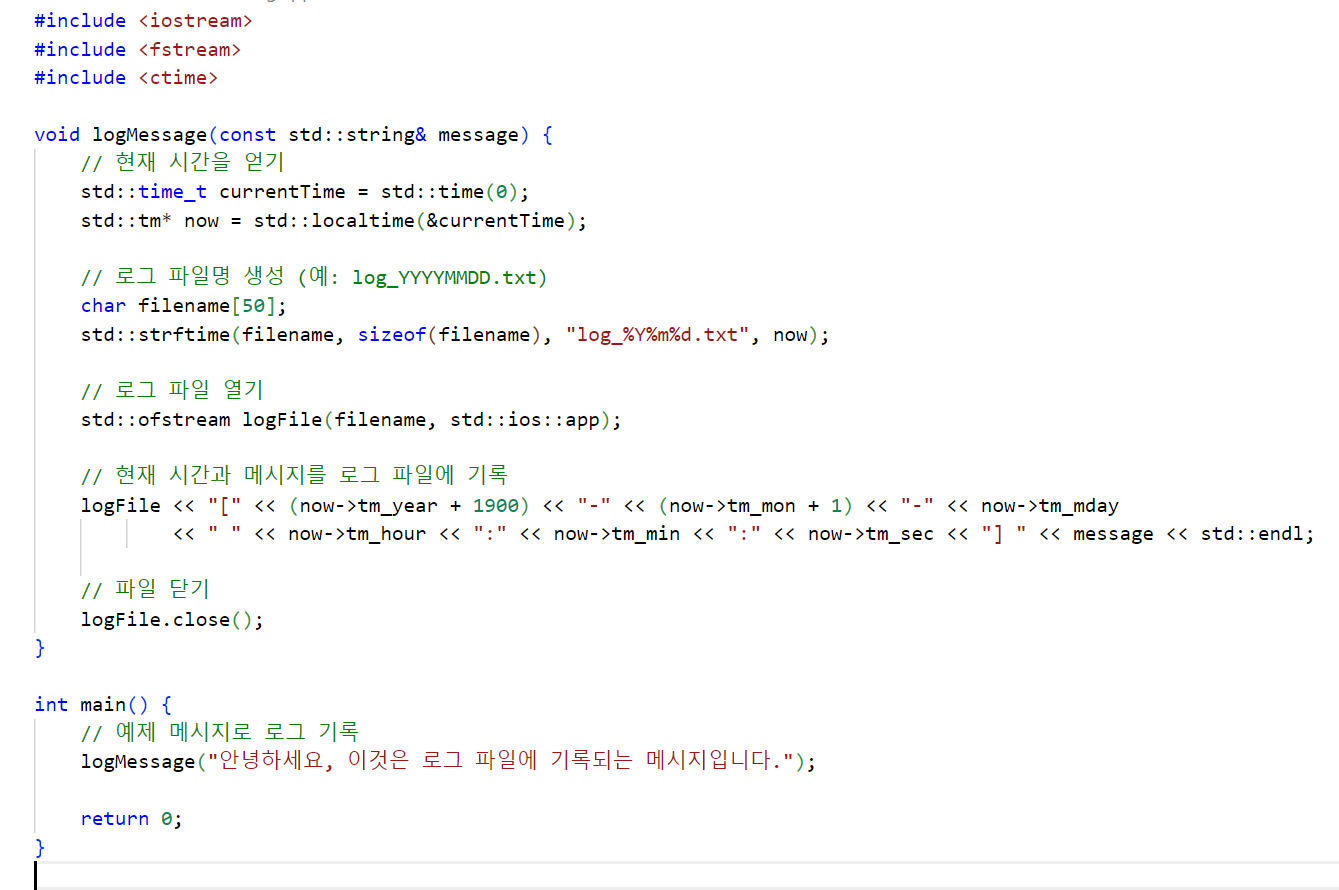
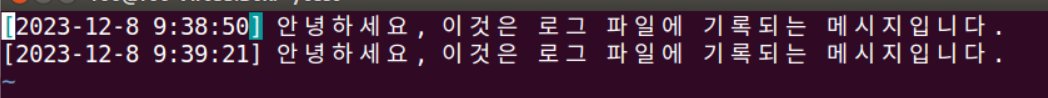
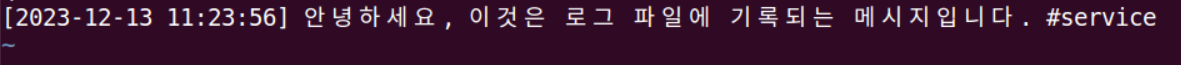
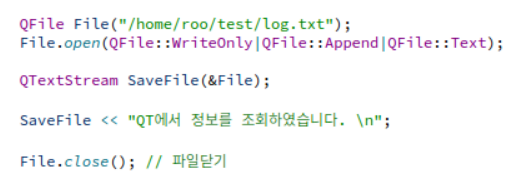
|  |  |
| --- | --- |
| MemTotal | 사용 가능한 총 메모리 |
| MemFree | 시스템에서 사용 하지 않는 실제 메모리 양 |
| MemAvailable | 스왑없이 새 응용 프로그램을 시작할 때 사용할 수 있는 메모리 양 추정 |
| Buffers | 버퍼 캐시의 메모리로, 원시 디스크 블록을 위한 비교적 저장 공간. |
| Cached | 페이지 캐시의 메모리 |
| SwapCached | 주 메모리 내에 있지만 스왑 파일에도 존재하는 메모리 |
| Active | 최근에 사용되었고 일반적으로 스왑 아웃되거나 재 확보되지 않은 메모리 |
| Inactive | 최근에 사용 되지 않았고 스왑 또는 재사용 가능한 메모리 |
| Active(anon) | 최근에 사용되었고 대개 스왑 아웃되지 않은 익명 메모리 |
| Inactive(anon) | 최근에 사용 되지 않았으며 교체 가능할 수 있는 익명 메모리 |
| Active(file) | 가장 최근에 사용되었고 일반적으로 필요한 때까지 재생되지 않은 캐시 메모리 |
| Inactive(file) | 막대한 성능 영향 없이 재생할 수 있는 캐시 메모리 |
| Unevictable | 여러 가지 이유로 단호한 페이지를 스왑 아웃 불가. |
| Mlocked | mlock() 시스템 호출을 사용하여 메모리에 잠근 페이지 . |
| SwapTotal | 사용 가능한 총 스왑 공간 |
| SwapFree | 남은 스왑 공간 |
| Dirty | 디스크에 다시 기록 대기중인 메모리 |
| Writeback | 디스크에 적극적으로 다시 쓰여지고 있는 메모리 |
| AnonPages | 사용자 공간 페이지 테이블에 매핑되는 비 파일 지원 페이지 |
| Mapped | 라이브러리와 같이 mmaped 된 파일 |
| Shmem | 총 사용 된 공유 메모리 |
| Slab | 커널 내 데이터 구조 캐시 |
| SReclaimable | 회수 예정인 Slab 영역 |
| SUnreclaim | 회수 불가한 Slab 영역 |
| KernelStack | 커널 스택이 사용하는 메모리. 회수 불가 |
| PageTables | 가장 낮은 레벨의 페이지 테이블 전용 메모리. 많은 프로세스가 동일한 공유 메모리 세그먼트에 연결되어 있으면 이 값이 커질 수 있다 |
| NFS\_Unstable | 서버에 보내지지만 저장소에 아직 커밋되지 않은 NFS 페이지 |
| Bounce | 블럭장치 bounce buffer를 위해 사용된 메모리 |
| WritebackTmp | FUSE에서 임시 쓰기 저장 버퍼로 사용하는 메모리 |
| CommitLimit | 오버 커밋 비율 (vm.overcommit\_ratio)에 따라 시스템에서 현재 할당 할 수 있는 총 메모리 크기 |
| Committed\_AS | 현재 시스템에 할당 된 메모리의 양. 커밋 된 메모리는 프로세스가 아직 사용하지 않았더라도 프로세스가 할당 한 모든 메모리의 합계 |
| VmallocTotal | vmalloc 메모리 영역의 전체 크기 |
| VmallocUsed | 사용 된 vmalloc 영역의 양 |
| VmallocChunk | 미사용 중인 vmalloc 영역중 가장 큰 |
| HardwareCorrupted | 커널이 손상되었거나 작동하지 않는 것으로 확인한 RAM의 양 |
| AnonHugePages | 사용자 공간 페이지 테이블에 매핑 된 비 파일 백 로그 페이지 |
| ShmemHugePages | 커널에서 관리하는 공유 메모리 (Shared Memory) 영역 중에서 특히 큰 페이지 (Huge Pages)로 할당된 부분의 크기 |
| ShmemPmdMapped | 페이지 테이블 엔트리의 큰 페이지 (PMD, Page Middle Directory)를 사용하여 할당된 공유 메모리 영역의 크기 |
| CmaTotal | Contiguous Memory Allocator (CMA)로 할당 가능한 전체 메모리 양 |
| CmaFree | 현재 사용 가능한 CMA 메모리 양 |
| HugePages\_Total | 커널에 의해 할당되는 거대한 페이지의 수 |
| HugePages\_Free | 프로세스에 의해 할당되지 않은 hugepage 수 |
| HugePages\_Rsvd | 풀에서 할당하려는 약속이 있지만 할당이 아직 이루어지지 않은 hugepage의 수 |
| HugePages\_Surp | vm.nr\_hugepages파라메터 값에 의해 정해진 풀의 hugepage의 수 |
| Hugepagesize | hugepage 크기 |
| DirectMap4k | 표준 4k 페이지에 매핑되는 메모리 양 |
| DirectMap2M | hgepages에 매핑되는 메모리의 양 |

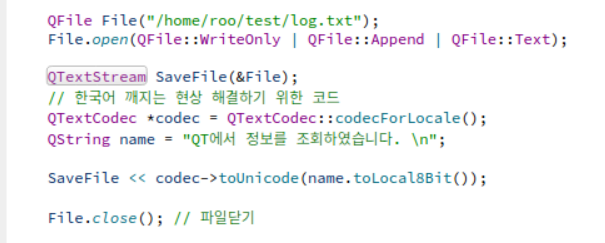
* memory의 경우 모든 데이터를 수집하여 저장한다.
* 위의 명령어를 C++ 코드를 이용해 출력  
  < HWdata.h 코드 >  
    
  < HWdata.cpp 코드>  
    
  출력 결과 (일부분)  
  
* 추후에는 OS, NIC, Memory, CPU 정보를 정리하여 SQLite DB에 저장한다.

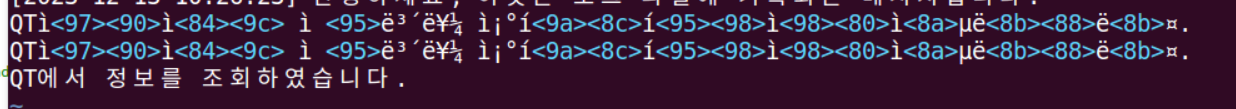
## SW 정보 수집 테스트

* C++ 언어를 통해 dpkg.log 파일을 읽어 출력하였다.  
  < SWdata.h 코드 >  
    
  < SWdata.cpp 코드 >  
    
  출력 결과 (일부분)  
    
    
  install, status half-installed, status installed 가 붙어있는 문장들을 순서대로 출력하는 모습을 확인할 수 있다.  
  [년-월-일] [시:분:초] [설치상태] [SW명] : [아키택쳐] [과거버전] [현재버전]
* 추후에는 출력 결과 중 SW 설치된 날짜, SW 명, SW 설치 버전을 수집하여 DB에 저장한다.

## 로그 파일 테스트

* 로그 파일을 생성하기 위한 코드  
  
  + #include <fstream> : std::ofstream 을 사용하기 위한 헤더
  + #include <ctime> : std::time, std::tm\* 을 사용하기 위한 헤더
  + void logMessage(const std::string& message) : 문자열 메시지를 매개변수로 받아 현재 시간과 함께 로그 파일에 기록
  + std::time(0) : 현재 시간을 수집
  + std::localtime(변수명) : 구조체로 변환
  + std::strftime() : 현재 날짜를 포함한 로그 파일 생성
  + std::ofstream : 파일 열기
  + std::ios::app : 파일의 끝에 내용을 추가하는 플래그
  + (now->tm\_year + 1900) : 현재 연도를 얻어온다. std::tm 구조체의 tm\_year 필드는 1900년부터의 연도를 나타내므로, 현재 연도를 얻기 위해 1900을 더한다.
  + (now->tm\_mon + 1): 현재 월을 얻어온다. std::tm 구조체의 tm\_mon 필드는 0부터 시작하므로, 1을 더해서 현재 월을 얻는다.
  + now->tm\_mday: 현재 월의 날짜를 얻는다.
  + now->tm\_hour, now->tm\_min, now->tm\_sec: 각각 현재 시간의 시, 분, 초를 얻는다.
  + 정보를 이어붙여서 "[YYYY-MM-DD HH:MM:SS]" 형식으로 만듭니다.
  + logFile.close : 파일 닫기
* 실행하여 생성한 TXT 파일  
    
  작성이 된 [년-월-일 시:분:초] 작성이 될 내용 형식으로 출력이 된다.
* 서비스에서 동작되는 것인지, QT에서 동작되는 것인지를 구별하기 위해 해시태그(#동작부분)를 남긴다.  
  
* QT에서 TXT 파일에 접근하여 남긴 코드  
  
  + 위와 같이 코드를 작성 시 txt 파일에 접근하여 문장은 남기지만 한국어가 깨지는 현상이 발생한다.  
      
    한국어가 깨지는 문제를 해결하기 위해 코드를 아래와 같이 수정하였다.



* + 입력이 될 내용을 QString 변수에 담고서 Local8Bit로 변경해주었다. 그 결과 한국어가 깨지는 문제가 해결되었다.  
    
  + QT Creator는 기본 UTF-8로 설정이 되어있는데 euc-kr 인코딩을 사용하기 때문에 코드상에서 한국어를 작성하면 깨지는 현상이 발생하는 것 같다.  
    QString::fromLocal8Bit(“하고 싶은 말”) 형식으로 작성해도 한국어를 사용할 수 있지만, \*fromUtf8()은 되지 않는다.
* 추후에는 Service와 QT에서 작동(DB접근, SW 저장, HW 저장, SW 데이터 접근 등)된 동작들을 전부 로그에 남겨둔다.