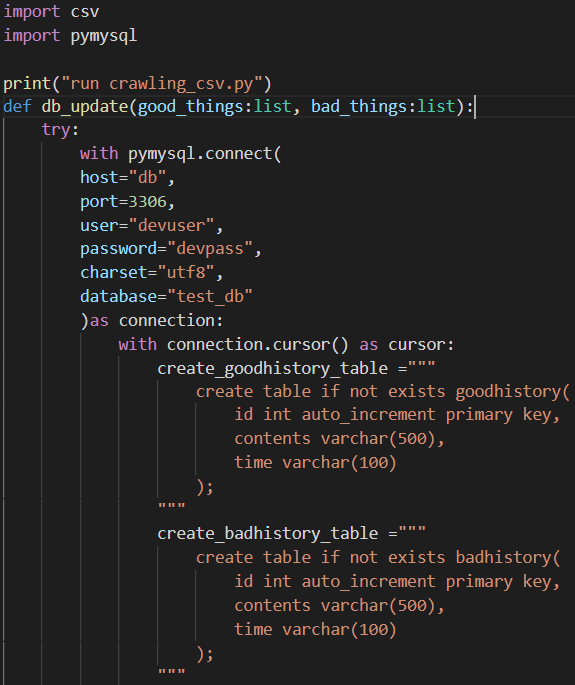
**5조 Chromw Browser History 소스코드**

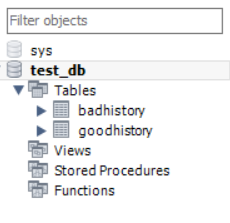
1. DB 생성스크립트

* **Crawling\_csv.py**



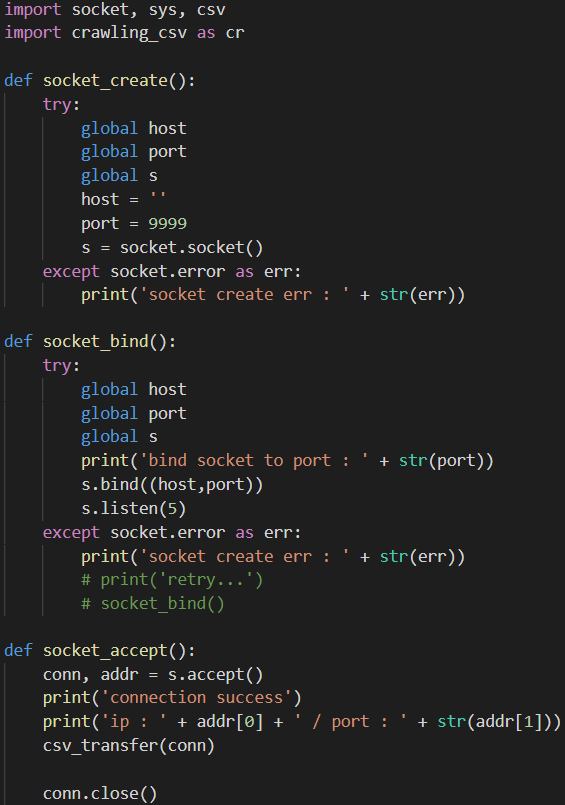
- DB(good,badhistory)테이블을 각각 생성

- DB테이블 생성화면



1. 크롤링 스크립트

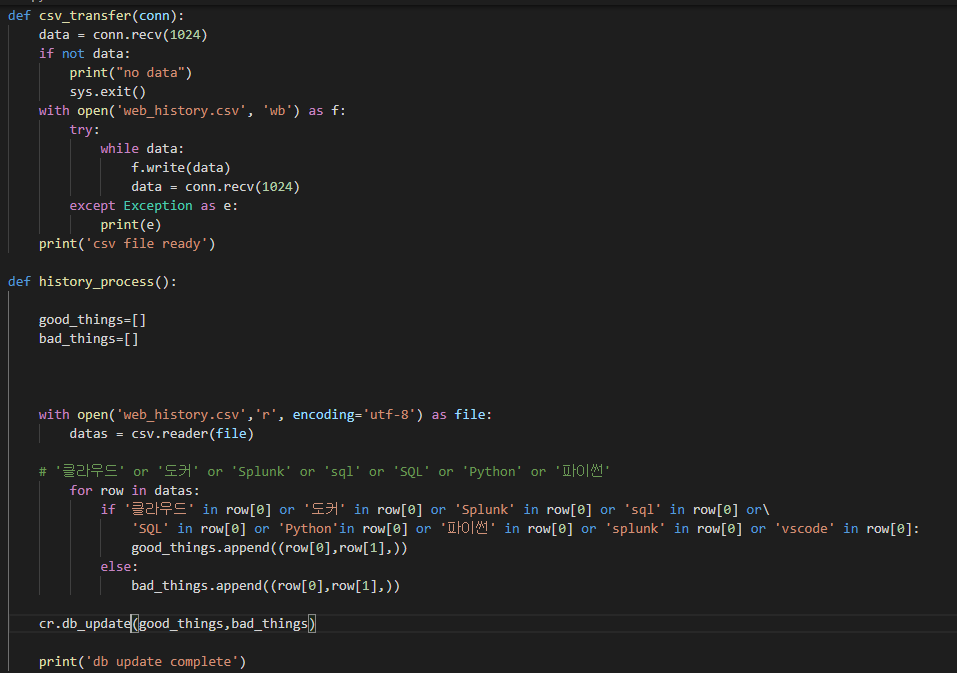
* **history\_csv.py**



* socket create : 서버쪽 소켓을 생성합니다, 포트번호 9999
* socket bind : 주어진 포트번호와 호스트 주소를 바탕으로 해당 소켓에 포트번호를 바인딩 줍니다
* socket accpet : 클라이언트가 접속하게 되면 connection을 만들고 주어진 작업 수행

(csv 파일을 클라이언트가 서버로 전송해주는 csv\_transfer 함수 호출)

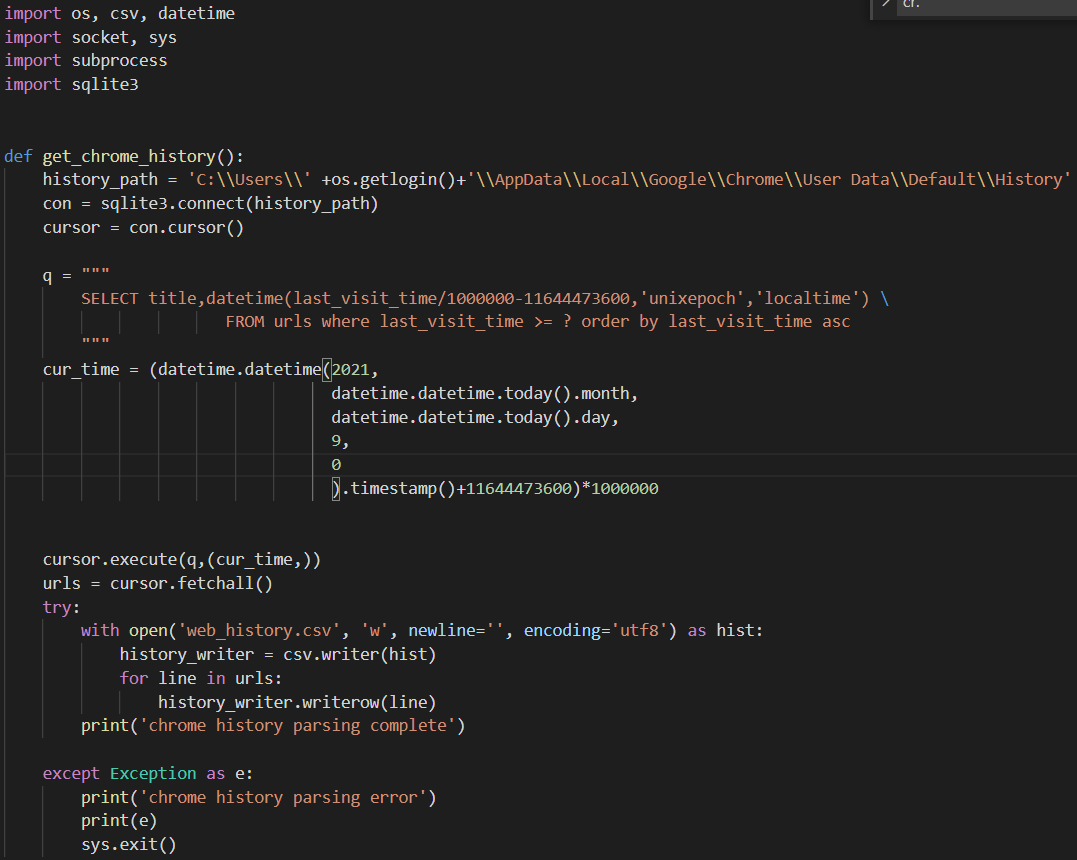
* conn.close : 다 완료되면 연결 종료



* csv\_transfer : 클라이언트가 연결된 후 서버쪽에서 클라이언트가 보내는 csv파일을 “recv” 함수를 통해 받아서 web\_history.csv 파일 생성.
* history\_process : 이건 정민씨가 한거니깐 )) cr.db\_update 함수의 경우 crawling\_csv.py를 import 시킨 후 거기에 있는 db\_update 함수를 불러오는것

(히스토리 프로세스 함수에서 처리한 굿띵즈와 배드띵즈 리스트를 crawling\_csv.py의 db\_update 함수에다 넘겨주고 db에 연동시키는 역할)

* **history\_client.py**



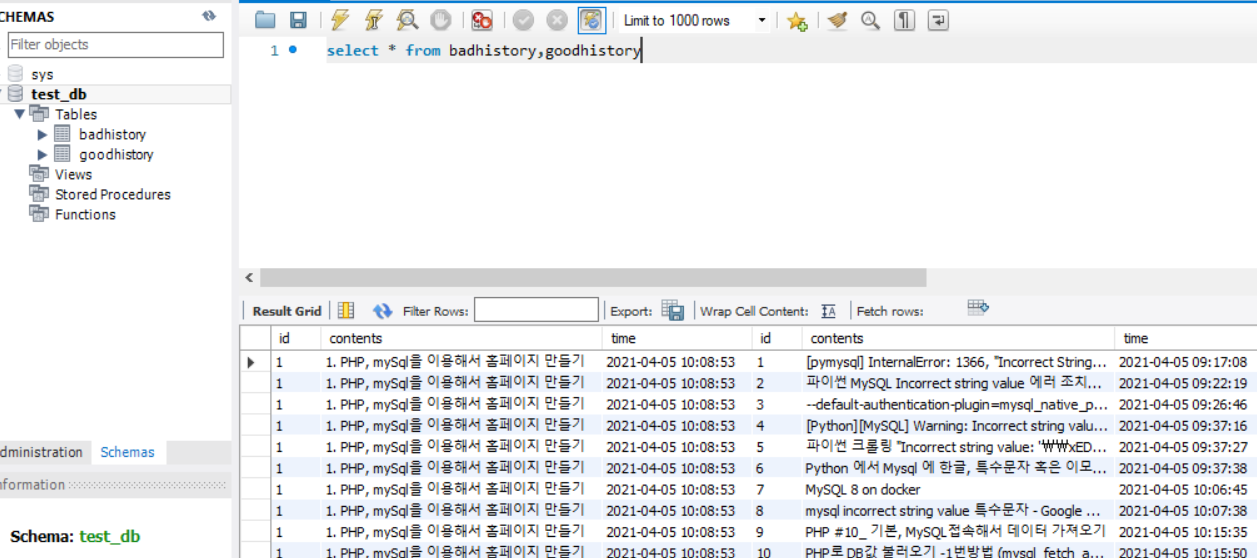
* get\_chrome\_history : 크롬 브라우저의 방문기록은 위에 주어진 history\_path 경로에 sqlite3 DB 파일로 저장되어 있습니다.
* os.getlogin : 현재 로컬PC 유저의 id를 가져오는 함수이며, path 경로를 통해 sqlite3 파이썬 내장 패키지를 이용하여 히스토리 DB를 불러오는 함수
* q : 히스토리 정보 긁어오는 sqlite3 select 쿼리문
* q의 쿼리문 : 크롬 히스토리는 webkit(웹킷) 시간으로 설정되어 있는데 (1601년 1월 1일 0시 0분 0초 기준 누적 시간을 마이크로세컨드로 표현), 이를 EPOCH 시간(에포크 시간 or 유닉스 시간) 으로 변환해주는게 저 숫자로 된 공식 (1970년 1월 1일 0시 0분 0초 기준 누적 시간을 초(seconds)로 표현), 에포크 시간으로 변환해야 datetime 함수로 우리가 쉽게 볼 수 있는 human date로 변환이 가능하기 때문입니다.

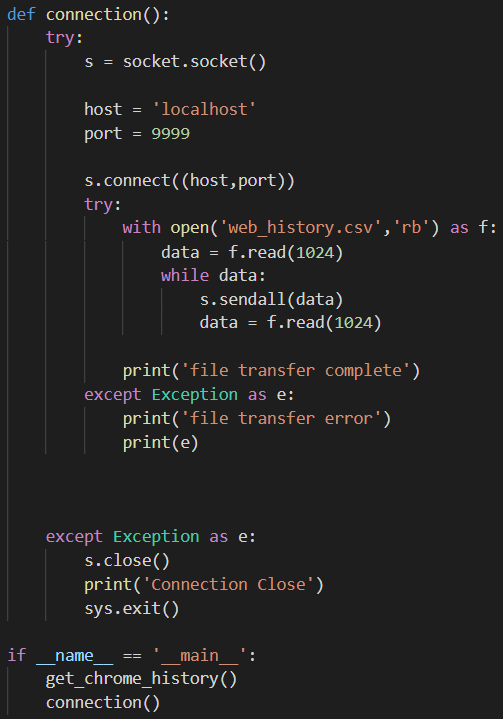
따라서 last\_visti\_time은 webkit 시간이니깐 이를 우선 뒤에 수식을 통해 EPOCH 시간으로 변환 후 datetime 함수에 넣어서 human date로 표현하도록 만들었고, 이것을select 쿼리문으로 히스토리의 방문 사이트 이름(title)과 방문날짜(datetime 이하)를 수집하였습니다.

* cur\_time : 현재 시간을 webkit 시간으로 변경하는 용도. 크롬 히스토리에서 쿼리문 전송할때 현재 시간(현재 월, 일 오전 9시 기준) 이후로 쌓인 history를 긁어오기 위한것. q에 보시면 **where last\_visit\_time >= ?**로 된 부분이 있는데 거기에 cur\_time 값이 들어가서 그 시간 보다 큰(현재 날짜 오전 9시 이후) last\_visit\_time 값을 가진 데이터를 뽑아오게 합니다.

(webkit 시간으로 바꿔야 하는지? -> 크롬에서 webkit을 쓰기 때문에 history 긁어오는 쿼리문을 돌릴려면 webkit 시간 기준으로 해야합니다.)

* **DB테이블에 데이터 삽입**

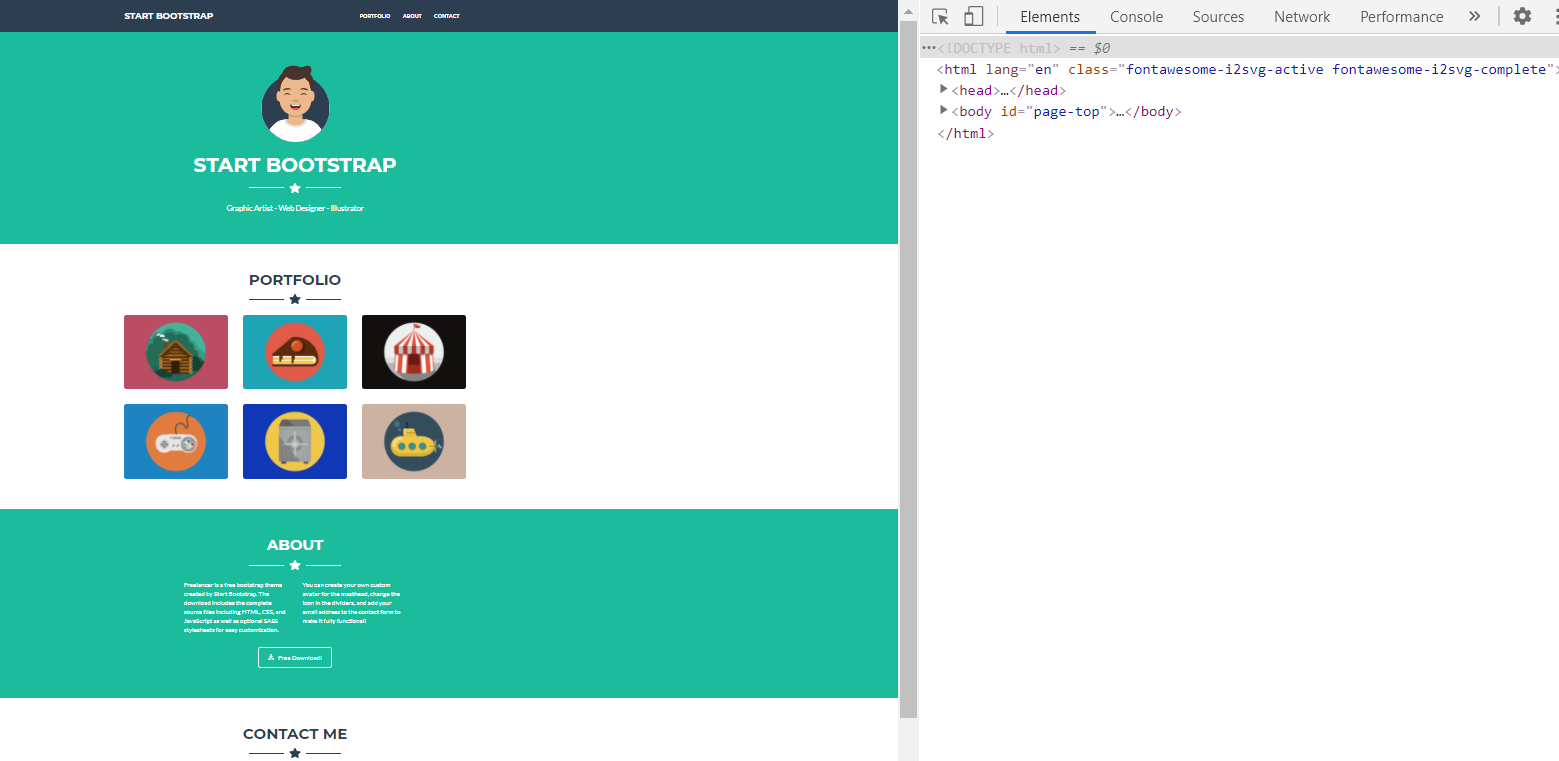


* 
* try ~ 구문 : web\_history.csv 파일 만들어서 쿼리로 불러온 히스토리 정보를 작성
* connection : 크롬 히스토리 파일을 csv로 다 만들고 나면 서버와 연결(s.connect),

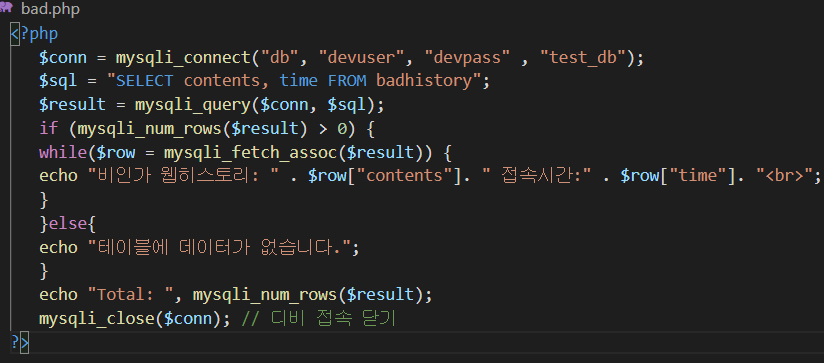
csv파일을 오픈해서 1024바이트씩 읽어온 후 sendall 함수를 통해 서버로 데이터 전송(바이트 스트림으로 전송)

1. 웹페이지 스크립트

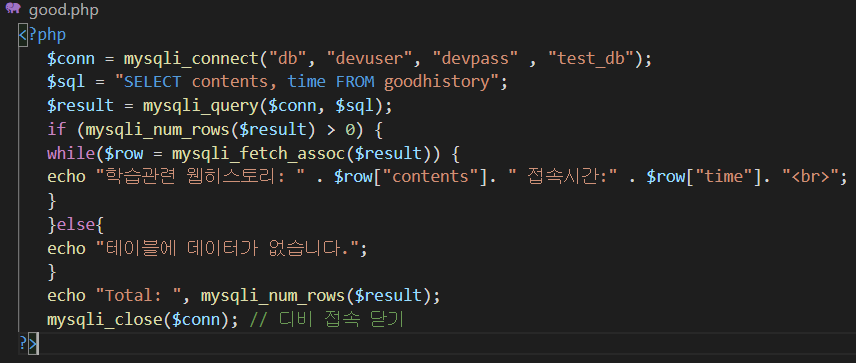
* **무료 웹페이지 템플릿 다운로드**



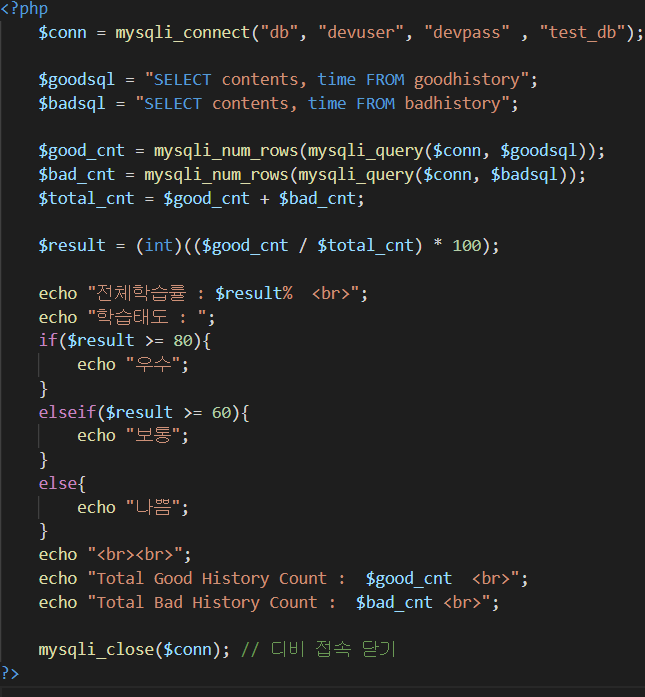
* 무료 웹페이지 템플릿 다운로드 후, html코드를 index.php에 그대로 가져와서 화면 설계한 내용대로 코드를 수정.
* **bad.php**



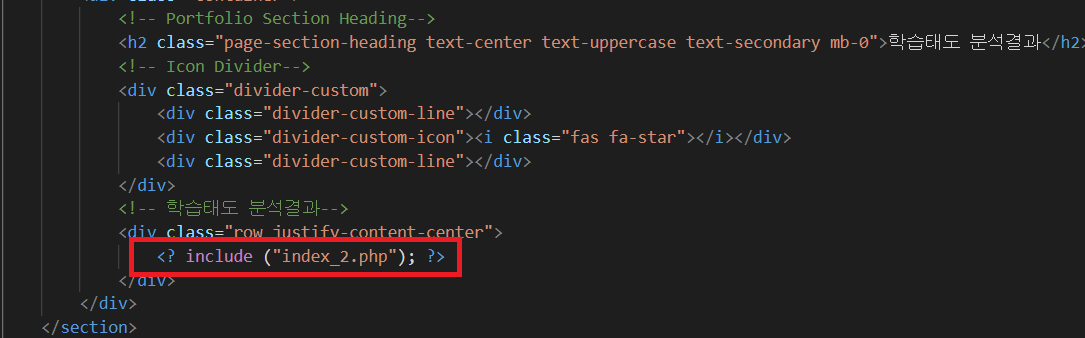
* DB에 있는 goodhistory 테이블을 가져와서 모든 데이터를 출력해주는 파일
* **good.php**



* DB에 있는 goodhistory 테이블을 가져와서 모든 데이터를 출력해주는 파일
* **Index\_2.php**

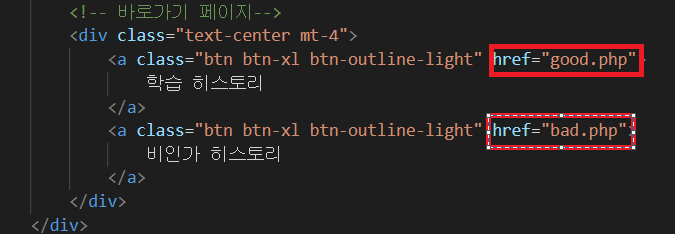
****

* DB에서 goodhistory, badhistory 데이터를 가져와서 전체학습률, 학습태도, 각 테이블 row count를 계산하여 echo를 통해 결과값들을 출력
* **Index.php(웹페이지의 본체)**

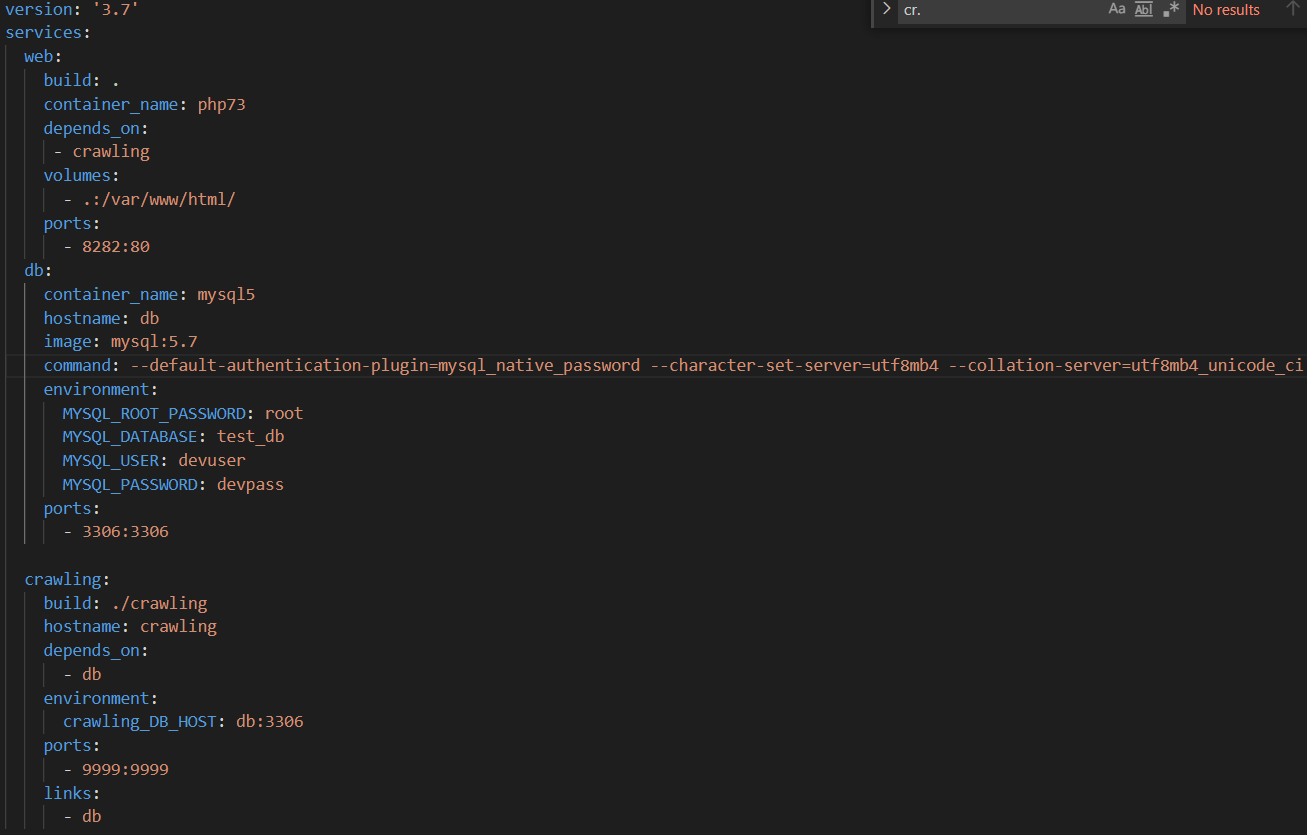
****

* <!—학습태도 분석결과--> 주석란에 index\_2.php를 삽입

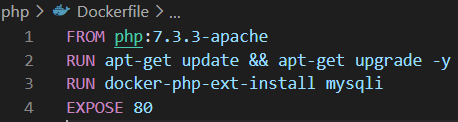
(index\_2.php는 DB에서 goodhistory, badhistory 데이터를 가져와서 전체학습률, 학습태도, 각 테이블 row count를 계산하여 출력해주는 파일)



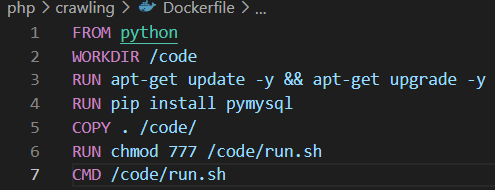
* 각각 goodhistory, badhistory의 데이터들을 모두 출력해주는 페이지로 각각 이동하기 위한 버튼을 생성
* **Docker-compose.yml**

****

* **Web 서버 구축 도커파일**

****

* **크롤링 서버 구축 도커파일**

****

* **실행 순서**

1. **현재 작업 디렉토리에서 history\_client.py 실행**
2. **Docker-compose up 실행**
3. **Localhost:8282/index.php 접속 후 데이터 확인**

