1. 프로젝트 개요

프로젝트 기간

2018.07.24 ~ 2018.08.02 (10일)

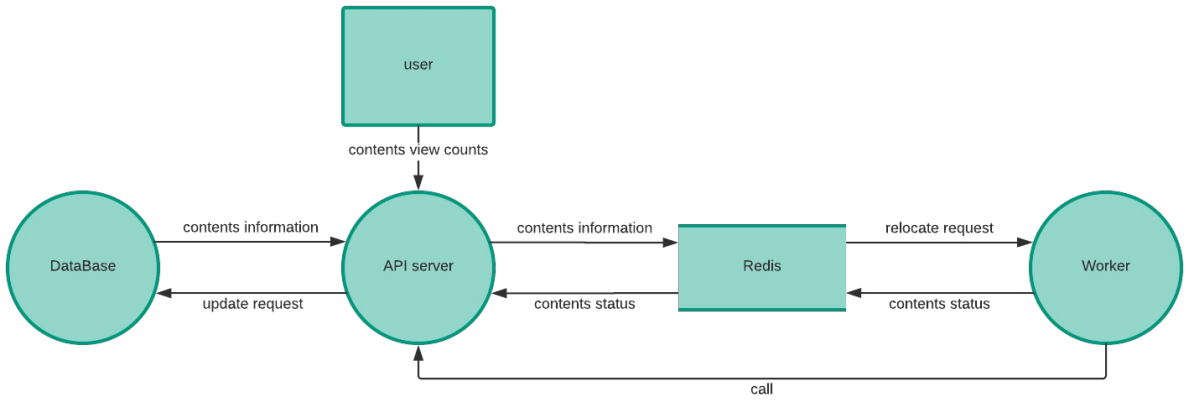
목적

컨텐츠에 대한 접근 수가 증가하면 서버에 대한 트래픽 양 또한 증가하여 용량이 더 큰 서버로의 컨텐츠 재배치가 요구됨
이를 실시간으로 수행하기 위하여 메모리 기반의 realtime database 인 redis 와 파이썬의 asyncio 라이브러리를 이용한 비동기 파일 재배치 프로그램을 제작

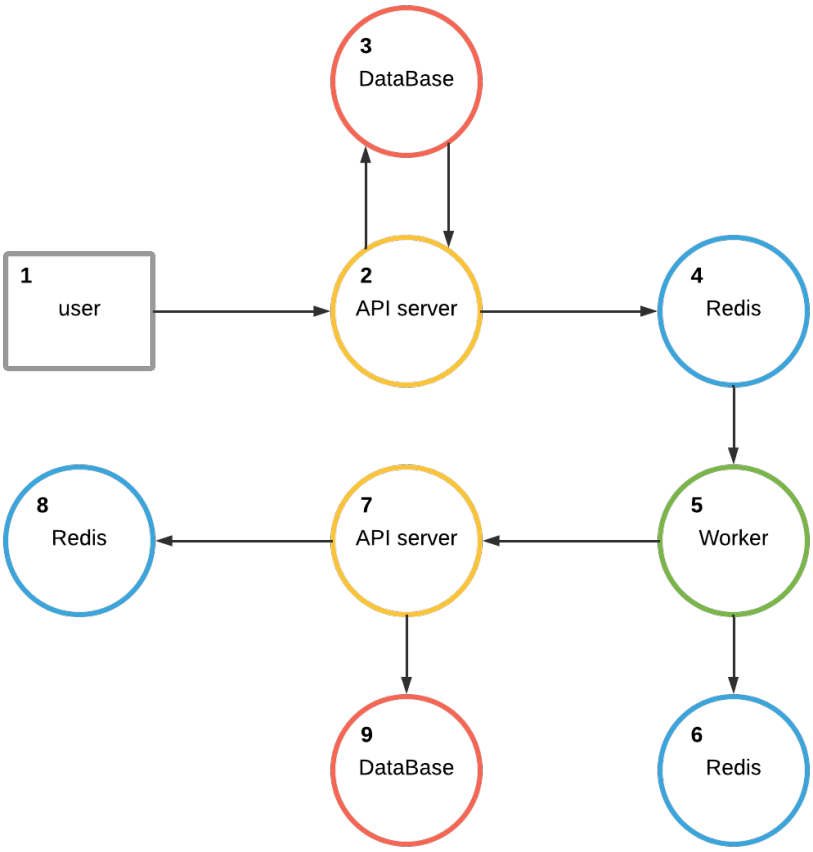
기대효과

트래픽 용량에 따른 컨텐츠 재배치가 자동으로 이루어져 서버의 부하를 실시간으로 조절하고 여러 개의 워커가 비동기적으로 작동하여 큰 용량의 파일이 이동할 때에도 병목 현상 방지 가능

Data Flow Diagram



Work Flow



2. API

담당자: 김지희

개발 환경

- OS : ubutu 16.04
- python==3.6.5
- MySQL==5.7
- Flask==1.0.2
- PyMySQL==0.9.2
- redis-server==4.0.10

프로세스 상세 및 실행 결과

컨텐츠 정보 쿼리 및 redis 업데이트(Work Flow. 2)

1. 사용자가 컨텐츠를 조회하면 컨텐츠의 cid 와 count 정보를 포함하여 API를 호출 (e.g.curl http://192.168.10.108:5001/post_sentence -d "cid=3&count=664")

2. db_que ry 모듈을 이용하여 database의 contents table 과 level table에서 post 된 cid를 가진 content의 현재위치와 목적위치를 반환

- contents

#	cid	content_level	filename	generate_time	update_time
1	1	gold	a.mp4	2018-07-29 ...	2018-08-02 18:30:17
2	2	bronze	b.mp4	2018-07-29 ...	2018-08-02 10:47:40
3	3	silver	c.mp4	2018-07-29 ...	2018-08-02 18:30:18
4	4	bronze	d.mp4	2018-07-29 ...	2018-08-01 16:25:19
5	5	silver	e.mp4	2018-07-29 ...	2018-08-02 18:30:18
6	6	gold	f.mp4	2018-07-29 ...	2018-08-01 17:55:23

- level
 https://i.imgur.com/OArMB1b.png " />

1. cid를 key 값으로 현재 위치와 counts에 따른 목적 위치가 다른 경우 status 에 'update' 라는 문자열을, 현재위치와 목적 위치가 같은 경우 status에 'done' 이라는 문자열을 삽입하여 redis database에 json 형식으로 set (worker_id 는 이 후 단계에서 할당되기 때문에 null값으로 초기화) (e.g. {'3':{"cid": "3", "count": "664", "target": "bronze", "db_level": "silver", "filename": "c.mp4", "worker_id": null, "status": "update"}} 형식으로 저장)

실행결과

```
# 컨텐츠 정보 쿼리 및 redis 업데이트 example
foo@bar:~/$ curl http://192.168.10.108:5000/post_sentence -d "cid=7&count=1964"
{"cid": "7", "count": "1964", "target": "silver", "db_level": "bronze",
"filename":"g.mp4", "worker_id": null, "status": "update"}
```

redis 값 체크 및 MySQL database 업데이트(Work Flow. 7)

1. worker가 파일을 재배치한 후 해당 contents 의 status 를 'done' 으로 바꾸고 API 를 호출 (e.g. curl http://192.168.10.108:5000/update_sentence -d "cid=3")
2. request를 받으면 cid 를 key 값으로 redis에서 해당 content의 status 가 'done' 인지 검사하고 MySQL의 contents table 에 새로운 level 과 update time 을 업데이트
   만약 status 가 'done' 이 아니면 "check your status again" 메세지를 반환

```
# redis 값 체크 및 MySQL database 업데이트 example
# db update 후 해당 content 의 cid 반환
```

```
foo@bar:~/$ curl http://192.168.10.108:5000/update_sentence -d "cid=7"
7
foo@bar:~/$ curl http://192.168.10.108:5000/update_sentence -d "cid=8"
check your status again
```

database 에서 cid 7 의 content_level과 update_time 업데이트

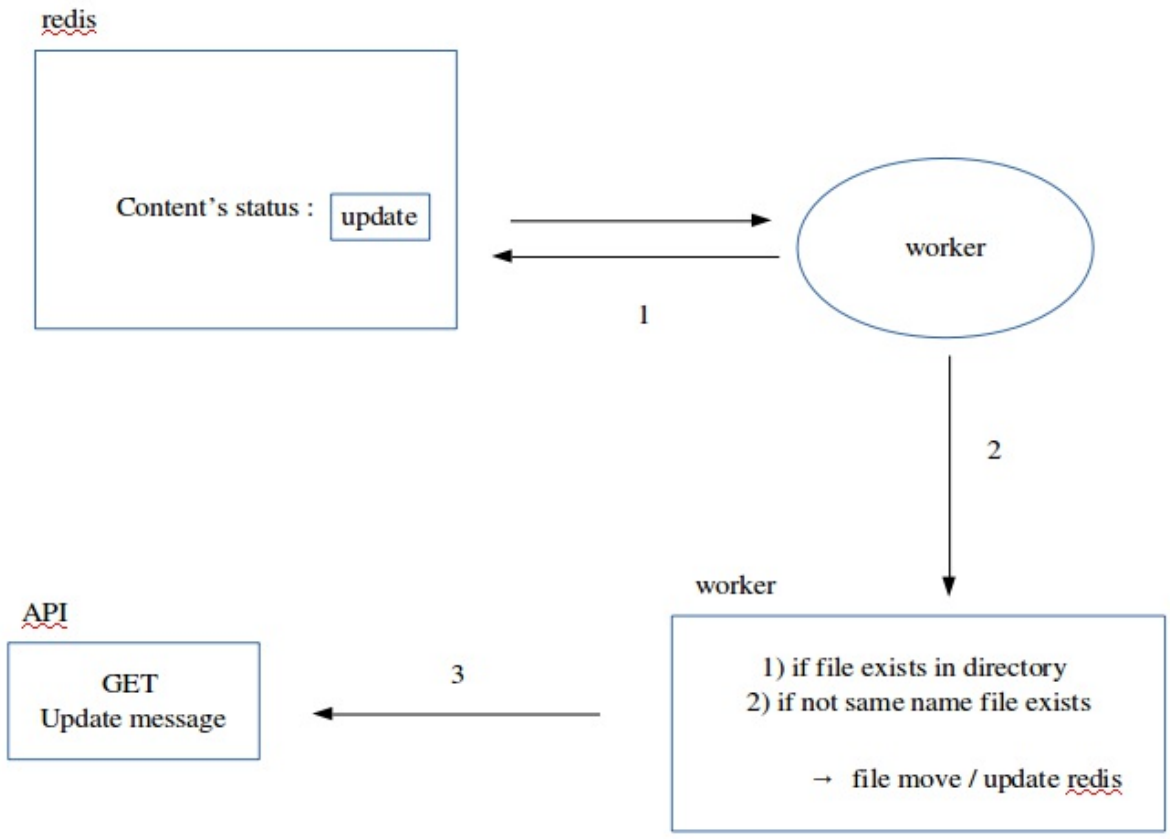
#	cid	content_level	filename	generate_time	update_time
7	7	silver	g.mp4	2018-07-29 …	2018-08-06 17:30:16

3. Worker

개발환경

- 서버
  - ubuntu (16.0.4 version)
- 백엔드
  - Framework : Python(3.5.2)
  - Database : mysql(14.14 Distrib 5.7.23)
  - Editor : vi

실행 흐름



1. redis의 콘텐츠 중 상태가 'update'라고 표시 된 것이 있으면 worker는 이를 확인하고 worker_id를 생성

2. 콘텐츠가 실제 경로에 존재하는지, 이동할 경로에 똑같은 이름의 파일이 있는지 확인 후, 콘텐츠의 이동 을 결정.이동 후 콘텐츠의 상태를 'done'으로 업데이트

3. API를 호출하여 성공적으로 worker의 수행이 끝났음을 알림

프로그램 설명과 결과물

프로그램 핵심 모듈

- import redis : redis에 저장된 정보를 확인 할 수 있는 모듈

```
redis_db = redis.StrictRedis(host='192.168.10.37', port=6379, db=1)
```

- import pymysql : 개발 서버의 데이터베이스에 저장된 정보를 확인 할 수 있는 모듈

```
conn = pymysql.connect(host='192.168.10.37',user='root',password='ini6223',db='redis_project',charset='utf8')
curs = conn.cursor()

sql = "select path from level"
curs.execute(sql)
```

- import shutiil : 콘텐츠 이동을 위한 모듈

```
shutil.move( goldpath + filename , silverpath + filename)
```

- import requests : API에게 worker의 작업이 끝났음을 알리기 위한 모듈

```
r = requests.post('http://192.168.10.108:5000/update_sentence', data = {'cid': cid})
```

- import asyncio : worker가 비동기로 작동하게 할 모듈

```
async def redis_func():
```

```
loop = asyncio.get_event_loop()
loop.run_until_complete(redis_func())
loop.close()
```

Worker의 비동기성

- async를 통하여 worker는 비동기로 실행
- 여러 개의 worker가 동시에 redis에 저장된 콘텐츠들의 상태를 확인하고 실행하여도 똑같은 파일이 동시에 업데이트 되는 일은 발생하지 않음

실행화면

- worker1

```
1s' status is not update and worker_id is not null
2
b.mp4 not exists
3s' status is not update and worker_id is not null
4
d.mp4 moves bronze to gold
5s' status is not update and worker_id is not null
```

- worker2

```
1s' status is not update and worker_id is not null
2
b.mp4 moves bronze to silver
3s' status is not update and worker_id is not null
4s' status is not update and worker_id is not null
5
e.mp4 moves silver to gold
```

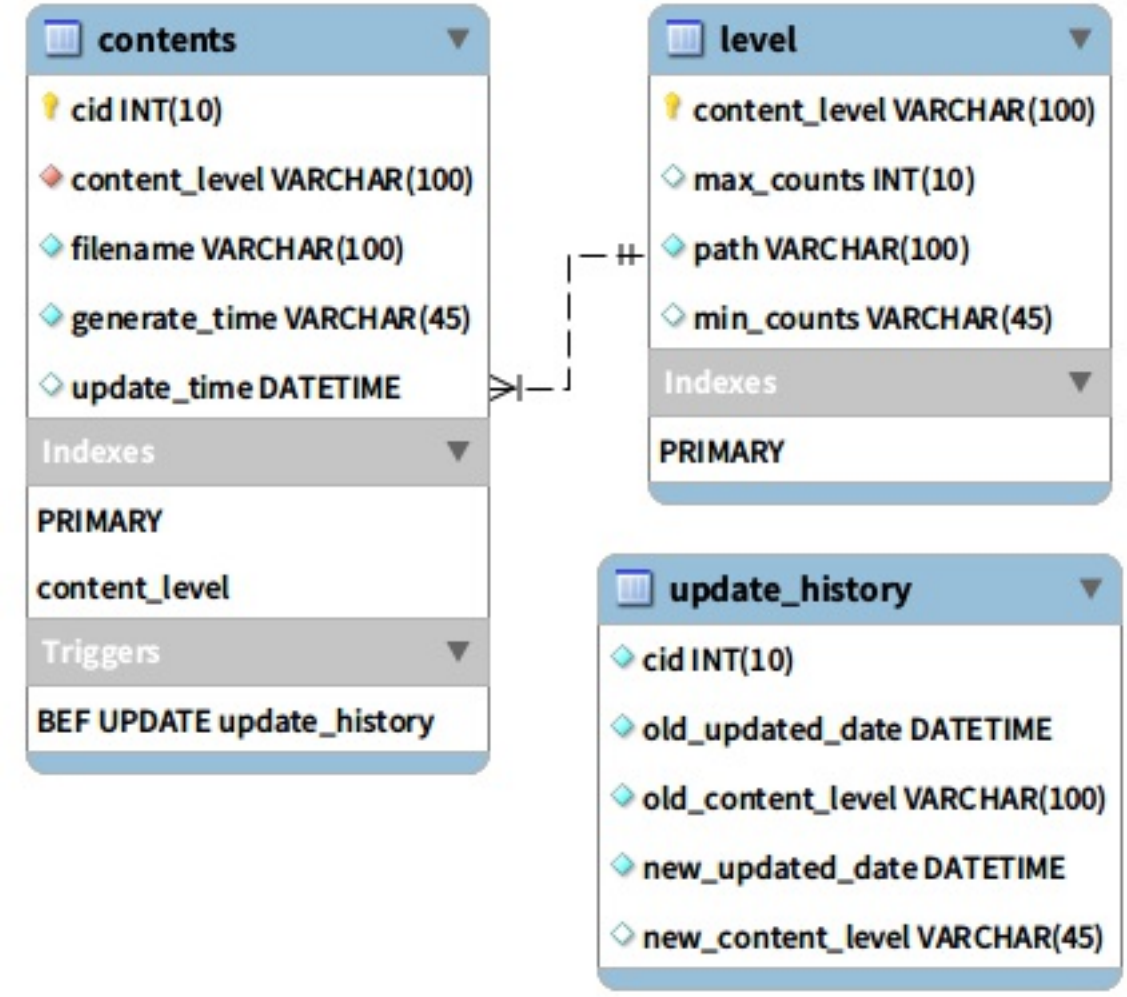
4. DataBase

개발 환경

- Ubuntu 16.04.5 LTS
- Python 3.5
- MySQL 5.7.23
- PyMySQL 0.9.2

DB 구축

1. ERD



2. DB 테이블

- level: content의 view count 수에 따른 level 정보를 저장하는 테이블
- contents: content의 정보를 저장하는 테이블
- update_history: contents level 정보가 업데이트 될 때마다 update history를 저장하는 테이블로 contents 테이블의 `content_level`에 업데이트 이벤트가 발생하면 데이터를 insert 하는 트리거와 연결

3. DB 트리거

- contents 테이블의 업데이트 이벤트가 발생하면 트리거가 작동되고 update_history에 업데이트 시점과 업데이트 된 content level이 기록됨
- 트리거

```
sql CREATE DEFINER = `root`@`localhost` trigger update_history before update on
```

`contents for each row begin insert into update_history values(old.cid, old.update_time, old.content_level, now(), new.content_level); end`

DB 쿼리 클래스 생성

프로세스 내에서 필요한 db 쿼리 클래스 생성

1. 클래스 구조

```
class db:
  def __init__(self):
    ...

  def select(self, table, column, where_clause=None, order_by=None):
    ...

  def insert_contents(self, table, cid, file_name):
    ...

  def update_level(self, cid, content_level):
    ...
```

2. 함수 설명

1. __init__

- db 클래스 생성 시 `PyMySQL` 로 MySQL 연결할 때 필요한 파라미터 설정
- `pymysql.connect()` 를 이용하여 MySQL 연결

2. select

- api 서버와 db 간 데이터 상호 교환 시 반복 사용되는 MySQL select 쿼리문을 모듈화 한 것
- `table` , `column` , `where_clause` , `order_by` 가 함수 인자이며 이 중 필수 인자는 `table` 과 `column`

3. insert_contents

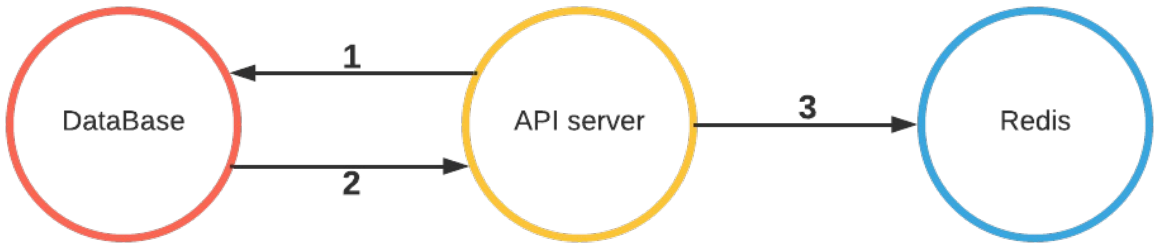
- content가 새로 업로드 될 때 콘텐츠에 대한 정보를 MySQL contents 테이블에 insert 해주는 기능을 모듈화 한 것
- min_count가 0인 level이 default `level`
- `generate time` 은 insert 쿼리문 실행 시점이 적용되므로 본 함수는 콘텐츠가 업로드 됨과 동시에 실행되어야 함

4. update_level

- api 서버가 본 함수를 호출하면 MySQL contents 테이블에서 content_level을 update 하는 쿼리 문을 모듈화 한 것
- update 하고자 하는 콘텐츠의 `cid` 와 relocate 된 `contet_level` 이 필수 인자
- 본 함수가 실행되면 contents table에서 content_level과 update_time column이 업데이트 이전에 update_history 테이블과 연결된 트리거가 먼저 작동

프로세스 상세 및 실행 결과

redis에 content 정보 업데이트



1. api 서버 - db 쿼리 함수 호출

- api 서버는 content의 현재 위치 level과 count가 해당되는 범위의 위치 level 비교를 위해 쿼리 모듈로 db에서 데이터를 쿼리 함
  - content 테이블에서 user로부터 받은 cid로 해당 content의 현재 위치 level을 쿼리 하는 `select 함수` 호출
  - `select 함수` 로 level 테이블의 모든 coulmn을 쿼리 한 후 user로부터 받은 count와 비교 연산하여 target level을 반환함

2. db - 쿼리 결과 api 서버에 반환

- db는 `select 함수` 로 쿼리 된 결과를 api 서버에 반환함
  - MySQL content 테이블 내의 cid 1 정보

#	cid	content_level	filename	generate_time	update_time
1	1	gold	a.mp4	2018-07-29 15:31:24	2018-08-02 18:30:17

  - api 서버에서 `select 함수` 를 호출하여 cid가 1인 content의 level 추출하는 함수 결과
    ```bash
    #python shell

> > > import redis_encode as re
> > > re.get_level_from_db(1)
> > > {'content_level': 'gold', 'file_name': 'a.mp4'}
```

```

- MySQL level 테이블

| # | content_level | max_counts | path | min_counts |
|---|---|---|---|---|
| 1 | bronze | 999 | /etc/inisoft/redis_project/bronze/ | 0 |
| 2 | gold | 10000000 | /home/inisoft/workspace/inisoft/redis_project/gold/ | 2000 |
| 3 | silver | 1999 | /var/www/html/redis_project/silver/ | 1000 |

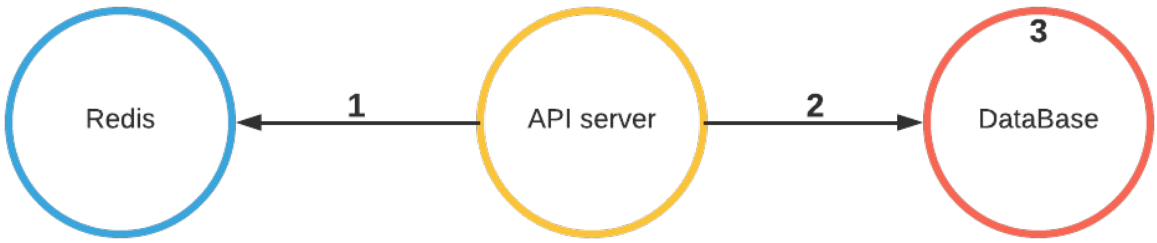
- api 서버에서 `select 함수`로 level 테이블을 쿼리 한 후 각 level의 count와 countent의 count를 비교 연산하여 target level을 반환하는 함수 결과
  ```bash
  #python shell

```
import redis_encode as re
re.get_target(2342)
'gold'
re.get_target(1548)
'silver'
re.get_target(356)
'bronze'
```
```

3. api 서버 - redis에 content에 대한 정보 업데이트

- db에서 쿼리 한 데이터를 알맞게 처리한 후 redis에 content 정보를 업데이트

relocate가 완료된 content 정보 업데이트



1. api 서버 - redis 업데이트

- content의 relocate를 마친 worker의 호출을 받은 api 서버는 해당 cid에 대해 redis의 content status가 'done' 인지 확인

2. api 서버 - db 업데이트
```

- redis의 content status가 'done' 인 것을 확인 한 api 서버가 쿼리 모듈로 db에서 content status 를 업데이트 하는 `update_level 함수`를 호출

  - `update_level 함수` 호출 전 MySQL content 테이블 내의 cid 12 정보

#	cid	content_level	filename	generate_time	update_time
12	12	silver	l.mp4	2018-07-29 15:31:27	2018-08-02 18:30:19

  - api 서버에서 `update_level 함수` 호출 결과
    ```bash

```
import redis_encode as re
re.update_db_level(12, 'gold')
1
```
```

- `update_level 함수` 호출 후 MySQL content 테이블 내의 cid 12 정보

#	cid	content_level	filename	generate_time	update_time
12	12	gold	l.mp4	2018-07-29 15:31:27	2018-08-07 14:27:44

3. db 업데이트

- api 서버의 쿼리 모듈 호출로 content 테이블이 업데이트 되면 db에 `insert 트리거`가 작동하여 content 테이블 업데이트 시점에 update_history 테이블에 row가 추가됨

  - MySQL content 테이블의 `insert 트리거` 작동 전 cid 12 의 update_history 테이블

#	cid	old_updated_date	old_content_level	new_updated_date	new_content_level
1	12	2018-05-01 12:32:45	gold	2018-07-26 11:26:18	silver
2	12	2018-07-26 11:26:18	silver	2018-08-01 17:55:23	gold
3	12	2018-08-01 17:55:23	gold	2018-08-02 18:30:19	silver
4	12	2018-08-02 18:30:19	silver	2018-08-06 12:34:08	silver

  - MySQL content 테이블의 `insert 트리거` 작동 후 cid 12 의 update_history 테이블

#	cid	old_updated_date	old_content_level	new_updated_date	new_content_level
1	12	2018-05-01 12:32:45	gold	2018-07-26 11:26:18	silver
2	12	2018-07-26 11:26:18	silver	2018-08-01 17:55:23	gold
3	12	2018-08-01 17:55:23	gold	2018-08-02 18:30:19	silver
4	12	2018-08-02 18:30:19	silver	2018-08-06 12:34:08	silver
5	12	2018-08-02 18:30:19	silver	2018-08-07 14:27:44	gold

5. 후기