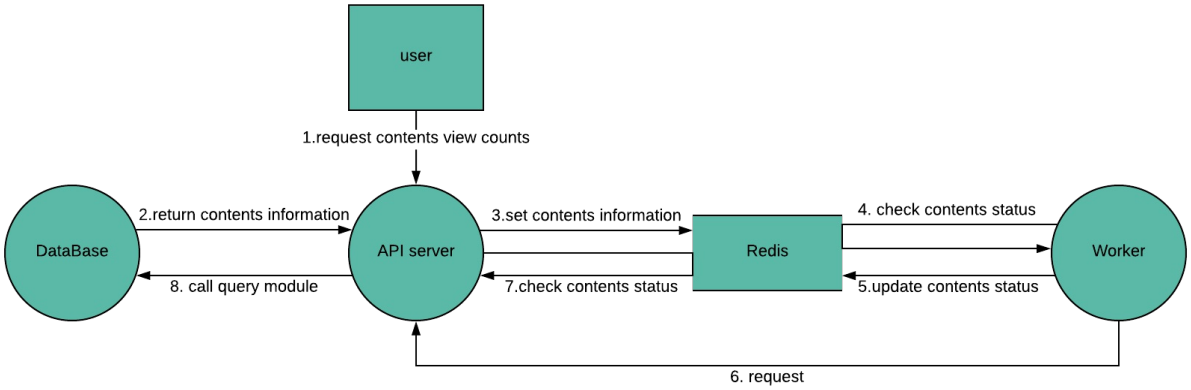
INI 2018 Intern Final Project

장예훈 (DB 구축 및 쿼리 모듈화)
18.07.27 Tue - 18.08.02 Thu

Project Process



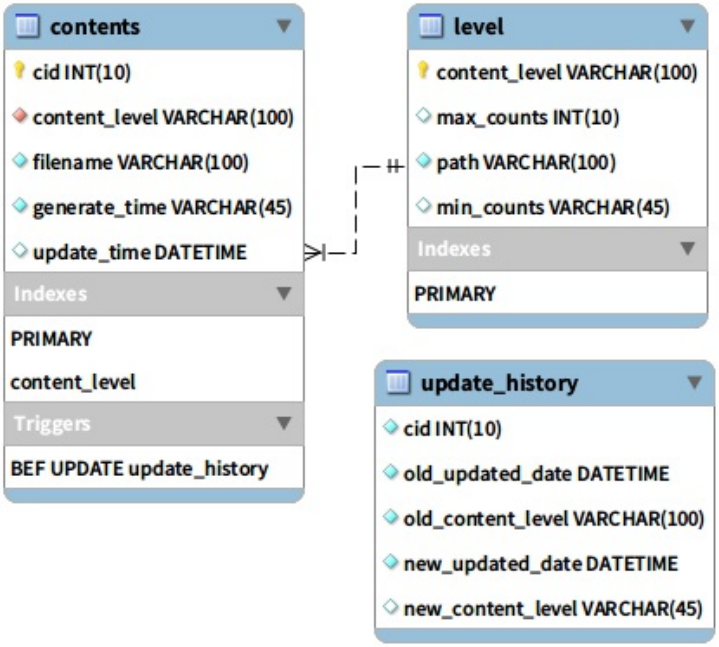
DB 프로세스 (Process 2 & 8)

1. 개발 환경

- Ubuntu 16.04.5 LTS
- Python 3.5
- MySQL 5.7.23
- PyMySQL 0.9.2
- Anaconda 4.3.11

2. DB 구축

2.1 ERD



2.2. DB 테이블

2.2.1 level

content의 view count 수에 따른 level 정보를 저장하는 테이블

- `content_level` : level의 이름
- `max_counts` : 해당 level의 maximum count
- `min_counts` : 해당 level의 minimum count
- `path` : 해당 level의 content가 저장 될 절대 경로

> ### Issue of 2.2.1.
>
> **api 서버에서 target level을 구하는 코드가 현재 level 구조 *(gold-silver-bronze)* 에 specific함**
>
> 1. Issue 설명
>
> - level 설정 수정에 영향 받지 않는 general 한 target level 추출 기능으로 수정이 요구됨
> - 수정된 target level 추출 기능에서 level count의 범위 필요
>
> 1. Issue 해결 방안
>
> - level 테이블의 기존 'count' 컬럼은 'max_count'로 이름이 변경되고 'min_count' 컬럼 추가
> - level의 구조가 변경되거나 count 수가 변경되어도 db 업데이트 외의 별도의 코드 수정은 요구되지 않음

2.2.2. contents

content의 정보를 저장하는 테이블

- `cid` : content를 구분하는 id
- `content_level` : content의 현재 level, level 테이블의 `content_level` 을 참조
- `filename` : content의 filename, worker가 relocate 할 때 경로 설정을 위해 필요
- `generate_time` : content 최초 생성 시간
- `update_time` : content의 level relocate 시간

> ## Issue of 2.2.2.
>
> **content generate time 이 요구**
>
> 1. Issue 설명
>
> - 초기의 content 테이블에는 'update_time' 컬럼만 존재
> - content의 level이 수정되면 'update_time'도 함께 수정되기 때문에 generate time 정보를 잃게 됨
> - content의 generate time을 확인하려면 'update_history' 테이블에서 해당 content의 로그를 전부 불러와야 하는 비효율적 작업 발생
>
> 1. Issue 해결 방안
>
> - content 테이블에 'generate_time' 컬럼 추가한 후 content가 업로드 되는 시점을 저장

2.2.3. 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

- `cid` : 업데이트 된 content의 id
- `old_updated_date` : content level의 현재 업데이트 이전 업데이트 시점
- `old_content_level` : content level의 현재 업데이트 이전 content level
- `new_updated_date` : content level의 현재 업데이트 시점
- `new_content_level` : content level의 현재 업데이트 content level

2.3. 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`

3. DB 쿼리 클래스 생성

프로세스 내에서 필요한 db 쿼리 클래스 생성

3.1. 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):
    ...

  def check_max_count(self, count):
    ...
```

3.2. 함수 설명

3.2.1. __init__

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

3.2.2. 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 파라미터이며 이 중 필수 파라미터는 `table` 과 `column`

> ### Issue of 3.2.2.
>
> **select sql 조건문 파라미터**
>
> 1. Issue 설명

> - 초기의 select 함수는 'table'과 'column' 파라미터만 존재
> - select 함수에서 where clause와 order_by clause에 해당하는 기능이 요구됨
> - where clause와 order_by clause는 상황에 따라 선택적으로 필요
>
> 1. Issue 해결 방안
>
> - select 함수에 'where_clause'와 'order_by' 파라미터 추가
> - 필수 파라미터는 'table'과 'column'이므로 'where_clause'와 'order_by'의 초기값은 None으로 설정
> - 각 파라미터에 값이 존재하는지 확인 후 조건에 맞는 sql문을 생성하여 db 쿼리
> - select 함수 하나와 파라미터 설정으로 프로젝트 내에서 필요한 select sql문을 구현

3.2.3. 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

3.2.4. update_level

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

3.2.5. check_max_count

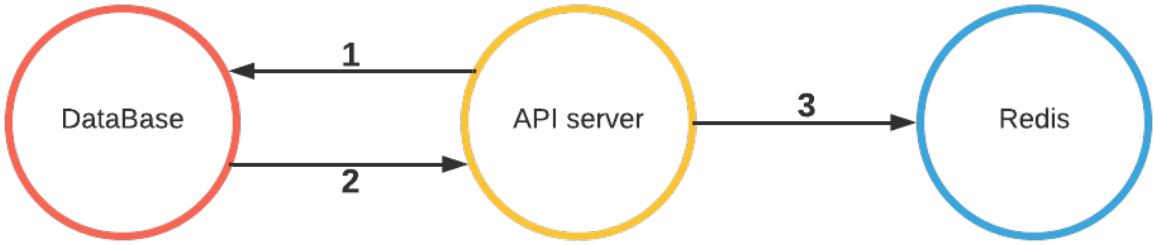
- MySQL level 테이블에서 최상위 level의 max_counts와 함수의 파라미터로 들어온 count의 크기를 비교하여 더 큰 count로 최상위 level의 max_counts를 update 해주는 기능을 모듈화 한 것
- api 서버에서 target level을 추출하기 위해 `select 함수`로 level 테이블을 쿼리하기 전에 실행됨

> ## Issue of 3.2.5.
>
> **최상위 level의 max_count 값에 대한 이슈**
>
> 1. Issue 설명
>
> - 초기의 최상위 level의 max_counts는 임의로 100000으로 설정
> - count가 100000을 초과한 값이 들어왔을 시 level을 할당 할 수 없는 이슈 발생

> 1. Issue 해결 방안
>
> - 최상위 level의 max_counts 값과 content 의 count 크기를 비교 후 max_counts 정보를 핸들링하는 모듈 추가
> - db에 저장 돼있던 max_counts가 더 클 때는 아무 이벤트가 발생하지 않음
> - content의 count가 더 클 때는 db의 최상위 level 의 max_counts 를 content의 count로 업데이트 함

4. 프로세스 상세 및 실행 결과

4.1. redis에 content 정보 업데이트



4.1.1. 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

4.1.2. 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

```
#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

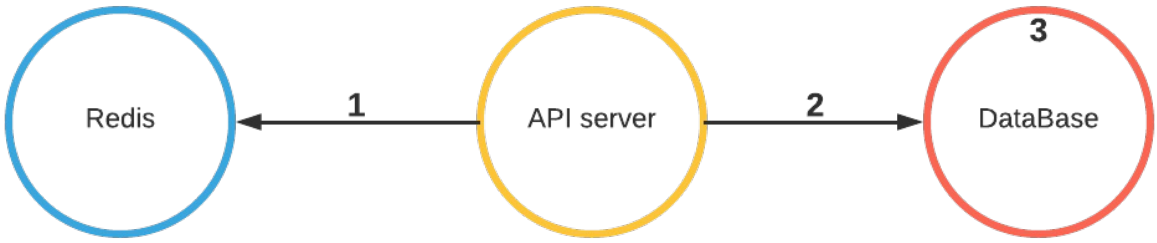
- api 서버에서 `select 함수`로 level 테이블을 쿼리 한 후 각 level의 count와 countent의 count를 비교 연산하여 target level을 반환하는 함수 결과

```
#python shell
\>>> import redis_encode as re
\>>> re.get_target(2342)
'gold'
\>>> re.get_target(1548)
'silver'
\>>> re.get_target(356)
'bronze'
```

4.1.3. api 서버 - redis에 content에 대한 정보 업데이트

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

4.2. relocate가 완료된 content 정보 업데이트



4.2.1. api 서버 - redis 업데이트

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

4.2.2. 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

```
\>>> 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

4.2.3. 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. 프로젝트 후기

이번 프로젝트를 진행하면서 가장 많이 배운 점은 코드의 완성도를 높이는 방법에 대한 고민과 각 구현

기능들이 요구사항에 얼마나 적합한 지 검토하는 방법인 것 같다. 다른 파트들에 비해 DB 모듈화는 개인적으로 러닝커브나 코딩 난이도, 진입장벽이 낮았기 때문에 스스로도 해당 기능 완성도에 대한 기대치가 높았다.
기한 내에 요구사항에 완벽히 들어 맞는 기능을 구현하기 위해서는 설계부터 사용자들의 피드백까지 체계적인 단계가 필수적이라고 생각하였다. 그래서 팀원들과 회의도 수차례 진행하고, 요구사항 명세화 작업도 하며 완성도에 집중하였고 결국엔 요구사항에 부합하는 프로세스를 구현하였다.
처음부터 내가 직접 설계하고 구현하며 피드백을 받아 보완하는 스텝을 차례대로 겪고 나니 성취감도 높았고, 프로젝트에 대한 애착이 생겼으며 다른 프로젝트를 진행할 때도 막막한 느낌 없이 잘 해낼 수 있을 것 같다.