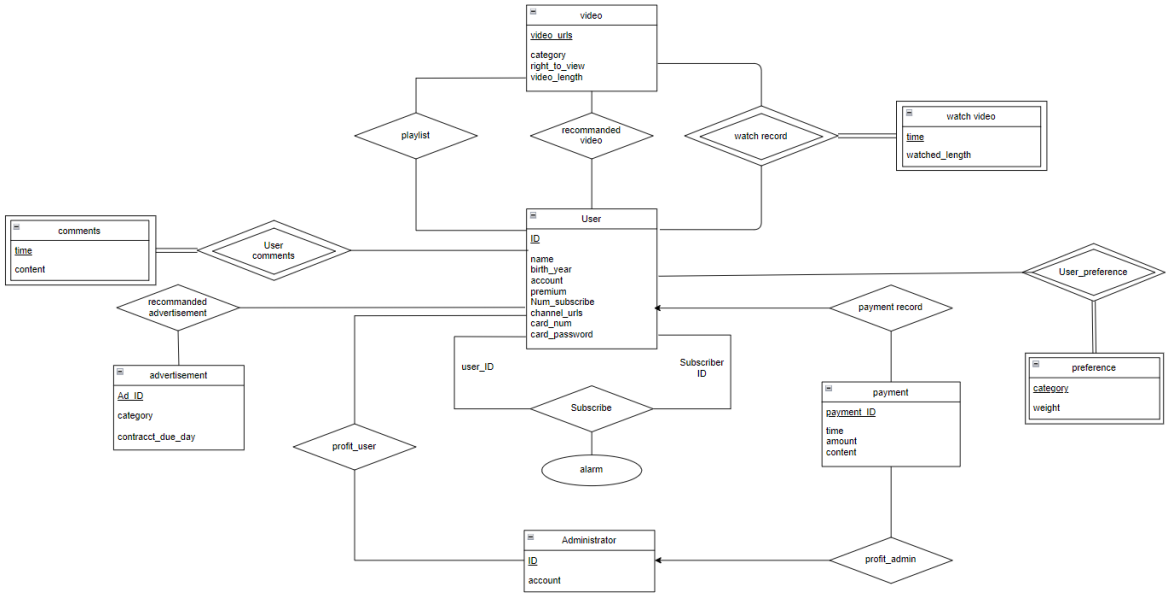
레포트 VI

소프트웨어학과 20186889 권용한

1. 응용분야의 제목

유투브 데이터 베이스

2. ERD 및 작성에 대한 설명



User entity는 youtube의 사용자에 대한 정보가 담겨 있습니다. Youtube는 User와 영상 또는 User와 다른 User사이의 상호작용을 이루게 됩니다. 이때 User와 다른 User가 상호작용을 하는 경우는 특정 사용자가 다른 사용자를 구독하는 경우입니다.

특정 사용자가 다른 사용자를 구독한다면 다른 구독자가 영상을 올리는 등 업데이트가 일어날 경우 알람을 받을 수 있는데, 이때 알람을 받을 것인지 받지 않을 것인지에 대한 정보가 alarm에 저장됩니다. 따라서 Subscribe관계는 alarm을 attribute로 가지게 됩니다. 이때 사용자는 여러 다른 사용자를 구독할 수 있으며, 때문에 이들의 관계가 다대다의 관계를 이룹니다. 이때 구독을 하는 사용자의 ID를 user_ID, 구독의 대상이 되는 사용자의 ID를 Subscriber_ID로 하여 저장하게 됩니다.

사용자가 다른 사용자에게 후원을 하거나 또는 유투브에서 유료로 제공하는 컨텐츠를 이용하게 되면 결제 정보가 남게 되며, 각 결제 정보는 개별적인 ID를 가지게 됩니다. 이때 각 결제 정보는 특정 사용자에 대해서만 연관되기 때문에 일대다의 관계를 이루게 됩니다. 각각의 결제는 관리자에게 모두 합산된 이후, 특정 사용자에게 후원 된 금액 등의 이익에 대해서는 정산 이후 특정 사용자에게 보내지게 됩니다. 또한 한 결제 건에 대하여 한 명의 관리자가 전담한다고 한다면, 결제와 관리자 사이에 일대다의 관계를 이룹니다. 하지만 관리자가 이후 사용자에게 수익을 배분하는 과정에서는 관리자가 특정 사용자에게 전담하는 것이 아닌 결제 건당 전담이기에 일대다의 관계가 아닌 다대다의 관계로

보는 것이 적절합니다.

사용자는 특정 영상이나 여러 활동에 대해서 댓글을 남길 수 있으며, 이런 댓글 또한 entity로 표현됩니다. 이때 comment는 단독으로 독립할 수는 없으며, 사용자 entity와 결합할 경우 독립적으로 구분될 수 있게 됩니다. 때문에 comment는 weak relation이 되고, comment를 구분하는 작성시각은 comment entity의 구분자가 됩니다.

사용자 이외의 entity로 영상이 있는데, 각 영상은 그 영상이 업로드 된 url 정보를 기본키로 구분됩니다. 영상의 경우 사용자와 관계를 이루는 경우가 많습니다. 우선 사용자가 영상을 시청하게 된다면 각 사용자의 시청기록에 대한 정보를 기록하게 됩니다. 이때 시청한 영상 중 중복으로 시청하는 경우도 있기에, 이런 경우 시청시각을 기준으로 구분해야 합니다. 때문에 시청영상이라는 weak entity를 만들게 되었으며, 시청영상은 영상entity와 사용자entity와 관계를 가지며 이 관계는 시청기록 관계로 기록됩니다. 이때 이 관계가 ternary 관계를 이루며, 그 중 시청영상은 weak entity입니다. 이를 통해 시청기록은 User_ID와 video_urls 그리고 시청영상의 구분자인 시청시각을 기본키로 가지가 됩니다.

이외에도 사용자는 youtube에서 추천되는 영상을 얻을 수 있으며, 사용자가 선호하는 영상을 따로 모아 재생목록으로 만들 수도 있습니다. 이 둘은 모두 사용자ID와 video_urls로 이루어졌지만 의미적으로 구분되는 관계입니다. 이들의 관계는 다대다의 관계를 이루게 됩니다.

사용자에게 영상을 추천하기 위해서는 사용자의 선호도에 대한 정보가 있어야 하며, 이 정보는 사용자선호도 관계에 저장됩니다. 사용자 선호도는 사용자와 선호도의 결합을 통해 얻어지며, 선호도는 각 영상의 category를 구분자로 가지며, 각 category별로 가중치를 가지게 됩니다. 선호도는 독립적으로는 구분될 수 없지만 사용자와의 결합을 통해 의미를 가지게 구분되는 weak entity이며 앞에서 언급했듯 category를 구분자로 가지게 됩니다.

마지막으로 유투브는 사용자에게 영상 추천과 유사한 방식으로 사용자의 취향이나 선호도에 따라 추천광고를 제공합니다. 이때 추천광고는 사용자와 광고entity의 관계를 통해 얻어지며, 광고는 각 광고 별로 ID를 가지며, 각 광고별로 category를 가지며 유투브와의 계약을 통해 제공되는 것이므로, 광고 계약 종료 기간에 대한 정보를 가지게 됩니다. 이 광고entity와 사용자entity의 결합을 통해 사용자에게 추천되는 광고를 얻을 수 있습니다.

3. RDB 테이블 스키마

테이블 스키마 :

User(ID, name, birth_year, premium, subscribe_num ,channel_urls, card_num, card_password, account)

administrator(ID, account)

payment(payment_ID, time, amount, content)//일대다

payment_record(User_ID, payment_ID)//일대다

profit_admin(payment_ID, admin_ID)//일대다

profit_user(User_ID, admin_ID)//다대다

Subscribe(User_ID, Subscriber_ID, alarm)

User_preference(User_ID, category, weight)

video(video_urls, category, right_to_view, video_length)

recommended_video(User_ID, video_urls)

playlist(User_ID, video_urls)

watch_record(User_ID, video_urls, time, watched_time)

comment(User_ID, time, content)

advertisement(Ad_ID, category, contract_due_day)

recommended_advertisement(User_ID, Ad_ID)

위 결과는 주어진 er diagram에서 모든 entity와 relation을 table로 최대한 교체한 결과이며, 선호도와 댓글, 시청영상 등의 weak한 entity의 경우 다른 entity와 결합하여 relation으로 table을 표현하였습니다.

이때 구매이력은 결제정보당 사용자ID가 고정되기 때문에 정보의 중복이 일어날 수 있다. 때문에 함수 종속을 고려하였을 때 구매이력에 대한 정보를 결제와 합쳐서 새로운 관계를 형성하는 것이 더 적절하다고 판단됩니다.

●수정 전

payment(payment_ID, time, amount, content)

payment_record(User_ID, payment_ID

●수정 후

Payment_record(payment_ID, User_ID, time, amount, content)

이때 결제와 사용자 사이에 일대다 관계가 성립하기 때문에 primary key를 다쪽인 결제 table의 primary key인 결제정보ID로 하였다.

Payment_record(payment_ID, User_ID, time, amount, content)

profit_admin(payment_ID, admin_ID)

그리고 다시 수정후의 결제table과 관리자 수익table을 본다면, 한 payment에 대해선 관리자가 딱 1명 정해지게 된다. 따라서 payment_ID의 정보가 중복으로 발생하게 되는데 이를 해결하기 위해

●수정 전

Payment_record(payment_ID, User_ID, time, amount, content)

profit_admin(payment_ID, admin_ID)

●수정 후

Payment_record(payment_ID, User_ID, admin_ID, time, amount, content)

로 수정을 할 수 있다. 이를 통해 불필요하게 중복된 정보를 줄일 수 있게 되었다.

재생목록과 추천영상의 attribute는 동일하지만 실제 적용에서 사용된다면 입력되는 값이 달라질 것이기에 데이터의 중복으로 볼 수 없다.

이를 통해 최종적으로 얻어진 테이블 스키마는

User(ID, name, birth_year, premium, subscribe_num ,channel_urls, card_num, card_password, account)

administrator(ID, account)

Payment_record(payment_ID, User_ID, admin_ID, time, amount, content)

profit_user(User_ID, admin_ID)//다대다

Subscribe(User_ID, Subscriber_ID, alarm)

User_preference(User_ID, category, weight)

video(video_urls, category, right_to_view, video_length)

recommended_video(User_ID, video_urls)

playlist(User_ID, video_urls)

watch_record(User_ID, video_urls, time, watched_time)

comment(User_ID, time, content)

advertisement(Ad_ID, category, contract_due_day)

recommended_advertisement(User_ID, Ad_ID)

총 12개의 테이블 스키마를 얻을 수 있습니다.

4. 응용도 복잡도 높은 기능을 지원하기 위한 자연어 질의 리스트 및 해당 SQL문

1. 특정 사용자(A)의 시청기록을 통해 사용자의 취향 선호도 구하기

$$\Pi_{category,count(category)}(\sigma_{\text{videoUrls(?)}=\Pi_{video}(\sigma_{watchRecord.UserID="A"}(watchRecord))}(video))$$

Insert into User_preference

Select category, count(category) as weight

From video

Where video_urls In Select video_urls

From watch_record

Where User_ID="A"

Group by category;

2. 선호도를 기반으로 사용자에게 영상 추천하기

$$\Pi_{User.ID,video.video_{urls}}(\sigma_{User.ID=A\ and(\ \Pi_{category}\left(\sigma_{User_{ID}=A}(UserPreference)\right)\ -(\Pi_{category}\left(\begin{matrix}\sigma_{a.UserID=A,a.UserID=b.UserID,a.weight<b.weight}(\\ \rho_a(UserPreference)\times\rho_b(UserPreference))\end{matrix}\right))}(video\times User))$$

Insert into User_preference

Select U.ID, V.video_urls

From video as V, User as U

Where U.ID="A" and V.category In (Select category

From User_preference

Where User_ID="A")

Except

(Select a.category

From User_preference as a, User_preference as b

Where a.User_ID=b.User_ID, a.User_ID="A",a.weight<b.weight)

3. 특정 사용자(A)가 끝까지 보지 않은 영상을 구하기

$$\Pi_{W.videoUrls}\left(\sigma_{User_{ID}=A,W.videoUrls=V.videoUrls,W.watchedTime<V.videoLength}\left(\rho_{W}(watchRecord)\times\rho_{V}(video)\right)\right)$$

Select W.video_urls

From watch_record as W, video as V

Where User_ID="A" and W.video.urls=V.video.urls and W.watched_time<V.video_length

4. 특정 사용자(A)를 구독한 사람중 알람설정을 한 사람 구하기

$$\Pi_{UserID}(\sigma_{SubscriberID="A"}(subscribe))$$

Select User_ID

From subscribe

Where Subscriber_ID="A" and alarm = true;

5. 특정 사용자(A)가 시청한 영상의 시청권한 구하기

$$\Pi_{videoUrls,rightToView}(\sigma_{\text{videoUrls(?)}=\Pi_{video}(\sigma_{watchRecord.UserID="A"}(watchRecord))}(video))$$

Select video_urls, right_to_view

From video

Where video_urls In Select video_urls

From watch_record

Where User_ID="A"

5. 대화식 SQL도구로 DB생성 및 데이터 적재

```
mysql> create table User(
    -> id varchar(20) not null,
    -> name varchar(20) not null,
    -> birth_year int not null,
    -> premium boolean,
    -> channel_urls varchar(80),
    -> card_num char(8),
    -> card_password varchar(20),
    -> account varchar(15),
    -> primary key(id)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

우선 User 테이블을 생성하였습니다.

```
mysql> select * from user;
+-------+------+------------+---------+--------------+----------+---------------+------------+
| id    | name | birth_year | premium | channel_urls | card_num | card_password | account    |
+-------+------+------------+---------+--------------+----------+---------------+------------+
| 00001 | A    |       1999 |       0 | AAA          | 12345678 | AA001         | 326593671  |
| 00002 | B    |       2000 |       0 | BBB          | 12349876 | BB002         | 532651236  |
| 00003 | C    |       2000 |       1 | CCC          | 43219876 | CC003         | 5345121236 |
| 00004 | D    |       1998 |       1 | DDD          | 13249786 | DD004         | 5455432453 |
| 00005 | E    |       1973 |       0 | EEE          | 41211635 | EE005         | 75545321   |
| 00006 | F    |       2002 |       1 | FFF          | 46842945 | FF006         | 83123723   |
| 00007 | G    |       1997 |       0 | GGG          | 63972942 | GG007         | 30572383   |
| 00008 | H    |       1999 |       1 | HHH          | 78539964 | HH008         | 28632314   |
| 00009 | I    |       2000 |       1 | III          | 92315432 | II009         | 13546873   |
| 00010 | J    |       2010 |       0 | JJJ          | 82359638 | JJ010         | 953120215  |
+-------+------+------------+---------+--------------+----------+---------------+------------+
10 rows in set (0.00 sec)
```

위 사진은 user에 데이터를 적재하고, 입력한 데이터가 잘 작성되었는지 확인한 모습입니다.

Video와 watch_record table도 이와 같이 table을 생성하고 직접 데이터를 입력하여 주었습니다.

```
mysql> create table Video(
    -> video_urls varchar(80) not null,
    -> category varchar(30),
    -> right_to_view boolean,
    -> video_length float,
    -> primary key(video_urls)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

Video 테이블의 생성 코드입니다.

아래는 데이터를 입력한 결과를 출력한 모습을 확인할 수 있습니다.

```
mysql> select *from video;
+------------+----------+---------------+--------------+
| video_urls | category | right_to_view | video_length |
+------------+----------+---------------+--------------+
| V0001      | game     |             1 |         12.5 |
| V0002      | cook     |             1 |         13.8 |
| V0003      | game     |             0 |           10 |
| V0004      | game     |             0 |           23 |
| V0005      | cook     |             1 |         53.7 |
| V0006      | music    |             1 |         43.6 |
| V0007      | music    |             1 |           15 |
| V0008      | animal   |             1 |         11.4 |
| V0009      | game     |             0 |        126.5 |
| V0010      | cook     |             1 |         31.8 |
| V0011      | music    |             1 |         32.5 |
| V0012      | animal   |             1 |         19.4 |
| V0013      | cook     |             1 |         13.5 |
| V0014      | game     |             0 |          8.5 |
| V0015      | animal   |             1 |         16.5 |
| V0016      | music    |             1 |          2.9 |
| V0017      | cook     |             1 |           51 |
| V0018      | game     |             1 |         21.7 |
| V0019      | animal   |             1 |         14.4 |
| V0020      | animal   |             1 |         19.4 |
+------------+----------+---------------+--------------+
20 rows in set (0.01 sec)
```

동일하게 watch_record table도 테이블을 생성하고 데이터를 입력해 줍니다.

```
mysql> create table watch_record(
    -> user_id varchar(20) not null,
    -> video_urls varchar(80) not null,
    -> time float not null,
    -> watched_time float,
    -> primary key(user_id,video_urls,time)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

마찬가지로 데이터를 입력한 결과를 출력한 모습입니다.

```
mysql> select * from watch_record;
+---------+------------+-------+--------------+
| user_id | video_urls | time  | watched_time |
+---------+------------+-------+--------------+
| 00001   | V0001      | 138.6 |         11.3 |
| 00001   | V0006      |  16.6 |         36.3 |
| 00001   | V0016      |   196 |          1.3 |
| 00001   | V0019      |  28.9 |         14.4 |
| 00002   | V0001      |   1.6 |         12.5 |
| 00002   | V0003      |  17.6 |           10 |
| 00002   | V0007      | 103.8 |         13.8 |
| 00002   | V0020      | 153.9 |         15.3 |
| 00003   | V0003      |   1.6 |            6 |
| 00003   | V0008      | 133.6 |         11.4 |
| 00003   | V0019      |   151 |          1.3 |
| 00003   | V0020      |  11.6 |         18.3 |
| 00004   | V0002      |   1.7 |         12.5 |
| 00004   | V0010      | 131.8 |           28 |
| 00004   | V0013      |  19.6 |         13.5 |
| 00004   | V0019      |  23.3 |         14.4 |
| 00004   | V0020      | 165.9 |         17.3 |
| 00005   | V0005      |  59.3 |         53.7 |
| 00005   | V0006      |  16.7 |         43.6 |
| 00005   | V0007      |  27.6 |           15 |
| 00006   | V0001      |   3.2 |         12.5 |
| 00006   | V0019      |  18.9 |         14.4 |
| 00006   | V0020      |   131 |          1.3 |
| 00007   | V0001      |     1 |           10 |
| 00007   | V0009      |  17.6 |          110 |
| 00007   | V0010      | 197.8 |         31.8 |
| 00008   | V0020      |    15 |         19.4 |
| 00009   | V0010      | 138.6 |         31.8 |
| 00009   | V0011      |  39.6 |         26.3 |
| 00009   | V0016      | 193.8 |          2.9 |
| 00010   | V0001      |  31.5 |         12.5 |
| 00010   | V0005      |  69.3 |         53.7 |
| 00010   | V0014      |     1 |          8.5 |
+---------+------------+-------+--------------+
33 rows in set (0.00 sec)
```

사용자에게 추천영상을 제공하기 위한 추천영상 table도 생성하였으며, 이때 데이터는 JDBC에서 구하기 위해 데이터를 입력하지 않았습니다. 아래 사진은 테이블 생성 코드입니다.

```
mysql> create table recommended_video(
    -> user_id varchar(20) not null,
    -> video_urls varchar(80) not null,
    -> primary key(user_id,video_urls)
    -> );
Query OK, 0 rows affected (0.04 sec)
```

6. JDBC/MySQL 프로그램

1. 시청기록 검색

2. 추천영상

3. 종료

3이 입력되기 전까지 1과 2가 입력되면 시청기록 검색 또는 추천영상을 검색할 수 있으며, 이때 검색 기준은 사용자의 ID로 하였습니다. 아래는 실행결과를 확인할 수 있습니다.

```
1.시청기록 검색
2.추천영상 검색
3. 종료
2
검색을 원하는 사용자의 ID를 입력하시오
00005
User id: 00005  video_urls: V0006
User id: 00005  video_urls: V0007
User id: 00005  video_urls: V0011
User id: 00005  video_urls: V0016
1.시청기록 검색
2.추천영상 검색
3. 종료
2
검색을 원하는 사용자의 ID를 입력하시오
00006
User id: 00006  video_urls: V0008
User id: 00006  video_urls: V0012
User id: 00006  video_urls: V0015
User id: 00006  video_urls: V0019
User id: 00006  video_urls: V0020
1.시청기록 검색
2.추천영상 검색
3. 종료
1
검색을 원하는 사용자의 ID를 입력하시오
00006
Id: 00006       video_urls: V0001       time: 3.2       watched_time: 12.5
Id: 00006       video_urls: V0019       time: 18.9      watched_time: 14.4
Id: 00006       video_urls: V0020       time: 131.0     watched_time: 1.3
1.시청기록 검색
2.추천영상 검색
3. 종료
2
검색을 원하는 사용자의 ID를 입력하시오
00010
User id: 00010  video_urls: V0001
User id: 00010  video_urls: V0003
User id: 00010  video_urls: V0004
User id: 00010  video_urls: V0009
User id: 00010  video_urls: V0014
User id: 00010  video_urls: V0018
1.시청기록 검색
2.추천영상 검색
3. 종료
3
```

1번 과정은 단순 select구문을 반복하는 코드임을 확인할 수 있습니다.

```
if(checker==1) {
    System.out.println("검색을 원하는 사용자의 ID를 입력하시오");
    checker_st=sc.next();
    selectsql="select * from watch_record where user_id="+checker_st;
    Statement stmt= conn.createStatement();
    ResultSet rset = stmt.executeQuery(selectsql);
    while(rset.next()) {
        System.out.println("Id: "+rset.getString(1)+'\t'+"video_urls: "+rset.getString(2)+'\t'+"time: "+rset.getDouble(3)+'\t'+"watched_time: "+rset.getDouble(4));
    }
```

반면 2번의 과정은 주어진 값을 단순 계산하는 것이 아닌 주어진 다른 테이블의 값을 통해 원하는 테이블에 값을 입력하고 그 값을 확인하게 되도록 하였습니다. 이때 추천영상은 계속 반복적으로 업데이트 된다고 가정하여 2번이 입력된다면 추천영상테이블은 초기화를 시키고 거기에 시청기록 테이블에서 데이터를 받아와 여러 sql구문을 통해 최종적으로 원하는 추천영상 table을 구하도록 하였습니다.

```
}else if(checker==2) {
    String query="truncate table recommended_video";
    Statement trunstmt=conn.createStatement();
    trunstmt.execute(query);
    int[] cate_check= {0,0,0,0};
    System.out.println("검색을 원하는 사용자의 ID를 입력하시오");
    checker_st=sc.next();
    String VU="select video_urls from watch_record where user_id="+checker_st;
    Statement stmt_t= conn.createStatement();
    ResultSet rset_temp = stmt_t.executeQuery(VU);
    while(rset_temp.next()) {
        String video_urls=rset_temp.getString(1);
        String temp="select category from video where video_urls="+"'"+video_urls+"'";
        Statement stemp=conn.createStatement();
        ResultSet rset_t=stemp.executeQuery(temp);
        while(rset_t.next()) {
            String cate=rset_t.getString(1);
            if(cate.equals("game"))
                cate_check[0]++;
            else if(cate.equals("cook"))
                cate_check[1]++;
            else if(cate.equals("music"))
                cate_check[2]++;
            else if(cate.equals("animal"))
                cate_check[3]++;
        }
    }
int max_index=0;
int max=cate_check[0];
for(int i=0;i<4;i++)
{
    if(max<cate_check[i])
    {
        max=cate_check[i];
        max_index=i;
    }
}
```

```
String max_category;
if(max_index==0)
    max_category="game";
else if(max_index==1)
    max_category="cook";
else if(max_index==2)
    max_category="music";
else
    max_category="animal";
String selectSql="select video_urls from video where category="+"'"+max_category+"'";
Statement stmt= conn.createStatement();
ResultSet rset = stmt.executeQuery(selectSql);
while(rset.next()) {
    String insertsql="insert into recommended_video values(?,?)";
    PreparedStatement pstmt = conn.prepareStatement(insertsql);
    pstmt.setString(1,checker_st);
    pstmt.setString(2,rset.getString(1));
    pstmt.executeUpdate();
}
String selectS="select * from recommended_video where user_id="+checker_st;
Statement Stmt= conn.createStatement();
ResultSet Rset = Stmt.executeQuery(selectS);
while(Rset.next()) {
    System.out.println("User id: "+Rset.getString(1)+'\t'+"video_urls: "+Rset.getString(2));
}
```

7. BCNF정규화 및 스키마 정제

테이블 스키마

●User(ID, name, birth_year, premium, subscribe_num ,channel_urls, card_num, card_password, account)

FD:ID->name,birth_year,premium,subscribe_num,channel_urls,card_num,card_password,account

정보의 중복이 없으며 ID에 의해 구분되므로 함수종속을 만족한다고 볼 수 있습니다.

●administrator(ID, account)

FD:ID->account

정보의 중복이 없고, ID로 administrator의 계좌정보를 구분할 수 있습니다.

●Payment_record(payment_ID, User_ID, admin_ID, time, amount, content)

FD: payment_ID->User_ID, admin_ID, time, amount, content

원래 User_ID와 payment_ID, 그리고 admin_ID와 User_ID로 이루어진 테이블이 더 있었으나, 정보의 중복으로 인해 payment table에 함께 표현하였고, 이를 통해 단순 지불 정보 뿐만 아니라 구매자(user)와 판매자(admin)에 대한 관계를 더욱 쉽게 파악할 수 있게 되었습니다. 함수종속을 만족하면서 잘 설계되었다고 볼 수 있습니다.

●profit_user(User_ID, admin_ID)

FD: non-trivial 함수 종속이 없습니다. 이 경우도 BCNF는 충족시킨다고 볼 수 있습니다.

●Subscribe(User_ID, Subscriber_ID, alarm)

FD: User_ID, Subscriber->alarm

위의 profit_user와 유사하게 다대다 관계로 이루어져 있으나 relation이 attribute를 가지게 되면서 non-trivial함수종속을 가지게 되었습니다.

●User_preference(User_ID, category, weight)

FD: User_ID, category->weight

Weak entity인 preference와 strong entity인 User의 relation을 통해 얻어져 User_ID와 category가 함께 primary key를 이루고 BCNF를 잘 만족함을 확인할 수 있습니다.

●video(video_urls, category, right_to_view, video_length)

FD: video_urls->category, right_to_view, video_length

●recommended_video(User_ID, video_urls)

FD: non-trivial 함수종속이 없습니다.

●playlist(User_ID, video_urls)

FD: non-trivial 함수종속이 없습니다.

●watch_record(User_ID, video_urls, time, watched_time)

FD: User_ID, video_urls, time-> watched_time

3중관계를 통해 이루어져 있으며, 각 entity의 primary key 및 구분자의 조합이 watch_record의 primary key를 이루게 됩니다.

●comment(User_ID, time, content)

FD: User_ID, time->content

Weak entity인 comment가 User와 relation을 가지며 comment 테이블을 이루게 되었습니다. 이를 통해 User의 primary key인 ID와 comment의 구분자인 time이 함께 comment 테이블의 primary key를 이룹니다.

●advertisement(Ad_ID, category, contract_due_day)

FD: Ad-ID->category, contract_due_day

정보의 중복이 없고, Ad_ID로 category와 contract_due_day 정보를 구분할 수 있습니다.

●recommended_advertisement(User_ID, Ad_ID)

FD: non-trivial 함수종속이 없습니다.