산업정보관리론 – Game DB 구현

2021 IIE Term Project

2014147033 손명현

0. INDEX

중요
DB설명서.pdf를 꼭 읽어주시고, 먼저 Table
을 생성하고 예시 데이터를 삽입해 주세요.

1. DATA REQUIREMENT ANALYSIS

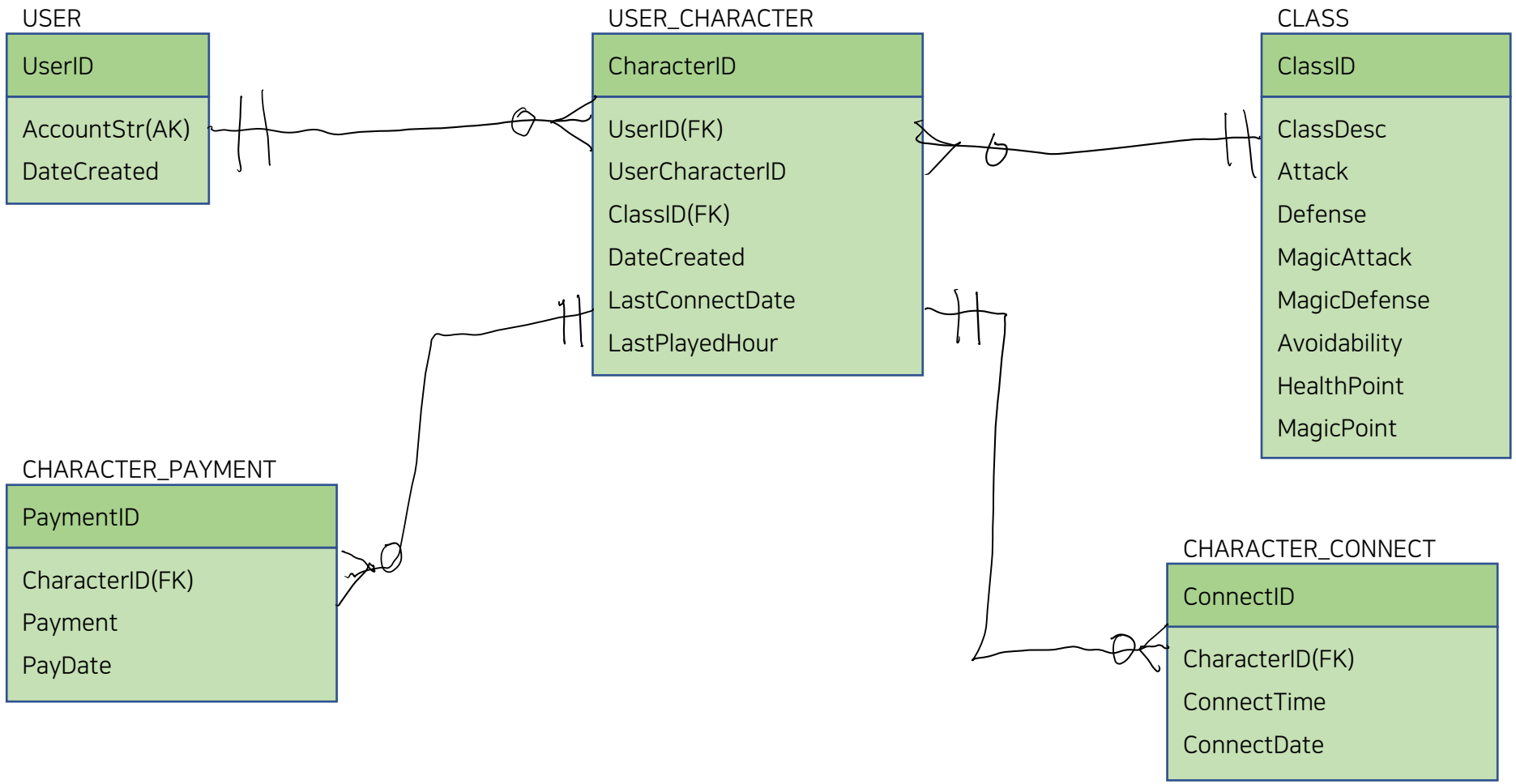
- 프로젝트 선정 배경
  - 2018, 2019 빅콘테스트 : RPG 게임 유저의 이탈을 예측하라 – 유저 정보를 통해 유저 이탈 원인 및 위험 유저 분석을 중요시

오래 전 출시했지만 아직까지 한국 게임순위 28위, 14위의 준수한 성적을 유지하고 있는 블레이드&소울과 리니지

- 시간이 지날수록 RPG게임은 더 복잡해지고 있고, 그에 따라 유저의 데이터 관리와 데이터 분석을 위한 데이터베이스는 계속해서 중요하게 인식됨
- 게임을 이용하면서 쌓이는 수많은 정보들 중 유저의 캐릭터 정보 및 유저의 접속 내역, 결제 내역 정보를 이용해서, 조건에 맞는 Insertion / Deletion 프로세스 및 RFM Analysis가 가능한 DB를 구현

2. E-R Model

- 유저의 정보와 유저의 접속 및 결제 내역 정보에 대한 ER 모델



2. E-R Model

- USER
  - UserID(PK), AccountStr, DateCreated
  - User 구분을 위한 Surrogate Key인 UserID와 계정명을 나타내는 AccountStr(AK), 그리고 생성된 날짜를 저장함

- USER_CHARACTER
  - 각 User는 최소 0개, 최대 6개까지의 게임 캐릭터(Character)를 생성할 수 있음
  - 각 Character는 어떤 직업군인지(Class)와 언제 생성되었는지, 언제 마지막으로 접속했으며 마지막 접속 시간은 몇 시간인지를 저장한다.
  - UserID 하나에 1~6까지의 UserCharacterID가 부여되며, CharacterID라는 surrogate Key가 위의 두 개 PK를 대체한다.

- CLASS
  - 캐릭터가 가질 수 있는 직업군(Worrier, Magician, Archer, Blacksmith)와 직업군 고유의 능력치를 표시
  - 고유 능력치는 물리 공격력/방어력, 마법 방어력/공격력과 회피능력, 체력과 마력으로 이루어져 있다.

- CHARACTER_PAYMENT
  - 게임 내 모든 캐릭터가 생성된 시점부터 현재까지의 게임에서의 결제 내역을 기록
  - 특정 캐릭터가 언제, 얼마의 Pay를 했는가를 기록

- CHARACTER_CONNECT
  - 게임 내 모든 캐릭터가 생성된 시점부터 현재까지 게임 접속 내역을 기록
  - 특정 캐릭터가 언제, 몇 시간 동안 접속했는지를 기록

2. E-R Model

- Relation 정의
  - 각 Table은 모두 Maximum Card. 1:N, Minimum Card. M-O의 관계를 가진다.
  - USER – USER_CHARACTER 사이에서는 각 User에 할당될 수 있는 Character의 최소 개수는 0개이며, 최대 개수는 6개이다.
  - CLASS – USER_CHARACTER 관계를 제외한 나머지 관계에서는 M-O 관계에서 'M' side가 삭제되면 'O' side의 개체들도 모두 Cascade Delete된다.
  - CLASS – USER_CHARACTER 관계에서는 CLASS의 개체 자체가 삭제되지 않음을 가정하며, 오직 CLASS 데이터의 Update만 가능하다.
  - USER_CHARACTER – CHARACTER_CONNECT 관계에서는 최신 Connect 내역이 기록될 때, USER_CHARACTER의 LastConnectDate와 LastPlayedHour가 갱신된다.

3. Data

- InsertData.xlsx 파일

UserID	Email	DateCreated
1	sinatrasinatra	2021-07-04
2	armstrong111	2021-07-21
3	colkingnat	2021-08-28
4	20bennett	2021-08-29
5	jamesbobby	2021-09-03
6	4_play	2021-09-09
7	GeorgeBenson	2021-09-20
8	aluminumeola	2021-10-01
9	mason0409	2021-10-11
10	jackson251	2021-10-13

ClassID	ClassDesc	Attack	Defence	MagicAtta	MagicDefe	Avoidabilit	HealthPoi	MagicPoin
1	Worrier	10	10	1	1	1	10	5
2	Magician	1	1	10	10	3	3	10
3	Archer	5	4	5	4	10	5	5
4	Blacksmith	4	4	4	4	10	8	4

CharacterID	UserID	UserCharacterID	ClassID	DateCreated
1	1	1	1	2021-07-04
2	1	2	4	2021-07-06
3	1	3	2	2021-09-09
4	1	4	3	2021-10-11
5	1	5	4	2021-12-10
6	1	6	4	2021-12-10
7	2	1	2	2021-07-21
8	2	2	1	2021-11-25
9	3	1	1	2021-08-28
10	4	1	1	2021-08-29
11	5	1	1	2021-09-03
12	5	2	2	2021-10-25
13	6	1	3	2021-09-09
14	7	1	1	2021-09-20
15	8	1	3	2021-10-01
16	9	1	4	2021-10-11
17	9	2	3	2021-10-14

왼쪽 위부터 시계방향으로 USER, CLASS, USER_CHARACTER에 들어갈 데이터들이다.

```python
User = pd.read_excel('InsertData.xlsx', sheet_name='User', header=0)

for i in range(len(User)):
    row = User.iloc[i, :].tolist()
    for idx in range(len(row)):
        if str(type(row[idx])) == "<class 'pandas._libs.tslibs.timestamps.Timestamp'>":
            new_factor = str(row[idx]).split()[0]
            row[idx] = new_factor
    print('INSERT INTO USER VALUES ', tuple(row), ';', sep='')
```

```
INSERT INTO USER VALUES (1, 'sinatrasinatra', '2021-07-04');
INSERT INTO USER VALUES (2, 'armstrong111', '2021-07-21');
INSERT INTO USER VALUES (3, 'colkingnat', '2021-08-28');
INSERT INTO USER VALUES (4, '20bennett', '2021-08-29');
INSERT INTO USER VALUES (5, 'jamesbobby', '2021-09-03');
INSERT INTO USER VALUES (6, '4_play', '2021-09-09');
INSERT INTO USER VALUES (7, 'GeorgeBenson', '2021-09-20');
INSERT INTO USER VALUES (8, 'aluminumeola', '2021-10-01');
INSERT INTO USER VALUES (9, 'mason0409', '2021-10-11');
INSERT INTO USER VALUES (10, 'jackson251', '2021-10-13');
```

Python을 이용해 INSERT문을 print해서 SQL문으로 구현

- InsertData.xlsx에 수작업으로 예시 데이터를 생성했으며, 생성된 예시 데이터는 DataGeneratingProc.ipynb 파일에서 MySQL 문법에 맞게 Insert문을 작성해 print하도록 하였다.
- Print된 INSERT문들은 Insert Data.sql 파일에 붙여넣기하여, 후에 MySQL Workbench에서 실행시킴으로써 Table에 insert하였다.

3. Data

- CharacterConnect.csv, CharacterPayment.csv
  - 수작업으로 생성한 USER, CLASS, USER_CHARACTER Table의 데이터를 바탕으로 랜덤하게 적절한 데이터를 생성하는 프로세스를 DataGeneratingProc.ipynb에 작성하여 csv파일로 저장
  - 각각 316행과 52행의 데이터로 구성되어 있다.

2. 삽입된 User, Class, User_Character 데이터를 바탕으로 조건에 부합하는 적당한 Character_Connect, Character_Payment 데이터 생성

- Character_Connect
  - ConnectID(PK), CharacterID(FK), ConnectTime(접속한 시간), ConnectDate(접속 날짜)
  - 게임 내의 모든 캐릭터에 대해, 그 캐릭터가 생성된 날짜부터 현재까지의 접속 내역을 보여주는 테이블
  - 캐릭터의 마지막 접속 날짜를 임의로 가정한 뒤, 접속 내역을 무작위로 생성하는 절차를 통해 예시 데이터를 만들었다.
  - 캐릭터의 생성 날짜와 마지막 접속 날짜와의 시간 차를 바탕으로, 차이가 클수록 더 많이 접속했을 가능성이 높도록 데이터를 생성했다.

- Character_Payment
  - PaymentID(PK), CharacterID(FK), Payment(결제한 금액), PayDate(결제 날짜)
  - 게임 내의 모든 캐릭터에 대해, 그 캐릭터가 생성된 날짜부터 현재까지의 현금결제 내역을 보여주는 테이블
  - 마찬가지로 캐릭터의 마지막 접속 날짜를 임의로 가정한 뒤, 결제 내역을 무작위로 생성하였다.
  - 캐릭터의 생성 날짜와 마지막 접속 날짜와의 시간 차를 바탕으로, 차이가 클수록 더 많이 게임에 현금결제를 했을 가능성이 높도록 데이터를 생성했다.

  - 자세한 내용은 DataGeneratingProc.ipynb에 작성됨
  - MySQL Workbench에서 Table Data Import Wizard 기능을 이용해 두 개의 csv파일을 table로 정리

4. 응용 서비스 목록 및 구현 프로세스

- Procedure : UserCharacter_Insert
  - USER_CHARACTER 테이블에 새로운 데이터가 들어왔을 때, 해당 User가 몇 개의 게임 캐릭터를 보유하고 있는지를 판단하고 만약 6개를 가지고 있을 경우 생성을 제한하고, 그렇지 않을 경우 생성을 허용하여 USER_CHARACTER에 새로운 캐릭터를 생성하는 PROCEDURE
  - UserID와 생성할 캐릭터의 ClassID를 선택하면, 조건에 맞을 시 나머지 column을 자동으로 생성하여 Insert한다.

```
CREATE PROCEDURE UserCharacter_Insert
    (
        IN varUserID        INT,
        IN varClassID       INT
    )
```

...

```
IF (varRowcount = 6) THEN
    SELECT 'Insertion Rejected: Too many Characters.' AS ErrorMessage;
    ROLLBACK;
    LEAVE block;
```

...

```
INSERT INTO USER_CHARACTER
    (UserID, UserCharacterID, ClassID, DateCreated)
    VALUES (varUserID, varUserCharacterID, varClassID, date(now()));
SELECT 'Insertion Succeeded.' AS SuccessMessage;
```

주요 Process만 정리

```
/* UserID = 1 의 User는 이미 6개의 Character를 가지고 있으므로 Insert가 일어나지 않는다. */
CALL UserCharacter_Insert(1, 2);
SELECT * FROM USER_CHARACTER WHERE UserID = 1;
```

Query Examples.sql의 Line46

- UserID = 1의 User가 ClassID = 2의 캐릭터를 새로 생성하는 쿼리
- 그러나 UserID = 1은 6개의 캐릭터를 이미 가지고 있기 때문에, 에러 코드가 출력되며 캐릭터가 생성되지 않는다.

	ErrorMessage
▶	Insertion Rejected: Too many Characters.

4. 응용 서비스 목록 및 구현 프로세스

- Procedure : DungeonCheck
  - 게임 캐릭터를 성장시킬 수 있는 중요한 요소 중 하나인 'Dungeon' : 제한된 인원이 하나의 팀을 이루어, 적의 소굴에 들어가 모든 적을 소탕하는 등의 미션을 수행하고 보상을 받는 시스템
  - 보통 Dungeon의 난이도에 따라 Dungeon을 어려움 없이 수행할 수 있도록 게임사에서(또는 유저 스스로) Dungeon의 입장 조건을 만들고, 이를 만족하는 팀만 입장할 수 있게끔 한다.
  - 지금 구축하는 DB에서는 팀의 직업군 구성에 따라 Dungeon의 입장을 제한하는 시스템을 만들어 간소화하였다.

```
CREATE PROCEDURE DungeonCheck
    (
        IN varUserNum       INT,    # should be 3 or more.
        IN varCharID1       INT,
        IN varCharID2       INT,
        IN varCharID3       INT,
        IN varCharID4       INT
    )

    DECLARE     TotalATK            INT;
    DECLARE     TotalDEF            INT;
    DECLARE     TotalMAT            INT;
    DECLARE     TotalMDF            INT;
    DECLARE     TotalAVD            INT;
    DECLARE     TotalHP             INT;
    DECLARE     TotalMP             INT;
```

주요 Process만 정리

```
    DECLARE     ATKLimit            INT DEFAULT 10;
    DECLARE     DEFLimit            INT DEFAULT 10;
    DECLARE     MATLimit            INT DEFAULT 10;
    DECLARE     MDFLimit            INT DEFAULT 10;
    DECLARE     AVDLimit            INT DEFAULT 10;
    DECLARE     HPLimit             INT DEFAULT 10;
    DECLARE     MPLimit             INT DEFAULT 10;
IF      ((TotalATK > ATKLimit)
        AND (TotalDEF > DEFLimit)
        AND (TotalMAT > MATLimit)
        AND (TotalMDF > MDFLimit)
        AND (TotalAVD > AVDLimit)
        AND (TotalHP > HPLimit)
        AND (TotalMP > MPLimit)) THEN
        SELECT 'OK' AS Message;
ELSE
        SELECT 'Rejected' AS Message;
END IF;
```

- 팀원들의 능력치 합은 Total~ 의 변수에, 던전의 고유 입장 제한 기준은 ~Limit의 변수에 담기게 된다.
- 팀원의 각 능력치 합이 고유 입장 제한 기준보다 높아야만 던전에 입장할 수 있다.
- ATK, DEF 등의 능력치는 CLASS table에 기록되어 있으며, 각 Class 개체마다 서로 특화된 능력치가 다르기 때문에, 같은 ClassID를 가진 캐릭터들이 팀이 된다면 던전은 입장을 Reject할 것이다.
- Query Examples.sql의 Line 87부터 이 Procedure를 테스트할 수 있다.

4. 응용 서비스 목록 및 구현 프로세스

- Trigger : After_CharacterConnectInsert_UpdateUserCharacter
  - CHARACTER_CONNECT에 새로운 데이터가 Insert되었을 때, 즉 어떤 User의 캐릭터의 접속 내역이 추가되었을 때 USER_CHARACTER 테이블의 LastConnectDate, LastPlayedHour를 그 접속 내역의 데이터로 자동으로 Update하도록 했다.

```
CREATE TRIGGER After_CharacterConnectInsert_UpdateUserCharacter
AFTER INSERT ON CHARACTER_CONNECT
FOR EACH ROW

BEGIN
    DECLARE varCharacterID          INT;
    DECLARE varLastConnectDate      Date;
    DECLARE varLastPlayedTime       INT;

    SET varCharacterID = new.CharacterID;
    SET varLastConnectDate = new.ConnectDate;
    SET varLastPlayedTime = new.ConnectTime;

    UPDATE USER_CHARACTER
        SET LastConnectDate = varLastConnectDate,
            LastPlayedHour = varLastPlayedTime
        WHERE CharacterID = varCharacterID;

END
```

Process 전체

Query Examples.sql의 Line 50

```
/* UserID = 9 의 최근 Connect 상태 */
SELECT CharacterID, UserID, UserCharacterID, LastConnectDate, LastPlayedHour
FROM USER_CHARACTER
WHERE UserID = 9;
```

CharacterID	UserID	UserCharacterID	LastConnectDate	LastPlayedHour
16	9	1	2021-10-31	5
17	9	2	2021-11-21	1
NULL	NULL	NULL	NULL	NULL

```
/* UserID = 9의 캐릭터인 CharacterID = 17의 Character가, 오늘 3시간 동안 접속했다고 가정. */
INSERT INTO CHARACTER_CONNECT
    (CharacterID, ConnectTime, ConnectDate)
    VALUES (17, 3, date(now()));

/* CHARACTER_CONNECT 의 TRIGGER에 의해, USER_CHARACTER의 LastConnectDate와 LastPlayedHour가 변경되었다. */
SELECT CharacterID, UserID, UserCharacterID, LastConnectDate, LastPlayedHour
FROM USER_CHARACTER
WHERE UserID = 9;
```

CharacterID	UserID	UserCharacterID	LastConnectDate	LastPlayedHour
16	9	1	2021-10-31	5
17	9	2	2021-12-19	3
NULL	NULL	NULL	NULL	NULL

4. 응용 서비스 목록 및 구현 프로세스

- Procedure : RFM_Analysis
  - 현재 날짜부터 최근 접속 날짜까지의 Day Gap을 Recentness의 기준으로, UserID 생성 날짜부터 현재까지 접속 시간을 Frequency의 기준으로, 현재까지의 결제 총액을 Money의 기준으로 하여 RFM Analysis를 하는 Procedure 정의₩
  - computeUser_R, computeUser_F, computeUser_M의 프로시져가 각각 R, F, M를 기준으로 유저를 순위별로 그룹화하는 테이블(USER_R, USER_F, USER_M)을 생성한다. 총 5개의 그룹을 생성하며, 상위 20%가 1번 그룹, 그 다음 20%가 2번 그룹, … 과 같은 법칙을 따른다.

  - Procedure : computeUser_R, computeUser_F, computeUser_M
    - computeUser_R, computeUser_F, computeUser_M은 Create Views.sql에 정의된 두 개의 View, UserLastConnectView와 UserTotalPaymentView의 데이터를 이용한다.
    - 각 Procedure는 View를 LastConnectDate, TotalConnectTime, TotalPayment의 내림차순에 따라 row를 돌아다니는 CURSOR를 정의하고, 그에 따라 UserID와 UserID 순위를 테이블 USER_R, USER_F, USER_M에 insert한다.
    - Insert된 테이블의 UserID 순위에 따라 각 User가 5개 그룹 중 어디에 들어가야 하는지를 판별하고 User순위 컬럼을 5개 그룹 순위로 변환한다.

  - computeUser_RFM은 위의 3개 프로시져를 실행한 뒤 UserID가 어느 그룹에 들어가는지의 컬럼만 따로 뽑아 USER_RFM 테이블에 저장한다.
  - USER_R, USER_F, USER_M, USER_RFM 테이블은 Create Tables.sql 파일에서 생성한다.
  - 자세한 내용은 Create Procedure RFM_Analysis에 정의

5. Query Examples

- Line 1 ~ Line 47 : Procedure – UserCharacter_Insert의 테스트 쿼리
- Line 50 ~ Line 64 : Trigger – After_CharacterConnectInsert_UpdateUserCharacter의 테스트 쿼리
- Line 66 ~ Line 84 : CHARACTER_PAYMENT, CHARACTER_CONNECT의 Cascade Delete의 테스트 쿼리
- Line 87 ~ Line 122 : Procedure – DungeonCheck의 테스트 쿼리
- Line 125 ~ : Procedure – computeUser_RFM의 테스트 쿼리