

# 게임 데이터베이스 프로그래밍 작업내역서\_ 이무준

***Hunter in the Gate — Database Programming & Transaction Study Log***

## 1. 작업 개요

본 문서는 헌터 인 더 게이트의 데이터베이스 설계 및 실습 과정을 기록한 문서로, MySQL을 중심으로 스키마 설계, 데이터 정제, CRUD, 트랜잭션, 보안 개념까지 포함한 데이터 중심 게임 프로그래밍 학습 내역서다.

모든 과정은 MySQL Workbench에서 수행되었다.

## 2. 개발 환경

항목	내용
DBMS	MySQL 8.0
관리 도구	MySQL Workbench
실습 데이터	자체 제작 CSV 시트 (헌터 인 더 게이트 카드/플레이어/인벤토리)

## 3. 작업 진행 상세

## [1] 스키마 및 환경 구성

### (초기 설정)

- hunter\_data 스키마를 생성하고, 프로젝트 단위로 독립된 데이터베이스 구조를 구성.
- USE hunter\_data 명령으로 스키마 전환 실습 수행.

### 학습 포인트

- MySQL의 **스키마 = 프로젝트 단위 데이터베이스** 개념 이해.
- Workbench에서 GUI 작업 또한 내부적으로 SQL 명령( CREATE SCHEMA , USE )을 수행한다는 점 확인.

## [2] 테이블 생성 — **PK, AUTO\_INCREMENT, 복합키**

### (기초 테이블 실습)

- new\_table , player 등 예제 테이블을 생성하면서 다음 항목을 실습:
  - PRIMARY KEY 및 AUTO\_INCREMENT 설정.
  - 복합키( player\_id + card\_id ) 구성 방식 실습.
  - 정규화 설계 흐름을 이해.

### 학습 포인트

- 기본키(PK)는 데이터의 유일성 보장 및 식별자 역할을 수행한다는 점 체득.
- 복합키는 정규화된 테이블 관계(1:N)를 표현할 때 사용됨.
- AUTO\_INCREMENT는 인덱스 기반 내부 정수형 키 자동 생성에 적합함.

## [3] 실제 게임 데이터 정제 — 카드 테이블 설계

### (CSV → Table)

- CSV를 UTF-8로 변환 후 임시 테이블로 import.
- 컬럼 타입 및 길이 지정, 텍스트/정수 구분, 제약조건 적용.
- carddata 로 최종 정리.

## 주요 컬럼 구성

컬럼명	설명	타입
id	내부 식별자 (PK)	INT (AUTO_INCREMENT)
card_code	외부 코드 (예: "C001")	VARCHAR(50) UNIQUE
name	카드명	VARCHAR(100)
type	카드 타입	VARCHAR(50)
rarity	희귀도	VARCHAR(20)
cost	코스트	INT
level	카드 레벨	INT
effect_code	효과 코드	VARCHAR(50)
desc	카드 설명	TEXT

## 학습 포인트

- 내부 id와 외부 코드(card\_code)를 분리 설계하는 이유 이해:
  - 내부 id → 빠른 조인/인덱싱, 안정된 참조
  - 외부 code → 클라이언트나 데이터 시트용 식별자
- CSV → MySQL import 시 인코딩(UTF-8) 문제 해결 절차 학습.

## [4] 인벤토리 테이블 설계

### (Inventory)

- inventory 테이블을 설계하여 **플레이어와 카드의 관계(1:N)**를 표현.
- player\_id 와 card\_id 를 복합키로 구성하여 동일 카드 중복을 방지.

## 학습 포인트

- 관계형 데이터 설계의 핵심: **정규화 + 외래키(FK) 참조**.
- “플레이어-카드” 구조는 **교차 테이블 (Join Table)** 개념으로 구현.
- 초기에는 단순 문자열 리스트 형태로 설계했으나,  
실제 운영 구조에서는 정규화를 통한 확장성과 무결성 확보가 중요함을 인식.

## [5] 데이터 조회 및 스키마 전환

### (Data Verification)

- `SELECT * FROM hunter_data.carddata` 를 통해 정제된 데이터를 검증.
- 스키마 전환(`USE`)과 FQN(Fully Qualified Name) 사용법 익힘.

### 학습 포인트

- `SELECT`는 단순 조회뿐 아니라 **스키마 구조/데이터 정합성 검증의 기본 도구**.
- 실제 개발 중에도 데이터 수정 후 `SELECT`로 검증하는 습관 형성.

## [6] 트랜잭션 실습 — 원자성 및 정합성

### (Transaction Test)

트랜잭션을 통해 데이터의 안정성과 원자성을 확인하는 실습 진행.

실습 순서	결과
<code>START TRANSACTION → DELETE → ROLLBACK</code>	데이터 취소 및 원상복귀
<code>START TRANSACTION → DELETE → COMMIT</code>	변경 확정 및 영구 반영
<code>COMMIT</code> 후 <code>ROLLBACK</code>	되돌리기 불가(트랜잭션 종료)

### 학습 포인트

- 트랜잭션은 “**모두 성공하거나, 모두 실패해야 하는 작업 단위**”.
- `COMMIT` 이후에는 `ROLLBACK` 불가능 — 확정 시점 명확히 인식.
- Workbench의 **Apply** 버튼 역시 내부적으로 `COMMIT`을 수행함.
- **Atomicity(원자성)** 과 **Consistency(정합성)** 개념을 실제로 체득.

## [7] ID 설계 및 PK 전략 이해

### ■ (고유키 분리 설계)

- 실제 게임에서 사용되는 `card_code` 등 외부 ID와 DB 내부의 `AUTO_INCREMENT id`를 분리하는 구조 설계.
- 내부 `id`는 빠른 검색과 안정된 관계 참조용, 외부 코드는 변경·식별용으로 사용.

### 학습 포인트

- 문자열 PK는 검색 효율이 떨어지며, 내부적으로 정수형 PK를 두는 것이 일반적임.
- 외부 ID는 사람이 읽기 쉬운 “논리적 키”, 내부 ID는 시스템이 관리하는 “물리적 키”로 구분.

## [8] 보안 및 데이터 무결성 개념 이해

### ■ (GPT를 통한 이론 학습)

- 가챠/결제/보상 등의 핵심 데이터는 반드시 **서버 트랜잭션**으로 관리.
- 요청마다 **HTTPS + HMAC** 서명 검증으로 무결성 보장.
- **Idempotency Key(고유 영수증 키)**를 사용하여 중복 지급 방지.
- DB 계정은 권한 최소화 원칙 적용 (**INSERT/UPDATE 한정 계정 생성**).

### 학습 포인트

- 클라이언트는 ‘의도(intent)’만 전송하고, 실제 연산(차감, 지급, 기록)은 서버가 담당해야 함.
- DB는 **단일 진실의 원천(single source of truth)** 역할을 수행.
- 트랜잭션과 보안 설계가 결합될 때 완전한 데이터 무결성이 달성됨.

## 4. 전체 실습 핵심 개념 요약

분야	학습한 개념
DB 구조	스키마, 테이블, PK, FK, AUTO_INCREMENT

분야	학습한 개념
데이터 정제	CSV → UTF-8 → TABLE Import / 타입 정비
트랜잭션	COMMIT / ROLLBACK / Atomicity / Isolation
ID 전략	내부 id + 외부 code 병행 구조
보안 설계	HMAC, HTTPS, Idempotency, 최소 권한 DB 계정
정합성 유지	트랜잭션 단위 제어 + 서버 검증 로직

## 5. 최종 산출물

- hunter\_data 스키마
- cards , players , inventories 테이블
- UTF-8 인코딩 데이터셋 및 정제 CSV
- 트랜잭션 및 보안 설계 학습 기록

## 6. 학습 성과 요약

이번 실습을 통해 데이터베이스가 단순히 저장소가 아니라,  
**“논리적 일관성과 보안의 중심축”** 으로 작동해야 한다는 점을 명확히 이해했다.

특히 트랜잭션, PK 설계를 직접 구현해보며  
**데이터 흐름 전체를 설계·통제할 수 있는 프로그래머 관점**을 확립했다.