# MLM 시스템 설계 문서

**버전**: 1.0 **작성일**: 2025년 9월 21일 **시스템명**: NanumPay MLM Management System

## 목차

1. [시스템 개요](#X8027de47315d76e77f1153ee70e622ea2bc586b)
2. [데이터베이스 설계](#X8600f41c7e4d698ff8a8bd8c9034de0ef938928)
3. [등급 계산 시스템](#X66fe59dbc4af6e4a68e02f891507a740c0b3f7e)
4. [매출 및 지급 시스템](#Xce818992c7c30a0b3d039c451061393c90eae53)
5. [시스템 아키텍처](#X1e158012178f91247ccb40b360d269f298d4aa3)
6. [운영 지침](#X2b3688ab3842a6e72bc531ab4778521f9f7dd95)

## 1. 시스템 개요

### 1.1 시스템 목적

MLM(Multi-Level Marketing) 비즈니스 모델을 지원하는 용역자 관리 및 수익 분배 시스템

### 1.2 핵심 기능

* **이진트리 구조** 기반 조직 관리
* **8단계 등급 시스템** (F1~F8)
* **자동 매출 계산** 및 **10회 분할 지급**
* **누적식 수익 배분** 방식

### 1.3 시스템 특징

* ✅ 완전 자동화된 등급 계산
* ✅ 투명한 수익 분배 구조
* ✅ 실시간 트리 구조 시각화
* ✅ 배치 처리 최적화

## 2. 데이터베이스 설계

### 2.1 컬렉션 구조

#### 2.1.1 기본 컬렉션 (MongoDB)

##### **admins 컬렉션**

{  
 \_id: ObjectId,  
 username: String,  
 password: String (hashed),  
 role: "admin",  
 createdAt: Date  
}

* 시스템 최상위 관리자
* 이진트리 외부 존재
* 등급 없음, 매출 제외

##### **users 컬렉션**

{  
 \_id: ObjectId,  
 loginId: String, // 한글이름 (중복시 A,B,C)  
 password: String, // 전화번호 뒷4자리  
 name: String,  
 phone: String,  
 bank: String,  
 accountNumber: String,  
 parentId: Mixed, // Admin ObjectId 또는 User loginId  
 leftChildId: String, // 왼쪽 자식 loginId  
 rightChildId: String, // 오른쪽 자식 loginId  
 position: "L" | "R", // 부모 기준 위치  
 grade: String, // F1~F8  
 createdAt: Date  
}

##### **monthlyrevenues 컬렉션**

{  
 \_id: ObjectId,  
 month: "2024-09",  
 totalRevenue: Number, // 총매출액  
 userCount: Number, // 신규 등록 인원  
 gradeDistribution: {  
 F1: Number,  
 F2: Number,  
 // ... F8까지  
 },  
 createdAt: Date  
}

##### **userpaymentplans 컬렉션**

{  
 \_id: ObjectId,  
 userId: String, // 용역자 loginId  
 revenueMonth: "2024-09", // 매출 발생월  
 grade: String, // 매출 시점 등급  
 totalAmount: Number, // 총 지급액  
 installmentAmount: Number, // 회차당 지급액  
 payments: [{  
 installment: Number, // 회차 (1~10)  
 scheduledWeek: "2024-10-W1",  
 amount: Number,  
 status: "pending" | "paid",  
 paidDate: Date,  
 transactionId: String  
 }],  
 createdAt: Date  
}

##### **weeklypayments 컬렉션**

{  
 \_id: ObjectId,  
 week: "2024-10-W1",  
 userId: String,  
 totalAmount: Number,  
 withholdingTax: Number, // 원천징수 3.3%  
 netAmount: Number, // 실지급액  
 details: [{  
 revenueMonth: String,  
 installment: Number,  
 amount: Number  
 }],  
 createdAt: Date  
}

### 2.2 계층 구조

관리자 (Admin)  
 [최상위]  
 |  
 +------+------+  
 | |  
 용역자A(F6) 용역자B(F4) <- parentId = Admin ObjectId  
 | |  
 +--+--+ +--+--+  
 | | | |  
 C(F3) D(F2) E(F2) F(F1) <- parentId = 부모 loginId

### 2.3 등록 프로세스

#### 2.3.1 엑셀 일괄 등록

**헤더 형식:**

순번, 날짜, 성명, 연락처, 주민번호, 은행, 계좌번호,  
판매인, 연락처, 설계사, 연락처, 보험상품명, 보험회사, 지사

**처리 단계:** 1. 판매인=“관리자” → parentId = Admin ObjectId 2. 판매인=기존용역자 → parentId = 해당 loginId 3. 자동 생성: loginId, password, position, grade

#### 2.3.2 배치 처리 최적화

// 효율적 배치 처리  
const allUsers = await User.find().lean();  
const gradeUpdates = calculateAllGrades(allUsers);  
await User.bulkWrite(gradeUpdates);  
await UserPaymentPlan.insertMany(plans);

## 3. 등급 계산 시스템

### 3.1 등급 정의

| 등급 | 조건 | 설명 |
| --- | --- | --- |
| **F1** | 자식 0~1개 | 리프 또는 불완전 노드 |
| **F2** | 좌우 자식 모두 | 완전 이진 노드 |
| **F3** | 좌우 각 F2 1개씩 | F2 균형 보유 |
| **F4** | 좌우 각 F3 1개씩 | F3 균형 보유 |
| **F5** | F4 총 3개 (2:1 또는 1:2) | F4 비대칭 허용 |
| **F6** | F5 총 3개 (2:1 또는 1:2) | F5 비대칭 허용 |
| **F7** | F6 총 3개 (2:1 또는 1:2) | F6 비대칭 허용 |
| **F8** | F7 총 3개 (2:1 또는 1:2) | 최고 등급 |

### 3.2 계산 알고리즘

async function calculateGrade(userId) {  
 // 1. 자식 확인  
 const leftChild = await User.findOne({ parentId: userId, position: 'L' });  
 const rightChild = await User.findOne({ parentId: userId, position: 'R' });  
  
 // 2. F1: 자식 불완전  
 if (!leftChild || !rightChild) return 'F1';  
  
 // 3. 서브트리 등급 수집  
 const leftGrades = await collectSubtreeGrades(leftChild.loginId);  
 const rightGrades = await collectSubtreeGrades(rightChild.loginId);  
  
 // 4. 등급 결정 로직  
 if (leftGrades.F2 >= 1 && rightGrades.F2 >= 1) {  
 if (leftGrades.F3 >= 1 && rightGrades.F3 >= 1) {  
 if (leftGrades.F4 >= 1 && rightGrades.F4 >= 1) {  
 // F5 이상 체크...  
 return 'F5';  
 }  
 return 'F4';  
 }  
 return 'F3';  
 }  
 return 'F2';  
}

### 3.3 실제 트리 예시 (21노드)

1(F4)  
 / \  
 2(F3) 3(F3)  
 / \ / \  
 4(F2) 5(F2) 6(F2) 7(F3)  
 / \ / \ / \ / \  
 8(F1)... [리프 노드들] ...21(F1)

**등급 분포:** - F1: 11명 (리프) - F2: 6명 - F3: 3명 - F4: 1명 (루트)

## 4. 매출 및 지급 시스템

### 4.1 매출 계산

**기본 공식:**

월 총매출 = 신규 등록 인원 × 100만원

**예시:** - 9월 신규 10명 등록 - 9월 총매출 = 10명 × 100만원 = 1,000만원

### 4.2 등급별 지급 비율

| 등급 | 기본 비율 | 누적 계산식 |
| --- | --- | --- |
| F1 | 24% | (총매출×24%) ÷ (F1+F2인원) |
| F2 | 19% | F1지급액 + (총매출×19%) ÷ (F2+F3인원) |
| F3 | 14% | F2지급액 + (총매출×14%) ÷ (F3+F4인원) |
| F4 | 9% | F3지급액 + (총매출×9%) ÷ (F4+F5인원) |
| F5 | 5% | F4지급액 + (총매출×5%) ÷ (F5+F6인원) |
| F6 | 3% | F5지급액 + (총매출×3%) ÷ (F6+F7인원) |
| F7 | 2% | F6지급액 + (총매출×2%) ÷ (F7+F8인원) |
| F8 | 1% | F7지급액 + (총매출×1%) ÷ F8인원 |

### 4.3 10회 분할 지급

#### 4.3.1 지급 스케줄

9월 매출 → 10월부터 10주간 지급  
  
10월: 1~4회차 (4주)  
11월: 5~8회차 (4주)  
12월: 9~10회차 (2주)

#### 4.3.2 실제 계산 예시

**시나리오:** - 총매출: 1,000만원 - F3 등급 김철수 - 등급 분포: F1(50), F2(10), F3(4), F4(2)

**계산:**

F3 기본액 = 1,000만원 × 14% = 140만원  
F3 개인 = 140만원 ÷ (4+2) = 233,333원  
F2 누적 = 175,714원  
F3 총액 = 233,333 + 175,714 = 409,047원  
회차당 = 409,047 ÷ 10 = 40,905원

### 4.4 중첩 지급

**여러 달 매출의 동시 지급:**

10월 1주차 수령액:  
├─ 7월 매출 7회차: 15,000원  
├─ 8월 매출 3회차: 20,000원  
├─ 9월 매출 1회차: 17,571원  
└─ 주간 합계: 52,571원

### 4.5 원천징수

지급액: 40,905원  
원천징수(3.3%): 1,350원  
실지급액: 39,555원

## 5. 시스템 아키텍처

### 5.1 기술 스택

* **Backend**: Node.js, Express
* **Database**: MongoDB
* **Frontend**: SvelteKit
* **인증**: JWT + Session

### 5.2 핵심 서비스

#### 5.2.1 배치 프로세서

// batchProcessor.js  
- 등급 자동 계산  
- 매출 집계  
- 지급 계획 생성  
- 트리 통계 업데이트

#### 5.2.2 트리 추출기

// treeExtractor.js  
- 전체 트리 한 번 로드  
- Map 구조로 메모리 계산  
- 캐싱 (TTL: 1분)

### 5.3 API 엔드포인트

| 엔드포인트 | 메소드 | 설명 |
| --- | --- | --- |
| /api/admin/users/bulk | POST | 엑셀 일괄 등록 |
| /api/admin/revenue/calculate | POST | 매출 계산 |
| /api/admin/payment/weekly | GET | 주간 지급 조회 |
| /api/tree/:userId | GET | 트리 구조 조회 |

### 5.4 성능 최적화

#### 5.4.1 인덱스 설정

// Users  
db.users.createIndex({ loginId: 1 }, { unique: true });  
db.users.createIndex({ parentId: 1 });  
db.users.createIndex({ grade: 1 });  
  
// PaymentPlans  
db.userpaymentplans.createIndex({  
 userId: 1,  
 revenueMonth: 1  
});

#### 5.4.2 배치 처리

* 개별 조회 대신 벌크 연산
* 100명 처리: 200회 → 2회 DB 접근
* 100배 성능 개선

## 6. 운영 지침

### 6.1 월간 운영 프로세스

[매월 말]  
1. 신규 등록 마감  
2. 총매출 집계  
3. 등급별 분포 확인  
4. 지급 계획 생성  
  
[다음달부터]  
5. 매주 자동 지급  
6. 지급 상태 모니터링

### 6.2 데이터 정합성

**검증 포인트:** - [ ] 총 지급액 = 총 매출액 - [ ] 10회 합계 = 개인별 총액 - [ ] 등급별 인원 × 개인액 = 등급 총액 - [ ] 이진트리 구조 무결성

### 6.3 백업 정책

* **일일 백업**: 전체 DB 스냅샷
* **주간 백업**: 지급 내역 아카이브
* **월간 백업**: 매출/등급 이력

### 6.4 보안 고려사항

* 관리자/용역자 권한 분리
* 지급 내역 변경 불가
* 모든 거래 로그 기록
* 개인정보 암호화

## 부록

### A. 용어 정의

| 용어 | 설명 |
| --- | --- |
| **용역자** | MLM 참여자 |
| **판매인** | 신규 용역자의 추천인 |
| **이진트리** | 각 노드가 최대 2개 자식을 갖는 구조 |
| **누적식 배분** | 하위 등급 지급액을 상위가 포함 |
| **회차** | 10회 분할 지급의 각 단위 |

### B. 주요 명령어

# 개발 서버 실행  
pnpm run dev  
  
# 관리자 생성  
pnpm run create:admin  
  
# 데이터 초기화  
node scripts/reset-all-data.js  
  
# 매출 재계산  
node scripts/fix-monthly-revenue.js  
  
# 트리 구조 확인  
node scripts/check-tree-structure.js

### C. 트러블슈팅

| 문제 | 원인 | 해결 |
| --- | --- | --- |
| 매출 0원 | MonthlyRevenue import 누락 | 모델 import 확인 |
| 등급 오류 | 0명 처리 누락 | 분모 0 체크 추가 |
| 트리 깨짐 | 2단계 처리 오류 | 부모 존재 확인 |

**문서 끝**

*이 문서는 NanumPay MLM 시스템의 공식 설계 문서입니다.* *수정 및 배포 시 버전 관리를 준수하시기 바랍니다.*