신일제약 실적관리 시스템 개발 요약 문서 (v3.1)

문서 버전: 3.1 (2025-07-30)

1. 개요 및 설계 철학

1.1. 문서의 목적

본 문서는 신일제약 실적관리 시스템의 **아키텍처, 핵심 기능, 데이터 구조, 그리고 주요 로직을 종합적으로 정리**한 공식 개발 기획 문서입니다. 시스템의 안정적인 유지보수, 신규 개발자의 온보딩, 그리고 향후 기능 확장을 위한 핵심 가이드라인으로 활용하는 것을 최우선 목적으로 합니다.

1.2. 시스템 설계 철학

본 시스템은 다음 세 가지 핵심 철학을 바탕으로 설계되었습니다.

1. **데이터 무결성 (Data Integrity):** 사용자가 입력한 원본 데이터는 절대 변경되지 않습니다. 모든 수정 및 변경 사항은 별도의 테이블(`performance_records_absorption`)에 이력으로 기록되어, 데이터의 정합성을 보장하고 모든 변경 과정을 추적할 수 있습니다.
2. **로직의 중앙화 (Centralized Logic):** 복잡한 데이터 처리, 집계, 계산 로직은 최대한 프론트엔드에서 분리하여 데이터베이스의 **뷰(View)와 함수(RPC)에 중앙화**합니다. 이를 통해 프론트엔드는 UI 표시에만 집중할 수 있어 코드의 복잡성이 감소하고 유지보수성이 향상됩니다.
3. **상태 기반 UI (State-Driven UI):** 모든 데이터와 UI 컴포넌트는 명확한 '상태'(`status` , `action` 등)를 가집니다. 이 상태 값에 따라 UI가 동적으로 변화하여, 사용자에게 현재 진행 상황을 명확하게 인지시키고 오작동을 방지합니다.

1.3. 개발 환경 및 기술 스택

1.3.1. 프론트엔드 (Frontend)

- **프레임워크**: Vue.js 3.5.13 (Composition API)
- **UI 라이브러리**: PrimeVue 4.2.0 (Aura 테마)
- **빌드 도구**: Vite 6.2.4
- **패키지 관리**: npm
- **라우팅**: Vue Router 4.5.0
- **상태 관리**: Vue 3 Composition API (Pinia 미사용)
- **HTTP 클라이언트**: Supabase JavaScript Client 2.49.4
- **추가 라이브러리**:
    - XLSX 0.18.5 (엑셀 처리)
    - File-saver 2.0.5 (파일 다운로드)
    - JSZip 3.10.1 (압축 파일 처리)

- html2canvas 1.4.1 (화면 캡처)
- jsPDF 3.0.1 (PDF 생성)
- CKEditor 5.41.4.2 (리치 텍스트 에디터)
- Quill 2.0.3 (텍스트 에디터)
- VueDraggable 4.1.0 (드래그 앤 드롭)

1.3.2. 백엔드 (Backend)

- **BaaS 플랫폼**: Supabase
- **데이터베이스**: PostgreSQL 15.x
- **인증**: Supabase Auth (Email/Password)
- **파일 저장소**: Supabase Storage
- **실시간 기능**: Supabase Realtime
- **Edge Functions**: Deno (TypeScript)

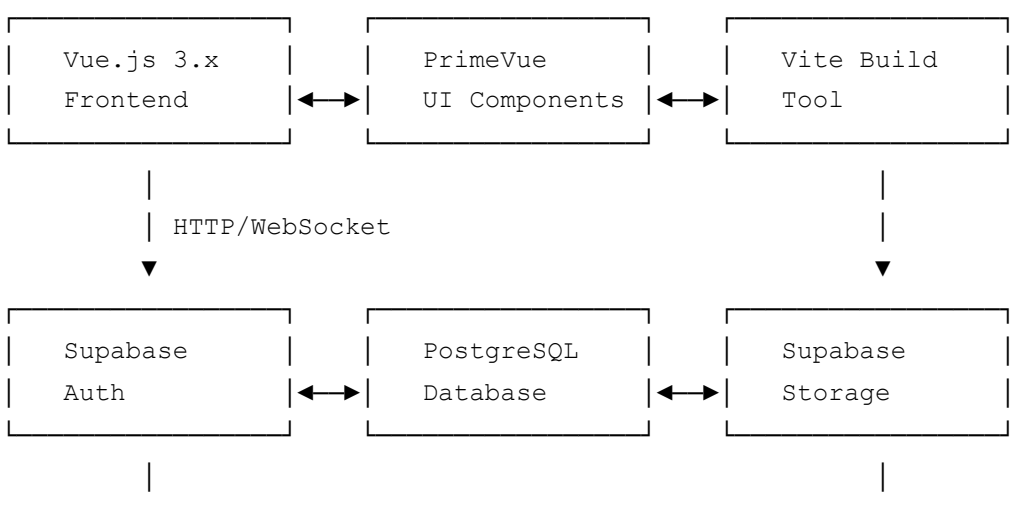
1.3.3. 개발 도구

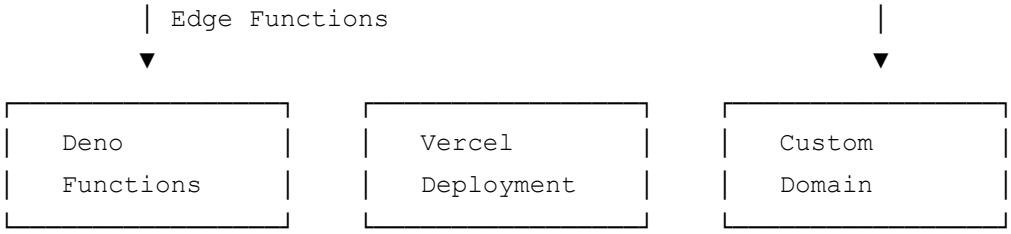
- **IDE**: Visual Studio Code
- **버전 관리**: Git
- **코드 품질**: ESLint 9.22.0
- **코드 포맷팅**: Prettier 3.5.3
- **형식 검사**: TypeScript (선택적)
- **API 테스트**: Supabase Dashboard
- **Vue 개발 도구**: Vite Plugin Vue DevTools 7.7.2

1.3.4. 배포 환경

- **프론트엔드 배포**: Vercel
- **백엔드 서비스**: Supabase Cloud
- **도메인**: 사용자 정의 도메인 (선택적)
- **SSL 인증서**: 자동 (Vercel/Supabase 제공)

1.4. 시스템 아키텍처





1.5. 개발 환경 설정

1.5.1. 필수 요구사항

- **Node.js**: 18.x 이상 (Vite 6.x 요구사항)
- **npm**: 9.x 이상
- **Git**: 2.x 이상
- **브라우저**: Chrome, Firefox, Safari, Edge (최신 버전)
- **메모리**: 최소 4GB RAM (대용량 엑셀 파일 처리 시)

1.5.2. 환경 변수 설정

```
# .env.local
VITE_SUPABASE_URL=your_supabase_project_url
VITE_SUPABASE_ANON_KEY=your_supabase_anon_key
VITE_APP_TITLE=신일제약 실적관리 시스템
VITE_APP_VERSION=3.0.0
```

1.5.3. 개발 서버 실행

```
# 의존성 설치
npm install

# 개발 서버 실행
npm run dev

# 프로덕션 빌드
npm run build

# 빌드 미리보기
npm run preview

# 코드 린팅
npm run lint

# 코드 포맷팅
npm run format
```

1.6. 데이터베이스 설계 원칙

1.6.1. 테이블 명명 규칙

- **일반 테이블**: `snake_case` (예: `performance_records`)
- **매핑 테이블**: `entity_entity_assignments` (예: `client_company_assignments`)
- **뷰**: `entity_details_view` (예: `review_details_view`)
- **함수**: `action_entity` (예: `calculate_absorption_rates`)

1.6.2. 컬럼 명명 규칙

- **기본 컬럼**: `snake_case` (예: `created_at`, `updated_at`)
- **상태 컬럼**: `status` (예: `review_status`, `user_status`)
- **액션 컬럼**: `action` (예: `review_action`)
- **수정 컬럼**: `field_modify` (예: `prescription_qty_modify`)
- **추가 컬럼**: `field_add` (예: `company_name_add`)

1.6.3. 인덱스 전략

- **기본키**: 모든 테이블에 `id` (UUID) 기본키
- **외래키**: 참조 무결성을 위한 인덱스
- **조회 최적화**: `review_status`, `settlement_month` 등 자주 조회되는 컬럼
- **복합 인덱스**: `(settlement_month, company_id, review_status)` 등

---

2. 핵심 데이터 흐름 (Data Flow Lifecycle)

하나의 실적 데이터가 생성되어 최종 정산되기까지의 여정은 다음과 같습니다.

1. `[생성]` **사용자가 실적 입력**:
    - 사용자가 UI를 통해 실적 정보를 입력하고 저장합니다.
    - 데이터는 `performance_records` 테이블에 저장됩니다.
    - 이때, `review_status` 는 **'대기'** 상태가 됩니다.
2. `[동기화]` **관리자가 검수 시작**:
    - 관리자가 '실적 검수' 화면에서 '불러오기'를 실행합니다.
    - `performance_records` 의 '대기' 상태 데이터를 '검수중'으로 변경합니다.
    - 동시에 `performance_records_absorption` 테이블에 복사하여 검수 작업을 진행합니다.
3. `[변경]` **관리자가 검수 진행**:
    - 관리자는 `performance_records_absorption` 의 데이터를 기반으로 수정, 추가, 삭제 작업을 수행합니다.
    - **수정**: `..._modify` 필드에 변경 값이 저장되고, `review_action` 은 '수정'이 됩니다.
    - **추가**: `..._add` 필드에 업체/거래처 정보가 저장되고, `review_action` 은 '추가'가 됩니다.
    - **삭제**: `review_action` 이 '삭제'로 업데이트됩니다. (실제 행 삭제 X)

4. **[확정] 관리자가 검수 완료**:
    - 관리자가 '검수 상태 변경' 버튼을 통해 검수를 마칩니다.
    - `performance_records_absorption`의 `review_status`와 원본 `performance_records`의 `review_status`가 모두 **'완료'**로 변경됩니다.
5. **[활용] 데이터 분석 및 공유**:
    - `review_status`가 '완료'된 데이터만이 '흡수율 분석'과 '정산내역서 공유' 화면에 집계될 자격을 얻습니다.
    - 모든 데이터 조회는 복잡한 테이블들을 미리 결합해놓은 `review_details_view`를 통해 이루어집니다.

---

3. 주요 기능 및 사용자 시나리오

3.1. 실적 검수 (`AdminPerformanceReviewView.vue`)

- **기능 목표**: 관리자가 사용자의 실적을 효율적으로 검토하고, 오류를 수정하며, 최종 데이터를 확정하는 인터페이스를 제공합니다.
- **사용자 시나리오**:
    1. 관리자는 '정산월'을 선택하고 '실적 정보 불러오기' 버튼을 클릭합니다.
    2. 시스템은 '대기' 상태의 모든 실적을 화면에 로드하고 '검수중' 상태로 전환합니다.
    3. 관리자는 각 행을 검토하며, 특정 행의 '수정(✎)' 버튼을 눌러 인라인 편집 모드로 전환합니다. 수량, 수수료율 등을 변경하고 '저장(✓)' 버튼을 누릅니다.
    4. 특정 행 아래에 새로운 실적을 추가하고 싶으면, '+' 버튼을 눌러 새 행을 만들고 정보를 입력 후 저장합니다.
    5. 검수를 마친 행들을 체크박스로 선택한 후, '검수 상태 변경' 버튼을 눌러 '완료' 상태로 확정합니다.

3.2. 흡수율 분석 (`AdminAbsorptionAnalysisView.vue`)

- **기능 목표**: 검수가 완료된 데이터를 기반으로, 제품별 처방액 대비 실제 매출(흡수율)을 분석하여 영업 성과를 다각도로 평가합니다.
- **사용자 시나리오**:
    1. 관리자는 '정산월', '업체', '병의원' 등 원하는 필터 조건을 설정합니다.
    2. '검수 완료 불러오기'를 클릭하여 조건에 맞는 '완료' 상태의 실적 데이터를 조회합니다.
    3. '흡수율 분석' 버튼을 클릭하면, 시스템은 백그라운드(DB 함수)에서 각 실적의 도매/직거래 매출을 합산하고 처방액과 비교하여 흡수율(%)을 계산한 후, 테이블의 '합산액' 및 '흡수율' 열을 업데이트합니다.

3.3. 정산내역서 공유 (`AdminSettlementShareView.vue`)

- **기능 목표**: 최종 확정된 실적을 **업체별로 깔끔하게 합산**하여 보여주고, 각 업체 담당자가 자신의 정산 내역을 조회할 수 있도록 공유 상태를 제어합니다.

- **사용자 시나리오**:
    1. 관리자는 '정산월'을 선택합니다.
    2. 시스템은 해당 월의 '완료'된 모든 실적을 업체별로 자동 합산하여, 업체당 한 줄의 요약 데이터를 보여줍니다. (총 처방액, 총 지급액 등)
    3. 관리자는 각 업체의 '공유' 체크박스를 선택/해제하고 '저장' 버튼을 눌러 공유 상태를 업데이트합니다.
    4. 공유가 활성화된 업체의 담당자는 자신의 계정으로 로그인하여 해당 월의 정산 내역을 조회할 수 있게 됩니다.

---

4. 최신 테이블 구조 개요

4.1. 핵심 테이블

- **`performance_records`** : 사용자 실적 원본 데이터 (review_status: '대기', '검수중', '완료')
- **`performance_records_absorption`** : 관리자 검수 및 분석용 테이블
- **`companies`** : 업체 정보 (user_type: 'admin', 'user')
- **`clients`** : 거래처(병의원) 정보
- **`products`** : 제품 정보 (commission_rate_a, commission_rate_b, commission_rate_c)
- **`pharmacies`** : 약국 정보
- **`wholesale_sales`** : 도매 매출 데이터
- **`direct_sales`** : 직거래 매출 데이터
- **`settlement_months`** : 정산월 관리
- **`settlement_share`** : 정산내역서 공유 설정
- **`notices`** : 공지사항 (is_pinned, view_count)
- **`performance_evidence_files`** : 실적 증빙 파일

4.2. 매핑 테이블

- **`client_company_assignments`** : 거래처-업체 매핑 (commission_grade)
- **`client_pharmacy_assignments`** : 거래처-약국 매핑

---

5. Database Functions 개요

5.1. 핵심 비즈니스 로직 함수들

- **`calculate_absorption_rates`** : 흡수율 분석 실행 (도매/직거래 매출액 계산)
- **`get_settlement_summary_by_company`** : 정산내역서 공유용 업체별 합산 데이터 조회
- **`create_settlement_summary`** : 정산월별 요약 데이터 생성

5.2. 변경사항 감지 함수들

- `check_for_updates` : 정산월별 변경사항 확인 (검수, 매핑, 매출 데이터 변경)
- `check_performance_changes` : 실적 데이터 변경사항 확인 (추가, 수정, 삭제 건수)

5.3. 데이터 조회 함수들

- `get_absorption_analysis_details` : 흡수율 분석 화면의 상세 데이터 조회
- `get_distinct_companies_from_analysis` : 업체 필터 목록 조회
- `get_distinct_clients_from_analysis` : 거래처 필터 목록 조회
- `get_distinct_settlement_months_from_analysis` : 정산월 필터 목록 조회

5.4. 디버깅 함수들

- `debug_absorption_calculation` : 흡수율 계산 디버깅용
- `debug_absorption_rates` : 흡수율 분석 디버깅용 (상세 매칭 정보)
- `debug_distribution_ratios` : 분배 비율 디버깅용

5.5. Edge Functions

- `register-user` : 사용자 회원가입 처리
- `reset-password` : 비밀번호 초기화

---

6. 향후 개선 과제 및 고려사항

- **권한 관리 고도화**: 현재는 'admin'과 'user' 역할만 존재하지만, 향후 특정 업체 담당자가 여러 업체를 관리하거나, 특정 메뉴에만 접근하는 등의 세분화된 권한 관리가 필요할 수 있습니다.
- **대시보드 기능**: 현재는 테이블 형태의 데이터 조회가 중심이지만, 주요 지표(월별 총 지급액, 상위 5개 업체 등)를 시각적인 차트와 그래프로 보여주는 대시보드 기능을 추가하여 사용성을 개선할 수 있습니다.
- **성능 최적화**: 데이터가 수십만 건 이상으로 증가할 경우, 현재의 View와 함수(RPC)의 성능을 모니터링하고 인덱스(Index) 추가, 쿼리 최적화 등의 작업이 필요할 수 있습니다.
- **사용자 경험(UX) 개선**: 엑셀 업로드/다운로드 기능의 편의성을 개선하고, 각 입력 필드에 대한 유효성 검사(Validation)를 강화하여 사용자 실수를 줄이는 방안을 고려해야 합니다.

7. 향후 개선 과제 및 고려사항

7.1. 기능 확장 및 개선

- **대시보드 기능**: 현재는 테이블 형태의 데이터 조회가 중심이지만, 주요 지표(월별 총 지급액, 상위 5개 업체 등)를 시각적인 차트와 그래프로 보여주는 대시보드 기능을 추가하여 사용성을 개선할 수 있습니다.
- **실시간 알림 시스템**: 실적 검수 완료, 새로운 공지사항 등 중요 이벤트에 대한 실시간 알림 기능을 추가하여 사용자 경험을 향상시킬 수 있습니다.

- **모바일 앱 개발**: 현재 웹 기반 시스템을 모바일 앱으로 확장하여 언제 어디서나 접근 가능한 시스템을 구축할 수 있습니다.

7.2. 성능 최적화

- **성능 최적화**: 데이터가 수십만 건 이상으로 증가할 경우, 현재의 View와 함수(RPC)의 성능을 모니터링하고 인덱스(Index) 추가, 쿼리 최적화 등의 작업이 필요할 수 있습니다.
- **캐싱 전략**: Redis 등을 활용한 데이터 캐싱 시스템을 도입하여 반복적인 쿼리 성능을 개선할 수 있습니다.
- **CDN 도입**: 정적 파일(이미지, CSS, JS)에 대한 CDN 도입으로 로딩 속도를 개선할 수 있습니다.

7.3. 사용자 경험 개선

- **사용자 경험(UX) 개선**: 엑셀 업로드/다운로드 기능의 편의성을 개선하고, 각 입력 필드에 대한 유효성 검사(Validation)를 강화하여 사용자 실수를 줄이는 방안을 고려해야 합니다.
- **접근성 개선**: 스크린 리더 지원, 키보드 네비게이션 강화 등 접근성을 개선하여 모든 사용자가 편리하게 사용할 수 있도록 해야 합니다.
- **다국어 지원**: 향후 해외 진출을 고려하여 다국어 지원 시스템을 구축할 수 있습니다.

7.4. 보안 강화

- **2단계 인증**: SMS, 이메일, OTP 앱 등을 활용한 2단계 인증 시스템을 도입할 수 있습니다.
- **감사 로그**: 모든 데이터 변경 사항에 대한 상세한 감사 로그를 기록하여 보안을 강화할 수 있습니다.
- **데이터 암호화**: 민감한 데이터에 대한 암호화를 강화하여 데이터 보안을 향상시킬 수 있습니다.

7.5. 시스템 확장성

- **마이크로서비스 아키텍처**: 현재 모놀리식 구조를 마이크로서비스로 분리하여 시스템 확장성을 개선할 수 있습니다.
- **컨테이너화**: Docker를 활용한 컨테이너화로 배포 및 운영 효율성을 향상시킬 수 있습니다.
- **자동화**: CI/CD 파이프라인 구축으로 개발 및 배포 프로세스를 자동화할 수 있습니다.

7.6. 데이터 분석 및 인사이트

- **BI 도구 연동**: Tableau, Power BI 등과 연동하여 고급 데이터 분석 기능을 제공할 수 있습니다.
- **머신러닝**: 실적 예측, 이상 패턴 감지 등 머신러닝 기능을 도입하여 비즈니스 인사이트를 제공할 수 있습니다.
- **API 확장**: 외부 시스템과의 연동을 위한 RESTful API를 확장하여 시스템 통합성을 향상시킬 수 있습니다.

신일제약 실적관리 시스템 데이터 구조 상세 (v3.1)

문서 버전: 3.1 (2025-07-30)

1. 개요

본 문서는 신일제약 실적관리 시스템의 핵심 데이터 구조, 테이블 간의 관계, 그리고 데이터 흐름의 중심이 되는 뷰(`View`)에 대해 상세히 기술합니다. 시스템의 모든 데이터 처리는 이 구조를 기반으로 이루어지므로, 유지보수 및 기능 확장 시 반드시 본 문서를 참고해야 합니다. 데이터 모델은 **원본 데이터의 불변성**을 유지하고, **관리자의 모든 변경 행위를 추적** 가능하게 하는 것에 중점을 두고 설계되었습니다.

---

2. 데이터 흐름도 (Lifecycle of Data)

하나의 실적 데이터가 생성되어 최종 정산되기까지의 여정은 다음과 같습니다.

```
(사용자)                          (관리자)
   |                                 |
[1. 실적 입력] ------------> [2. 데이터 로드 & 동기화] -------> [3. 검수 진행 (수정/추가/삭제)] ----
   | (performance_records)       | (performance_records_absorption) | (performance_records_abs
   | - review_status: '대기'     | - pr.status -> '검수중'          | - review_action 기록
   |                             | - pra.status -> '검수중'         | - _modify, _add 필드에 값 저
   |                             | - _add 필드 -> NULL로 초기화     |
```

---

3. 테이블 관계도 (Text-based ERD)

```
[auth.users] 1--* [performance_records] 1--1 [performance_records_absorption] *--1 [companies
   |                                             |                          |
   *--1 [companies] (user_id)                    *--1 [clients]             *--1 [clients]
                                                 *--1 [products]
```

- `1--*` : 일대다 (One-to-Many)
- `1--1` : 일대일 (One-to-One)

---

4. 주요 테이블 상세 정의

4.1. `performance_records` (사용자 실적 원본 테이블)

- **역할**: 사용자가 입력하는 모든 실적 데이터의 **원본(Raw Data)** 을 저장하는 가장 기초적인 테이블입니다.

- **특징**: 이 테이블의 데이터는 **절대 직접 수정되거나 삭제되지 않습니다.**
- **컬럼 정의**:
    - `id` (bigint, PK): 레코드 고유 ID.
    - `company_id` (uuid, FK -> `companies.id`): 실적의 주체인 업체 정보.
    - `settlement_month` (varchar): 정산월 정보.
    - `prescription_month` (varchar): 처방월 정보.
    - `client_id` (bigint, FK -> `clients.id`): 관련 거래처 정보.
    - `product_id` (uuid, FK -> `products.id`): 관련 제품 정보.
    - `prescription_qty` (numeric): 처방 수량.
    - `prescription_type` (varchar): 처방 유형 ('EDI' 기본값).
    - `remarks` (text): 비고.
    - `registered_by` (uuid, FK -> `auth.users.id`): 실적을 입력한 사용자 정보.
    - `review_status` (text): 레코드의 검수 상태 ('대기', '검수중', '완료'). 데이터 흐름을 제어하는 핵심 필드.
    - `review_action` (text): 검수 작업 유형.
    - `commission_rate` (numeric): 수수료율.
    - `updated_by` (uuid, FK -> `auth.users.id`): 수정자 정보.
    - `created_at`, `updated_at` (timestamptz): 생성/수정 시각.

4.2. `performance_records_absorption` (관리자 검수 및 분석용 테이블)

- **역할**: 관리자의 모든 검수 작업(수정, 추가, 삭제)의 **결과와 이력을 기록**하는 시스템의 핵심 운영 테이블입니다.
- **컬럼 정의**:
    - `id` (bigint, PK): 검수 레코드 고유 ID.
    - `settlement_month` (varchar): 정산월.
    - `company_id` (uuid, FK -> `companies.id`): 업체 ID.
    - `client_id` (bigint, FK -> `clients.id`): 거래처 ID.
    - `product_id` (uuid, FK -> `products.id`): 제품 ID.
    - `prescription_month` (varchar): 처방월.
    - `prescription_qty` (numeric): 처방 수량.
    - `prescription_type` (varchar): 처방 유형.
    - `commission_rate` (numeric): 수수료율.
    - `remarks` (text): 비고.
    - `registered_by` (uuid, FK -> `auth.users.id`): 등록자.
    - `review_status` (text): 관리자의 검수 상태 ('검수중', '완료').
    - `review_action` (text): 관리자의 작업 유형 (`'수정'`, `'추가'`, `'삭제'`).
    - `wholesale_revenue` (numeric): 도매 매출액.
    - `direct_revenue` (numeric): 직거래 매출액.
    - `total_revenue` (numeric): 총 매출액.
    - `absorption_rate` (numeric): 흡수율.
    - `updated_by` (uuid, FK -> `auth.users.id`): 수정자.
    - `created_at`, `updated_at` (timestamptz): 생성/수정 시각.

4.3. 마스터 테이블

- **companies**:
  - id (uuid, PK), user_id (uuid, FK -> auth.users.id)
  - company_name, business_registration_number, representative_name
  - business_address, landline_phone, contact_person_name, mobile_phone, mobile_phone_2
  - email, default_commission_grade (text, 'A' 기본값), remarks
  - approval_status (text, 'pending' 기본값), status (text, 'active' 기본값)
  - user_type (text, 'user' 기본값), company_group (text)
  - assigned_pharmacist_contact, receive_email (text)
  - created_by, approved_at, updated_by (uuid, FK -> auth.users.id)
  - created_at, updated_at (timestamptz)
- **clients**:
  - id (bigint, PK), client_code (text)
  - name, business_registration_number, owner_name, address, remarks
  - status (text, 'active' 기본값)
  - created_at, updated_at (timestamptz)
- **products**:
  - id (uuid, PK), product_name, insurance_code, price (integer)
  - commission_rate_a, commission_rate_b, commission_rate_c (numeric)
  - standard_code, unit_packaging_desc, unit_quantity (integer)
  - base_month (text), status (text, 'active' 기본값)
  - created_at, updated_at (timestamptz)
- **pharmacies**:
  - id (bigint, PK), pharmacy_code (text)
  - name, business_registration_number, address, remarks
  - status (text, 'active' 기본값)
  - created_at, updated_at (timestamptz)
- **settlement_months**:
  - id (bigint, PK), settlement_month (varchar)
  - start_date, end_date (date), notice (text)
  - status (varchar, 'active' 기본값), remarks (text)
  - created_at (timestamptz)
- **settlement_share**:
  - id (bigint, PK), settlement_month (text)
  - company_id (uuid, FK -> companies.id)

- `share_enabled` (boolean), `created_at` (timestamptz)

4.4. 매핑 테이블

- **`client_company_assignments`**:
  - `id` (bigint, PK), `client_id` (bigint, FK -> `clients.id`)
  - `company_id` (uuid, FK -> `companies.id`)
  - `company_default_commission_grade` (text, 'A' 기본값)
  - `modified_commission_grade` (text)
  - `created_at` (timestamptz)
- **`client_pharmacy_assignments`**:
  - `id` (bigint, PK), `client_id` (bigint, FK -> `clients.id`)
  - `pharmacy_id` (bigint, FK -> `pharmacies.id`)
  - `created_at` (timestamptz)

4.5. 매출 데이터 테이블

- **`wholesale_sales`**:
  - `id` (bigint, PK), `pharmacy_code`, `pharmacy_name`
  - `business_registration_number`, `address`, `standard_code`
  - `product_name`, `sales_amount` (numeric), `sales_date` (date)
  - `created_at`, `updated_at` (timestamptz)
  - `created_by`, `updated_by` (text)
- **`direct_sales`**:
  - `id` (bigint, PK), `pharmacy_code`, `pharmacy_name`
  - `business_registration_number`, `address`, `standard_code`
  - `product_name`, `sales_amount` (numeric), `sales_date` (date)
  - `created_at`, `updated_at` (timestamptz)
  - `created_by`, `updated_by` (text)

4.6. 기타 테이블

- **`notices`**:
  - `id` (uuid, PK), `title`, `content`
  - `is_pinned` (boolean, false 기본값), `view_count` (integer, 0 기본값)
  - `author_id` (uuid, FK -> `auth.users.id`)
  - `file_url`, `links` (text)
  - `created_at`, `updated_at` (timestamptz)

- **`performance_evidence_files`**:
  - `id` (uuid, PK), `company_id` (uuid, FK -> `companies.id`)
  - `client_id` (bigint, FK -> `clients.id`)
  - `settlement_month` (text), `file_name`, `file_path`
  - `file_size` (bigint), `uploaded_by` (uuid, FK -> `auth.users.id`)
  - `uploaded_at`, `created_at` (timestamp)

- **`absorption_analysis`**:
  - `id` (bigint, PK), `performance_record_id` (bigint, FK -> `performance_records.id`)
  - `review_status` (text), `review_action` (text)
  - `user_edit_status` (text), `update_by` (uuid, FK -> `auth.users.id`)
  - `company_id_add`, `client_id_add`, `product_id_modify`, `prescription_month_modify`
  - `prescription_qty_modify`, `prescription_type_modify`, `remarks_modify`
  - `commission_rate_modify`, `created_add_at`, `settlement_month_add`
  - `registered_add_by` (uuid, FK -> `auth.users.id`)
  - `wholesale_revenue`, `direct_revenue`, `total_revenue`, `absorption_rate`
  - `created_at`, `updated_at` (timestamptz)

---

5. 핵심 데이터 뷰 (View): `review_details_view`

`review_details_view`는 시스템의 **데이터 조회 로직을 극적으로 단순화**시키는 가장 중요한 가상 테이블입니다.

5.1. View의 SQL 구조 (실제 코드)

```
CREATE OR REPLACE VIEW public.review_details_view AS
WITH analysis_base AS (
    SELECT
        aa.id AS absorption_analysis_id,
        aa.performance_record_id,
        aa.review_status,
        aa.review_action,
        pr.user_edit_status,
        -- 등록자 ID 결정 (관리자가 추가했으면 관리자 ID, 아니면 원본 등록자 ID)
        COALESCE(aa.registered_add_by, pr.registered_by) AS final_registered_by_id,
        aa.update_by, -- 수정자 ID
        COALESCE(aa.company_id_add, pr.company_id) AS final_company_id,
        COALESCE(aa.client_id_add, pr.client_id) AS final_client_id,
        COALESCE(aa.product_id_modify, pr.product_id) AS final_product_id,
        COALESCE(aa.prescription_month_modify, pr.prescription_month) AS prescription_month,
        COALESCE(aa.prescription_qty_modify, pr.prescription_qty) AS prescription_qty,
        COALESCE(aa.prescription_type_modify, (pr.prescription_type)::text) AS prescription_t
        COALESCE(aa.remarks_modify, pr.remarks) AS remarks,
        aa.commission_rate_modify,
        -- 날짜 결정 (관리자 추가일 우선)
```

```
        COALESCE(aa.created_add_at, pr.created_at) AS final_created_at,
        COALESCE(aa.settlement_month_add, pr.settlement_month) AS final_settlement_month,
        aa.updated_at
    FROM absorption_analysis aa
    LEFT JOIN performance_records pr ON aa.performance_record_id = pr.id
)
SELECT
    ab.absorption_analysis_id,
    ab.performance_record_id,
    ab.review_status,
    ab.review_action,
    ab.user_edit_status,
    ab.final_company_id AS company_id,
    ab.final_client_id AS client_id,
    ab.final_product_id AS product_id,
    ab.prescription_month,
    ab.prescription_qty,
    ab.prescription_type,
    ab.remarks,
    ab.final_created_at AS created_at,
    ab.final_settlement_month AS settlement_month,
    ab.updated_at,
    -- 최종 등록자 이름 결정 로직 수정
    CASE
        WHEN registrant.role = 'admin' THEN registrant.contact_person_name
        ELSE registrant.company_name
    END AS registered_by_name,
    updater.contact_person_name AS updated_by_name, -- 수정자 이름
    c.company_name,
    c.company_group,
    c.business_registration_number,
    c.representative_name,
    c.assigned_pharmacist_contact AS manager_name,
    cl.name AS client_name,
    p.product_name AS product_name_display,
    p.insurance_code,
    p.price,
    (ab.prescription_qty * p.price) AS prescription_amount,
    COALESCE(
        ab.commission_rate_modify,
        CASE
            WHEN c.default_commission_grade = 'B' THEN p.commission_rate_b
            ELSE p.commission_rate_a
        END
    ) AS commission_rate
FROM analysis_base ab
-- 이름 조인을 위한 조인
LEFT JOIN companies registrant ON ab.final_registered_by_id = registrant.user_id
LEFT JOIN companies updater ON ab.update_by = updater.user_id
-- 나머지 데이터 조인
LEFT JOIN companies c ON ab.final_company_id = c.id
LEFT JOIN clients cl ON ab.final_client_id = cl.id
LEFT JOIN products p ON ab.final_product_id = p.id;
```

5.2. `COALESCE` 로직의 중요성

`COALESCE` 는 "NULL이 아닌 첫 번째 값을 반환하라"는 의미의 SQL 함수입니다. 이 View는 `COALESCE` 를 적극적으로 사용하여, 관리자의 수정/추가 이력을 종합하고 **최종적으로 사용해야 할 데이터를 동적으로 결정**합니다.

- **예시 시나리오**:
    1. **원본 데이터**: `performance_records` 에 `prescription_qty = 100` 인 데이터가 있음.
    2. **수정 발생**: 관리자가 이 데이터를 120으로 수정하면, `performance_records_absorption` 에 `prescription_qty_modify = 120` 이 저장됨.
    3. **View 조회**: `COALESCE(aa.prescription_qty_modify, pr.prescription_qty)` 는 `aa.prescription_qty_modify` 가 `NULL` 이 아니므로 **120**을 반환함.
    4. **수정 미발생**: 만약 수정되지 않았다면, `aa.prescription_qty_modify` 는 `NULL` 이므로, `COALESCE` 는 두 번째 값인 `pr.prescription_qty` 즉, **100**을 반환함.
- **기대 효과**: 이 패턴 덕분에 프론트엔드는 데이터의 수정 여부나 추가 여부를 전혀 신경 쓸 필요 없이, `review_details_view` 의 최종 필드(`prescription_qty` , `company_id` 등)만 조회하면 **항상 올바른 최신 데이터**를 얻을 수 있습니다. 이는 코드의 가독성과 유지보수성을 극대화합니다.

---

7. 최신 추가 테이블 및 기능

7.1. 실적 증빙 파일 관리 (`performance_evidence_files`)

목적: 실적 데이터의 증빙 자료를 안전하게 저장하고 관리

컬럼 정의:

- `id` (uuid, PK): 파일 고유 ID
- `company_id` (uuid, FK -> `companies.id`): 업체 ID
- `client_id` (bigint, FK -> `clients.id`): 거래처 ID
- `settlement_month` (text): 정산월
- `file_name` (text): 원본 파일명
- `file_path` (text): Supabase Storage 내 파일 경로
- `file_size` (bigint): 파일 크기 (bytes)
- `uploaded_by` (uuid, FK -> `auth.users.id`): 업로드한 사용자
- `uploaded_at` (timestamp): 업로드 시각
- `created_at` (timestamp): 생성 시각

특징:

- Supabase Storage를 활용한 안전한 파일 저장
- 업체별 접근 권한 제어 (RLS 정책)
- 파일 메타데이터 추적으로 관리 효율성 향상

6.2. 실적입력기간 관리 (`settlement_months` 확장)

기존 컬럼에 추가된 기능:

- `start_date` (date): 실적입력 시작일
- `end_date` (date): 실적입력 종료일
- `notice` (text): 전달 사항
- `status` (varchar, 'active' 기본값): 활성/비활성 상태

핵심 기능:

- **일반 사용자**: 실적입력기간 내에서만 실적 등록/수정 가능
- **관리자**: 실적입력기간과 무관하게 언제든지 실적 등록/수정 가능
- **UI 제어**: 기간 외에는 모든 입력 버튼 비활성화

6.3. 사용자 편집 상태 추적 (`performance_records` 확장)

추가된 컬럼:

- `user_edit_status` (text): 사용자 편집 상태
- `updated_by` (uuid, FK -> `auth.users.id`): 수정자 정보
- `updated_at` (timestamptz): 수정 시각

상태 값:

- `'editable'` : 편집 가능
- `'locked'` : 편집 불가 (검수중/완료)
- `'pending'` : 대기 중

6.4. 제품 정보 월별 관리 (`products` 확장)

핵심 기능:

- **월별 제품 정보**: `base_month` 필드로 월별 제품 정보 관리
- **보험코드 유니크**: 같은 보험코드의 제품은 월별로 하나만 유지
- **자동 업데이트**: 처방월 변경 시 제품 정보 자동 업데이트

데이터 무결성:

- 보험코드 기준 유니크 처리로 중복 방지
- 월별 제품 정보 캐싱으로 성능 최적화
- 제품 정보 변경 시 실적 데이터 자동 반영

6.5. 키보드 네비게이션 상태 관리

프론트엔드 상태 변수:

- `currentCell` (ref): 현재 포커스된 셀 정보 `{ row: number, col: string }`
- `fieldRefs` (ref): 각 입력 필드의 DOM 참조
- `productInputRefs` (ref): 제품명 입력 필드의 DOM 참조

키보드 이벤트 처리:

- **전역 이벤트**: document 레벨에서 키보드 이벤트 처리
- **조건부 처리**: 편집 가능한 상태에서만 키보드 기능 활성화
- **충돌 방지**: 제품 검색 중 특정 키 차단

6.6. 제품 검색 시스템 상태 관리

상태 변수:

- `productSearchForRow` (ref): 제품 검색 상태
  - `query` : 검색 쿼리
  - `results` : 검색 결과 배열
  - `selectedIndex` : 선택된 항목 인덱스
  - `show` : 드롭다운 표시 여부
  - `activeRowIndex` : 활성 행 인덱스

캐싱 시스템:

- `productsByMonth` (ref): 월별 제품 목록 캐시
- `productDropdownStyle` (ref): 드롭다운 위치 스타일

6.7. 실시간 검증 시스템

입력 검증 규칙:

- **제품명**: 제품 선택 필수 (자동완성 지원)
- **처방수량**: 숫자만 입력 가능, 0보다 큰 값
- **처방액**: 자동 계산, 수정 불가
- **필수 필드**: 제품명 + 처방수량 조합 필수

실시간 피드백:

- **처방액 계산**: 수량 입력 시 즉시 계산
- **제품 정보 업데이트**: 처방월 변경 시 자동 업데이트
- **상태 표시**: 검수상태별 색상 구분

6.8. 정렬 규칙 시스템

사용자별 정렬 규칙:

- **일반 사용자**: 처방건수 0건 → 1건 이상 (입력된 것이 아래로)
- **관리자**: 처방건수 0건 → 1건 이상 (입력된 것이 위로)

- **실적 검수**: 검수상태별 정렬 (신규 → 검수중 → 완료)

정렬 로직:

- **가나다순**: `localeCompare()` 메서드 사용
- **상태별 그룹화**: 조건에 따라 그룹별로 분리 후 정렬
- **실시간 정렬**: computed 속성으로 실시간 정렬

6.9. 상태별 편집 권한 시스템

권한 규칙:

- **일반 사용자**: '대기' 상태만 편집 가능
- **관리자**: 모든 상태 편집 가능
- **상태별 UI**: 편집 불가 상태의 시각적 표시

구현 방식:

- `isRowEditable()` 함수로 행별 편집 가능 여부 판단
- `isInputEnabled` ref로 전체 편집 상태 제어
- 키보드 이벤트 조건부 처리

6.10. 성능 최적화 시스템

제품 데이터 캐싱:

- **월별 캐시**: `productsByMonth` 객체로 월별 제품 목록 캐시
- **중복 제거**: 보험코드 기준 유니크 처리
- **지연 로딩**: 필요한 월의 제품만 로드

키보드 이벤트 최적화:

- **전역 이벤트**: document 레벨에서 키보드 이벤트 처리
- **조건부 처리**: 편집 가능한 상태에서만 키보드 기능 활성화
- **충돌 방지**: 제품 검색 중 특정 키 차단

---

7. 데이터 흐름 최적화

7.1. 제품 정보 자동 업데이트 흐름

```
1. 사용자가 처방월 변경
   ↓
2. 기존 제품의 보험코드로 새 처방월에서 검색
   ↓
3. 검색 결과에 따라 분기:
   - 결과 있음: 새 제품 정보로 자동 업데이트
```

```
   - 결과 없음: 제품 선택 해제
   ↓
4. 처방수량이 있으면 처방액 재계산
```

7.2. 키보드 네비게이션 흐름

```
1. 사용자가 키보드 입력
   ↓
2. 전역 키보드 이벤트 핸들러에서 처리
   ↓
3. 편집 가능 여부 확인
   ↓
4. 제품 검색 상태 확인
   ↓
5. 적절한 동작 실행:
   - 화살표 키: 셀 간 이동
   - Enter 키: 필드별 특별 동작
   - 특수 키: 행 관리 동작
```

7.3. 실시간 검증 흐름

```
1. 사용자가 데이터 입력
   ↓
2. 입력값 실시간 검증
   ↓
3. 검증 결과에 따라 피드백 제공:
   - 성공: 정상 처리
   - 실패: 오류 메시지 표시
   ↓
4. 관련 필드 자동 업데이트 (처방액 계산 등)
```

7.4. 상태별 편집 권한 흐름

```
1. 화면 로드 시 사용자 권한 확인
   ↓
2. 실적 데이터의 검수 상태 확인
   ↓
3. 권한과 상태에 따라 편집 가능 여부 결정
   ↓
4. UI 요소 활성화/비활성화
   ↓
5. 키보드 이벤트 조건부 처리
```

이러한 최신 기능들을 통해 신일 PMS는 더욱 효율적이고 사용자 친화적인 실적 관리 시스템으로 발전했습니다. 모든 기능은 데이터 무결성을 보장하면서 사용자 경험을 최적화하는 방향으로 설계되었습니다.

신일제약 실적관리 시스템 주요 기능 및 핵심 로직 (v3.1)

문서 버전: 3.1 (2025-07-30)

1. 개요

본 문서는 신일제약 실적관리 시스템의 핵심 기능인 '실적 검수', '흡수율 분석', '정산내역서 공유'의 상세한 동작 방식과 내부 로직을 설명합니다. 각 기능은 데이터베이스의 뷰(View)와 함수(RPC)를 적극적으로 활용하여 프론트엔드의 복잡성을 낮추고, 시스템 전체의 안정성과 일관성을 확보하는 방향으로 설계되었습니다.

---

2. 실적 검수 (`AdminPerformanceReviewView.vue`)

관리자가 사용자의 실적을 검수하고 확정하는 가장 중요하고 복잡한 기능입니다. 모든 데이터 변경 이력은 이 화면에서 발생하며, 프론트엔드는 사용자의 인터랙션을 `performance_records_absorption` 테이블의 변경 이력으로 변환하는 역할을 수행합니다.

2.1. 주요 상태 변수 (State Variables)

- `loading` (ref): 데이터 로딩 상태를 제어하는 boolean. API 호출 시 `true` 로 설정되어 로딩 인디케이터를 표시.
- `displayRows` (ref): 화면의 데이터 테이블에 실제로 표시되는 데이터 배열.
- `selectedRows` (ref): 사용자가 체크박스를 통해 선택한 행들의 데이터 배열.
- `isAnyEditing` (computed): 현재 편집 중인 행이 하나라도 있는지 여부를 계산하는 boolean. 편집 중일 때는 다른 액션 버튼들을 비활성화하는 데 사용됨.
- `fieldRefs`, `productInputRefs`: 인라인 편집 시 각 입력 필드(input, select)에 대한 참조(reference)를 저장하여, 키보드 네비게이션 등으로 직접 포커스를 제어하는 데 사용.

2.2. 데이터 조회 및 필터링 (`loadPerformanceData`)

- **역할**: 필터 조건에 맞는 데이터를 조회하여 `displayRows` 에 채워 넣는 메인 함수.
- **동작 순서**:
  1. 필터링의 기준이 되는 `performance_records` 테이블을 직접 조회합니다.
  2. `selectedSettlementMonth`, `selectedCompanyId`, `selectedHospitalId` 등 UI 필터 값을 바탕으로 `WHERE` 조건을 동적으로 추가합니다.
  3. `selectedReviewStatus` 값에 따라 분기 처리됩니다.
     - `'신규'`: '대기' 상태의 실적을 '검수중'으로 전환시킨 후, '검수중' 상태의 데이터를 조회합니다.

        - `'전체'` : 마찬가지로 '대기' 상태 데이터를 '검수중'으로 전환한 후, 상태 조건 없이 모든 데이터를 조회합니다.
    4. 조회된 데이터를 `map()` 을 통해 순회하며, 화면 표시에 필요한 형식(예: 숫자 포맷팅)으로 가공합니다.
    5. 가공된 데이터를 정해진 규칙(상태 > 작업 > 업체명 순)에 따라 `sort()` 하고 최종적으로 `displayRows.value` 에 할당합니다.

2.3. 데이터 동기화 (`syncNewRecordsToAnalysis`)

- **역할**: 사용자가 입력한 '대기' 상태의 실적을 관리자가 검수할 수 있도록 `performance_records_absorption` 테이블로 가져오는 핵심 백그라운드 작업.
- **상세 로직**:
    1. `performance_records` 테이블에서 현재 선택된 '정산월'에 해당하는 `'대기'` 상태의 실적을 모두 `select` 합니다.
    2. 찾아낸 실적들의 데이터를 기반으로 `performance_records_absorption` 테이블에 삽입(INSERT)할 레코드 객체 배열을 생성합니다.
    3. **가장 중요한 부분**: 이 때, 관리자가 직접 추가한 데이터와 구분하기 위해 `company_id_add` 와 `client_id_add` 필드의 값을 **반드시 `NULL` 로 명시**하여 삽입합니다. `review_status` 는 '검수중'으로 설정합니다.
    4. 준비된 레코드 목록을 `performance_records_absorption` 에 `insert` 합니다.
    5. 성공적으로 삽입되면, 원본 실적의 `review_status` 를 '대기'에서 **'검수중'**으로 `update` 하여 중복해서 불러오는 것을 방지합니다.

2.4. 인라인 수정 및 저장 (`saveEdit`)

- **역할**: 사용자가 수정한 내용을 데이터베이스에 저장하는 함수. `isNewRow` 플래그로 수정과 추가 로직을 분기합니다.
- **수정 로직 (`isNewRow === false`)**:
    1. 사용자가 수정한 값들(처방월, 제품, 수량 등)을 모아 `recordData` 객체를 생성합니다. 이 객체에는 `..._modify` 필드들만 포함됩니다.
    2. `review_action` 필드를 **'수정'**으로 설정하여 변경 이력을 남깁니다.
    3. **핵심**: 이 로직에서는 `company_id_add` 나 `client_id_add` 필드를 **절대 건드리지 않습니다.** 따라서 `NULL` 상태가 유지됩니다.
    4. `supabase.from('performance_records_absorption').update(recordData).eq('id', row.id)` 를 호출하여 DB를 업데이트합니다.
- **추가 로직 (`isNewRow === true`)**:
    1. `review_action` 필드를 **'추가'**로 설정하여 신규 데이터임을 명시합니다.
    2. **핵심**: `company_id_add` 와 `client_id_add` 필드에 **기준이 되었던 바로 위 행의 업체 ID (`row.company_id`)와 거래처 ID (`row.client_id`)**를 저장합니다.
    3. 나머지 수정된 정보(`..._modify` 필드)와 함께 `supabase.from('performance_records_absorption').insert(recordData)` 를 호출하여 DB에 삽입합니다.

---

3. 흡수율 분석 (`AdminAbsorptionAnalysisView.vue`)

- **역할**: 검수가 완료된 데이터를 기준으로, 제품별 처방액 대비 실제 매출(도매+직거래)의 비율을 분석합니다.
- **핵심 로직**:
  - 프론트엔드에서는 복잡한 계산을 수행하지 않고, **`calculate_absorption_rates`** 라는 데이터베이스 함수(RPC)를 호출합니다.
  - **`calculate_absorption_rates` 함수의 역할**:
    1. `settlement_month`를 인자로 받습니다.
    2. 해당 월의 모든 '완료'된 실적(`review_status` = '완료')을 조회합니다.
    3. 각 실적의 `client_id`와 `product_id`를 기준으로, 관련된 도매/직거래 매출 테이블에서 해당 월의 매출 데이터를 찾아 합산합니다.
    4. `(합산된 매출액 / 처방액) * 100` 공식을 통해 흡수율(%)을 계산합니다.
    5. `performance_records_absorption`의 `id`와 계산된 결과를 함께 반환합니다.
  - 프론트엔드는 이 함수가 반환한 계산 결과를 기존에 `displayRows`에 있던 데이터와 `id` 기준으로 매핑하여 '합산액', '흡수율' 열을 업데이트합니다.

---

4. 정산내역서 공유 (`AdminSettlementShareView.vue`)

- **역할**: 검수가 끝난 최종 실적을 **업체별로 합산**하여 보여주고, 각 업체에게 해당 내역을 공유할지 여부를 설정합니다.
- **핵심 로직**:
  1. **데이터 조회 (`loadSettlementData`)**:
     - 이 함수는 **`get_settlement_summary_by_company`** 라는 전용 데이터베이스 함수(RPC)를 호출하는 것이 전부입니다. 다른 프론트엔드 계산 로직이 전혀 없습니다.
  2. **`get_settlement_summary_by_company` 함수의 내부 동작**:
     - `performance_records_absorption`에서 특정 '정산월'과 `review_status`가 '완료'인 데이터를 모두 조회합니다.
     - **`GROUP BY company_id`** 구문을 사용하여, 다른 어떤 필드에도 방해받지 않고 오직 **업체 ID를 기준으로** 모든 데이터를 완벽하게 합산합니다.
     - 합산 항목: `COUNT(DISTINCT client_id)`로 병의원 수를, `COUNT(*)`로 처방 건수를, `SUM()`으로 총 처방액/지급액을 계산합니다.
     - `company_name` 등 그룹화되지 않은 다른 문자열 필드는 `MAX()` 집계 함수를 사용하여 대표값을 선택합니다. (업체 ID가 동일하면 이 값들은 모두 동일하므로 어떤 집계 함수든 무방합니다.)
     - `settlement_share` 테이블과 `LEFT JOIN` 하여 현재 공유 상태까지 가져와 최종 결과를 한 번의 쿼리로 반환합니다.
  3. **공유 상태 저장 (`saveShareStatus`)**:
     - 사용자가 체크박스를 변경하면, 변경된 내용(`company_id`, `is_shared`, `settlement_share_id`)만 `shareChanges` 객체에 추적됩니다.
     - '저장' 버튼을 누르면, `shareChanges`에 기록된 업체들만 `settlement_share` 테이블에 `INSERT` (기존 공유 기록이 없던 업체) 또는 `UPDATE` (기존 공유 기록이 있던 업체)를 수

행하여 불필요한 DB 접근을 최소화합니다.

4. **기대 효과**: 모든 계산과 그룹화가 데이터베이스 안에서 완벽하게 처리되므로, 프론트엔드 코드는 극적으로 단순해졌고 데이터 정합성 오류가 발생할 가능성이 원천적으로 차단되었습니다.

---

5. 실적 등록 시 초기 상태 차이

5.1. 이용자 vs 관리자 실적 등록 초기 상태

신일제약 실적관리 시스템에서는 **등록자의 권한에 따라 실적의 초기 검수 상태가 다르게 설정**됩니다.

5.1.1. 이용자 실적 등록

- **초기 상태**: `review_status = '대기'`
- **설명**: 이용자가 등록한 실적은 관리자의 검수를 거쳐야 하므로 초기 상태가 '대기'로 설정됩니다.
- **코드 위치**: `vue-project/src/views/user/PerformanceRegisterEdit.vue`

```
// 관리자가 입력하는 경우 바로 완료 상태로 저장
const reviewStatus = (route.query.companyId && isAdminUser.value) ? '완료' : '대기';
```

5.1.2. 관리자 실적 등록

- **초기 상태**: `review_status = '완료'`
- **설명**: 관리자가 등록한 실적은 이미 검수가 완료된 것으로 간주하여 바로 '완료' 상태로 설정됩니다.
- **조건**: `route.query.companyId` 가 존재하고 `isAdminUser.value` 가 `true` 인 경우
- **코드 위치**: `vue-project/src/views/user/PerformanceRegisterEdit.vue`

```
// 관리자가 입력하는 경우 바로 완료 상태로 저장
const reviewStatus = (route.query.companyId && isAdminUser.value) ? '완료' : '대기';
```

5.1.3. 관리자 실적 검수 화면에서 추가

- **초기 상태**: `review_status = '검수중'`
- **설명**: 관리자가 실적 검수 화면에서 새로 추가하는 실적은 검수 과정에 포함되어야 하므로 '검수중' 상태로 설정됩니다.
- **코드 위치**: `vue-project/src/views/admin/AdminPerformanceReviewView.vue`

```
// 추가 (INSERT)
saveData = {
  ...saveData,
  created_at: new Date().toISOString(),
  registered_by: adminUserId,
  review_action: '추가',
```

```
  review_status: '검수중'
};
```

5.2. 상태별 의미 및 데이터 흐름

상태	의미	등록자	데이터 흐름
대기	이용자가 등록한 실적, 관리자 검수 대기	이용자	이용자 등록 → 관리자 검수 화면에서 '검수중'으로 변경
검수중	관리자가 검수 중인 실적	관리자	관리자 검수 화면에서 추가/수정 → 검수 완료 시 '완료'로 변경
완료	검수가 완료된 최종 실적	관리자/이용자	흡수율 분석, 정산내역서 공유 대상

5.3. 구현 상세

5.3.1. 이용자 실적 등록 시

```
// INSERT 시
const dataToInsert = rowsToInsert.map(row => ({
  // ... 기타 필드들
  registered_by: currentUserUid,
  review_status: '대기', // 이용자는 항상 '대기' 상태
  commission_rate: commissionRate
}));

// UPDATE 시
const updatePromises = rowsToUpdate.map(row =>
  supabase.from('performance_records').update({
    // ... 기타 필드들
    review_status: '대기', // 이용자는 항상 '대기' 상태
    updated_by: currentUserUid,
    updated_at: new Date().toISOString()
  }).eq('id', row.id)
);
```

5.3.2. 관리자 실적 등록 시

```
// 관리자 권한 확인
const reviewStatus = (route.query.companyId && isAdminUser.value) ? '완료' : '대기';

// INSERT 시
const dataToInsert = rowsToInsert.map(row => ({
  // ... 기타 필드들
  registered_by: currentUserUid,
  review_status: reviewStatus, // 관리자는 '완료' 상태
  commission_rate: commissionRate
}));

// UPDATE 시
```

```
const updatePromises = rowsToUpdate.map(row =>
  supabase.from('performance_records').update({
    // ... 기타 필드들
    review_status: reviewStatus, // 관리자는 '완료' 상태
    updated_by: currentUserUid,
    updated_at: new Date().toISOString()
  }).eq('id', row.id)
);
```

5.4. 비즈니스 로직의 의의

1. **검수 프로세스 보장**: 이용자 등록 실적은 반드시 관리자 검수를 거치도록 강제
2. **관리자 권한 인정**: 관리자가 등록한 실적은 즉시 활용 가능하도록 설계
3. **데이터 무결성**: 상태에 따른 명확한 데이터 흐름으로 시스템 일관성 확보
4. **업무 효율성**: 관리자의 즉시 등록과 이용자의 검수 대기 프로세스 분리

---

6. Database Functions 상세 분석

6.1. 핵심 비즈니스 로직 함수들

`calculate_absorption_rates(p_settlement_month text)`

```
-- Returns: TABLE(analysis_id bigint, wholesale_revenue numeric, direct_revenue numeric)
-- Security: Invoker
```

- **목적**: 흡수율 분석을 실행하고, 각 실적별로 연결된 도매/직거래 매출액을 계산
- **핵심 로직**:
    1. `review_details_view` 에서 '완료' 상태의 실적 데이터 조회
    2. 병원-약국 매핑 정보를 통한 약국 연결
    3. 도매/직거래 매출 데이터와 매칭하여 매출액 계산
    4. 약국별 병원 수에 따른 매출 분배 로직 적용

`get_settlement_summary_by_company(p_settlement_month text)`

```
-- Returns: TABLE(company_id, company_name, client_count, total_records, total_prescription_a
-- Security: DEFINER
```

- **목적**: 정산내역서 공유용 업체별 합산 데이터 조회
- **핵심 로직**:
    1. `performance_records_absorption` 에서 '완료' 상태 데이터 조회
    2. 업체별로 GROUP BY하여 합산 계산
    3. `settlement_share` 테이블과 JOIN하여 공유 상태 포함

5.2. 변경사항 감지 함수들

check_for_updates(p_settlement_month text)

```
-- Returns: JSON
-- Security: Invoker
```

- **목적**: 정산월별 변경사항 확인 (검수, 매핑, 매출 데이터 변경)
- **반환 데이터**:

```
{
  "updated_reviews": 0,
  "updated_mappings": 0,
  "updated_sales": 0,
  "has_changes": false
}
```

check_performance_changes(p_settlement_month text)

```
-- Returns: jsonb
-- Security: Invoker
```

- **목적**: 실적 데이터 변경사항 확인 (추가, 수정, 삭제 건수)
- **반환 데이터**:

```
{
  "added_count": 0,
  "modified_count": 0,
  "deleted_count": 0
}
```

5.3. 데이터 조회 함수들

get_absorption_analysis_details(p_settlement_month, p_company_id, p_client_id)

```
-- Returns: TABLE(id, company_name, client_name, prescription_month, product_name, prescripti
-- Security: Invoker
```

- **목적**: 흡수율 분석 화면의 상세 데이터 조회
- **필터링**: 정산월, 업체, 거래처별 필터링 지원

get_distinct_companies_from_analysis(p_settlement_month)

```
-- Returns: TABLE(id uuid, company_name text)
-- Security: Invoker
```

- **목적**: 업체 필터 목록 조회
- **로직**: '완료' 상태 실적에 연결된 모든 업체 중복 제거

5.4. 디버깅 함수들

`debug_absorption_rates(p_settlement_month text)`

```
-- Returns: TABLE(absorption_analysis_id, pharmacy_brn_from_prescription, standard_code_from_
-- Security: Invoker
```

- **목적**: 흡수율 분석 디버깅용 (상세 매칭 정보)
- **용도**: 매출 데이터 매칭 과정에서 발생하는 문제 진단

`debug_distribution_ratios(p_settlement_month text)`

```
-- Returns: TABLE(absorption_analysis_id, company_name, client_name, product_name, pharmacy_b
-- Security: Invoker
```

- **목적**: 분배 비율 디버깅용
- **용도**: 약국별 매출 분배 로직 검증

---

6. 최신 RLS 정책 (Row Level Security)

6.1. 핵심 정책 개요

시스템은 Supabase의 RLS를 활용하여 데이터 접근을 제어합니다:

- **`companies` 테이블**:
  - 관리자는 모든 업체 데이터 조회 가능
  - 일반 사용자는 자신의 업체 데이터만 조회/수정 가능
  - 회원가입 시 자동으로 업체 정보 생성
- **`performance_records` 테이블**:
  - 관리자는 모든 실적 데이터 접근 가능
  - 일반 사용자는 자신의 업체 실적만 조회/수정 가능
- **`notices` 테이블**:
  - 모든 인증된 사용자가 공지사항 조회 가능
  - 관리자만 공지사항 작성/수정/삭제 가능
- **`products` 테이블**:
  - 모든 인증된 사용자가 활성 제품 조회 가능
  - 관리자만 제품 관리 가능

6.2. 권한 관리 체계

- **`user_type`** : 'admin' 또는 'user'로 구분
- **`approval_status`** : 'pending', 'approved', 'rejected'로 업체 승인 상태 관리
- **`status`** : 'active', 'inactive'로 데이터 활성 상태 관리

---

7. 파일 업로드 시스템

7.1. 실적 증빙 파일 (`performance_evidence_files`)

- **저장 위치**: Supabase Storage
- **접근 권한**: 업체별로 자신의 파일만 관리 가능
- **파일 정보**: 파일명, 경로, 크기, 업로드자, 업로드 시각 추적

7.2. 공지사항 첨부 파일

- **파일 URL**: `notices.file_url` 필드에 저장
- **링크 정보**: `notices.links` 필드에 외부 링크 저장
- **관리 권한**: 관리자만 파일 업로드/관리 가능

---

8. 실적 등록 시 처방월 변경 로직

8.1. 제품 정보 자동 업데이트 규칙

목적: 사용자가 실적 등록 중 처방월을 변경할 때, 해당 월의 제품 정보(약가, 수수료율)를 자동으로 업데이트

핵심 로직:

1. **검색 기준**: 이미 선택된 제품의 **보험코드**를 사용
2. **검색 대상**: 변경된 처방월의 `base_month` 에서 해당 보험코드 검색
3. **결과 처리**:
    - **여러 결과가 있는 경우**: 첫 번째 결과만 사용 (`.limit(1)` 적용)
    - **결과가 없는 경우**: 제품 선택 해제 (제품명, 보험코드, 약가, 수수료율 등 모든 정보 초기화)
    - **결과가 있는 경우**: 새로운 제품 정보로 자동 업데이트 (약가, 수수료율, 처방액 재계산)

구현 위치: `PerformanceRegisterEdit.vue - updateProductInfoForMonthChange()` 함수

동작 시나리오:

- **시나리오 1**: 2025-05에서 가모피드정 5mg 선택 → 2025-04로 변경 → 보험코드로 검색하여 약가 4원, 수수료율 40%로 자동 업데이트
- **시나리오 2**: 2025-05에서 가모피드정 5mg 선택 → 2025-03으로 변경 → 보험코드가 2025-03에 없으므로 제품 선택 해제

기술적 특징:

- `.single()` 대신 `.limit(1)` 사용으로 여러 결과 처리 가능
- 에러 처리 개선으로 쿼리 에러와 데이터 없음을 분리
- 처방수량이 있는 경우 처방액 자동 재계산

---

9. 실적입력기간 제한 시스템

9.1. 개요

시스템은 실적 등록을 관리자가 설정한 기간 내에서만 허용하는 제한 시스템을 운영합니다. 이는 데이터 입력의 일관성과 관리 효율성을 위한 핵심 기능입니다.

9.2. 데이터 구조

`settlement_months` 테이블:

- `settlement_month` : 정산월 (예: '2025-01')
- `start_date` : 실적입력 시작일 (YYYY-MM-DD)
- `end_date` : 실적입력 종료일 (YYYY-MM-DD)
- `notice` : 전달 사항
- `status` : 활성/비활성 상태
- `remarks` : 비고

9.3. 권한별 실적입력기간 제한

9.3.1. 일반 사용자 (CSO 업체)

- **제한 적용**: 실적입력기간 내에서만 실적 등록/수정 가능
- **검증 로직**: `checkInputPeriod()` 함수에서 현재 시간과 `start_date`, `end_date` 비교
- **UI 제어**: 실적입력기간 외에는 모든 입력 버튼 비활성화
- **구현 위치**: `PerformanceRegister.vue`, `PerformanceRegisterEdit.vue`

9.3.2. 관리자 (신일제약)

- **제한 없음**: 실적입력기간과 무관하게 언제든지 실적 등록/수정 가능
- **우회 로직**: `checkInputPeriod()` 함수에서 `isInputPeriod.value = true` 로 강제 설정
- **구현 위치**: `AdminPerformanceRegisterView.vue`

9.4. 핵심 로직

9.4.1. 실적입력기간 체크 함수

```
// 일반 사용자용
const checkInputPeriod = async () => {
  if (!selectedSettlementMonth.value) return
  const { data, error } = await supabase
    .from('settlement_months')
    .select('start_date, end_date')
    .eq('settlement_month', selectedSettlementMonth.value)
    .single()

  if (!error && data) {
    const now = new Date()
    const startDate = new Date(data.start_date)
    const endDate = new Date(data.end_date)
    endDate.setHours(23, 59, 59, 999) // 종료일을 그날의 마지막 시간으로 설정
    isInputPeriod.value = now >= startDate && now <= endDate
  } else {
    isInputPeriod.value = false
  }
}

// 관리자용 (우회)
const checkInputPeriod = async () => {
  // 관리자는 항상 입력 가능
  isInputPeriod.value = true
  return
  // ... 일반 사용자 로직 (실행되지 않음)
}
```

9.4.2. UI 제어 로직

- **버튼 비활성화**: `:disabled="!isInputPeriod"`
- **실적 등록**: `:disabled="!isInputPeriod"`
- **실적 수정**: `:disabled="!isInputPeriod"`
- **파일 업로드**: `:disabled="!isInputPeriod"`

9.5. 관리자 설정 화면

`AdminSettlementMonthsCreateView.vue`:

- 정산월 설정
- 실적입력 시작일/종료일 설정
- 전달 사항 입력
- 상태 관리 (활성/비활성)

9.6. 동작 시나리오

9.6.1. 일반 사용자 시나리오

1. **기간 내**: 실적 등록/수정 버튼 활성화 → 정상 입력 가능

2. **기간 외**: 실적 등록/수정 버튼 비활성화 → 입력 불가
3. **기간 변경**: 관리자가 기간을 수정하면 즉시 반영

9.6.2. 관리자 시나리오

1. **언제든지**: 실적 등록/수정 버튼 항상 활성화
2. **기간 무관**: 설정된 실적입력기간과 관계없이 입력 가능
3. **긴급 대응**: 기간 외에도 실적 데이터 수정 가능

9.7. 보안 및 데이터 무결성

- **데이터베이스 레벨**: RLS 정책으로 추가 보안 강화
- **프론트엔드 레벨**: UI 제어로 사용자 경험 개선
- **관리자 권한**: `user_type = 'admin'` 확인 후 우회 로직 적용

---

10. 실적 등록 시 제품 선택 및 키보드 네비게이션 시스템

10.1. 제품 선택 방식

10.1.1. 제품 선택 방식 (2가지)

방식 1: 제품명 검색 + 드롭다운

- **검색 방식**: 제품명 또는 보험코드로 실시간 검색
- **드롭다운 표시**: 검색 결과를 드롭다운으로 표시 (제품명 + 보험코드)
- **선택 방법**:
  - 마우스 클릭으로 선택
  - 키보드 상하 화살표로 탐색 후 Enter로 선택

방식 2: 드롭다운 아이콘 클릭

- **동작**: 드롭다운 아이콘(▼) 클릭 시 해당 처방월의 모든 제품 표시
- **유니크 처리**: 보험코드 기준으로 중복 제거하여 유니크하게 표시
- **표시 내용**: 제품명 + 보험코드
- **구현 위치**: `PerformanceRegisterEdit.vue - toggleProductDropdown()` 함수

10.1.2. 검색 로직

방식 1: 제품명 실시간 검색

```
// 제품명 입력 시 실시간 검색
function handleProductNameInput(rowIndex, event) {
  const query = event.target.value.toLowerCase();
  const month = inputRows.value[rowIndex].prescription_month;
```

```
  productSearchForRow.value.activeRowIndex = rowIndex;
  productSearchForRow.value.results = productList.filter(
    product => product.product_name.toLowerCase().includes(query) ||
               product.insurance_code.toLowerCase().includes(query)
  );
  productSearchForRow.value.selectedIndex = -1;
  productSearchForRow.value.show = productSearchForRow.value.results.length > 0;
}
```

방식 2: 드롭다운 아이콘 클릭 (전체 제품 표시)

```
// 드롭다운 아이콘 클릭 시 해당 월의 모든 제품 표시
function toggleProductDropdown(rowIndex) {
  if (!isInputEnabled.value) return;

  // 이미 열려있으면 닫기
  if (productSearchForRow.value.show && productSearchForRow.value.activeRowIndex === rowIndex
    productSearchForRow.value.show = false;
    productSearchForRow.value.activeRowIndex = -1;
    return;
  }

  // 처방월에 맞는 제품만 필터링 (보험코드 기준 유니크)
  const month = inputRows.value[rowIndex].prescription_month;
  productSearchForRow.value.activeRowIndex = rowIndex;
  productSearchForRow.value.results = productsByMonth.value[month] || [];
  productSearchForRow.value.selectedIndex = -1;
  productSearchForRow.value.show = productSearchForRow.value.results.length > 0;
}
```

제품 유니크 처리 로직

```
// 보험코드 기준으로 유니크하게 제품 처리
async function fetchProductsForMonth(month) {
  const { data, error } = await supabase
    .from('products')
    .select('*')
    .eq('status', 'active')
    .eq('base_month', month)
    .range(0, 2999);

  if (!error && data) {
    const uniqByMonthAndInsurance = {};
    const noInsurance = [];

    data.forEach(p => {
      const key = `${p.base_month}_${p.insurance_code || ''}`;
      if (p.insurance_code) {
        // 보험코드가 있으면 유니크 처리
        if (!uniqByMonthAndInsurance[key]) uniqByMonthAndInsurance[key] = p;
      } else {
        // 보험코드가 없으면 별도 배열에 추가
```

```
        noInsurance.push(p);
      }
    });

    // 유니크 처리된 제품 + 보험코드 없는 제품
    productsByMonth.value[month] = [...Object.values(uniqByMonthAndInsurance), ...noInsurance
  }
}
```

10.2. 키보드 네비게이션 규칙

10.2.1. Enter 키 동작

- **제품명 필드**: 선택된 제품 적용 후 처방수량 필드로 이동
- **처방수량 필드**: 제품명과 수량이 모두 입력된 경우 다음 행으로 이동 (자동 행 추가)
- **비고 필드**: 제품명과 수량이 모두 입력된 경우 다음 행으로 이동 (자동 행 추가)

10.2.2. 화살표 키 동작

- **상하 화살표**: 같은 열에서 위아래 행으로 이동
- **좌우 화살표**: 같은 행에서 좌우 열로 이동
- **이동 순서**: 처방월 → 제품명 → 처방수량 → 처방구분 → 비고
- **제품 검색 중**: 상하 화살표는 드롭다운 탐색, 좌우 화살표는 차단

10.2.3. 특수 키 동작

- **ESC 키**: 현재 행의 모든 입력 정보 초기화 (제품명, 처방수량, 비고 등)
- **Delete 키**: 현재 행 삭제 (최소 1행은 유지)
- **Insert 키**: 현재 행 아래에 새 행 추가

10.3. 자동 행 관리 시스템

10.3.1. 자동 행 추가

- **조건**: 마지막 행에 제품명 또는 처방수량이 입력되면 자동으로 새 행 추가
- **구현**: `watch(inputRows)` 감시자로 실시간 모니터링
- **기본값**: 새 행의 처방월은 정산월 - 1M, 처방구분은 'EDI'

10.3.2. 행 초기화 로직

```
function resetRow(rowIdx) {
  const row = inputRows.value[rowIdx];
  row.prescription_month = getDefaultPrescriptionMonth();
  row.product_name_display = '';
  row.product_id = null;
  row.insurance_code = '';
  row.price = '';
```

```
  row.prescription_qty = '';
  row.prescription_amount = '';
  row.prescription_type = 'EDI';
  row.remarks = '';
  row.commission_rate_a = null;
  row.commission_rate_b = null;

  // 제품 검색 드롭다운 숨기기
  if (productSearchForRow.value.show && productSearchForRow.value.activeRowIndex === rowIdx)
    productSearchForRow.value.show = false;
    productSearchForRow.value.activeRowIndex = -1;
  }

  // 해당 행의 제품명 필드로 포커스 이동
  nextTick(() => {
    focusField(rowIdx, 'product_name');
  });
}
```

10.4. 전역 키보드 이벤트 처리

10.4.1. 핸들러 등록

```
// 컴포넌트 마운트 시 등록
onMounted(() => {
  document.addEventListener('keydown', handleGlobalKeydown);
  document.addEventListener('click', handleGlobalClick);
});

// 컴포넌트 언마운트 시 해제
onUnmounted(() => {
  document.removeEventListener('keydown', handleGlobalKeydown);
  document.removeEventListener('click', handleGlobalClick);
});
```

10.4.2. 전역 키 처리 로직

```
function handleGlobalKeydown(e) {
  // 제품 검색 드롭다운이 열려있으면 Insert/Delete/Escape 키 차단
  if (isProductSearchOpen.value) {
    if (e.key === 'Delete' || e.key === 'Insert' || e.key === 'Escape') {
      e.preventDefault();
      return;
    }
  }

  if (e.key === 'Delete') {
    e.preventDefault();
    const currentRowIdx = currentCell.value.row;
    if (inputRows.value.length > 1) {
```

```
      confirmDeleteRow(currentRowIdx);
    }
  } else if (e.key === 'Insert') {
    e.preventDefault();
    const currentRowIdx = currentCell.value.row;
    confirmAddRowBelow(currentRowIdx);
  } else if (e.key === 'Escape') {
    e.preventDefault();
    const currentRowIdx = currentCell.value.row;
    if (isRowEditable(inputRows.value[currentRowIdx])) {
      resetRow(currentRowIdx);
    }
  }
}
```

10.5. 포커스 관리 시스템

10.5.1. 셀 포커스 추적

- **현재 셀**: `currentCell.value = { row: rowIdx, col: colName }`
- **포커스 스타일**: `cellClass()` 함수로 현재 셀 하이라이트
- **자동 포커스**: 행 추가/삭제 시 적절한 필드로 자동 포커스

10.5.2. 필드별 포커스 규칙

- **처방월**: 드롭다운 선택 후 제품명으로 이동
- **제품명**: 제품 선택 후 처방수량으로 이동
- **처방수량**: 입력 완료 후 다음 행 제품명으로 이동
- **처방구분**: 선택 후 비고로 이동
- **비고**: 입력 완료 후 다음 행 제품명으로 이동

10.6. 사용자 경험 최적화

10.6.1. 편집 상태별 제어

- **편집 가능**: 모든 키보드 기능 활성화
- **편집 불가**: 키보드 기능 비활성화, 읽기 전용 모드
- **관리자**: 모든 상태에서 편집 가능

10.6.2. 실시간 피드백

- **처방액 자동 계산**: 처방수량 입력 시 실시간 계산
- **제품 정보 자동 업데이트**: 처방월 변경 시 제품 정보 자동 업데이트
- **처방구분 자동 복사**: 한 행의 처방구분 변경 시 아래 행들에 자동 적용

---

11. 정렬 규칙 시스템

11.1. 개요

시스템은 사용자 권한과 화면 목적에 따라 다른 정렬 규칙을 적용합니다. 기본적으로 가나다순 정렬을 사용하되, 입력 상태와 검수 상태에 따라 우선순위를 조정합니다.

11.2. 사용자별 정렬 규칙

11.2.1. 일반 사용자 (CSO 업체)

- **정렬 대상**: 거래처 목록
- **정렬 방식**:
  - **1순위**: 처방건수 0건 → 1건 이상 (입력된 것이 아래로)
  - **2순위**: 각 그룹 내 가나다순 정렬
- **구현 위치**: `PerformanceRegister.vue`, `PerformanceRegisterList.vue`

```
// 정렬: 처방건수 0건 → 1건 이상, 각 그룹 내 가나다순
const noData = unsortedList
  .filter((c) => !c.performance_count || Number(c.performance_count) === 0)
  .sort((a, b) => a.name.localeCompare(b.name, 'ko'))
const hasData = unsortedList
  .filter((c) => Number(c.performance_count) > 0)
  .sort((a, b) => a.name.localeCompare(b.name, 'ko'))
clientList.value = [...noData, ...hasData] // 입력된 것이 아래로
```

11.2.2. 관리자 (신일제약)

- **정렬 대상**: 업체 목록, 거래처 목록
- **업체 정렬**: 가나다순 (company_name 기준)
- **거래처 정렬**:
  - **1순위**: 처방건수 0건 → 1건 이상 (입력된 것이 위로)
  - **2순위**: 각 그룹 내 가나다순 정렬
- **구현 위치**: `AdminPerformanceRegisterView.vue`, `AdminPerformanceReviewView.vue`

```
// 업체 목록 정렬 (가나다순)
const { data, error } = await supabase
  .from('companies')
  .select('id, company_name')
  .eq('approval_status', 'approved')
  .eq('status', 'active')
  .eq('user_type', 'user')
  .order('company_name', { ascending: true }) // 가나다순

// 거래처 목록 정렬 (입력된 것이 위로)
const noData = unsortedList
  .filter((c) => !c.performance_count || Number(c.performance_count) === 0)
  .sort((a, b) => a.name.localeCompare(b.name, 'ko'))
```

```
const hasData = unsortedList
  .filter((c) => Number(c.performance_count) > 0)
  .sort((a, b) => a.name.localeCompare(b.name, 'ko'))
clientList.value = [...hasData, ...noData] // 입력된 것이 위로
```

11.3. 실적 검수 화면 정렬 규칙

11.3.1. 검수 상태별 정렬

- **정렬 우선순위**: 신규 > 검수중 > 완료
- **상태 표시**:
  - `review_status = '대기'` → `display_status = '신규'`
  - `review_status = '검수중'` → `display_status = '검수중'`
  - `review_status = '완료'` → `display_status = '완료'`

11.3.2. 필터 옵션 정렬

```
// 업체 필터 정렬 (가나다순)
monthlyCompanies.value = companies.sort((a, b) =>
  a.company_name.localeCompare(b.company_name, 'ko')
);

// 거래처 필터 정렬 (가나다순)
monthlyHospitals.value = clients.sort((a, b) =>
  a.name.localeCompare(b.name, 'ko')
);
```

11.4. 정렬 로직 상세

11.4.1. 가나다순 정렬

- **사용 언어**: 한국어 (`'ko'` 로케일)
- **정렬 방식**: `localeCompare()` 메서드 사용
- **대소문자**: 구분하지 않음

11.4.2. 상태별 그룹화 정렬

1. **데이터 분리**: 조건에 따라 그룹별로 분리
2. **그룹 내 정렬**: 각 그룹 내에서 가나다순 정렬
3. **그룹 결합**: 우선순위에 따라 그룹들을 결합

11.4.3. 실시간 정렬

- **Computed 속성**: `displayRows` computed로 실시간 정렬
- **필터 적용**: 처방월, 업체, 거래처 필터와 함께 적용
- **성능 최적화**: 필요한 경우에만 정렬 수행

11.5. 정렬 규칙 요약

사용자 유형	정렬 대상	1순위	2순위	입력 데이터 위치
일반 사용자	거래처	처방건수 (0건 → 1건+)	가나다순	아래로
관리자	업체	가나다순	-	-
관리자	거래처	처방건수 (0건 → 1건+)	가나다순	위로
관리자	실적 검수	검수상태 (신규 → 검수중 → 완료)	가나다순	-

---

12. 이용자 실적 등록 프로세스 상세 가이드

12.1. 개요

이용자(CSO 업체)의 실적 등록은 **단계별 입력 프로세스**와 **고급 키보드 네비게이션**을 통해 효율적인 데이터 입력을 지원합니다. 모든 입력은 실시간으로 검증되며, 관리자의 검수를 거쳐 최종 확정됩니다.

12.2. 단계별 실적 등록 프로세스

12.2.1. 1단계: 거래처 선택 및 화면 진입

- **화면**: `PerformanceRegister.vue` (거래처 목록)
- **정렬 규칙**:
  - 처방건수 0건 → 1건 이상 (입력된 것이 아래로)
  - 각 그룹 내 가나다순 정렬
- **선택 방법**: 거래처명 클릭 또는 키보드 네비게이션

12.2.2. 2단계: 실적 등록 화면 진입

- **화면**: `PerformanceRegisterEdit.vue` (실적 입력)
- **초기 상태**:
  - 기존 실적이 있으면 로드
  - 없으면 빈 행 1개 생성
  - 마지막 행은 항상 입력 가능한 빈 행으로 유지

12.2.3. 3단계: 처방월 설정

- **기본값**: 정산월 - 1개월 (예: 2025-01 정산 → 2024-12 처방)
- **옵션**: 정산월 - 1~3개월 선택 가능
- **변경 시**: 선택된 제품의 보험코드로 새 처방월에서 제품 정보 자동 업데이트

12.2.4. 4단계: 제품 선택 (2가지 방식)

방식 1: 제품명 실시간 검색

1. 제품명 필드에 제품명 또는 보험코드 입력

2. 실시간으로 검색 결과 드롭다운 표시
3. 상하 화살표로 탐색 후 Enter 선택
4. 또는 마우스 클릭으로 선택

방식 2: 드롭다운 아이콘 클릭

1. 제품명 필드 옆 ▼ 아이콘 클릭
2. 해당 처방월의 모든 제품 표시 (보험코드 기준 유니크)
3. 제품명 + 보험코드 형태로 표시
4. 마우스 클릭 또는 키보드로 선택

12.2.5. 5단계: 처방수량 입력

- **자동 계산**: 수량 × 단가 = 처방액 실시간 계산
- **포맷팅**: 천 단위 콤마 자동 적용
- **검증**: 숫자만 입력 가능

12.2.6. 6단계: 처방구분 설정

- **기본값**: 'EDI'
- **옵션**: EDI, ERP직거래자료, 매출자료, 약국조제, 원내매출, 원외매출, 차감
- **자동 복사**: 한 행의 처방구분 변경 시 아래 행들에 자동 적용

12.2.7. 7단계: 비고 입력

- **선택사항**: 추가 정보 입력
- **자동 이동**: Enter 키로 다음 행으로 이동

12.2.8. 8단계: 자동 행 관리

- **조건**: 마지막 행에 제품명 또는 처방수량 입력 시
- **동작**: 자동으로 새 행 추가
- **기본값**: 처방월(정산월-1M), 처방구분(EDI)

12.2.9. 9단계: 저장 및 검수 대기

- **검증**: 제품명과 처방수량 필수 입력 확인
- **상태**: `review_status = '대기'` 로 저장
- **이후**: 관리자 검수 대기

12.3. 고급 키보드 네비게이션 시스템

12.3.1. 기본 이동 키

키	동작	설명
Tab	다음 필드로 이동	표준 탭 순서

키	동작	설명
Shift+Tab	이전 필드로 이동	역방향 탭 순서
Enter	특별 동작	필드별 다른 동작

12.3.2. 화살표 키 네비게이션

키	동작	제한사항
↑	같은 열에서 위 행으로 이동	편집 가능한 행만
↓	같은 열에서 아래 행으로 이동	편집 가능한 행만
←	같은 행에서 왼쪽 열로 이동	제품 검색 중 차단
→	같은 행에서 오른쪽 열로 이동	제품 검색 중 차단

이동 순서: 처방월 → 제품명 → 처방수량 → 처방구분 → 비고

12.3.3. Enter 키 특별 동작

필드	Enter 키 동작	조건
제품명	선택된 제품 적용 후 처방수량으로 이동	제품이 선택된 경우
처방수량	다음 행 제품명으로 이동 (자동 행 추가)	제품명과 수량 모두 입력
비고	다음 행 제품명으로 이동 (자동 행 추가)	제품명과 수량 모두 입력

12.3.4. 특수 키 동작

키	동작	설명
Delete	현재 행 삭제	최소 1행은 유지
Insert	현재 행 아래 새 행 추가	편집 가능한 행에서만
Escape	현재 행 초기화	편집 가능한 행에서만

12.3.5. 제품 검색 중 키보드 제어

상황	허용 키	차단 키	설명
제품 검색 드롭다운 열림	↑↓ (드롭다운 탐색), Enter (선택)	←→ (좌우 이동), Delete, Insert, Escape	드롭다운 탐색에 집중

12.4. 실시간 검증 및 피드백

12.4.1. 입력 검증

- **제품명**: 제품 선택 필수 (자동완성 지원)
- **처방수량**: 숫자만 입력 가능, 0보다 큰 값
- **처방액**: 자동 계산, 수정 불가
- **필수 필드**: 제품명 + 처방수량 조합 필수

12.4.2. 실시간 피드백

- **처방액 계산**: 수량 입력 시 즉시 계산
- **제품 정보 업데이트**: 처방월 변경 시 자동 업데이트

- **상태 표시**: 검수상태별 색상 구분 (대기/검수중/완료)
- **편집 제한**: 검수중/완료 상태 행은 편집 불가

12.4.3. 오류 처리

- **부족한 정보**: 저장 시 누락된 필드 확인
- **중복 제품**: 보험코드 기준 유니크 처리
- **네트워크 오류**: 저장 실패 시 재시도 안내

12.5. 상태별 편집 권한

12.5.1. 일반 사용자 (CSO 업체)

- **대기 상태**: 편집 가능 (모든 키보드 기능 활성화)
- **검수중 상태**: 편집 불가 (읽기 전용)
- **완료 상태**: 편집 불가 (읽기 전용)

12.5.2. 관리자 (신일제약)

- **모든 상태**: 편집 가능 (모든 키보드 기능 활성화)
- **특별 권한**: 실적입력기간 제한 우회

12.6. 성능 최적화

12.6.1. 제품 데이터 캐싱

- **월별 캐시**: `productsByMonth` 객체로 월별 제품 목록 캐시
- **중복 제거**: 보험코드 기준 유니크 처리
- **지연 로딩**: 필요한 월의 제품만 로드

12.6.2. 키보드 이벤트 최적화

- **전역 이벤트**: document 레벨에서 키보드 이벤트 처리
- **조건부 처리**: 편집 가능한 상태에서만 키보드 기능 활성화
- **충돌 방지**: 제품 검색 중 특정 키 차단

12.7. 사용자 경험 개선

12.7.1. 시각적 피드백

- **포커스 표시**: 현재 셀 하이라이트 (`cell-focused` 클래스)
- **상태 표시**: 검수상태별 색상 구분
- **드롭다운 위치**: 화면 공간에 따른 자동 위치 조정

12.7.2. 접근성

- **키보드 접근**: 모든 기능 키보드로 접근 가능
- **스크린 리더**: 적절한 ARIA 라벨과 역할
- **색상 대비**: 충분한 색상 대비로 가독성 확보

12.7.3. 오류 복구

- **실행 취소**: Escape 키로 행 초기화
- **부분 저장**: 필수 정보가 있는 행만 저장
- **상태 복원**: 페이지 새로고침 시 상태 유지

12.8. 데이터 무결성 보장

12.8.1. 입력 검증

- **클라이언트 검증**: 실시간 입력 검증
- **서버 검증**: 저장 시 서버 측 검증
- **데이터 타입**: 적절한 데이터 타입 강제

12.8.2. 상태 관리

- **원본 데이터**: `originalRows` 로 변경사항 추적
- **변경 감지**: `canSave` computed로 저장 가능 여부 판단
- **동기화**: 저장 후 원본 데이터 업데이트

12.8.3. 이력 관리

- **등록자 추적**: `registered_by` 필드로 등록자 기록
- **수정 이력**: `updated_by`, `updated_at` 필드로 수정 이력
- **상태 이력**: `review_status` 변경 이력 추적

이러한 상세한 프로세스와 키보드 네비게이션 시스템을 통해 이용자는 효율적이고 정확한 실적 등록을 수행할 수 있으며, 시스템은 데이터의 무결성과 일관성을 보장합니다.

---

13. 최신 추가 기능 및 개선사항

13.1. 파일 업로드 및 관리 시스템

13.1.1. 실적 증빙 파일 업로드

- **구현 위치**: `AdminPerformanceFileViewModal.vue`
- **기능**: 실적 데이터에 대한 증빙 파일 업로드 및 관리
- **저장 방식**: Supabase Storage 활용
- **권한 관리**: 업체별 접근 권한 제어

13.1.2. 공지사항 첨부 파일

- **구현 위치**: `AdminNoticeCreateView.vue` , `AdminNoticeEditView.vue`
- **기능**: 공지사항에 파일 첨부 및 외부 링크 추가
- **에디터**: CKEditor 5.41.4.2 활용
- **파일 형식**: 이미지, PDF, 엑셀 등 다양한 형식 지원

13.2. 엑셀 처리 및 데이터 내보내기

13.2.1. 엑셀 업로드 기능

- **라이브러리**: XLSX 0.18.5
- **기능**: 대용량 엑셀 파일 업로드 및 데이터 파싱
- **적용 화면**:
  - `AdminWholesaleRevenueCreateView.vue`
  - `AdminDirectRevenueCreateView.vue`
  - `AdminPerformanceRegisterView.vue`

13.2.2. 엑셀 다운로드 기능

- **라이브러리**: XLSX 0.18.5 + File-saver 2.0.5
- **기능**: 데이터를 엑셀 형식으로 내보내기
- **적용 화면**: 모든 리스트 화면에서 데이터 내보내기 지원

13.3. PDF 생성 및 보고서 시스템

13.3.1. PDF 생성 기능

- **라이브러리**: jsPDF 3.0.1 + html2canvas 1.4.1
- **기능**: 화면 캡처 후 PDF 생성
- **적용 화면**: 정산내역서, 성과 보고서 등

13.3.2. 압축 파일 처리

- **라이브러리**: JSZip 3.10.1
- **기능**: 여러 파일을 ZIP 형태로 압축하여 다운로드
- **적용**: 증빙 파일 일괄 다운로드

13.4. 고급 UI/UX 개선사항

13.4.1. 드래그 앤 드롭 기능

- **라이브러리**: VueDraggable 4.1.0
- **기능**: 리스트 항목의 순서 변경 및 재정렬
- **적용 화면**: 공지사항 관리, 메뉴 순서 변경 등

13.4.2. 실시간 알림 시스템

- **구현**: PrimeVue ToastService 활용
- **기능**: 작업 완료, 오류 발생, 권한 부족 등 실시간 알림
- **적용**: 모든 CRUD 작업에서 사용자 피드백 제공

13.4.3. 확인 다이얼로그 시스템

- **구현**: PrimeVue ConfirmationService 활용
- **기능**: 삭제, 수정 등 중요 작업 전 사용자 확인
- **적용**: 데이터 삭제, 상태 변경 등 위험 작업

13.5. 성능 최적화 개선사항

13.5.1. 데이터 캐싱 시스템

- **제품 데이터**: 월별 제품 목록 캐싱으로 반복 쿼리 최소화
- **사용자 정보**: 로그인 시 사용자 정보 캐싱
- **필터 데이터**: 자주 사용되는 필터 조건 캐싱

13.5.2. 지연 로딩 (Lazy Loading)

- **컴포넌트**: Vue Router의 동적 import 활용
- **데이터**: 페이지네이션과 무한 스크롤 적용
- **이미지**: 이미지 지연 로딩으로 초기 로딩 속도 개선

13.5.3. 메모리 최적화

- **이벤트 리스너**: 컴포넌트 언마운트 시 이벤트 리스너 정리
- **대용량 데이터**: 가상 스크롤링으로 대용량 리스트 처리
- **파일 처리**: 스트리밍 방식으로 대용량 파일 업로드/다운로드

13.6. 보안 강화 기능

13.6.1. 파일 업로드 보안

- **파일 형식 검증**: 허용된 파일 형식만 업로드 가능
- **파일 크기 제한**: 최대 파일 크기 제한으로 서버 보호
- **바이러스 스캔**: 업로드된 파일의 보안 검사

13.6.2. 데이터 접근 제어

- **RLS 정책 강화**: 더 세밀한 행 레벨 보안 정책
- **API 요청 제한**: 과도한 API 호출 방지
- **세션 관리**: 자동 로그아웃 및 세션 만료 처리

13.7. 모바일 대응 및 반응형 디자인

13.7.1. 반응형 레이아웃

- **PrimeVue Aura 테마**: 모바일 친화적인 디자인 시스템
- **그리드 시스템**: 화면 크기에 따른 자동 레이아웃 조정
- **터치 인터페이스**: 모바일 터치 제스처 지원

13.7.2. 모바일 최적화

- **터치 네비게이션**: 모바일에서 편리한 터치 조작
- **화면 크기 대응**: 작은 화면에서도 사용 가능한 UI
- **성능 최적화**: 모바일 기기에서의 성능 최적화

13.8. 접근성 (Accessibility) 개선

13.8.1. 키보드 접근성

- **전체 키보드 네비게이션**: 모든 기능을 키보드로 접근 가능
- **포커스 관리**: 명확한 포커스 표시 및 이동
- **단축키 지원**: 자주 사용하는 기능의 키보드 단축키

13.8.2. 스크린 리더 지원

- **ARIA 라벨**: 적절한 ARIA 속성으로 스크린 리더 지원
- **의미론적 HTML**: 의미에 맞는 HTML 태그 사용
- **색상 대비**: 충분한 색상 대비로 가독성 확보

13.9. 국제화 (i18n) 준비

13.9.1. 다국어 지원 구조

- **텍스트 외부화**: 모든 UI 텍스트를 외부 파일로 분리
- **날짜/시간 형식**: 지역별 날짜/시간 형식 지원
- **숫자 형식**: 지역별 숫자 형식 (천 단위 구분자 등) 지원

13.9.2. RTL 언어 지원

- **우측에서 좌측**: 아랍어, 히브리어 등 RTL 언어 지원 준비
- **레이아웃 조정**: RTL 언어에 맞는 레이아웃 자동 조정

13.10. 모니터링 및 로깅 시스템

13.10.1. 에러 추적

- **클라이언트 에러**: 브라우저 콘솔 에러 로깅
- **API 에러**: Supabase API 호출 에러 추적
- **사용자 행동**: 주요 사용자 행동 로깅

13.10.2. 성능 모니터링

- **페이지 로딩 시간**: 각 페이지의 로딩 시간 측정
- **API 응답 시간**: 데이터베이스 쿼리 성능 모니터링
- **사용자 경험**: 사용자 인터랙션 시간 측정

이러한 최신 기능들과 개선사항들을 통해 신일 PMS는 더욱 강력하고 사용자 친화적인 시스템으로 발전했습니다. 모든 기능은 데이터 무결성을 보장하면서 사용자 경험을 최적화하는 방향으로 설계되었습니다.