Zapier/Airtable 기반 자동화 및 데이터 운영 플랫폼 구축

[자동차 부품 거래 및 운영 플랫폼]

제출일	2025년 월 일	학과	컴퓨터공학전공, 인공지능학과
과목	데이터베이스설계	학번	2021111968, 2023112468
담당교수	이강만 교수님	이름	정유현, 박현서

1. 데이터베이스 구축 목표

본 프로젝트는 프로토타입 단계의 NoSQL(Airtable) 기반 데이터 구조를 엔터프라이즈급 RDBMS(PostgreSQL)로 전환하여, 대규모 트랜잭션 처리, 안정적 데이터 무결성, 고성능 검색을 지원하는 '자동차 부품 거래 및 운영 플랫폼'의 물리적 데이터베이스 구축을 목표로 한다.

1.1 엄격한 데이터 무결성 및 정합성 확보

기존 Airtable 기반 구조는 유연하나 스키마 구조가 느슨해 데이터 불일치와 무결성 저해 위험이 존재했다, 이를 해결하기 위해 관계형 데이터베이스의 스키마 설계를 적용한다.

- **복합 비즈니스 로직의 제약조건화:** 파트너 유형에 따른 배타적 소유권(Exclusive OR) 등 기존에 애플리케이션 레벨에서 처리하던 복잡한 비즈니스 규칙을 CHECK - UNIQUE 등 DBMS 제약조건으로 강제하여 데이터 오염을 차단한다.
- **고아 레코드 방지:** Foreign Key 와 참조 무결성 옵션(Cascade, Restrict)을 명확히 설정함으로 삭제,갱신시 발생할 수 있는 데이터 불일치 문제를 구조적으로 예방한다.

1.2 고차원 벡터 데이터와 관계형 데이터의 하이브리드 설계

자연어 기반의 부품 검색 기능을 위해 정형 데이터(부품 스펙, 가격)와 비정형 데이터(부품 설명 임베딩 벡터)를 단일 DB 에서 통합 관리하는 구조를 설계한다.

- **Vector DB 통합:** PostgreSQL 의 확장 모듈인 pgvector 를 활용하여 1536 차원의 임베딩 데이터를 저장하고, 코사인 유사도 연산이 가능한 구조를 설계한다.
- **검색 최적화 전략:** 메타데이터 기반 필터(상태, 카테고리 등)과 시맨틱 검색이 결합될 수 있도록 적절한 인덱싱 전략(HNSW)전략을 수립한다.

1.3 대규모 트랜잭션 처리를 위한 확장성과 성능 최적화

다수의 파트너와 관리자가 동시에 이용하는 환경을 고려해, 안정적인 OLTP 성능과 확장성을 확보하는 방향으로 설계를 진행한다.

- **정규화-성능의 균형:** 제 3 정규형(3NF)을 준수해 중복을 제거하되, 실제 쿼리 패턴을 분석하여 B-Tree, Hash 인덱스를 적절히 배치함으로써 조인 비용과 응답 지연 시간(Latency)을 최소화한다.
- **타입 최적화:** 각 속성의 성격에 따라 VARCHAR, TEXT, TIMESTAMPTZ, DECIMAL 등 최적의 물리적 데이터 타입을 매핑하여 저장 효율성을 높이고 I/O 비용을 감소시킨다.

2. 관계형 데이터베이스 스키마 설계

본 장에서는 개념적 ER 모델을 실제 PostgreSQL 환경에 적합한 관계형 스키마로변환한 결과를 정리한다. 모든 테이블명과 컬럼명은 일관된 snake_case 규칙을 적용한다. 또한 파트너, 핵심 자산, 거래 공통 영역으로 논리적 모듈을 구분하여 관리와 확장을 용이하게 했다.

2.1 파트너 및 계약 (Partner & Contract)

비즈니스 요구사항에 따라 속성과 생명주기가 상이한 파트너를 Ambassor, Shop, Individual 세 개의 릴레이션으로 분리하였다. 이를 통해 불필요한 NULL 컬럼을 최소화하고 제 3 정규형(3NF)을 만족시켰다. 계약 정보는 하나의 계약서가 정확히 하나의 파트너 유형에만 귀속되도록 배타적 관계(Exclusive OR)를 적용한다. 주요 테이블은 다음과 같다.

- **Ambassador** (ambassador_id (PK), code, name, email, commission_rate, created_at)
- **Shop** (shop_id (PK), name, business_reg_number, main_phone, referred_by_ambassador_id (FK), created_at)
- **Individual** (individual_id (PK), name, personal_phone, email, referred_by_shop_id (FK), referred_by_ambassador_id (FK), created_at)
- **Contact** (contact_id (PK), name, email, role, phone_number, is_primary, shop_id (FK))
- **Contract** (contract_id (PK), contract_type, file_url, signed_at, expired_at, status, shop_id (FK), individual_id (FK), ambassador_id (FK))

Shop 과 Individual 은 모두 Ambassador 를 통해 유입될 수 있고, Individual 은 Shop 을 통해서도 유입될 수 있는 구조를 갖는다. Contract 테이블은 shop_id, individual_id, ambassador_id 중 정확히 하나만 값을 갖도록 설계하여, 동일 계약이 여러 파트너 유형에 중복 귀속되는 것을 방지한다.

```sql
CREATE TABLE contract (
    contract_id BIGSERIAL PRIMARY KEY,
    contract_type VARCHAR(50),
    file_url VARCHAR(255),
    signed_at TIMESTAMPTZ,
    expired_at TIMESTAMPTZ,
    status VARCHAR(20),
    shop_id BIGINT REFERENCES shop(shop_id),
    individual_id BIGINT REFERENCES individual(individual_id),
    ambassador_id BIGINT REFERENCES ambassador(ambassador_id)
);
```

```sql
CREATE TABLE ambassador (
    ambassador_id   BIGSERIAL PRIMARY KEY,
    code            VARCHAR(50),
    name            VARCHAR(50) NOT NULL,
    email           VARCHAR(100) NOT NULL,
    commission_rate DECIMAL(5, 2),
    created_at      TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE individual (
    individual_id   BIGSERIAL PRIMARY KEY,
    name            VARCHAR(50) NOT NULL,
    personal_phone  VARCHAR(20) NOT NULL,
    email           VARCHAR(100) NOT NULL,
    referred_by_shop_id BIGINT REFERENCES shop(shop_id),
    referred_by_ambassador_id BIGINT REFERENCES ambassador(ambassador_id),
    created_at      TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);
```

```sql
CREATE TABLE shop (
    shop_id         BIGSERIAL PRIMARY KEY,
    name            VARCHAR(100) NOT NULL,
    business_reg_number VARCHAR(20) NOT NULL,
    main_phone      VARCHAR(20),
    referred_by_ambassador_id BIGINT REFERENCES ambassador(ambassador_id),
    created_at      TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE contact (
    contact_id      BIGSERIAL PRIMARY KEY,
    name            VARCHAR(50) NOT NULL,
    email           VARCHAR(100),
    role            VARCHAR(50),
    phone_number    VARCHAR(20),
    is_primary      BOOLEAN DEFAULT FALSE,
    shop_id         BIGINT REFERENCES shop(shop_id)
);
```

2.2 핵심 자산 및 물류 (Assets & Logistics)

부품은 가격,상태 등의 정형 데이터와 이미지, 텍스트 임베딩 벡터와 같은 비정형 데이터가 결합된 핵심 자산이다. AI 기반 자연어 검색을 지원하기위해 pgvector 를 통해 임베딩 벡터를 저장하는 구조를 도입한다. 물류 관점에서는 '현재 상태'와 '이동 이력'을 분리하여 조회성능과 이력 추적성을 동시에 확보한다. Part 테이블은 부품의 현재 위치만 유지하고 WarehouseTransfer 테이블에서 모든 이동 경로를 관리한다. 주요 테이블은 다음과 같다.

- **Part** (part_id (PK), qr_code, name, status, description_vector, owner_shop_id (FK), owner_individual_id (FK), current_warehouse_id (FK), created_at)
- **MediaAsset** (media_id (PK), file_url, media_type, part_id (FK), uploaded_at)
- **Warehouse** (warehouse_id (PK), code, name, capacity, address_id (FK), is_active)
- **WarehouseTransfer** (transfer_id (PK), transfer_date, status, part_id (FK), from_warehouse_id (FK), to_warehouse_id (FK))

```sql
CREATE TABLE warehouse (
    warehouse_id BIGSERIAL PRIMARY KEY,
    code VARCHAR(50),
    name VARCHAR(100),
    capacity INTEGER,
    address_id BIGINT,
    is_active BOOLEAN DEFAULT TRUE
);

CREATE TABLE warehouse_transfer (
    transfer_id BIGSERIAL PRIMARY KEY,
    transfer_date TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20),
    part_id BIGINT REFERENCES part(part_id),
    from_warehouse_id BIGINT REFERENCES warehouse(warehouse_id)
    to_warehouse_id BIGINT REFERENCES warehouse(warehouse_id)
);
```

```sql
CREATE TABLE part (
    part_id BIGSERIAL PRIMARY KEY,
    qr_code VARCHAR(100),
    name VARCHAR(200) NOT NULL,
    status VARCHAR(20),
    description_vector VECTOR(1536),
    owner_shop_id BIGINT REFERENCES shop(shop_id),
    owner_individual_id BIGINT REFERENCES individual(individual_id),
    current_warehouse_id BIGINT REFERENCES warehouse(warehouse_id),
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE media_asset (
    media_id BIGSERIAL PRIMARY KEY,
    file_url VARCHAR(255) NOT NULL,
    media_type VARCHAR(20),
    part_id BIGINT REFERENCES part(part_id),
    uploaded_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);
```

Part 는 Shop 또는 Individual 중 하나만 소유자로 가질 수 있으며, WarehouseTransfer 는 from_warehouse_id, to_warehouse_id 를 통해 창고 간 이동 이력을 세밀하게 기록한다. MediaAsset 은 부품별 이미지·동영상 등의 파일 메타데이터를 관리해, Part 와의 1:N 관계를 형성한다.

2.3 거래 및 공통 (Transaction & Common)

판매와 배송은 밀접한 관계에 있으나, 처리 과정과 생명주기가 상이하므로 독립된 릴레이션으로 분리한다. 하나의 Sale 레코드가 하나의 Delivery 와만 연결되는 1:1 구조를 통해 중복 배송 기록을 방지한다. 주소는 전사 공통 마스터 데이터로 관리하며, PartnerAdress 교차 엔터티를 통해 동일 주소를 '배송지', '청구지' 등 다양한 용도로 재상용할 수 있도록 설계한다. 주요 테이블은 다음과 같다.

- **Sale** (sale_id (PK), price, sales_channel, sold_at, part_id (FK))
- **Delivery** (delivery_id (PK), tracking_number, carrier_name, status, sale_id (FK), shipping_address_id (FK))
- **Address** (address_id (PK), street_address, city, zip_code, country)
- **PartnerAddress** (partner_address_id (PK), type, address_id (FK), shop_id (FK), individual_id (FK), ambassador_id (FK))

Sale 은 개별 부품의 판매 이력을 관리하고, Delivery 는 배송사, 송장번호, 배송 상태 등을 관리한다. Address 는 한 번만 저장한 뒤 PartnerAddress 를 통해 Shop, Individual, Ambassador 와 다대다 관계로 연결할 수 있어, 주소 중복을 효과적으로 줄였다

```sql
CREATE TABLE address (
    address_id BIGSERIAL PRIMARY KEY,
    street_address VARCHAR(255) NOT NULL,
    city VARCHAR(100) NOT NULL,
    zip_code VARCHAR(20),
    country VARCHAR(50)
);
CREATE TABLE delivery (
    delivery_id BIGSERIAL PRIMARY KEY,
    tracking_number VARCHAR(100),
    carrier_name VARCHAR(50),
    status VARCHAR(20),
    sale_id BIGINT REFERENCES sale(sale_id),
    shipping_address_id BIGINT REFERENCES address(address_id)
);
```

```sql
CREATE TABLE partner_address (
    partner_address_id BIGSERIAL PRIMARY KEY,
    type VARCHAR(20),
    address_id BIGINT REFERENCES address(address_id),
    shop_id BIGINT REFERENCES shop(shop_id),
    individual_id BIGINT REFERENCES individual(individual_id),
    ambassador_id BIGINT REFERENCES ambassador(ambassador_id)
);

    CREATE TABLE sale (
        sale_id BIGSERIAL PRIMARY KEY,
        price DECIMAL(15, 2) NOT NULL,
        sales_channel VARCHAR(50),
        sold_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
        part_id BIGINT REFERENCES part(part_id)
    );
```

3. 무결성 제약조건 및 트랜잭션 제어 설계

본 장에서는 앞서 정의한 스키마에 대해 DBMS 레벨의 무결성 제약조건과, 다중 사용자 환경을 가정한 트랜잭션,동시성 제어, 성능 및 운영 전략을 정리한다.

3.1 파트너 및 계약 영역 제약조건

```sql
ALTER TABLE shop ADD CONSTRAINT uq_shop_biz_num UNIQUE (business_reg_number);
ALTER TABLE individual ADD CONSTRAINT uq_individual_email UNIQUE (email);

ALTER TABLE contract
ADD CONSTRAINT chk_contract_exclusive_owner CHECK (
    (shop_id IS NOT NULL AND individual_id IS NULL AND ambassador_id IS NULL) OR
    (shop_id IS NULL AND individual_id IS NOT NULL AND ambassador_id IS NULL) OR
    (shop_id IS NULL AND individual_id IS NULL AND ambassador_id IS NOT NULL)
);

ALTER TABLE contract
ADD CONSTRAINT chk_contract_valid_period CHECK (expired_at > signed_at);
```

a. 중복 방지 파트너의 고유 식별 정보인 사업자 등록번호와 개인 이메일에 UNIQUE 제약조건을 설정하여, 시스템 내 중복 가입을 막는다.

b. 배타적 관계 강제 (Exclusive OR) Contract 테이블은 논리적으로 단 하나의 파트너 유형과만 연결되어야 한다. 이를 위해 CHECK 제약조건을 사용하여 세 개의 외래키 컬럼 중 정확히 하나만 값을 갖고 나머지는 NULL 이 되도록 강제한다

c. 데이터 논리 검증 계약 만료일이 서명보다 과거일 수 없다는 논리적 유효성을 DB 레벨에서 보장한다.

3.2 핵심 자산 및 물류영역 제약조건

```sql
ALTER TABLE part
ADD CONSTRAINT chk_part_owner_xor CHECK (
    (owner_shop_id IS NOT NULL AND owner_individual_id IS NULL) OR
    (owner_shop_id IS NULL AND owner_individual_id IS NOT NULL)
);

ALTER TABLE part
ADD CONSTRAINT chk_part_status CHECK (
    status IN ('Draft', 'Inspection', 'Listed', 'Sold', 'Disposed')
);

ALTER TABLE media_asset
ADD CONSTRAINT fk_media_part_cascade FOREIGN KEY (part_id)
REFERENCES part(part_id) ON DELETE CASCADE;

ALTER TABLE part
ADD CONSTRAINT fk_part_warehouse_restrict FOREIGN KEY (current_warehouse_id)
REFERENCES warehouse(warehouse_id) ON DELETE RESTRICT;
```

a. 소유권 무결성 부품 역시 상점 또는 개인 중 하나의 소유주만 가져야 하므로, 계약 테이블과 동일한 XOR CHECK 제약조건을 적용한다.

b. 상태 값 통제 부품의 상태 컬럼에 오타나 정의되지 않은 값이 입력되는 것을 방지하기 위해, 허용된 5 가지 상태 값만 입력되도록 제한한다.

c. 트랜잭션 전략 창고 간 이동 시 재고의 정합성을 위해, 해당 트랜잭션이 완료될 때까지 관련 부품 레코드에 비관적 락을 걸어 동시 수정을 방지한다.

c. 참조 무결성 정책 (Cascade vs Restrict) MediaAsset: 부품이 삭제되면 해당 부품의 이미지 데이터는 불필요하므로 CASCADE 를 적용하여 자동 삭제한다. Warehouse: 창고에 보관 중인 부품이 실수로 삭제되면 재고 불일치가 발생하므로, 위치 정보가 존재하는 한 삭제할 수 없도록 RESTRICT 를 적용하여 데이터를 보호한다.

3.3 거래 및 공통 영역 제약 조건

```
ALTER TABLE sale ADD CONSTRAINT uq_sale_part_id UNIQUE (part_id);

ALTER TABLE sale
ADD CONSTRAINT fk_sale_part_protect FOREIGN KEY (part_id)
REFERENCES part(part_id) ON DELETE RESTRICT;

ALTER TABLE sale ADD CONSTRAINT chk_sale_price_positive CHECK (price >= 0);

ALTER TABLE delivery ADD CONSTRAINT uq_delivery_tracking UNIQUE (tracking_number);
```

a. 중복 판매 방지 (1:1 관계) 하나의 부품은 단 한 번만 판매될 수 있다. Sale 테이블의 part_id 에 UNIQUE 제약조건을 걸어, 이미 판매된 부품에 대해 중복 판매 데이터가 생성되는 것을 물리적으로 막는다.

b. 재무 데이터 보존 판매(Sale) 이력이 존재하는 부품은 매출 데이터로서의 가치를 지니므로, 원본 부품 데이터가 삭제되지 않도록 RESTRICT 삭제 정책을 적용한다.

c. 트랜잭션 및 동시성 제어 인기 부품에 대해 다수의 구매자가 동시에 결제를 시도하는 '경쟁 상태(Race Condition)'를 방지하기 위해, 결제 프로세스 진입 시 SELECT ... FOR UPDATE 구문을 사용하여 해당 부품을 선점하. 또한 교착 상태(Deadlock) 방지를 위해 트랜잭션 대기 시간을 제한하는 설정을 적용한다.

3.4 시스템 운영 및 트랜잭션 환경 설정

```
ALTER DATABASE kickback_db SET timezone = 'Asia/Seoul';

ALTER DATABASE kickback_db SET default_transaction_isolation = 'read committed';

ALTER DATABASE kickback_db SET lock_timeout = '10s';

ALTER DATABASE kickback_db SET idle_in_transaction_session_timeout = '300s';
```

a. 타임존 설정: 본 플랫폼은 국내 물류 및 배송을 처리하므로, 모든 TIMESTAMPTZ 데이터가 한국 표준시(Asia/Seoul) 기준으로 저장 및 조회되도록 강제하여 시간 데이터의 정합성을 확보한다.

b. 트랜잭션 격리 수준: 데이터의 일관성과 처리량 균형을 위해 PostgreSQL 표준인 READ COMMITTED 를 기본값으로 채택한다. 이는 Dirty Read 를 방지하면서도 높은 트랜잭션 처리량을 확보한다.

c. 교착 상태 방지: lock_timeout 을 10 초로 설정하여, 특정 트랜잭션이 자원을 획득하지 못한 채 무기한 대기하는 상황을 방지한다.

d. 좀비 트랜잭션 차단: idle_in_transaction_session_timeout 을 300 초(5 분)로 설정하여 사용자의 이탈 또는 애플리케이션 오류로 인해 커밋,롤백 없이 유휴 상태로 남은 세션을 종료한다.

3.5 성능 최적화를 위한 인덱스 설계

대규모 데이터 조회 및 AI 검색 성능을 보장하기 위해 다음과 같은 인덱싱 전략을 수립하고 이를 DBMS 에 반영한다.

```
CREATE INDEX idx_part_description_vector
ON part USING hnsw (description_vector vector_cosine_ops);

CREATE INDEX idx_part_owner_shop ON part(owner_shop_id);
CREATE INDEX idx_part_current_warehouse ON part(current_warehouse_id);
CREATE INDEX idx_sale_part_id ON sale(part_id);

CREATE INDEX idx_part_status ON part(status);
CREATE INDEX idx_part_created_at ON part(created_at DESC);
```

a. AI 검색 최적화: part 테이블의 description_vector 컬럼에 pgvector 의 HNSW(Hierarchical Navigable Small World) 알고리즘을 적용한다. 특히 vector_cosine_ops 옵션을 명시하여, 단순 거리가 아닌 코사인 유사도 기반의 자연어 검색 속도를 획기적으로 개선한다.

b. 조인 성능 향상: PostgreSQL 은 외래키(FK) 컬럼에 대해 자동으로 인덱스를 생성하지 않습니다. 따라서 빈번하게 조인이 발생하는 owner_shop_id, current_warehouse_id, sale.part_id 등에 수동으로 B-Tree 인덱스를 생성하여, 연관 데이터 조회 시 Full Table Scan 을 방지하고 응답 속도를 단축시킨다.

c. 조회 및 정렬 최적화: 사용자가 가장 자주 필터 조건인 '부품 상태(status)'와 '최신순 정렬(created_at DESC)'에 복합 인덱스를 적용하여, 리스트 조회 화면에서의 페이징, 정렬 기능을 최적화한다.

3.6 대용량 로그 데이터 처리를 위한 파티셔닝 전략

서비스 운영 중 발생하는 시스템 오류,감사 로그는 장기간 누적 시 수백만 건 이상으로 커질 수 있다. 이를 단일 테이블에 적재시 조회 성능 저하와 관리 부담이 커지므로, system_error_log 데이블에 날짜 기 Range Partitioning 을 적용한다.

```sql
CREATE TABLE system_error_log (
    log_id BIGSERIAL,
    error_code VARCHAR(50),
    error_message TEXT,
    stack_trace TEXT,
    created_at TIMESTAMPTZ NOT NULL,
    CONSTRAINT pk_system_error_log PRIMARY KEY (log_id, created_at)
) PARTITION BY RANGE (created_at);

CREATE INDEX idx_error_log_created_at ON system_error_log(created_at DESC);
CREATE INDEX idx_error_log_code ON system_error_log(error_code);

CREATE TABLE error_log_2025_11 PARTITION OF system_error_log
    FOR VALUES FROM ('2025-11-01 00:00:00+09') TO ('2025-12-01 00:00:00+09');

CREATE TABLE error_log_2025_12 PARTITION OF system_error_log
    FOR VALUES FROM ('2025-12-01 00:00:00+09') TO ('2026-01-01 00:00:00+09');
```

a. Range Partitioning 적용: 로그 발생 일시(created_at)를 기준으로 데이터를 월(Month) 단위 물리적 테이블로 분산 저장한다. 이를 통해 특정 기간(예: 2025 년 12 월)의 장애 내역을 조회할 때, DB 엔진이 불필요한 파티션을 스캔하지 않고 해당 월의 테이블만 탐색(Partition Pruning)하여 검색 성능을 극대화한다.

b. PK 제약조건의 특수성: 파티셔닝 테이블에서 유니크 제약조건(PK 포함)을 설정할 때는 반드시 파티션 키(Partition Key)가 포함되어야 한다는 PostgreSQL 의 제약사항을 준수하여, created_at 컬럼을 기본키 복합 구성요소로 포함한다.

c. 데이터 수명주기 관리 (ILM): 수천만 건의 데이터를 DELETE 연산으로 삭제하면 막대한 시스템 부하가 발생한다. 파티셔닝을 활용하면 보존 기한이 지난 월 단위의 파티션을 DROP 하는 방식으로, 부하 없이 데이터를 효율적으로 정리할 수 있다.

3.7 데이터 거버넌스를 위한 메타데이터 정의

개발자간의 원활한 커뮤니케이션과 데이터 명세의 모호함을 없애기 위해, COMMENT 구문을 사용하여 DB 객체 자체에 비즈니스 정의를 메타데이터로 기록한다.

```sql
COMMENT ON TABLE part IS '핵심 자산: 자동차 중고 부품 마스터 데이블';
COMMENT ON COLUMN part.status IS '생명주기 상태 (Draft: 작성중, Listed: 판매중, Sold
COMMENT ON COLUMN part.description_vector IS 'pgvector: 1536차원 임베딩 벡터 (Open

COMMENT ON TABLE contract IS '계약 관리: 파트너 유형별 배타적 관계(XOR) 적용';
COMMENT ON COLUMN sale.sales_channel IS '판매 채널 구분 (Web, App, B2B_Partner)';
```

3.8 보안 강화를 위한 사용자 및 권한 분리 전략

데이터베이스 접근 제어를 위해 애플리케이션 연결 계정(app_user)과 운영자 계정(db_admin)의 역할을 분리하고 최소 권한 원칙을 적용한다.

```sql
CREATE ROLE app_role;

GRANT CONNECT ON DATABASE kickback_db TO app_role;
GRANT USAGE ON SCHEMA public TO app_role;

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO app_role;
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO app_role;

CREATE USER kkb_service_account WITH PASSWORD 'secure_password_placeholder';
GRANT app_role TO kkb_service_account;
```

a. 역할 기반 접근 제어: 개별 사용자에게 권한을 직접 부여하는 대신, app_role 이라는 역할을 생성하여 권한을 부여한 후 길제 계정에서 역할 할당하는 구조를 사용한다. 이를 통해 애플리케이션 서버증설이나 계정이 변경시에도 역할만 재사용하면 되어 관리가 단순해진다.

b. 최소 권한 원칙: 실제 서비스 애플리케이션이 사용하는 kkb_service_account 에는 데이터를 조작(DML: SELECT, INSERT, UPDATE, DELETE)할 수 있는 권한만 부여한다. 테이블을 삭제하거나 구조를 변경하는 DDL 권한은 배제하여, SQL Injection 공격 등이 발생하더라도 데이터베이스 구조가 파괴되는 것을 방지한다.

c. 시퀀스 사용 권한: BIGSERIAL PK 를 사용하는 테이블에 대해, 시퀀스 객체에 대한 USAGE 권한을 명시적으로 부여하여 INSERT 시 권한 부족으로 인한 오류를 사전에 방지한다. 이를 통해 서비스 운영 중 예기치 않은 권한 문제로 인한 장애를 줄일 수 있다.

4. 결론

본 프로젝트는 Airtable 기반 구조를 PostgreSQL 중심의 정규화된 스키마로 전환하여 데이터의 안정성과 일관성을 확보하였다. 파트너·계약·부품 등 핵심 엔터티를 분리하고 XOR·참조 무결성·상태 제약을 적용해 데이터 품질을 구조적으로 강화하였다. pgvector 기반 임베딩 저장, 인덱스 최적화, 파티셔닝을 통해 AI 검색성과 대용량 처리 성능을 향상시켰고, 트랜잭션 제어와 락 전략을 통해 실시간 운영 환경에서도 안전한 동시성을 보장하였다.
이로써 향후 서비스 확장에도 견고하게 대응할 수 있는 데이터베이스 기반을 구축하였다,