Modernizing the Database of AI Litigation (DAIL)

SchemaForge Team

Table of contents

1. What is DAIL . . . 2
2. Legacy System and Current Limitations . . . 2
    Identified Limitations . . . 2
3. Redesign Objectives . . . 3
4. Normalization and Database Design Principles . . . 3
    First Normal Form (1NF) . . . 3
    Second Normal Form (2NF) . . . 3
    Third Normal Form (3NF) . . . 3
    Referential Integrity . . . 4
5. Final Entity Design . . . 4
5.1 Cases . . . 4
5.2 Jurisdictions . . . 5
5.3 Dockets . . . 5
5.4 Documents . . . 5
5.5 Secondary Sources . . . 6
5.6 Areas of Application . . . 6
5.7 Issues . . . 6
5.8 Causes of Action . . . 7
5.9 Algorithms . . . 7
5.10 Organizations . . . 7
6. ER Diagram . . . 8
7. ETL Process . . . 8
8. Supabase Deployment . . . 9
9. API Layer (FastAPI) . . . 9
    Architecture . . . 9
    Capabilities . . . 10
10. Render Deployment . . . 10

11. Improvements Over Legacy System 10
12. Data Integrity and Validation 11
13. Conclusion 11

1. What is DAIL

The Database of AI Litigation (DAIL) is a public-interest legal research resource that catalogs court cases involving artificial intelligence technologies. It provides structured information about litigation concerning algorithms, automated decision systems, generative AI, and related technologies.

The goal of DAIL is to support:

- Legal research
- Policy analysis
- Academic study
- Public transparency

As AI adoption expands, litigation involving AI systems has increased in volume and complexity. A scalable and structured backend is required to support long-term sustainability.

2. Legacy System and Current Limitations

The legacy system stored data across multiple Excel sheets:

- Case Table
- Docket Table
- Document Table
- Secondary Source Table

The structure was flat and denormalized. Relationships were implied through matching values rather than enforced by database constraints.

Identified Limitations

1. No enforced primary keys
2. Slug used as identifier but not structurally enforced
3. No referential integrity between sheets
4. Multi-value fields stored as comma-separated strings
5. No foreign key constraints
6. No uniqueness guarantees
7. No transactional consistency

8. No programmatic API access
9. Limited scalability
10. Manual consistency enforcement

This structure created risks of duplication, inconsistency, and long-term maintenance issues.

3. Redesign Objectives

The redesign aimed to:

1. Introduce a relational database structure
2. Enforce referential integrity
3. Normalize multi-value attributes
4. Support structured querying
5. Enable API-based access
6. Ensure long-term scalability

The system was redesigned using PostgreSQL, deployed via Supabase, and exposed through a FastAPI application deployed on Render.

4. Normalization and Database Design Principles

The redesign follows relational database theory and normalization principles.

First Normal Form (1NF)

- All fields contain atomic values.
- Multi-value text fields were extracted into separate tables.
- No repeating groups remain.

Second Normal Form (2NF)

- All non-key attributes depend entirely on the primary key.
- No partial dependencies exist.

Third Normal Form (3NF)

- No transitive dependencies.
- Lookup values stored in dedicated reference tables.

Referential Integrity

All relationships are enforced through foreign key constraints. Cascade rules ensure consistent deletion behavior.

5. Final Entity Design

Below is the explanation of each entity.

5.1 Cases

Purpose: Represents a litigation case involving AI.

Primary Key:

- case_id (SERIAL, surrogate key)

New Columns Introduced:

- case_id (not present in raw data)
- jurisdiction_id (foreign key reference)

Foreign Keys:

- jurisdiction_id → jurisdictions.jurisdiction_id

Unique Constraints:

- slug
- record_number

How Records Are Accessed:

- By slug
- By record_number
- By filters (issue, area, algorithm, jurisdiction, date)

This is the central entity of the system.

5.2 Jurisdictions

Purpose: Represents the court context in which a case is filed.

Primary Key:

- jurisdiction_id

Unique Constraint:

- (court_name, jurisdiction_type, jurisdiction_name)

Foreign Key Usage:

- Referenced by cases

This prevents duplicate jurisdiction entries.

5.3 Dockets

Purpose: Represents docket records associated with a case.

Primary Key:

- docket_id

Foreign Keys:

- case_id → cases.case_id

Cardinality: One case → many dockets

Cascade Rule: Deleting a case deletes associated dockets.

5.4 Documents

Purpose: Represents filings within a docket.

Primary Key:

- document_id

Foreign Keys:

- docket_id → dockets.docket_id

Cardinality: One docket → many documents

Cascade Rule: Deleting a docket deletes associated documents.

5.5 Secondary Sources

Purpose: Represents external coverage or research references.

Primary Key:

- source_id

Foreign Keys:

- case_id → cases.case_id

Cardinality: One case → many secondary sources

5.6 Areas of Application

Purpose: Categorizes cases by AI domain.

Primary Key:

- area_id

Unique Constraint:

- name

Relationship: Many-to-many with cases via case_areas bridge table.

5.7 Issues

Purpose: Represents legal issues involved in the case.

Primary Key:

- issue_id

Unique Constraint:

- name

Many-to-many relationship via case_issues.

5.8 Causes of Action

Purpose: Legal claims or defenses asserted.

Primary Key:

- cause_id

Unique Constraint:

- name

Many-to-many relationship via case_causes.

5.9 Algorithms

Purpose: Represents AI systems involved in litigation.

Primary Key:

- algorithm_id

Unique Constraint:

- name

Many-to-many relationship via case_algorithms.

5.10 Organizations

Purpose: Represents companies or institutions involved.

Primary Key:

- organization_id

Unique Constraint:

- name

Many-to-many relationship via case_organizations.

6. ER Diagram

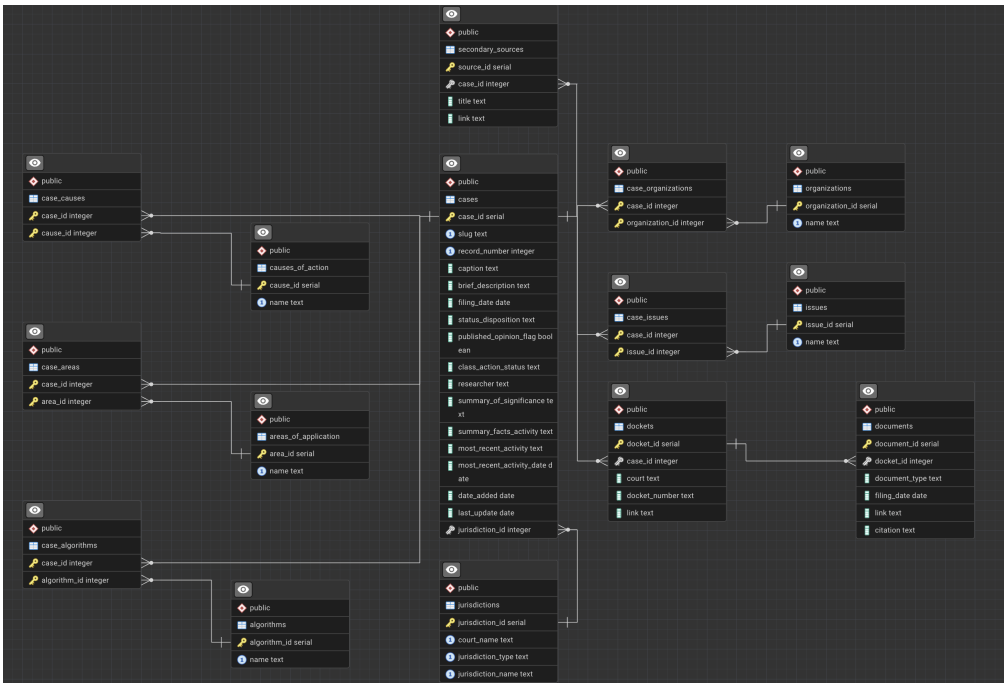

Figure 1: Entity Relationship Diagram

The diagram illustrates:

- One-to-many relationships (cases → dockets → documents)
- Many-to-many relationships via bridge tables
- Foreign key enforcement

This visualizes the normalized relational structure.

7. ETL Process

Data was migrated from Excel files using a structured ETL pipeline.

Steps:

1. Load Excel sheets using pandas
2. Clean column names
3. Convert NaT to null
4. Remove incomplete rows
5. Deduplicate values

6. Normalize multi-value fields
7. Insert base entities first
8. Insert reference tables
9. Insert bridge relationships
10. Commit transactional changes

Foreign keys were resolved dynamically during insertion.

Integrity constraints prevented duplicate or orphaned records.

8. Supabase Deployment

The PostgreSQL database is hosted on Supabase.

Advantages:

- Managed PostgreSQL
- Secure remote access
- Scalable infrastructure
- Automatic backups
- Role-based authentication

The schema was deployed directly to Supabase. Data was loaded using the ETL pipeline.

9. API Layer (FastAPI)

The API layer provides structured programmatic access.

Framework: FastAPI ORM: SQLAlchemy Validation: Pydantic Deployment: Render

Architecture

Router Layer → Handles HTTP endpoints

CRUD Layer → Business logic and queries

Database Layer → Session management

Capabilities

- Retrieve cases
- Retrieve related dockets
- Retrieve documents
- Retrieve secondary sources
- Filter by jurisdiction
- Filter by issue
- Filter by area
- Filter by algorithm
- Filter by date
- Pagination

The API automatically generates OpenAPI documentation.

10. Render Deployment

The FastAPI application is deployed on Render.

Advantages:

- Free tier hosting
- Continuous deployment from GitHub
- Public API endpoint
- Auto-scaling capability

The API communicates with Supabase using secure environment variables.

11. Improvements Over Legacy System

Legacy System:

- Flat Excel sheets
- No referential integrity
- No enforced uniqueness
- No normalization
- Manual consistency

Modernized System:

- Fully normalized relational schema
- 3NF compliance
- Enforced primary keys

- Enforced foreign keys
- Cascade rules
- Programmatic API access
- Cloud deployment
- Scalable architecture

12. Data Integrity and Validation

The final system ensures:

- Unique case slugs
- Unique record numbers
- No duplicate reference values
- No orphan foreign keys
- Many-to-many integrity via composite primary keys
- Atomic fields (1NF compliance)

The schema enforces integrity at the database level rather than relying on application logic.

13. Conclusion

The SchemaForge redesign transforms DAIL from a flat research dataset into a structured, scalable, and production-ready backend system.

The final system:

- Enforces relational integrity
- Supports complex legal research queries
- Enables API-based access
- Provides cloud deployment
- Ensures long-term maintainability

This establishes a sustainable foundation for continued expansion of AI litigation tracking.