AcademiaLens Database Schema

This document outlines the comprehensive database schema for the AcademiaLens application, including all tables, relationships, indexes, and constraints.

Database Technology

The AcademiaLens application uses PostgreSQL as its primary relational database, with Prisma as the ORM (Object-Relational Mapping) layer. The schema is defined using Prisma Schema Language.

Core Schema

```
// prisma/schema.prisma

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

// ==================== User Management ====================

model User {
  id             String    @id @default(cuid())
  name           String?
  email          String    @unique
  emailVerified  DateTime?
  image          String?
  password       String?   // Hashed password for email/password auth
  createdAt      DateTime  @default(now())
  updatedAt      DateTime  @updatedAt

  // Profile information
  title          String?
  institution    String?
  department     String?
  bio            String?
  researchInterests String[]

  // Subscription and billing
```

```
  stripeCustomerId  String?   @unique
  subscriptionId    String?
  subscriptionStatus SubscriptionStatus? @default(FREE)
  subscriptionPeriodEnd DateTime?

  // Relations
  accounts         Account[]
  sessions         Session[]
  projects         Project[]
  documents        Document[]
  analyses         Analysis[]
  glossaries       Glossary[]
  aiInteractions   AIInteraction[]
  notifications    Notification[]
  sharedProjects   ProjectMember[]

  @@index([email])
}

model Account {
  id                String  @id @default(cuid())
  userId            String
  type              String
  provider          String
  providerAccountId String
  refresh_token     String? @db.Text
  access_token      String? @db.Text
  expires_at        Int?
  token_type        String?
  scope             String?
  id_token          String? @db.Text
  session_state     String?

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([provider, providerAccountId])
  @@index([userId])
}

model Session {
  id           String   @id @default(cuid())
  sessionToken String   @unique
  userId       String
  expires      DateTime

  user         User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@index([userId])
}

model VerificationToken {
  identifier String
```

```
  token      String   @unique
  expires    DateTime

  @@unique([identifier, token])
}

enum SubscriptionStatus {
  FREE
  BASIC
  PROFESSIONAL
  ENTERPRISE
  CANCELLED
  PAST_DUE
}

// ==================== Project Management ====================

model Project {
  id          String    @id @default(cuid())
  name        String
  description String?
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt
  isArchived  Boolean   @default(false)

  // Relations
  userId      String    // Owner
  user        User      @relation(fields: [userId], references: [id], onDelete: Cascade)
  documents   Document[]
  analyses    Analysis[]
  glossaries  Glossary[]
  members     ProjectMember[]

  @@index([userId])
  @@index([isArchived])
}

model ProjectMember {
  id        String      @id @default(cuid())
  role      ProjectRole  @default(VIEWER)
  joinedAt  DateTime    @default(now())

  // Relations
  projectId String
  project   Project     @relation(fields: [projectId], references: [id], onDelete:
Cascade)
  userId    String
  user      User        @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([projectId, userId])
  @@index([projectId])
  @@index([userId])
```

```
}

enum ProjectRole {
  OWNER
  EDITOR
  VIEWER
}

// ==================== Document Management ====================

model Document {
  id            String    @id @default(cuid())
  title         String
  description   String?
  fileUrl       String    // URL to stored file
  fileType      FileType
  fileSize      Int       // Size in bytes
  pageCount     Int?      // For PDFs
  extractedText String?   @db.Text
  metadata      Json?     // Extracted metadata (authors, publication date, etc.)
  createdAt     DateTime  @default(now())
  updatedAt     DateTime  @updatedAt
  isProcessed   Boolean   @default(false)
  processingError String?

  // Relations
  userId        String
  user          User      @relation(fields: [userId], references: [id], onDelete: Cascade)
  projectId     String?
  project       Project?  @relation(fields: [projectId], references: [id], onDelete:
SetNull)
  chunks        DocumentChunk[]
  analyses      Analysis[]
  citations     Citation[]
  entities      DocumentEntity[]

  @@index([userId])
  @@index([projectId])
  @@index([fileType])
  @@index([isProcessed])
}

model DocumentChunk {
  id         String    @id @default(cuid())
  content    String    @db.Text
  chunkIndex Int
  pageNumber Int?
  embedding  Bytes?    // Vector embedding for semantic search

  // Relations
  documentId String
  document   Document  @relation(fields: [documentId], references: [id],
```

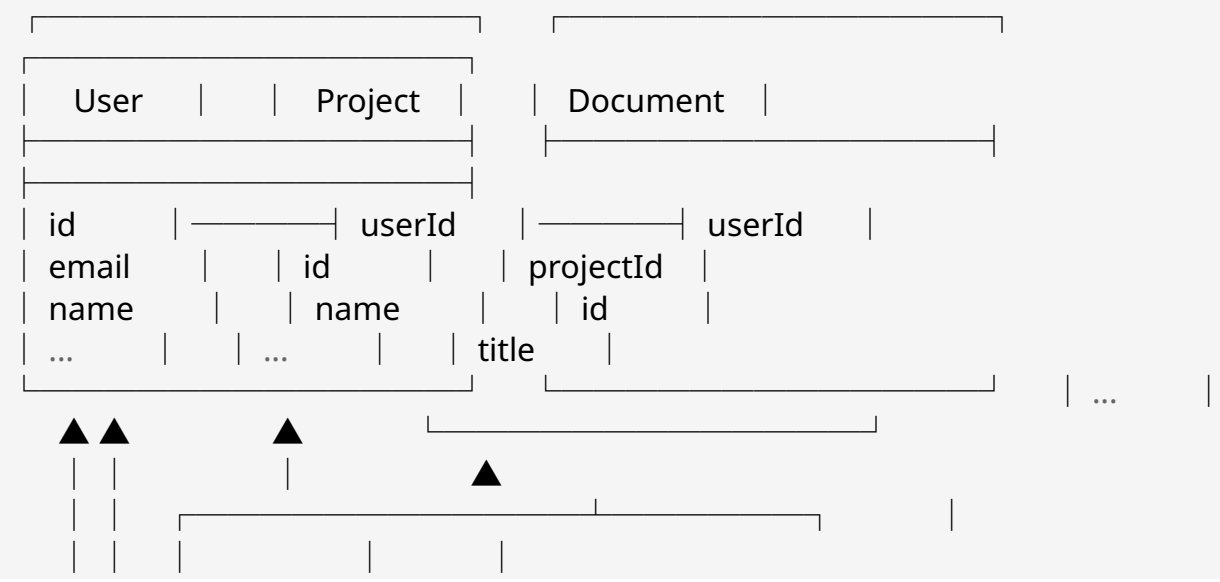
```
onDelete: Cascade)

  @@unique([documentId, chunkIndex])
  @@index([documentId])
}

model DocumentEntity {
  id          String      @id @default(cuid())
  name        String
  type        EntityType
  definition  String?     @db.Text
  occurrences Json        // Array of {pageNumber, position} objects

  // Relations
  documentId  String
  document    Document    @relation(fields: [documentId], references: [id],
onDelete: Cascade)
  glossaryId  String?
  glossary    Glossary?   @relation(fields: [glossaryId], references: [id])

  @@index([documentId])
  @@index([type])
  @@index([name])
}

enum FileType {
  PDF
  TEXT
  DOCX
  URL
  VIDEO
}

enum EntityType {
  KEYWORD
  PERSON
  ORGANIZATION
  LOCATION
  CONCEPT
  METHOD
  DATASET
  VARIABLE
  ACRONYM
  JARGON
}

// ==================== Analysis & AI Features ====================

model Analysis {
  id          String       @id @default(cuid())
  type        AnalysisType
  status      AnalysisStatus @default(PENDING)
```

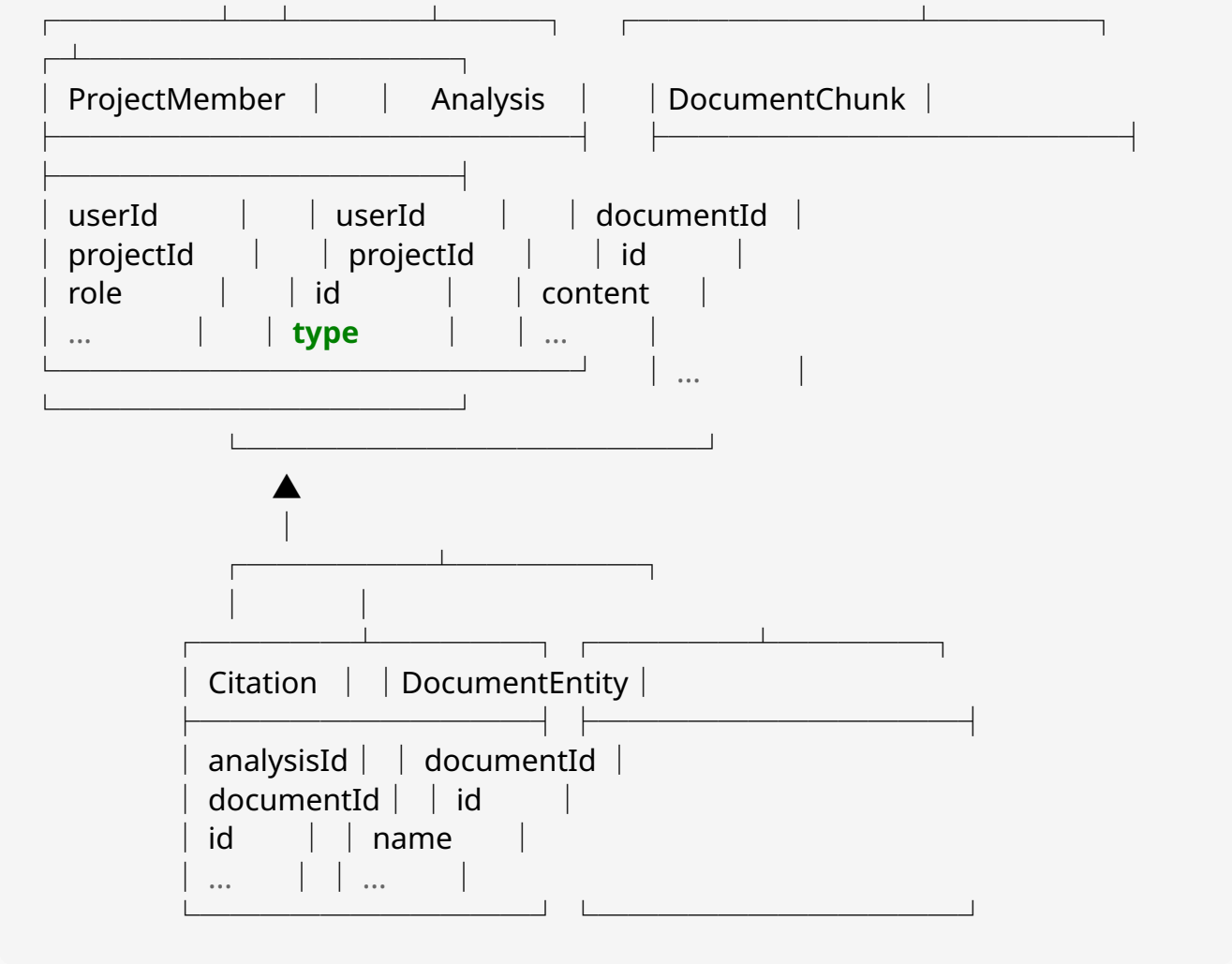
```
  parameters     Json?       // Analysis parameters
  result         Json?       // Analysis results
  createdAt      DateTime    @default(now())
  completedAt    DateTime?
  error          String?

  // Relations
  userId         String
  user           User        @relation(fields: [userId], references: [id], onDelete:
Cascade)
  projectId      String?
  project        Project?    @relation(fields: [projectId], references: [id], onDelete:
SetNull)
  documents      Document[]   // Many-to-many for cross-document analysis
  citations      Citation[]
  aiInteractions  AIInteraction[]

  @@index([userId])
  @@index([projectId])
  @@index([type])
  @@index([status])
}

model Citation {
  id             String   @id @default(cuid())
  quote          String   @db.Text
  context        String?  @db.Text
  pageNumber     Int?
  section        String?
  confidence     Float?   // Confidence score (0-1)

  // Relations
  documentId     String
  document       Document  @relation(fields: [documentId], references: [id],
onDelete: Cascade)
  analysisId     String?
  analysis       Analysis? @relation(fields: [analysisId], references: [id], onDelete:
SetNull)

  @@index([documentId])
  @@index([analysisId])
}

model AIInteraction {
  id             String   @id @default(cuid())
  prompt         String   @db.Text
  response       String   @db.Text
  tokensUsed     Int
  createdAt      DateTime  @default(now())

  // Relations
  userId         String
```

```
  user          User     @relation(fields: [userId], references: [id], onDelete: Cascade)
  analysisId    String?
  analysis      Analysis? @relation(fields: [analysisId], references: [id], onDelete:
SetNull)

  @@index([userId])
  @@index([analysisId])
}

model Glossary {
  id            String    @id @default(cuid())
  name          String
  description   String?
  isAutoUpdated Boolean   @default(true)
  createdAt     DateTime  @default(now())
  updatedAt     DateTime  @updatedAt

  // Relations
  userId        String
  user          User     @relation(fields: [userId], references: [id], onDelete: Cascade)
  projectId     String?
  project       Project?  @relation(fields: [projectId], references: [id], onDelete:
SetNull)
  entries       DocumentEntity[]

  @@index([userId])
  @@index([projectId])
}

enum AnalysisType {
  SUMMARY
  EXPLANATION
  METHODOLOGY
  CLAIMS_EVIDENCE
  CROSS_DOCUMENT
  COMPARISON
  APPLICATION
  SWOT
  ETHICAL_IMPACT
  INNOVATION
  FUTURE_RESEARCH
  QA
}

enum AnalysisStatus {
  PENDING
  PROCESSING
  COMPLETED
  FAILED
}

// ==================== Notifications & System ====================
```

```
model Notification {
  id           String          @id @default(cuid())
  type          NotificationType
  title         String
  message       String
  isRead        Boolean         @default(false)
  createdAt     DateTime        @default(now())

  // Relations
  userId        String
  user          User            @relation(fields: [userId], references: [id], onDelete:
Cascade)

  @@index([userId])
  @@index([isRead])
}

model SystemSetting {
  id           String    @id @default(cuid())
  key          String    @unique
  value        String    @db.Text
  updatedAt    DateTime  @updatedAt
}

enum NotificationType {
  DOCUMENT_PROCESSED
  ANALYSIS_COMPLETED
  PROJECT_SHARED
  SUBSCRIPTION_UPDATED
  SYSTEM_ANNOUNCEMENT
}
```

Schema Relationships Diagram





Detailed Entity Descriptions

User Management

1. **User**
2. Core user entity with authentication and profile information
3. Tracks subscription status and billing details

4. Connected to all user-created content

5. **Account**

6. OAuth provider accounts linked to a user

7. Supports multiple authentication methods per user

8. **Session**

9. Active user sessions

10. Used by NextAuth for session management

11. **VerificationToken**

12. Email verification tokens
13. Password reset tokens

Project Management

1. **Project**
2. Container for related documents and analyses
3. Supports collaboration through ProjectMember

4. Can be archived rather than deleted to preserve history

5. **ProjectMember**

6. Defines user roles within a project
7. Supports collaboration with different permission levels

Document Management

1. **Document**
2. Represents uploaded files (PDFs, text, etc.)
3. Stores metadata and processing status

4. Contains extracted text for analysis

5. **DocumentChunk**

6. Segments of document content for efficient processing
7. Stores vector embeddings for semantic search

8. Linked to original document with position information

9. **DocumentEntity**

10. Named entities extracted from documents
11. Includes keywords, concepts, methods, etc.
12. Tracks occurrences within documents

Analysis & AI Features

1. **Analysis**
2. Represents an AI analysis operation
3. Can be applied to one or multiple documents

4. Stores parameters and results of the analysis

5. **Citation**

6. Extracted quotes and references from documents
7. Links to source documents with page/section information

8. Used to provide verifiable sources for AI outputs

9. **AIInteraction**

10. Records of user interactions with AI
11. Tracks token usage for billing and optimization

12. Preserves prompt-response pairs for reference

13. **Glossary**

14. Collection of terms and definitions
15. Can be project-specific or user-specific
16. Can be automatically updated from document entities

Notifications & System

1. **Notification**
2. User notifications for system events

3. Tracks read status and creation time

4. **SystemSetting**

5. Global application settings
6. Configurable parameters for system operation

Indexes and Performance Optimization

The schema includes strategic indexes to optimize query performance:

1. **Primary Lookups**
2. Indexes on all foreign keys (userId, projectId, documentId, etc.)

3. Unique constraints where appropriate (email, provider+providerAccountId)

4. **Filtering Indexes**

5. Status fields (isProcessed, isArchived, isRead)

6. Type fields (fileType, entityType, analysisType)

7. **Search Optimization**

8. Text search on document content via DocumentChunk

9. Entity name indexing for quick lookup

10. **Relationship Optimization**

11. Composite indexes for many-to-many relationships
12. Cascading deletes where appropriate

Vector Search Implementation

For semantic search capabilities, the schema includes vector embeddings:

1. **Document Chunks**
2. Each chunk stores a vector embedding (Bytes field)

3. PostgreSQL with pgvector extension provides vector similarity search

4. **Implementation Notes**

5. Embeddings are generated during document processing
6. Vector similarity search is used for semantic retrieval
7. Chunk size is optimized for context preservation and query performance

Data Migration Strategy

The Prisma schema supports a structured migration approach:

1. **Initial Schema**
2. Core tables for MVP functionality

3. Basic relationships and indexes

4. **Incremental Migrations**

5. Add new features through schema migrations

6. Preserve existing data during schema evolution

7. **Migration Scripts**

8. Generated automatically by Prisma
9. Version controlled in the repository

Example Queries

User Authentication

```
// Find user by email
const user = await prisma.user.findUnique({
  where: { email: 'user@example.com' },
  include: { accounts: true }
});

// Create new user
const newUser = await prisma.user.create({
  data: {
    email: 'newuser@example.com',
    name: 'New User',
    password: hashedPassword
  }
});
```

Document Management

```
// Upload new document
const document = await prisma.document.create({
  data: {
    title: 'Research Paper',
    description: 'Important research findings',
    fileUrl: 'https://storage.example.com/documents/paper.pdf',
    fileType: 'PDF',
    fileSize: 1024000,
    userId: currentUser.id,
    projectId: projectId
  }
});

// Get all documents for a project
const projectDocuments = await prisma.document.findMany({
  where: { projectId: projectId },
  orderBy: { createdAt: 'desc' }
});

// Get document with chunks
const documentWithChunks = await prisma.document.findUnique({
  where: { id: documentId },
  include: { chunks: true }
});
```

Analysis Operations

```
// Create new analysis
const analysis = await prisma.analysis.create({
  data: {
    type: 'SUMMARY',
    userId: currentUser.id,
    projectId: projectId,
    documents: {
      connect: { id: documentId }
    },
    parameters: {
      length: 'medium',
      focus: 'methodology'
    }
  }
});

// Update analysis with results
const updatedAnalysis = await prisma.analysis.update({
  where: { id: analysisId },
  data: {
    status: 'COMPLETED',
    completedAt: new Date(),
    result: analysisResults,
    citations: {
      create: [
        {
          quote: 'Important finding from the paper',
          pageNumber: 42,
          section: 'Results',
          confidence: 0.95,
          documentId: documentId
        }
      ]
    }
  }
});

// Get all analyses for a user
const userAnalyses = await prisma.analysis.findMany({
  where: { userId: currentUser.id },
  include: {
    documents: true,
    citations: true
  },
  orderBy: { createdAt: 'desc' }
});
```

Project Collaboration

```
 // Share project with another user
const projectMember = await prisma.projectMember.create({
 data: {
  projectId: projectId,
  userId: collaboratorId,
  role: 'EDITOR'
 }
});

// Get all projects shared with user
const sharedProjects = await prisma.project.findMany({
 where: {
  members: {
   some: {
    userId: currentUser.id
   }
  }
 },
 include: {
  user: true, // Owner
  members: {
   include: {
    user: true
   }
  }
 }
});
```

Schema Evolution Considerations

As the application evolves, the schema will need to adapt. Key considerations include:

1. **Backward Compatibility**
2. Add nullable fields for new features
3. Use default values for new required fields

4. Maintain existing relationships

5. **Performance Scaling**

6. Monitor query performance as data grows
7. Add indexes for common query patterns

8. Consider table partitioning for large tables

9. **Feature Extensions**

10. Plan for additional analysis types
11. Support for more document formats
12. Enhanced collaboration features

Conclusion

This database schema provides a comprehensive foundation for the AcademiaLens application, supporting all core features while maintaining flexibility for future expansion. The schema is designed with performance, scalability, and data integrity in mind, using Prisma's powerful ORM capabilities to simplify database interactions.

The relationships between entities are carefully structured to support the complex interactions between users, documents, analyses, and collaborative features, while maintaining clear boundaries and efficient query patterns.