# Data Model Suggestions

> **Source File:** generated-documents\technical-analysis\data-model-suggestions.md
> **Generated:** 16/07/2025 at 14:00:08
> **Generated by:** Requirements Gathering Agent - PDF Converter

# Data Model Suggestions

**Generated by adpa-enterprise-framework-automation v3.2.0**
**Category:** technical-analysis
**Generated:** 2025-07-14T21:24:46.974Z
**Description:** Database architecture and data model recommendations

---

# Data Model Suggestions for ADPA (Advanced Document Processing & Automation Framework)

---

## 1. Entity Relationship Analysis

### 1.1 Core Business Entities

Based on the requirements, the following primary entities are identified:

- **User**: End users including admins, analysts, project managers, stakeholders, etc.
- **Team**: Groups of users collaborating on projects.
- **Project**: High-level workspaces for managing document generation, templates, and workflows.
- **Document**: Generated outputs (PDF, Markdown, etc.) and associated metadata.
- **Template**: Document templates compliant with BABOK, PMBOK, DMBOK, etc.
- **Framework**: Supported standards (BABOK v3, PMBOK 7, DMBOK 2.0).
- **AI Provider**: Registered AI backends (OpenAI, Google AI, etc.).
- **Integration**: External system connections (Confluence, SharePoint, Adobe, etc.).
- **Job**: Asynchronous document generation or conversion operations.
- **Role**: Permission grouping for RBAC.
- **Permission**: Granular access rights.
- **Audit Log**: Security and compliance event tracking.

## 1.2 Entity Relationships & Cardinality

- **User** ↔ **Team**: Many-to-many (users can belong to multiple teams; teams have multiple users).
- **Team** ↔ **Project**: One-to-many (a team owns many projects; a project belongs to one team).
- **User** ↔ **Project**: Many-to-many (for sharing or access control beyond teams).
- **Project** ↔ **Document**: One-to-many (project contains documents).
- **Document** ↔ **Template**: Many-to-one (each document is generated from a template).
- **Template** ↔ **Framework**: Many-to-one (templates are linked to a standard).
- **Project** ↔ **Integration**: Many-to-many (projects may have multiple integrations).
- **User** ↔ **Role**: Many-to-many (users can have multiple roles).
- **Role** ↔ **Permission**: Many-to-many (RBAC).

- **Document** ↔ **Job**: One-to-many (a document may have multiple jobs for generation/updates).
- **AI Provider** ↔ **Job**: One-to-many (track which AI provider handled which job).
- **User** ↔ **Audit Log**: One-to-many (user triggers logs).

## 1.3 Key Attributes

#### User

- id (PK, UUID)
- username (unique)
- email (unique)
- hashed_password
- status (active, suspended, etc.)
- created_at, updated_at

#### Team

- id (PK, UUID)
- name
- description
- created_at, updated_at

#### Project

- id (PK, UUID)
- team_id (FK)
- name
- description
- status (active, archived)
- created_at, updated_at

#### Document

- id (PK, UUID)
- project_id (FK)

- template_id (FK)
- name
- format (pdf, docx, md, etc.)
- content_url / file_path
- version
- status (draft, published, archived)
- created_at, updated_at

**Template**

- id (PK, UUID)
- framework_id (FK)
- name
- description
- category
- content (JSON/Markdown/HTML)
- variables (JSON)
- created_by (FK User)
- created_at, updated_at

**Framework**

- id (PK, UUID)
- code (babok, pmbok, dmbok)
- name
- description
- version

**AI Provider**

- id (PK, UUID)
- name
- provider_type (openai, google, etc.)
- config_json
- status (active, deprecated)
- created_at, updated_at

**Integration**

- id (PK, UUID)
- type (confluence, sharepoint, adobe, vcs, etc.)
- config_json
- status
- created_at, updated_at

**Job**

- id (PK, UUID)
- document_id (FK)
- ai_provider_id (FK)
- type (generation, conversion, etc.)
- status (pending, running, completed, failed)
- result_url
- started_at, completed_at

**Role**

- id (PK, UUID)
- name
- description

**Permission**

- id (PK, UUID)
- resource
- action

**Audit Log**

- id (PK, UUID)
- user_id (FK)
- action
- resource
- resource_id
- timestamp

- details (JSON)

### 1.4 Primary & Foreign Keys

- PK: All `id` fields (UUID).
- FK: As indicated above (e.g., `project.team_id` → `team.id`).

---

## 2. Data Model Recommendations

### 2.1 Logical Data Model Structure

- Use a **relational model** for core business entities (users, teams, projects, documents, templates, frameworks, RBAC).
- Store large document/template content as external objects (object storage or BLOB columns), reference via URLs/file paths.
- Store AI provider and integration configurations as JSON/BLOB columns for flexibility.
- Use **junction tables** for:
  - user_team (user_id, team_id, role_id, joined_at)
  - user_role (user_id, role_id)
  - role_permission (role_id, permission_id)
  - project_integration (project_id, integration_id)
  - user_project (user_id, project_id, role_id)

### 2.2 Physical Implementation Considerations

- Prefer **UUIDs** for primary keys for distributed scalability and security.
- For high concurrency (collaboration, real-time editing), consider optimistic locking/versioning on Document.
- For audit and compliance, dedicate a write-optimized log table (partitioned by date).
- Use **PostgreSQL** or **Azure SQL** for core data; consider **NoSQL** (Cosmos DB, MongoDB) for audit logs or versioned document content if needed.

## 2.3 Normalization Recommendations

- Normalize up to 3NF for core operational tables (avoid redundant data, simplify updates).
- Denormalize for read-heavy analytics/reporting tables as appropriate (materialized views for dashboards).
- Store large/unstructured template content as external BLOB (S3, Azure Blob) and reference via URL if size becomes an issue.

## 2.4 Index Strategy Suggestions

- Unique indexes on all natural keys: username, email, team name, project name (scoped to team), template name (scoped to framework).
- Composite indexes for frequent queries:
  - (project_id, status) on Document
  - (team_id, status) on Project
  - (user_id, team_id) on user_team
  - (role_id, permission_id) on role_permission
  - (document_id, status) on Job
- GIN/JSONB indexes for querying into config_json or variables columns.

---

# 3. Database Design Guidelines

## 3.1 Table Structure Recommendations

- Use **singular, lower_snake_case** table names: user, team, project, document, template, framework, ai_provider, integration, job, role, permission, audit_log.
- Use **junction tables** for all many-to-many relationships, include audit fields (created_at, updated_at).
- For versioned documents/templates, add: version (integer), previous_version_id (FK).

## 3.2 Data Type Selections

- UUIDs for all PKs and FKs.
- VarChar(255) for names, emails, and slugs.
- Text/BLOB for large content (documents, templates).
- JSONB for config and variable fields.
- Enum types for status fields (where supported).
- Timestamps (with timezone) for all created_at, updated_at, etc.

## 3.3 Constraint Definitions

- **NOT NULL** on all required fields.
- **UNIQUE** on all natural keys (username, email, etc.).
- **FOREIGN KEY** constraints with ON DELETE CASCADE/SET NULL as appropriate.
- **CHECK** constraints for enums (status, type).
- **Default values** for status, created_at, updated_at.

## 3.4 Performance Optimization Strategies

- Partition large tables (audit_log, job) by date or project for fast querying and archival.
- Use read replicas for analytics/reporting workloads.
- Implement connection pooling.
- Periodically archive or purge historical document/job/audit data.
- Use caching (Redis) for frequently accessed metadata (frameworks, templates).
- Consider database-level encryption for sensitive columns (PII, API keys).

---

# 4. Data Governance Framework

## 4.1 Data Quality Standards

- Enforce schema validation at the API and DB layer.

- Require all core fields (names, emails, document/template metadata) to be populated.
- Use referential integrity for all FKs.
- Automate duplicate detection for templates, projects, documents.

## 4.2 Data Validation Rules

- Validate email format, password strength, and unique usernames on user creation.
- Enforce allowed file formats during document upload/generation.
- Check that referenced templates/frameworks exist before document generation.
- Validate JSON config fields using JSON schema validation libraries (AJV, Joi).

## 4.3 Data Security Considerations

- Use salted hash (bcrypt) for password storage.
- Store API tokens/credentials encrypted at rest.
- Apply RBAC at the database and application layers.
- Log all access to sensitive data in the audit log.
- Implement row-level security (PostgreSQL RLS) for multitenancy if needed.
- Use database encryption (TDE) for persistent storage.
- Mask PII in logs and exports.

## 4.4 Backup & Recovery Strategies

- Daily full backups, hourly incrementals for core DB.
- Store backups in geographically redundant storage (Azure, AWS).
- Point-in-time recovery enabled for production environments.
- Regular test restores to validate backup integrity.
- Retention policy: 30 days standard, 7 years for audit/compliance logs.
- Store versioned documents/templates for rollback.

---

## Summary Table: Entity Overview

| Entity | Description | PK | Key Relationships |
|---|---|---|---|
| User | System user | id (UUID) | Team (M:M), Project (M:M), Role (M:M) |
| Team | Group of users | id (UUID) | User (M:M), Project (1:M) |
| Project | Workspace for documents/templates | id (UUID) | Team (M:1), User (M:M), Document (1:M) |
| Document | Generated output | id (UUID) | Project (M:1), Template (M:1), Job (1:M) |
| Template | Document blueprint | id (UUID) | Framework (M:1), Document (1:M) |
| Framework | Standard (BABOK, PMBOK, etc.) | id (UUID) | Template (1:M) |
| AI Provider | AI backend configuration | id (UUID) | Job (1:M) |
| Integration | External system connection | id (UUID) | Project (M:M) |

| Entity | Description | PK | Key Relationships |
|---|---|---|---|
| Job | Processing task (generation, conversion) | id (UUID) | Document (M:1), AI Provider (M:1) |
| Role | RBAC grouping | id (UUID) | User (M:M), Permission (M:M) |
| Permission | Access right on resource | id (UUID) | Role (M:M) |
| Audit Log | Security & compliance event | id (UUID) | User (M:1) |

---

## Closing Notes

- This model provides a scalable, extensible foundation for ADPA's document processing, automation, and enterprise integration needs.
- It aligns with industry best practices for security, compliance, and maintainability.
- The design supports both immediate project requirements and long-term enterprise growth (multi-tenancy, high concurrency, auditability, and integration extensibility).

---

**Next Steps:**

- Review entity relationships with business/product owners.
- Prioritize implementation of core tables and RBAC.
- Prototype schema using PostgreSQL or Azure SQL.
- Validate with integration and API use cases.

- Establish data validation and governance processes early.

---

**End of Data Model Suggestions**