Database Schemas - Intelligent CV Analysis and Candidate Database System

This document describes the database schemas used in the application, which utilizes both SQL (here I use MariaDB) and vector databases (ChromaDB) for comprehensive CV storage and retrieval.

SQL Database Schemas (SQLModel)

- The ORM I used is SQLModel which is the most used in FastAPI and is based in SQLAlchemy and Pydantic.
- The SQL database uses three main tables defined in `schema/Application.py`:

1. Application Table

The main table storing applicant information:

Field	Type	Description	Constraints
id	Integer	Primary key	Auto-generated
vectorDbUuid	String	UUID linking to vector database document in Chroma	Required
name	String	Applicant's name	Required, Indexed
email	String	Email address	Optional, Indexed
phone	String	Phone number	Optional, Indexed
linkedIn	String	LinkedIn profile URL	Optional, Indexed
gitRepo	String	Git repository URL of the candidate (Github, GitLab, Bitbucket,...)	Optional, Indexed
yearsOfExperience	Integer	Total years of experience	Optional
lastUpdated	DateTime	Timestamp of last update	Auto-generated

Relationships: One-to-many with Education and ExperiencedSkill tables

2. Education Table

Stores educational background:

Field	Type	Description	Constraints
id	Integer	Primary key	Auto-generated
application_id	Integer	Foreign key to Application table	Required
degree	String	Degree obtained	Required, Indexed
institution	String	Educational institution	Required, Indexed
year	String	Graduation year or period	Optional, Indexed
gpa	Float	Grade Point Average	Optional

3. ExperiencedSkill Table

Stores skills with experience levels:

Field	Type	Description	Constraints
id	Integer	Primary key	Auto-generated
application_id	Integer	Foreign key to Application table	Required
skill	String	Skill name with experience	Required, Indexed
yearsOfExperience	Integer	Years of experience in that skill (will be rounded to the nearest 0.5 year)	Optional

Vector Database Schema

The vector database uses `Document` objects from LangChain for storing embeddings:

Document Structure

- **page_content**: Text content containing skills and experience information (skills, experiences, projects are stored in the same document)
- **id**: Unique identifier (matches vectorDbUuid from Application table)

Vector Store Implementation

- **Database**: ChromaDB (with Langchain integration)
- **Embedding Model**: HuggingFace sentence-transformers/all-MiniLM-L6-v2
- **Purpose**: Semantic search and similarity matching

Database Controllers

The application uses two specialized query controllers:

1. SqlQueryObject (`database/SqlQuery.py`)

- Handles CRUD operations for SQL database

- Manages relationships between tables
- Provides structured data queries

2. VectorQueryObject (`database/VectorQuery.py`)

- Handles vector similarity search operations (to search), search documents
- Performs semantic matching of CV content
- Supports natural language queries.

Both controllers inherit from `BaseQueryObject` (`database/BaseQuery.py`) which defines the common interface for adding/updating/deleting records in the database.

Database Integration

The `DBController` (`controller/DBController.py`) class coordinates between both databases:

Key Responsibilities

- **Data Conversion**: Converts parsed CV data into appropriate schemas.
- **Referential Integrity**: Maintains links between SQL and vector databases via UUID
- **Unified Operations**: Provides single interface for CRUD operations across both storage systems, and synchronize between the

Data Flow

1. CV data is parsed and extracted
2. Structured data is stored in SQL database (candidate personal information, their education and insitutions, and skills with real world experiences are stored in Application, Education, ExperiencedSkill tables respectively).
3. Detailed work experience (company, position, duration, job description, achievements), skills, projects participated in, certifications are concatenated, embedded and stored in vector database.
4. UUID links maintain relationship between both representations, and will be updated when adding, updating or deleting candidates' CVs.

Schema Design Benefits

- **Structured Queries**: SQL database enables precise filtering and sorting for information that needs exact matching (like personal information.)
- **Semantic Search**: Vector database enables natural language and similarity-based queries for complex information like detail work experiences and projects.
- **Scalability**: Optimized storage for different query patterns and storing candidate information efficiently.

File Locations

- SQL Schemas: `schema/Application.py`
- SQL Query Controller: `database/SqlQuery.py`
- Vector Query Controller: `database/VectorQuery.py`
- Base Query Interface: `database/BaseQuery.py`

- Database Controller: `controller/DBController.py`
- Vector Database Initiation: `database/VectorDB.py`