Group 10: Job Application Management System

Contributors

Ameya Mahendra Naik
ameya.naik@utdallas.edu AMN240001

Chinmyee Kiran Gurav
chinmyee.gurav@utdallas.edu CKG240000

Jay Pradeep Tarde
jaypradeep.tarde@utdallas.edu JXT230069

Megha Manoj Nair
meghamanoj.nair@utdallas.edu MXN230084

Urvi Chalodiya
urvi.chalodiya@utdallas.edu UXC240001

Course Information

Course Title: BUAN 6320.006 Database Foundations for Business Analytics

Instructor: Dr.Lidong Wu

Date: December 8, 2024

Contents

Chapter 1

Scope and Functional Requirements

1.1 Scope

The platform bridges the gap between job seekers and hiring managers by providing:

- **Tools for application tracking:** Ensures users can monitor the status of their job applications in real-time.
- **Personalized job recommendations:** Matches job seekers with roles aligned to their skills and preferences.
- **Streamlined recruitment workflows:** Allows employers to manage postings, applications, and analytics efficiently.
- **Role-specific access:**
  - **Admin role:** Full control over job postings, applications, and analytics for performance insights, powered by MongoDB.
  - **User role:** Ability to view and apply for job postings, managed using MySQL (phpMyAdmin).

By addressing challenges such as tracking applications, verifying job listings, and organizing job seeker data, JobConnect ensures an effective and efficient hiring process for all stakeholders.

1.2 Functional Requirements

The JobConnect system includes the following functional requirements:

1. **Login Authentication:**
   - Two distinct user roles:
     - **Admin:** Accesses job postings, applications, and analytics, with data stored in MongoDB.
     - **User:** Views and applies for jobs, with data stored in MySQL (phpMyAdmin).
   - Secure login system with role-based access management.
2. **Database Management:**
   - **Admin Operations (MongoDB):**
     - All admin-related data, including job postings, application insights, and analytics, is stored and managed in MongoDB.
     - MongoDB integration ensures efficient handling of unstructured and semi-structured data.

- The system uses **MyPHPmongo** as an interface for MongoDB to manage admin-related operations.

- **User Operations (MySQL via phpMyAdmin):**
  - All user-related data, such as job applications and user profiles, is stored and managed in MySQL (phpMyAdmin).
  - Tabs such as "Jobs" and "Applications" dynamically fetch and display user-specific data from the SQL database.

3. **Job Postings Tab (Admin Only):**
   - Admins can post new jobs, edit or delete existing jobs, and view all job postings. All changes are stored in MongoDB.

4. **Application Submission (User Access):**
   - Users can view job postings and submit applications, with data stored in MySQL.
   - Application data is linked to specific jobs for admin review.

5. **Integration of SQL and NoSQL Databases:**
   - MongoDB is used for admin-related operations.
   - MySQL (phpMyAdmin) is used for user-related operations.
   - MyPHPmongo serves as the interface for MongoDB, while phpMyAdmin is used for SQL database interactions.

6. **Frontend-Backend Architecture:**
   - The system is built using React.js for the user interface and Node.js for server-side operations.
   - The UI integrates SQL and NoSQL databases for efficient data handling and scalability.

1.3 Additional Notes

- **User Role:**
  - Can only view and apply for jobs.
  - All user data and operations are handled by MySQL (phpMyAdmin).

- **Admin Role:**
  - Has full access to jobs, applications, and analytics tabs.
  - Admin-related data and operations are stored in MongoDB and managed through MyPHPmongo.

- **Project Summary:** JobConnect is a robust, role-based system that leverages the strengths of both SQL and NoSQL databases, providing efficient data handling and scalability for the hiring process.

Chapter 2

Information and Tools Needs

2.1 Data and Information Needs

To develop the JobConnect platform, the following data and information are essential:

- **Job Postings:** Includes metadata such as title, description, location, salary, job type, and expiration date.
- **User Profiles:** Contains user-specific data, including skills, qualifications, preferences, and personal details.
- **Application Data:** Tracks the status of job applications, linking users to job postings for seamless monitoring.
- **Admin Data:** Stores admin-specific operations, such as job postings, application insights, and analytics.

2.2 Tools and Software Used

The tools and software utilized in this project are as follows:

- **MongoDB:** A scalable NoSQL database used for managing admin-related data, including job postings, analytics, and application insights.
- **MySQL (phpMyAdmin):** A relational database used for handling user-related data, such as profiles and application tracking.
- **MyPHPmongo:** A user-friendly interface for managing MongoDB operations, specifically for admin-related tasks.
- **React.js:** A front-end library used to create a dynamic and responsive user interface for JobConnect.
- **Node.js:** A JavaScript runtime used to handle server-side logic and integrate MongoDB and MySQL databases.
- **XAMPP:** A local environment for testing and deploying the MySQL database for user-related operations.
- **SQL Workbench:** A tool used to manage and interact with the MySQL database for structured data.
- **Draw.io:** For creating diagrams such as ER diagrams and workflows for database design.
- **Microsoft Excel:** Used for analyzing and visualizing data during project development.
- **Generatedata.com:** A platform for generating realistic dummy data to populate the databases during testing.

Chapter 3

Implementation

3.1 Data Normalization and Cleaning

To ensure data integrity and eliminate redundancy, the raw data was normalized and cleaned. The following steps outline the process:

3.1.1 Functional Dependencies Identification

The raw data provided in the form of Excel sheets (e.g., job postings, job categories, and applications) was analyzed to identify functional dependencies. For example:

- A **job_id** uniquely determines the job title, salary range, job type, and associated category.
- An **application_id** uniquely determines the applicant's name, job applied for, and application status.
- A **category_id** uniquely determines the category name (e.g., IT, Engineering).

3.1.2 Normalization Process

The schemas were normalized to Third Normal Form (3NF) to eliminate redundancy and improve data integrity. The normalization process involved:

1. **First Normal Form (1NF):**
   - Ensured that all attributes contained atomic values.
   - Example: Split multi-valued attributes like skills into separate rows or tables.
2. **Second Normal Form (2NF):**
   - Removed partial dependencies by creating separate tables for attributes dependent on only part of a composite primary key.
   - Example: Separated job categories into a `job_categories` table linked by `category_id`.
3. **Third Normal Form (3NF):**
   - Removed transitive dependencies by ensuring non-key attributes were dependent only on the primary key.
   - Example: Moved location and company details to separate tables linked by 'location_id' and 'company_id'.

3.1.3 Data Cleaning

The Excel sheets were cleaned before importing into the database:

- Removed duplicate rows and corrected inconsistencies in job titles and categories.
- Standardized data formats (e.g., dates in 'YYYY-MM-DD' format, consistent salary ranges).
- Ensured missing values were either replaced with default values or handled appropriately (e.g., NULL values for optional fields).
- Verified data types to match database schema requirements (e.g., integers for IDs, strings for names, etc.).

3.2 Database Implementation

3.2.1 SQL Database Implementation (MySQL)

Using the normalized schema, relational tables were implemented in MySQL (phpMyAdmin). The following steps were followed:

1. **Schema Creation:** SQL scripts were written to create normalized tables, such as:
   - `job_postings`: Contains fields like `job_id`, `title`, `salary_range`, `job_type`, `category_id`, `location_id`, etc.
   - `applications`: Contains fields like `application_id`, `job_id`, `applicant_name`, `status`, etc.
   - `job_categories`: Contains fields like `category_id`, `category_name`.

   Example SQL script for table creation:

```
CREATE TABLE job_postings (
    job_id INT PRIMARY KEY,
    title VARCHAR(255),
    salary_range VARCHAR(50),
    job_type VARCHAR(50),
    category_id INT,
    location_id INT,
    FOREIGN KEY (category_id) REFERENCES job_categories(category_id)
);
```

2. **Data Insertion:** Data from cleaned Excel sheets was imported into the database using SQL scripts or the phpMyAdmin import functionality.
3. **Database Operations:** SQL queries were written to demonstrate functionality, such as:
   - Retrieve all job postings in a specific category.
   - Track the status of applications by `job_id`.

3.2.2 NoSQL Database Implementation (MongoDB)

Parts of the refined model suited for unstructured data were implemented in MongoDB:

- Selected relational schemas, such as `job_postings`, were converted into MongoDB collections.
- MongoDB scripts were written to create collections and insert sample data.

  Example MongoDB script for collection creation and data insertion:

```
db.job_postings.insertOne({
    job_id: 1,
    title: red"redDatared redScientistred",
    salary_range: red"red$70kred-red$120kred",
    job_type: red"redFullred-redTimered",
    category_id: 101,
    location_id: 201
});
```

- MongoDB operations demonstrated functionality such as:
  - Querying job postings by category or location.
  - Aggregating data for analytics (e.g., total job postings by category).

3.2.3 Hybrid Integration of SQL and NoSQL Databases

To leverage the strengths of both database types:

- **SQL (MySQL):** Used for user-related operations such as application tracking and profile management.
- **NoSQL (MongoDB):** Used for admin-related operations, such as managing job postings and analytics.
- Data synchronization was achieved by using MyPHPmongo for MongoDB and phpMyAdmin for MySQL.

3.3 Data Population

- Sufficient sample data was populated in both databases to demonstrate functionality.
- Excel sheets, such as `job_postings_clean.csv` and `applications_clean.csv`, were imported directly into MySQL.
- Equivalent sample data was manually inserted into MongoDB collections to match SQL functionality.

Chapter 4

ER/EER Diagram

The entity-relationship diagram (ERD) for JobConnect is shown below:

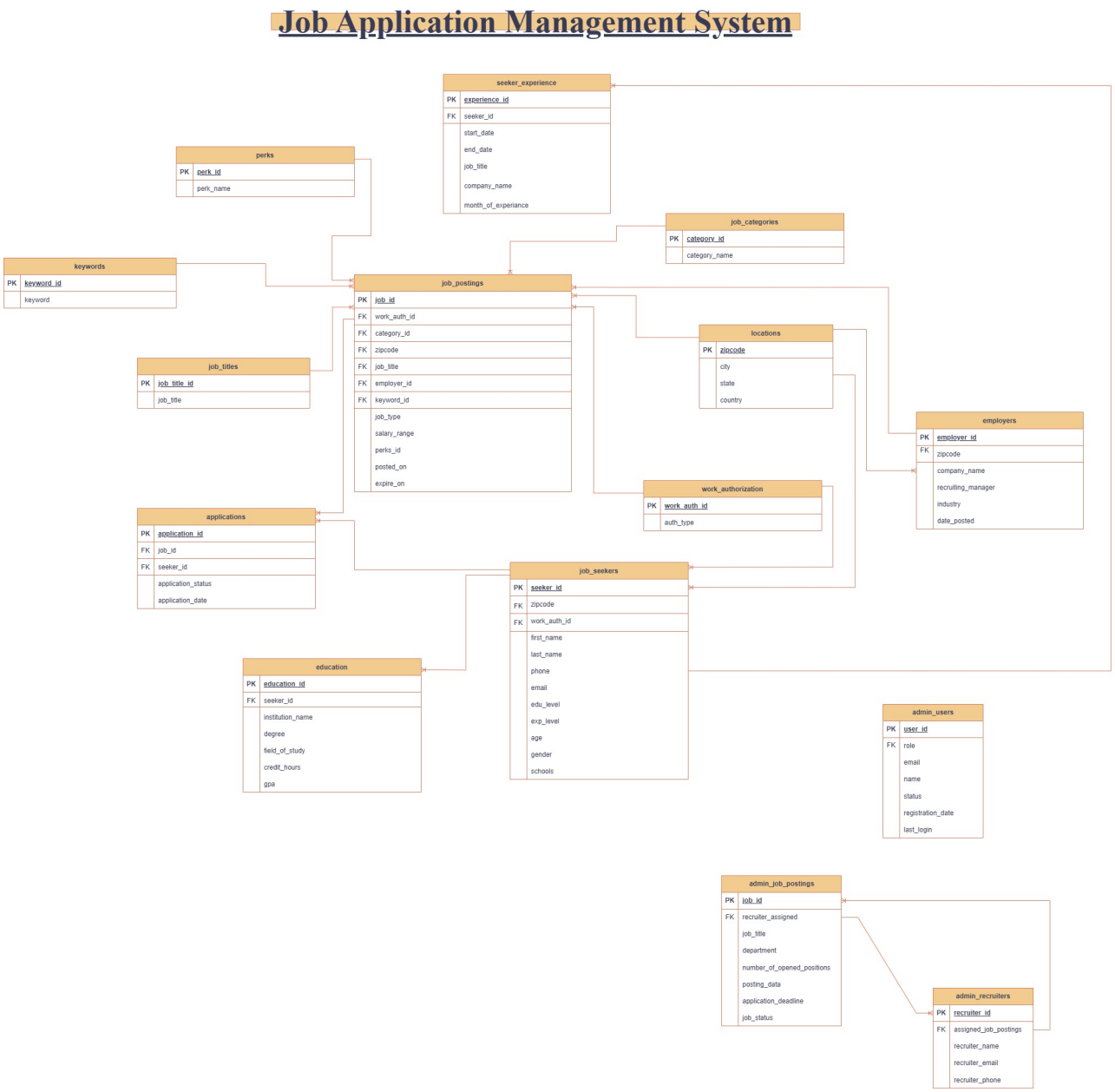

Figure 4.1: ER Diagram for JobConnect

Chapter 5

ER Diagram Explanation

5.1 Explanation of the ER Diagram

The ER Diagram for the Job Application Management System models the entities, attributes, and relationships necessary for managing job postings, applications, and administrative operations efficiently. Below is a detailed explanation of the primary entities and their relationships:

5.1.1 Entities and Attributes

- **Job_Postings:**
  - Represents jobs created by employers.
  - Attributes:
    - `job_id` (Primary Key): Unique identifier for each job posting.
    - `title`: The title of the job.
    - `salary_range`: Salary offered for the job.
    - `job_type`: Specifies if the job is full-time, part-time, or contract.
    - `category_id`: Foreign key referencing job categories.
    - `location_id`: Foreign key referencing job location.
- **Job_Seekers:**
  - Represents individuals applying for job postings.
  - Attributes:
    - `seeker_id` (Primary Key): Unique identifier for each job seeker.
    - `first_name`, `last_name`: The name of the seeker.
    - `email`: Contact email of the seeker.
    - `edu_level`: Educational level of the seeker.
    - `experience_years`: Total years of experience.
- **Applications:**
  - Tracks job applications submitted by job seekers.
  - Attributes:
    - `application_id` (Primary Key): Unique identifier for each application.
    - `job_id`: Foreign key referencing the job applied for.
    - `seeker_id`: Foreign key referencing the job seeker.
    - `application_status`: Tracks the status of the application (e.g., pending, accepted, rejected).
- **Employers:**

  - Represents companies or individuals posting jobs.
  - Attributes:
    * `employer_id` (Primary Key): Unique identifier for each employer.
    * `company_name`: Name of the employer.
    * `industry`: Industry to which the company belongs.
    * `date_posted`: Date the job was posted.

- **Job_Categories:**
  - Represents job categories such as IT, Finance, etc.
  - Attributes:
    * `category_id` (Primary Key): Unique identifier for each category.
    * `category_name`: Name of the category (e.g., IT, Engineering).

- **Admin_Users:**
  - Represents administrators managing the system.
  - Attributes:
    * `user_id` (Primary Key): Unique identifier for each admin user.
    * `email`: Admin email for login.
    * `role`: Admin role in the system.
    * `last_login`: Timestamp of the last login.

- **Locations:**
  - Represents geographical locations associated with job postings.
  - Attributes:
    * `zipcode` (Primary Key): Unique identifier for the location.
    * `city`, `state`, `country`: Details about the location.

5.1.2 Relationships

- **Job_Postings** is linked to:
  - **Job_Categories** through `category_id`.
  - **Locations** through `location_id`.
  - **Employers** through `employer_id`.
- **Job_Seekers** is related to:
  - **Applications** through `seeker_id`.
  - **Education** (not detailed here) to capture their educational qualifications.
- Admin entities (**Admin_Users** and related tables) track system-wide operations and job postings.

5.2 Summary

The ER Diagram effectively represents the data flow and relationships in the system, enabling seamless interaction between job seekers, employers, and administrators. By ensuring data normalization and clear relationships, the system ensures scalability, integrity, and efficient management of the recruitment process.

Chapter 6

Relational Schema and Explanation

6.1 Relational Schema for Admin Users

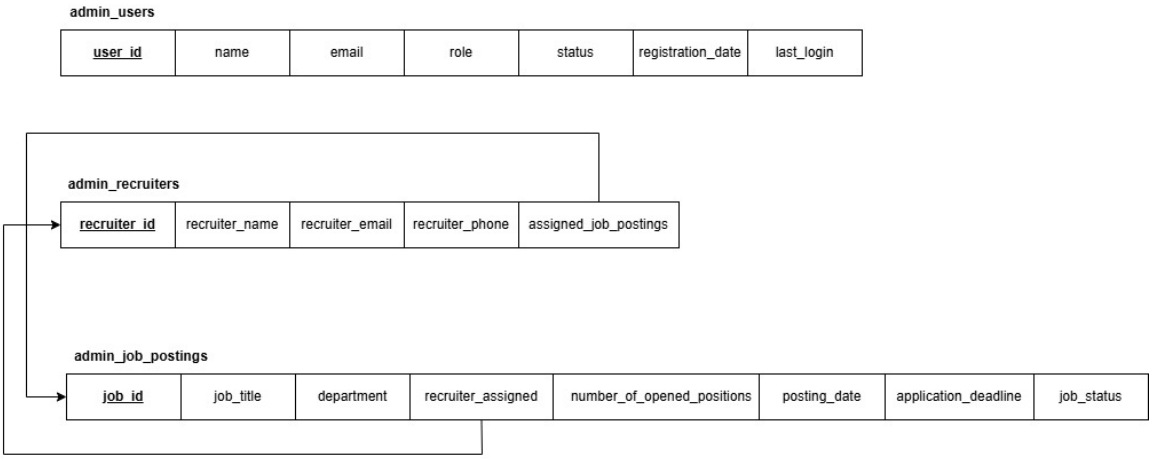

Figure 6.1: Relational Schema for Admin Users

6.1.1 Relational Schema for Admin Users (Figure 6.1)

- **Admin_Users:**
  - Represents administrators managing the platform.
  - Attributes:
    - `user_id` (Primary Key): Unique identifier for each admin user.
    - `name`, `email`, `role`: Admin details.
    - `registration_date`: The date of account creation.
    - `last_login`: Timestamp of the last login.
- **Admin_Recruiters:**
  - Represents recruiters assigned to manage specific job postings.
  - Attributes:
    - `recruiter_id` (Primary Key): Unique identifier for recruiters.
    - `recruiter_name`, `recruiter_email`, `recruiter_phone`.
    - `assigned_job_postings`: Foreign key linking to `Admin_Job_Postings`.
- **Admin_Job_Postings:**
  - Stores information about job postings created by admins.

– Attributes:

* `job_id` (Primary Key): Unique identifier for each job posting.
* `job_title`, `department`, `recruiter_assigned`.
* `number_of_opened_positions`, `posting_date`, `application_deadline`, `job_status`.

6.2 Relational Schema for Job Seekers and Applications

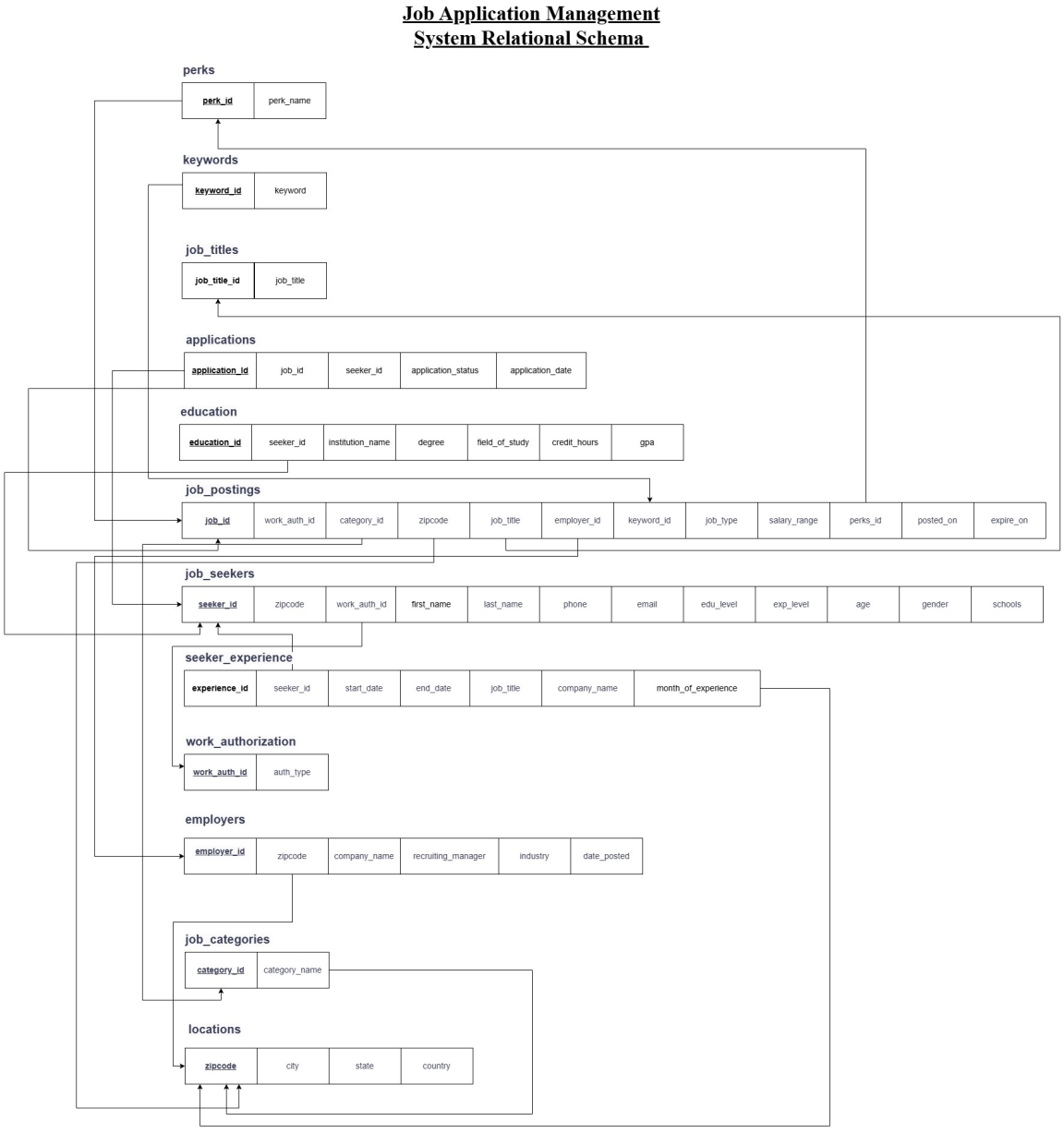

Figure 6.2: Relational Schema for Job Seekers and Applications

6.3 Explanation of Relational Schemas

The relational schemas provided above outline the structure and relationships between various entities in the Job Application Management System.

6.3.1 Relational Schema for Job Seekers and Applications (Figure 6.2)

- **Job_Seekers:**
  - Represents individuals applying for jobs.
  - Attributes:
    - `seeker_id` (Primary Key): Unique identifier for each job seeker.
    - `first_name`, `last_name`, `email`, `phone`, `edu_level`, `age`, `gender`.
- **Applications:**
  - Tracks applications submitted by job seekers.
  - Attributes:
    - `application_id` (Primary Key): Unique identifier for each application.
    - `job_id`: Foreign key referencing `Job_Postings`.
    - `seeker_id`: Foreign key referencing `Job_Seekers`.
    - `application_status`, `application_date`.
- **Job_Postings:**
  - Stores details of job openings.
  - Attributes:
    - `job_id` (Primary Key): Unique identifier.
    - `title`, `salary_range`, `job_type`, `category_id`, `location_id`.

6.4 Summary

The relational schemas ensure efficient organization and management of data in the Job Application Management System. These schemas establish clear relationships between entities while maintaining data integrity and normalization.

Chapter 7

User Interface (Admin Login)

This chapter provides an overview of the User Interface (UI) for the admin login of the JobConnect platform. The admin interface is designed to provide complete control over job postings, applications, analytics, and other administrative operations. The UI is developed using React.js for the front-end and Node.js for the backend, ensuring a dynamic and responsive experience.

7.1 Dashboard

The dashboard serves as the central hub for admins, providing key statistics and insights into the system's activities. It includes the following features:

- Total Applications: Displays the number of applications submitted by users.
- Available Jobs: Shows the number of job postings currently active.
- Pending Applications: Highlights applications that are yet to be reviewed.
- Completed Applications: Displays the number of successfully processed applications.

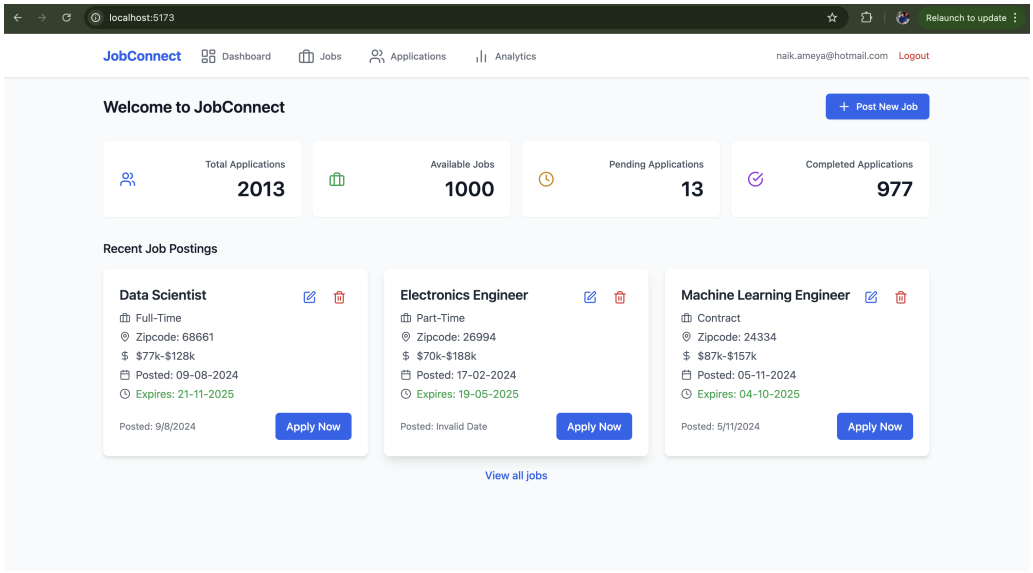

Figure 7.1: Admin Dashboard Interface

7.2 Jobs Page

The Jobs page allows admins to manage job postings efficiently. It provides the following functionalities:

- Post New Jobs: Admins can create new job postings by filling in relevant details.
- Edit/Delete Jobs: Admins have the flexibility to modify or remove job postings.
- View Job Listings: A comprehensive view of all active and inactive job postings is available.

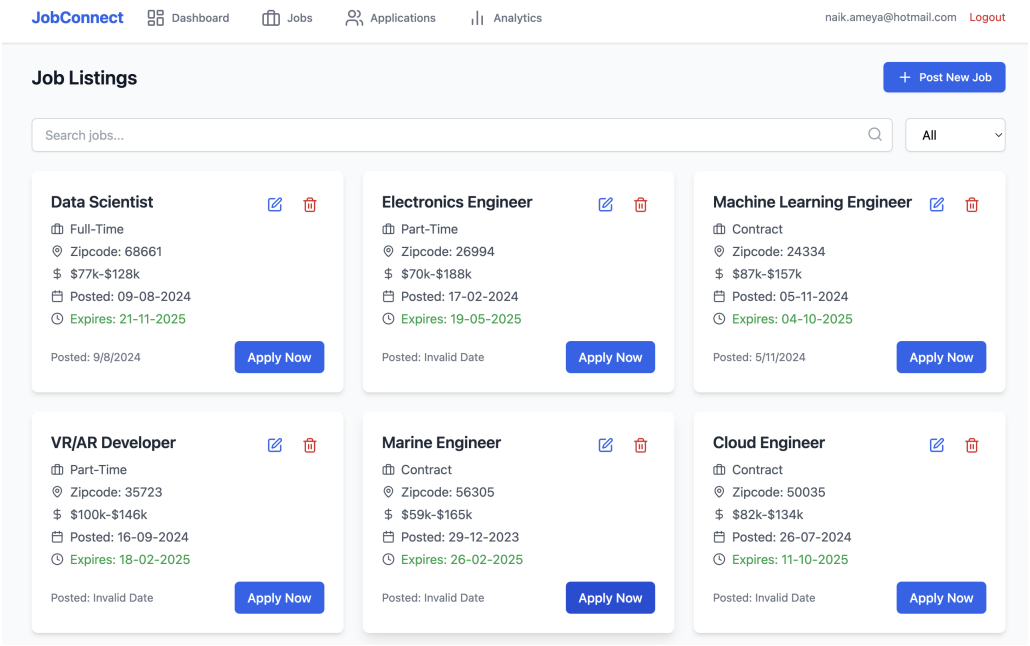

Figure 7.2: Admin Jobs Page Interface

7.3 Applications Page

The Applications page is designed to allow admins to manage applications submitted by job seekers. Key functionalities include:

- View Applications: Admins can view applications for specific jobs.
- Filter/Search: Admins can search for applications using keywords or filters.
- Application Status Management: Update the status of each application (e.g., Pending, Reviewed, Accepted, or Rejected).

7.4 Analytics Dashboard

The Analytics Dashboard provides insights into the platform's performance. Features include:

- Total Jobs Posted: Displays the cumulative count of jobs posted by the admin.
- Total Applications: Tracks the number of applications submitted across all jobs.
- Success Rate: Calculates the percentage of successfully processed applications.
- Active Locations: Displays the number of unique job locations.
- Job Categories and Types: Provides analytics for job categories and job types such as Full-Time, Part-Time, and Contract positions.

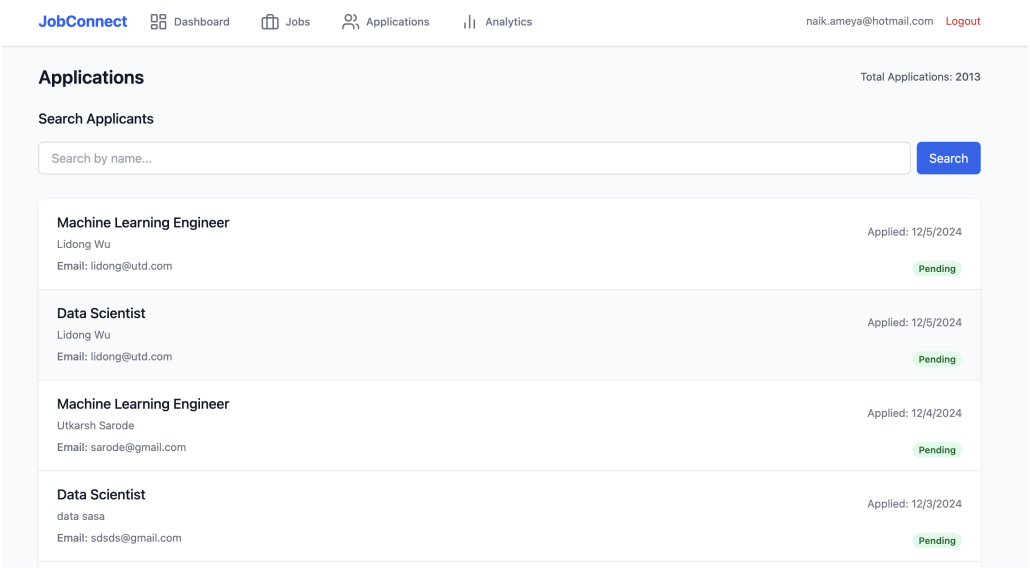

Figure 7.3: Admin Applications Page Interface

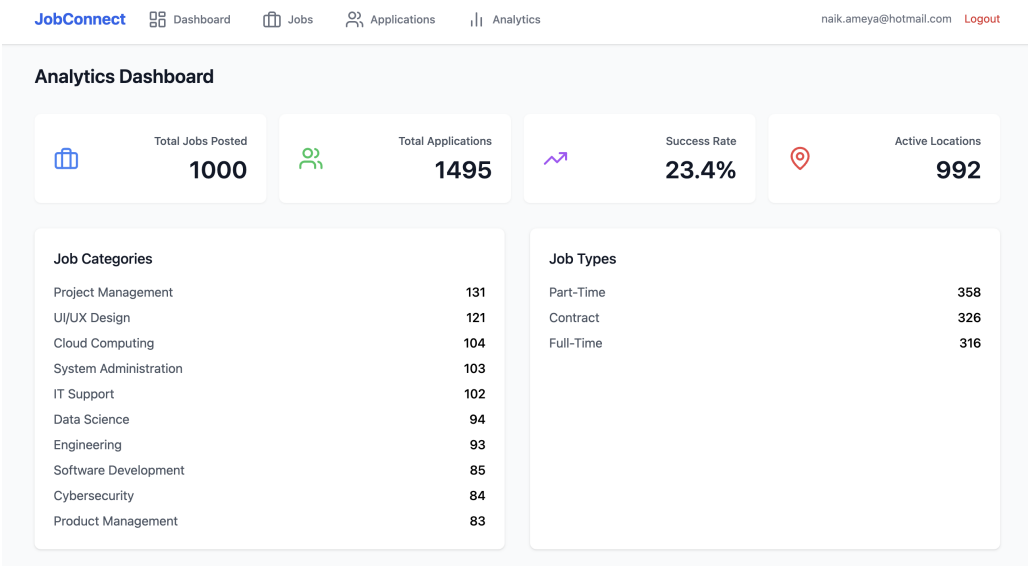

Figure 7.4: Admin Analytics Dashboard Interface

7.5 Summary

The admin UI is a critical component of the JobConnect platform, designed to streamline administrative tasks and provide comprehensive insights into job and application data. By integrating SQL and NoSQL databases, the UI ensures seamless management of structured and unstructured data while delivering an intuitive and user-friendly interface.

Chapter 8

Conclusion

JobConnect is a comprehensive and modern solution designed to address the inefficiencies and challenges in the recruitment process. By bridging the gap between job seekers and hiring managers, the platform simplifies the job application journey through its innovative features, intuitive design, and robust functionality. It is a scalable, efficient, and user-friendly system that empowers both candidates and employers by creating a collaborative and transparent environment.

The development of JobConnect included the integration of advanced technologies, such as SQL and NoSQL databases, to manage structured and unstructured data efficiently. This hybrid approach allowed the system to cater to diverse user requirements, with MongoDB handling admin-specific operations and MySQL (phpMyAdmin) managing user-related tasks. The use of React.js and Node.js further ensured a dynamic and responsive user interface while maintaining seamless communication with the back-end systems.

The platform's core features, such as real-time application tracking, personalized job recommendations, and role-based access management, offer tailored experiences for both users and administrators. The analytics dashboard provides actionable insights into job postings, applications, and success rates, equipping hiring managers with data-driven decision-making capabilities. Similarly, job seekers benefit from an organized and efficient job search process, eliminating the traditional pain points associated with job hunting.

The project emphasized best practices in data normalization and cleaning to ensure data integrity and eliminate redundancy. The relational schema and entity-relationship modeling provided a solid foundation for database design, ensuring scalability and maintainability. Comprehensive scripts and workflows were created for database creation, data population, and operational testing, highlighting the platform's technical rigor.

In conclusion, JobConnect represents a significant advancement in job application management by fostering collaboration, enhancing communication, and streamlining recruitment workflows. The project demonstrates the practical application of database foundations, software development, and data analytics in creating a scalable and impactful solution. With its versatile features and robust architecture, JobConnect positions itself as a valuable tool for addressing current and future challenges in the job market, making the hiring process more effective and efficient for all stakeholders.