

# CS 411 STAGE 2: Conceptual and Logical Database Design

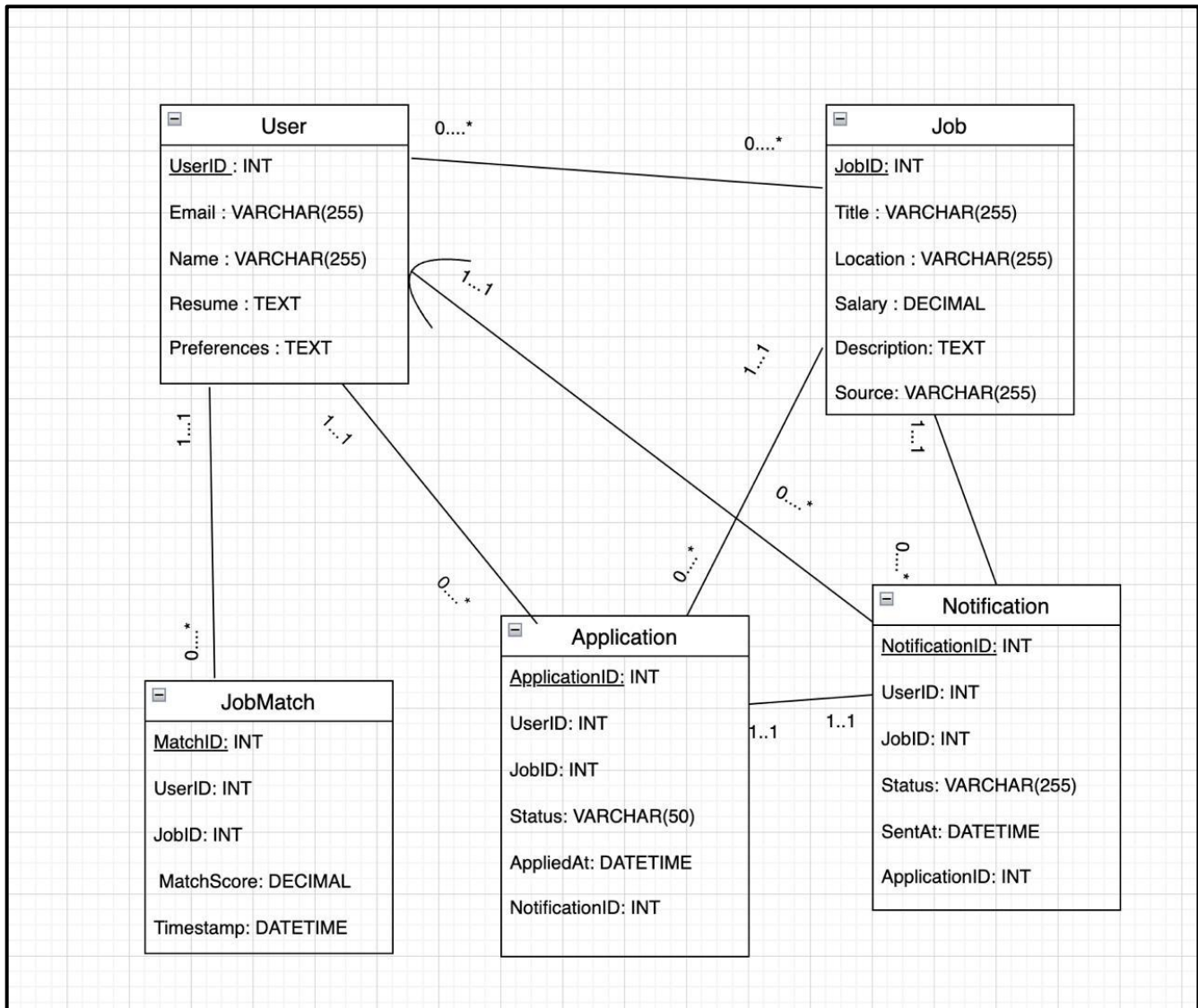
## AI-Powered Job Recommendation and Notification System (Job-Genie)

Devansh Agarwal, Aaryan Gusain, Nakul Kuppu

### Introduction

This document presents the conceptual and logical database design for our project, Job-Genie, an AI-powered job recommendation and notification system. The design follows the required ER/UML model, ensuring that our schema adheres to normalization principles (3NF) for efficiency and correctness.

### Conceptual Design (ER/UML Diagram)



# **Assumptions and Relationship Justifications**

Each entity is designed based on our system's requirements:

## **1. User**

- Stores essential user details, resumes, and preferences.
- A user can receive multiple job matches, notifications, and applications, but each job match, application, or notification belongs to only one user.
- The relation between User and Job is many-to-many.
- 1 User → Many JobMatches, Many Applications, Many Notifications, Many Jobs → Many Users

## **2. Job**

- Represents unique job postings from multiple sources.
- The relation between User and Job is many-to-many.
- A job can be matched to multiple users and receive multiple applications, but each job match or application is associated with a single job.
- 1 Job → Many JobMatches, Many Applications, Many Jobs → Many Users

## **3. JobMatch**

- Tracks AI-generated job recommendations.
- Stores the match score (how well a job fits a user's profile).
- A user can be matched with multiple jobs, and a job can be matched to multiple users.
- JobMatch is not related to Application, as it comes from the similarity score for the user's resume and not from applying.
- Many Users ↔ Many Jobs

## **4. Application**

- Tracks applications submitted by users.
- Applications are separate from job matches (not all matches turn into applications).
- A user can apply to multiple jobs, and each job can have multiple applicants.
- A Notification will be sent when an application is completed
- Many Users ↔ Many Jobs, 1 Application → 1 notification

## 5. Notification

- Stores alerts sent to users for job recommendations.
- Tracks which jobs were notified to the user and the timestamp.
- A user can receive multiple notifications, and each notification is linked to one job.
- A Notification will be sent when an application is completed
- 1 User → Many Notifications and 1 Job → Many Notifications, 1 Application → 1 notification

## **Logical Database Design (Relational Schema)**

Below is the relational schema derived from our ER diagram:

User(UserID: INT [PK], Email: VARCHAR(255), Name: VARCHAR(255), Resume: TEXT, Preferences: TEXT)

Job(JobID: INT [PK], Title: VARCHAR(255), Company: VARCHAR(255), Location: VARCHAR(255), Salary: DECIMAL, Description: TEXT, Source: VARCHAR(255))

JobMatch(MatchID: INT [PK], UserID: INT [FK to User.UserID], JobID: INT [FK to Job.JobID], MatchScore: DECIMAL, Timestamp: DATETIME)

Notification(NotificationID: INT [PK], UserID: INT [FK to User.UserID], JobID: INT [FK to Job.JobID], ApplicationID: INT [FK to Application.ApplicationID] Status: VARCHAR(50), SentAt: DATETIME)

Application(ApplicationID: INT [PK], UserID: INT [FK to User.UserID], JobID: INT [FK to Job.JobID], NotificationID: INT [FK to Notification.NotificationID] Status: VARCHAR(50), AppliedAt: DATETIME)

UserJob(UserID : INT [PK] [FK to User.UserID] JobID : INT [PK] [FK to Job.JobID])

# **SQL BASED DATABASE SCHEMA**

Below is the database schema:

## **CREATE TABLE User (**

UserID INT PRIMARY KEY AUTO\_INCREMENT,

Email VARCHAR(255) UNIQUE NOT NULL,

Name VARCHAR(255) NOT NULL,

Resume TEXT,

Preferences TEXT

);

## **CREATE TABLE Job (**

JobID INT PRIMARY KEY AUTO\_INCREMENT,

Title VARCHAR(255) NOT NULL,

Company VARCHAR(255),

Location VARCHAR(255),

Salary DECIMAL(10,2),

Description TEXT,

Source VARCHAR(255)

);

## **CREATE TABLE JobMatch (**

MatchID INT PRIMARY KEY AUTO\_INCREMENT,

UserID INT NOT NULL,

JobID INT NOT NULL,

MatchScore DECIMAL(5,2) NOT NULL,

Timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE,

FOREIGN KEY (JobID) REFERENCES Job(JobID) ON DELETE CASCADE

);

**CREATE TABLE Application (**

ApplicationID INT PRIMARY KEY AUTO\_INCREMENT,

UserID INT NOT NULL,

JobID INT NOT NULL,

NotificationID INT NOT NULL

Status VARCHAR(50) NOT NULL CHECK (Status IN ('Pending', 'Accepted', 'Rejected')),

AppliedAt DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE,

FOREIGN KEY (JobID) REFERENCES Job(JobID) ON DELETE CASCADE

FOREIGN KEY (NotificationID) REFERENCES Notification(NotificationID) ON DELETE CASCADE

);

**CREATE TABLE Notification (**

NotificationID INT PRIMARY KEY AUTO\_INCREMENT,

UserID INT NOT NULL,

JobID INT NOT NULL,

ApplicationID INT NOT NULL

Status VARCHAR(255) NOT NULL,

SentAt DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE,

FOREIGN KEY (JobID) REFERENCES Job(JobID) ON DELETE CASCADE

FOREIGN KEY (ApplicationID) REFERENCES Application(ApplicationID) ON DELETE CASCADE

);

**CREATE TABLE UserJob (**

UserID INT,

JobID INT,

PRIMARY KEY (UserID, JobID),

FOREIGN KEY (UserID) REFERENCES User(UserID) on DELETE CASCADE

FOREIGN KEY (JobID) REFERENCES Job(JobID) on DELETE CASCADE

);

## **Normalization (3NF)**

Our schema follows Third Normal Form (3NF):

### **1. No Partial Dependencies**

- All attributes depend on the entire primary key.
- Example: MatchScore in JobMatch depends on both UserID and JobID, ensuring no partial dependency.

### **2. No Transitive Dependencies**

- Each attribute is fully dependent on the primary key and not on any other non-key attribute.
- Example: Title, Location, and Salary in Job are all dependent only on JobID, not on each other.

### **3. Separation of Concerns**

- JobMatch is separate from Application because matching is an AI-generated recommendation, while applications are initiated by the user.
- Notification is stored separately to track sent alerts without redundancy.

Thus, our schema is fully normalized to 3NF, preventing data anomalies while maintaining efficiency and reducing redundancy.