Artjoms Kucajevs: 22385231

Databases assignment 1:

The database I created contains 7 tables, 4 as per brief(hospital_details, candidate_details, position_details, interview_details).The other 3 consist of skill tables, one for the skills required for a given position (position_skills), one for the skills of given candidates(candidate_skills) and one to link the skillIDs in both tables to its names and potentially other details(skill_details).

Breakdown of each table:

hospital_details

- **hospitalID** INT PRIMARY KEY
- **hospitalName** VARCHAR(255)
- **address** VARCHAR(255)
- **telephoneNum** VARCHAR(8)

candidate_details

- **candidateID** INT PRIMARY KEY
- **firstname** VARCHAR(255)
- **surname** VARCHAR(255)
- **address** VARCHAR(255)
- **telephoneNum** VARCHAR(8)

skill_details

- **skillID** INT PRIMARY KEY
- **skillName** VARCHAR(45)

position_details

- **positionID** INT PRIMARY KEY
- **hospitalID** INT
- **positionName** VARCHAR(255)
- FOREIGN KEY (**hospitalID**) REFERENCES **hospital_details** (**hospitalID**)

  *(connects tables **position_details** with **hospital_details** using foreign key, allowing to see what hospital the position is offered in)*

candidate_skills

- **candidateID** IN

- **skillID** INT
- PRIMARY KEY (**candidateID, skillID**)
- FOREIGN KEY (**candidateID**) REFERENCES **candidate_details** (**candidateID**)
- FOREIGN KEY (**skillID**) REFERENCES **skill_details** (**skillID**)

  (Here I have both int values forming a composite primary key, allowing to have a single candidate with multiple skills and or multiple candidates with the same skill)

  (foreign key links allows to see the details of specific candidates/skills to be found from ID attributes in the table)

position_skills

- **positionID** INT
- **skillID** INT
- PRIMARY KEY (**positionID**, **skillID**)
- FOREIGN KEY (**positionID**) REFERENCES **position_details** (**positionID**)
- FOREIGN KEY (**skillID**) REFERENCES **skill_details** (**skillID**)

  (Here I have both int values forming a composite primary key, allowing to have a single position needing multiple skills and or multiple positions needing the same skill)
  (foreign keys allow to see position/skill details, as only IDs shown in table)

interview_details

- **interviewID** INT PRIMARY KEY UNIQUE
- **candidateID** INT
- **positionID** INT
- **date** DATE
- **jobOffered** TINYINT
- FOREIGN KEY (**candidateID**) REFERENCES **candidate_details** (**candidateID**)
- FOREIGN KEY (**positionID**) REFERENCES **position_details** (**positionID**)
- FOREIGN KEY (**hospitalID**) REFERENCES **hospital_details** (**hospitalID**)

  (foreign keys allow to see the details of: the candidate being interviewed, the position the interview is for and the hospital in which the interview will take place)

  *(I also had a choice whether to make **candidateID, positionID, hospitalID** primary keys creating a single composite primary key, but I found this doesn't add any benefit as **interviewID** already uniquely identifies each row)*

• **A discussion of any assumptions or additions you made.**

Assumptions:

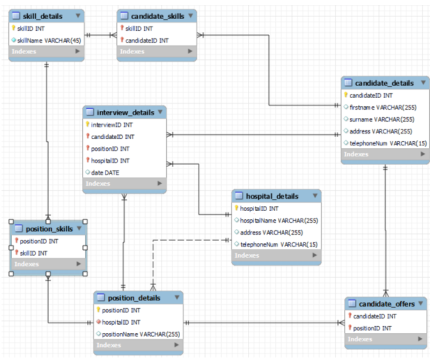
Relationships: The schema assumes certain relationships between tables, like the association of skills with candidates and positions. These relationships are essential for maintaining data integrity.

Data Types: The data types chosen for columns (such as VARCHAR for names and addresses) are assumed to be suitable for the expected data length. Adjustments might be necessary based on real-world data requirements.

Unique Identifiers: Each table has a primary key, assuming uniqueness for IDs. Ensuring uniqueness is crucial for accurate data retrieval and updates.

Additions:

-I removed the **candidate_offers** table (SEE PREVIOUS EER DIAGRAM BELOW) and added **jobOffered** as a(TINYINT) parameter in **interview_details** instead for clarity. Though a TINYINT can hold a byte of memory I created a trigger in interview_details before insertion, and before updating not allowing any values other than 1 or 0 to be entered in that field(mimicking a Boolean true(1)/false(0)).



The hospitalID column in interview_details table conundrum…

*Initially I had **hospitalID** as a column in **interview_details**, thinking it would make things much clearer having an attribute directly linking **interview_details** with **hospital_details**, despite there already being a link from **position_details** to **hospital_details** making it accessible from **interview_details**. I thought that an extra column wouldn't hurt for the sake of clarity, and ease of writing queries. A thought then came to me that, what if there was a million tuples for **hospitalID**, that would have to be entered an extra 1 million times in the **interview_details** table, and although Rome was built on columns due to information redundancy this column had to be removed.*

-In **interview_details** I removed all primary keys bar **interviewID** as the rows were already uniquely identified by the **interviewID** attribute. Having a composite primary key consisting of(**interviewID**, **candidateID**, **positionID**, **hospitalID**) gave no benefit.

Other potential additions include:

-adding extra security encrypting private user data.

-Including timestamp columns to track when records are created or modified could be beneficial for audit purposes

• A discussion of reaction policies used and why they were used.

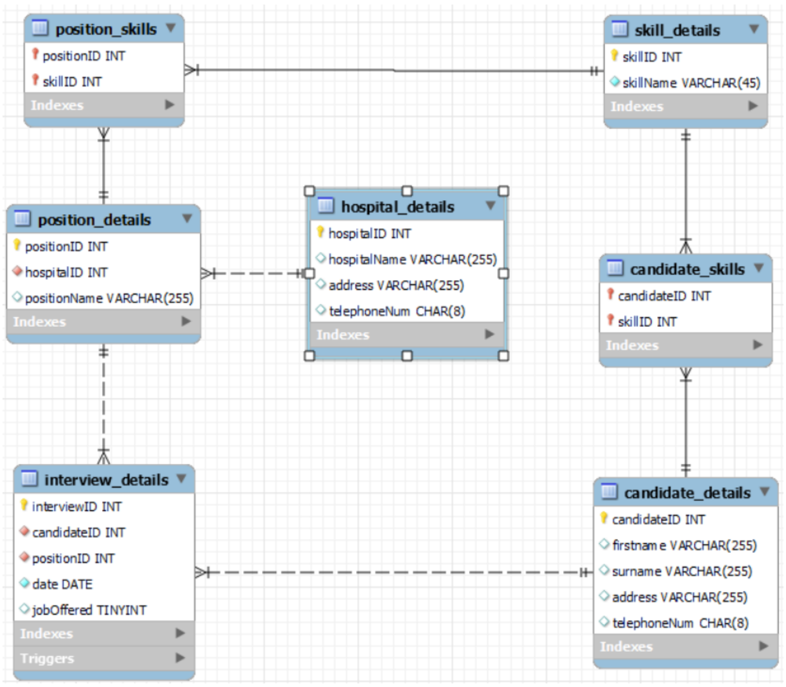
Data Consistency: By employing CASCADE on foreign keys, the reaction policy ensures that if a related parent record is altered or removed, the associated child records are updated or deleted accordingly, maintaining data consistency.

These actions also align with business logic. For instance, when a candidate is removed, it is necessary to delete all their interview details to maintain accurate data representation.

Triggers for **jobOffered** Field: The triggers were employed to restrict the values inserted or updated in the **jobOffered** column to only allow 0 or 1. This policy is intended to enforce data integrity and ensure that only valid values are entered for this field. This had to be done as there is no unit with only 1 bit of memory in mySQL(that I know of), so I restricted the TINYINT unit which can store up to a byte, mimicking a Boolean true/false value.

• The Entity-Relationship (ER) diagram of your database (generated as described in lab Exercise 5).

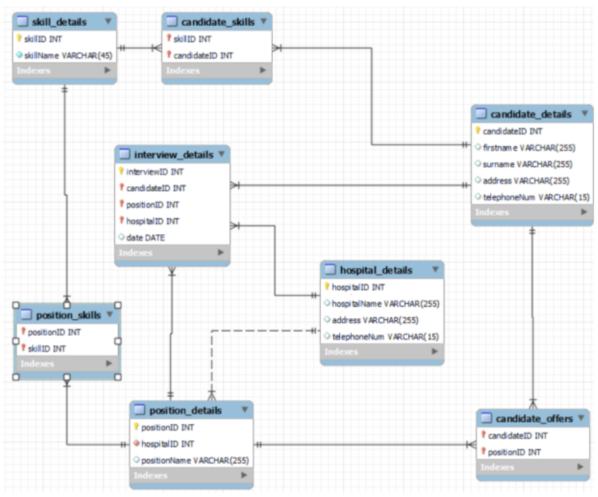
FINAL EER DIAGRAM.

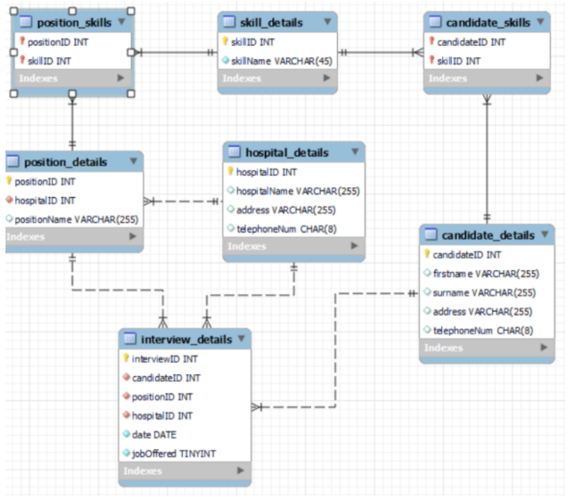


• Please indicate clearly, in your documentation, the operating system you used (Mac, Windows, Ubuntu, etc).

WINDOWS

Some previous EER diagrams showing thought process :


skill_details
skillID INT
skillName VARCHAR(45)
candidate_skills
skillID INT
candidateID INT
candidate_details
candidateID INT
firstname VARCHAR(255)
surname VARCHAR(255)
address VARCHAR(255)
telephoneNum VARCHAR(15)
interview_details
interviewID INT
candidateID INT
positionID INT
hospitalID INT
date DATE
hospital_details
hospitalID INT
hospitalName VARCHAR(255)
address VARCHAR(255)
telephoneNum VARCHAR(15)
position_skills
positionID INT
skillID INT
position_details
positionID INT
hospitalID INT
positionName VARCHAR(255)
candidate_offers
candidateID INT
positionID INT
Indexes


position_skills
positionID INT
skillID INT
skill_details
skillID INT
skillName VARCHAR(45)
candidate_skills
candidateID INT
skillID INT
position_details
positionID INT
hospitalID INT
positionName VARCHAR(255)
hospital_details
hospitalID INT
hospitalName VARCHAR(255)
address VARCHAR(255)
telephoneNum CHAR(8)
candidate_details
candidateID INT
firstname VARCHAR(255)
surname VARCHAR(255)
address VARCHAR(255)
telephoneNum CHAR(8)
interview_details
interviewID INT
candidateID INT
positionID INT
hospitalID INT
date DATE
jobOffered TINYINT
Indexes