COMP20070: Databases and Info. Systems I

2024 / 2025

MySQL DB assignment

Departments Vacancies Database

Simon Lasak

23721209

UCD School of Computer Science,

University College Dublin,

Belfield, Dublin 4.

Supporting Documentation

This document aims to explain my decisions in creating this database, its tables, relationships, procedures, reaction policies used and my assumptions in design.

Description:

The Departments Vacancies Database is a relational database that contains information about vacant positions of a single company, divided into numerous departments.

My database contains 7 tables:

- **candidate_details**, containing essential information about a candidate, candidate's skills are accessed through a many-to-many relationship table -> **candidate_skills** that links the Candidate's ID to a Skill ID in the **skills** table.
- **department_details**
- **interview_details** has 3 attributes that are distinct from any table -> InterviewID (the unique primary key), InterviewDate and the Outcome of the interview that is a Boolean variable which is used to determine whether a candidate was successful in their interview. Outside of these attributes the table contains the CandidateID, DepartmentID and the PositionID as foreign keys.
- **position_details** has unique primary key for each position and with it we can access the **position_skills** table that maps each position to a number of skills that are required, just like in candidate_details -> candidate_skills -> skills relation.

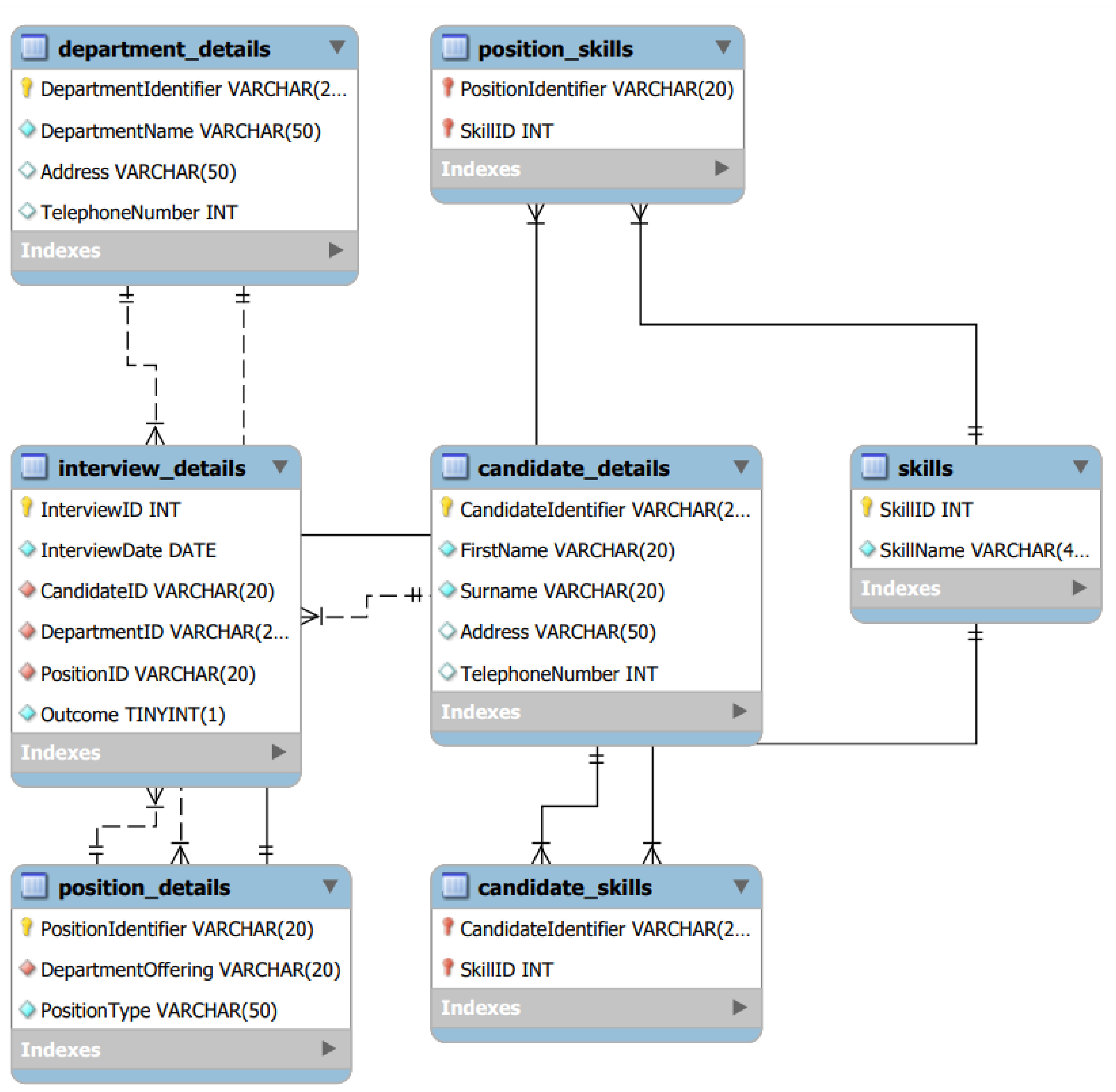
Database population:

- **candidate_details** is populated with 15 distinct candidates from Ireland (15 rows)
- **candidate_skills** gives each candidate 1-8 distinct skills (67 rows)
- **department_details** contains 10 unique departments (10 rows)
- **interview_details** gives details of 35 different interviews (35 rows)
- **position_details** is populated with two positions for each department (20 rows)
- **position_skills** each PositionID gets 2-5 skills from the skill list, ensuring a diverse skill set for each position. (69 row)
- **skills** contains the ID and Name for the 10 basic skills that are used by positions and candidates (10)

Reaction policies

I chose to use the NO ACTION policy in my database to ensure strict data integrity and prevent unintentional changes. This approach stops operations, like deleting a parent row, if they would violate referential constraints, safeguarding dependent child data. It gives us explicit control, allowing my application logic to handle related updates or deletions as needed. By avoiding automatic actions like CASCADE, we reduce the risk of accidental data loss or unintended consequences. This design aligns with our need for precision and oversight in managing critical relationships.

ER Diagram



Additional information

I created my SQL database in Windows OS

Task4.4 – **distinct** keyword makes sure we get each candidate only once (each candidate can have multiple required skills)

Task4.8 do not include candidates that have interviews outside of the given date.

Task4.10 sorted by default

```
Alter TABLE candidate_details

   ADD CHECK (CandidateIdentifier like CAND)

;
```

every candidate identifier starts with CAND

Outcome can be null because we might not know the outcome of an interview in future.