

**Operating system: Windows 10**

**1 Why I chose my tables:**

In the requirements we were told to include candidate details, department details, position details, and interview details. Therefore, I created a candidate details table, a department details table, a position details table, and interview details. I decided that as we were told a candidate can have many skills, and a position could have many required skills, I would create a skills table, which would include the skills relevant to the company. In this way, a candidate won’t have any skills assigned to them which aren’t sought after by the company. Candidate\_skills and position\_skills were created as associative tables, in order to provide many to many relationships. For example, a candidate can have many skills, and skills can be had by many candidates. An associative table was also used to make a many to many relationship between department and position, as a department could have many positions open, and the same position type could be available in many departments. This brought the total table count to 8.

**2 Why I chose the columns in my tables:**

Skills has just the skill name, as I didn’t feel that anything else was needed. I made the skill name the primary key, as I felt that the list of skills used by the company wouldn’t be very large, and mostly static. I could have made the primary key an int, which is a smaller memory footprint than a varchar, but then a skill with a different id number could be added with the same skill but in a different case, for example id:1 skill\_name: admin, id:2 skill\_name: Admin. To avoid this I’d need to make both the id and skill\_name composite primary keys, and then suddenly I’d have a situation where more data is being used to store the primary key than just with skill\_name. Therefore, I chose skill\_name only as the primary id. I was tempted to have position\_details’ primary key to be the position type only following my logic behind the skill table, but I saw in the requirements that position identifier and position type were both mentioned, and I was unsure whether I was allowed to combine these requirements or not, so I included both to be on the safe side.

In interview\_details, I made interview\_id the primary key, as I wanted to ensure that a candidate could have many interviews for a position, a position could have many interviews, and many interviews could happen on the same date. As for recording a job offer, I included a column of varchar length 1, (I didn’t try include char or binary as I wanted to stick to the data types we covered most in the practicals). This column has a trigger which sets the cell to ‘N’ if the entered value isn’t ‘Y’; N for no, position not offered, and Y for yes. This ensures that only N or Y is entered.

In department positions, the primary key is an int, and not a composite key of department\_id and position\_id, as otherwise a department could hire for a position once, for example if the department

**3 Reaction policies:**

I chose cascade for most of my foreign keys. The reasons being:

If a candidate is updated or deleted, then this change should occur throughout the tables; if a candidate ID is changed, then we still want that candidate linked to the right interviews and skills etc. This is true for positions and departments also. Likewise, if the candidate is deleted, there is no point keeping that candidate’s interviews and list of skills. This is the same for position and departments; if they are deleted, then the candidates, interviews, skills and so on related to them become pointless, and thus should be deleted too. As for skills, in the event that a company no longer needs a certain type of skill, for example they sell off their programming side of the business making programming skills undesirable, then the position-skill entry for this should be removed too, as it would be pointless to keep that skill as a requirement when it’s no longer wanted. Also, tuples where a candidate has that skill should be removed too, as we are no longer interested in whether someone has a skill we no longer need.

However, I have used Restrict twice, for the skills foreign keys in candidate\_skills and position\_skills on their On Update policies. This was to stop the scenario where someone might change the skill value in the cell, for example from computer skills to cooking skills, and then everyone with computer skills would now seem to be master chefs according to the database. For the other tables, only ids would change, and as these are primary keys, they couldn’t change to be an existing record, and thus cascading is desired to keep the links. With skills being name only, any changes could have big consequences.

**4 Additional information:**

I added auto increment to the primary id ints of the interview, position, candidate and departments tables, so that the user isn’t required to add them, which will reduce the amount of data the user has to enter, and keep order to the data.

I also chose to have int id only columns in department\_positions, as even though it isn’t the most attractive table to look at, it reduces the amount of data the user needs to enter, and achieves the desired function of creating a many to many relationship.

I chose to make the address columns varchar(100), as I felt many addresses are usually longer than 45 characters.