COMP20240: Assignment 1 - Hospital HR Database

Adam Ryan

November 24, 2020

1 Introduction

As outlined in the MySQLAssignment document, I create a MySQL Database for use in HR system with a particular slant towards a health-context. This database stores candidate details, hospital information, position information, interview details, interview assessment, and skill information. In addition, a number of stored procedures have been developed to meet analytics requirements.

This database was created using macOS Catalina version 10.15.7 and MySQL Workbench Version 8.0.21.

Contents

2 Database Design

This section outlines my database design and why some of the design decisions were made.

2.1 Schema and Design Decisions

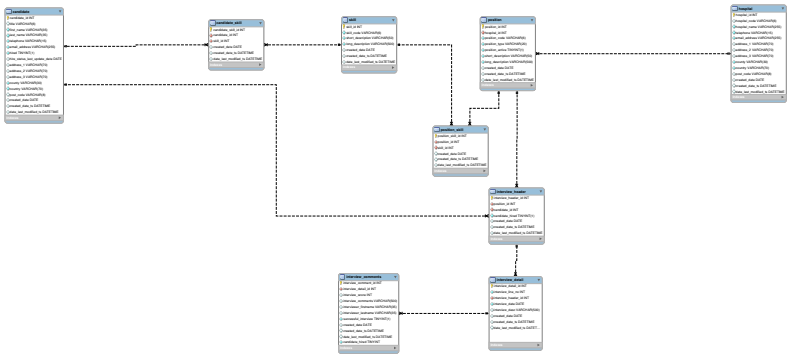

Figure 1: The Schema of my Database

The database at present consists of the following tables:

candidate This table stores candidate information. In particular, it stores the candidates' title (Mr, Ms, Mr, Mrs, Prof, Dr), first name (varchar 35), last name (varchar 35), telephone number (varchar 15), email address (which must be unique among candidate), whether the candidate is hired, when their hire status last changed, three lines of address (varchar 70), the candidates' county, candidates' country, the candidates' post code, and for auditing purposes when the candidate was created (as a shortdate), when the candidate was created (with timestamp), and when the candidate record was last modified. The primary key is candidate_id which is auto-generated as an int. By default, the candidate should be inserted as not hired and this should only be altered via actually getting hired via an interview.

hospital This table stores hospital information. It stores it stores the hospital code (e.g. a business code to identify the hospital), the hospital name, the hospital's email, address information (split as three lines, county, country, and post code), and the create date (as shortdate), create date (with timestamp), and the date the record was last modified with timestamp. The primary key is hospital_id which is an auto-generated int.

position This table stores all of the positions which are advertised. It includes a position code (business code to identify positions), the position type (e.g. what type of position it is, a junior doctor, senior doctor, etc.) a short description of the position (e.g. 'ward doctor', 'assistant to surgeon', etc.), a verbose description (this would be used to pull into HR systems for the lengthy role description), and auditing info on the creation. The primary key is position_id which is an auto-generated int, and it has a foreign key of hospital_id to identify which hospital is advertising the position. The link to hospital is a one-to-many relationship as one hospital may advertise many positions. There is a flag for active positions and although this is not touched at present, is something which should be contained as a flag so as to hide inactive positions.

skill The table outlines a list of skills which are possible both for positions and candidates. Each skill has a skill code (business code to identify skill), a short description which serves as the skill name, and a verbose description which identifies details on the particular skill; in addition there is auditing info. the primary key is skill_id which is an auto-generated int.

candidate_skill This table is used to identify which candidates have what skills. It has foreign keys from the candidate table and skill table as a candidate may have many skill. It has auditing informtion on when the skill was added to the candidate, and has a primary key which is an auto-generated int.

position_skill This table is used to identify which positions require what skills. It has foreign keys from the position table and skill table as a position may require many skills. It has auditing informtion on when the skill was added to the position, and has a primary key which is an auto-generated int.

interview_header This contains information which is common among all interview line items. In particular, things such as whether the candidate was hired at the end of the interview process, which candidate is being interview for what position, and auditing information on when the interview header was created.

interview_detail This contains line item detail for the interview of a candidate. It contains details such as which header it relates to, what interview number it is (interview line number), what date the interview line took place, a description of the interview, and auditing information. It has a primary key which is an auto-generated int, and a foreign key to the interview header item.

interview_comments Interview comments is designed for internal use by HR to comment and give feedback on an interview post-interview. It has information on who is making the comment, the score of the interview, comments of the interview, whether the interview was successful, and whether the candidate was hired within an interview. It has a primary key which is an auto-generated int, and a foreign key with the interview detail to identify which interview number for a candidate the comment is referring to.

All tables in the database have auditing information on creation, and all tables in the database use an auto-generated int as the primary key.

Based on prior experience in solution design, an auto-generated primary key is best practice in developing primary keys as opposed to other business identifiers. As such, this has been implemented for all tables within my schema. As other business codes are often used such as a candidate number, a hospital BU code, etc. as these may change and massive data issues can arise from somebody interfering with the database and adding an overlapping identifier, using an autogenerated INT or BIGINT as the primary key is important for quality reasons. I've included the possibility of business codes being used on tables where this may make sense or be useful to a HR personal.

Email, first name, last name, address split, phone number, and approprate limits are in accordance with best data quality practices (e.g. UK Business Data Limits SOP)

In reality, the County and Country should be foreign keys to a country and county table linking to ISO codes, however for the purposes of this assignment this seemed like overkill and as such

are simply fields within the table.

Candidate information such as address should in a production system not be embedded within a candidate table and should be kept separate as a candidate_address table with a foreign key to candidate id and a flag as to which address is active since one candidate may have multiple addresses, but for the purpose of this assignment these tables are embedded as one.

For the interview table, the transaction header and transaction detail design practice is used. As one candidate may have multiple interviews, and each interview is a detail of an overall application, this design structure seems to be the most sensible implementation as each interview line shares certain common details such as the candidate, whether the candidate was hired at the end of the application, etc. For one position, a candidate should only have one interview header item which would be controlled on an ETL level. As the candidate receives multiple interviews, a detail item is added for each subsequent interview with the line number being an autogenerated int partitioned over the interview header id so as to identify which interview it is for the candidate. The interview comments are designed for internal use by HR to describe each interview the candidate has partaken in, and whether the interview was successful, if the candidate was hired during the interview (only one line should have this per interview application), a score and any comments while capturing the HR Lead responsible for the interview. In reality, a HR lead table with a list of all possible HR people should be added with the HR person details in the interveiw comment table replaced with a FK to the HR user table, however for the purposes of this assignment they have been embedded as one.

Note: Additional development is required to implement the 'complete' the updating of certain flags and characteristics within this design, however I believe these have not yet been covered in the module and are outside the scope of this assignment. In particular, a trigger needs to be added on the interview comments table such that on insert of a line with candidate hired the following tables are updated:

candidate The candidate hired flag should be flagged as true and hire last status change should be flagged as GETDATE()

interview_header The candidate hired flag should be flagged to highlight as part of the interview process the candidate was ultimately hired.

subsequently, the position table should be updated to set the position to inactive if the position is no longer hiring, but as one position may take on many candidates this should be driven from elsewhere.

2.2 Stored Procedures

Two types of stored procedures have been implemented in the database, parametrised inserts, and the stored procedures to meet the requirements of the assignment questions. This section discusses both types of procedures

2.2.1 Inserts

The insert stored procedures are as follows:

ins_candidate This is used to insert a candidate into the candidate table. This procedure allows the user to insert title, first name, surname, phone, email, address line 1 to 3, county, country, and postcode. The primary key is auto-generated so cannot be inserted, a candidate inserted into the table by default is not hired so this cannot be forcably entered, and the auditing info cannot be entered.

ins_hospital This is used to enter details into the hospital table. The hospital code, and hospital details can be inserted, but the auditing info and primary key cannot be.

ins_position This is used to insert open positions. The hospital id, position code, position type, and short and long description of the position can be entered while the primary key and auditing info cannot be.

ins_skill This is used to insert skills into the skill table. A skill code, short description, and verbose description can be entered, while the primary key and auditing informaation cannot be.

candidate_skill This is used to insert candidate skills. The user can enter the candidate ID and skill ID. The user cannot forcibly insert the primary key or auditing info.

ins_position_skill This is used to insert position skills. The user can enter the position ID and skill ID. The user cannot forcibly insert the primary key or auditing info.

ns_interview_header This is used to insert a row into the interview header. The user can insert the position ID, candidate ID, and whether the candidate was hired (techically speaking this should not be entered, should default to 0, and driven through the comments however I've allowed for the possibility that it may be needed to enter interviews that previously happened).

ins_interview_detail This is used to insert a line into the interview detail table. The interview header, interview date, and interview description can be entered. This SP is more complicated than the other inserts as I am automatically calculcating the correct interview line number and using this as an insert. It works by initially declaring a variable called line number. I'm setting this to be coalesce of the max line number from the interview detail table + 1 or 1 of the interview header with the interview header Id. I'm using coalesce so that if no interview detail already exists, the query will return null and thus the line number value will be 1. I'm then using this in the standard insert. As such, it's important that this drives inserts. Ideally, I would lock table prior to entering however MySQL doesn't allow this as part of the query which is a bit silly so instead the table should be locked in whatever calls the query.

interview_comments This is used to enter into the interview comment table. You can enter which interview detail ID it relates to, the interview score, interviewer details, whether it was a successful interview i.e. proceeded to the next round, and if the candidate was hired.

2.2.2 Questions

The question stored procedures are all denoted with sp(n) where n indicates that it's the n-th requested stored procedure.

sp1 This selects all details from the hospital table with a given hospital id.

sp2 This selects all details from the hospital table where the hospital name exactly matches what the user entered.

sp3 This selects all details from the candidate table where the candidate last name matches a last name entered by the user.

sp4 This selects all candidate details where the candidate has at least one skill required by a position. It works by getting the skills required for the position, and inner joining this onto the candidate table and selecting candidates contained in this list.

sp5 Technically there are many ways to do this in my schema, however as these other flags are driven via interview comments as the hired flag here is key. I count the distinct number of candidates flagged as hired from the candidate comments table.

sp6 I assume the skill is being searched for via the short description. I select all positions requiring the skill description.

sp7 Essentially Sp6 however the short description is nursing.

sp8 I select a combination of hospital details and position details order by the hospital primary key. I assume this is what the ordering should be based off of.

sp9 I select various details from the interview tables and candidate tables. There's a left join on comments as an interview may have been scheduled but not yet been commented on. I select details pertaining to all interviews that happened on a certain date.

sp10 I select all distinct candidate ids where candidates were interviewed on one date entered by the user but not on any other date.

sp11_p1 To answer Q11 I've done it two ways as I'm not sure which is needed. In this way, I select the candidate ID and candidate names of people with more than one interview line. This would mean interviews total.

sp11_p2 To answer Q11, I've done it two ways as I'm not sure whch is needed. In this way, I select candidate details of customers who have interviewed for more than one position (e.g. they have two or more distinct interview headers which have at least one line each in interview details). This corresponds with at least two separate applications with interviews which is different from two interviews at all, but I wasn't sure which version was needed.

2.3 Collation

I have used On Update: Collate and On Delete: Restrict across all tables. On Update: Collate as I wish references to remain valid on update. On Delete: Restrict is done as I want to ensure deletion is done purposefully at big header items. For GDPR purposes, as candidate details are stored in

a PROD environment a deletion stored procedure would need to be added to delete from bottom up all candidate details. While this could be avoided using on delete cascade, that can represent a signficant risk to a business and it would be better to fulfill right to be forgotten requests via a purposely built stored procedure which would need to work from interview comments upwards. Many industrial production B2B systems such as Salesforce Core operate similar sps (however on delete cascade can be enabled). Similarly, hospital deletions should not happen. If a hospital is no longer operational it should not be deleted with the corresponding records as this drops analytics info.

3 Assumptions

Key assumptions taken throughout this assignment include the following:

- A candidate can only have one persistent address and should only have a single record in the candidate table. A candidate's email must be unique.
- All tables should have an auto-generated int as a key.
- One candidate may apply for many positions
- One candidate may have many skills.
- One hospital may have many positions available.
- One position may require many skills.
- The candidates and positions use a shared skill-space.
- One candidate per one position may have one application (interview header) but may have many interviews for this application (interview detail).
- A HR personnel will rate each interview, and for each interview state whether the candidate was hired as part of the interview.
- The interview line number should be partitioned over the interview header. For each interview header there should not be any overlap in interview line numbers.
- Once an interivew comment is added flagging the candidate as hired, the interview header table and candidate table will be updated to flag that the candidate was hired.
- If the position is no longer active, the flag will be updated to highlight that the position is no longer open to applications.
- In the insert stored procedures, there are certain items which the user should not be able to force via an insert. These include items such as not being able to overwrite the create date and date last modified fields, not being able to force insert a primary key into any of the tables. Outlined in section above.
- In SP6 I assume it's based on the skill short description.
- In SP8 I assumed it's based on the hospital ID as opposed to hospital name that it should be ordered by.

- In Sp10, I assume it is looking for candidates who interviewed on one date entered by the user e.g. 13th February but have not been interviewed on any other date. I select the list of candidate IDs who meet this criteria (interviewed on one date and one date only).
- The ETL process will drive most data quality checks and ensure population of data in the correct order (hospital, skill, then candidate and candidate_skill, then position and position_skill, then interview_header and at least one interview_detail, followed by interview_comment and any subsequent interview_detail)